



Creating A Single Global Electronic Market

1 **Message Service Specification**

2 **Version 2.0 rev C**

3 **OASIS ebXML Messaging Services Technical Committee**

4 21 February 2002

5 **Status of this Document**

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

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

10 Note: Implementers of this specification should consult the OASIS ebXML Messaging Services Technical
11 Committee web site for current status and revisions to the specification
12 (<http://www.oasis-open.org/committees/ebxml-msg/>).

13 *Specification*

14 Version 1.0 of this Technical Specification document was approved by the ebXML Plenary in May 2001.

15 Version 2.0 of this Technical Specification document was approved by the OASIS Messaging Team as a
16 Technical Committee(TC) Specification, January 22, 2002.

17 Version 2.0 of this Technical Specification document is presented to the OASIS membership for
18 consideration as an OASIS Technical Specification, April 2002.

19 *This version*

20 V2.0 – http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf

21 *Errata to this version*

22 V2.0 – http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0_errata.html

23 *Previous version*

24 V1.0 – <http://www.ebxml.org/specs/ebMS.doc>

25 **ebXML Participants**

26 The authors wish to acknowledge the support of the members of the Messaging Services Team who
27 contributed ideas, comments and text to this specification by the group's discussion eMail list, on
28 conference calls and during face-to-face meetings.

Arvola Chan	RosettaNet/TIBCO	Doug Bunting	Sun Microsystems, Inc
Aynur Unal	E2Open	Himagiri Mukkamala	Sybase
Bob Miller	GE Global eXchange	Ian Jones	British Telecom
Brad Lund	Intel™ Corporation	Jeff Turpin	Cyclone Commerce
Brian Gibb	Sterling Commerce	Jim Hughes	Hewlett Packard
Bruce Pedretti	Hewlett-Packard	Kazunori Iwasa	Fujitsu Limited
Cedrec Vessell	DISA	Martin Sachs	IBM Research
Chris Ferris	Sun Microsystems, Inc	Pete Wenzel	RosettaNet/SeeBeyond
Cliff Collins	Sybase	Philippe DeSmedt	Agentis Software
Colleen Evans	Sonic Software	Prasad Yendluri	WebMethods
Jim Galvin	Drummond Group	Ralph Berwanger	BTrade
Dale Moberg	Cyclone Commerce	Sanjay Cherian	Sterling Commerce
Daniel Weinreb	eXcelon	Scott Hinkelman	IBM
David Burdett	Commerce One	Sinisa Zimek	SAP
David Fischer	Drummond Group	Yukinori Saito	Ecom
Dick Brooks	Systrends, Inc		

29

29 The UN/CEFACT-OASIS v1.0 Team – see Acknowledgments

30 Table of Contents

31	Status of this Document	2
32	ebXML Participants	2
33	Introduction	6
34	1 Summary of Contents of this Document	6
35	1.1.1 Document Conventions	7
36	1.1.2 Audience	7
37	1.1.3 Caveats and Assumptions	7
38	1.1.4 Related Documents	7
39	1.2 Concept of Operation	8
40	1.2.1 Scope	8
41	1.2.2 Background and Objectives	8
42	1.2.3 Operational Policies and Constraints	9
43	1.2.4 Modes of Operation	10
44	1.3 Minimal Requirements for Conformance	11
45	Part I. Core Functionality	12
46	2 ebXML with SOAP	12
47	2.1 Packaging Specification	12
48	2.1.1 SOAP Structural Conformance	13
49	2.1.2 Message Package	13
50	2.1.3 Header Container	13
51	2.1.4 Payload Container	14
52	2.1.5 Additional MIME Parameters	14
53	2.1.6 Reporting MIME Errors	15
54	2.2 XML Prolog	15
55	2.2.1 XML Declaration	15
56	2.2.2 Encoding Declaration	15
57	2.3 ebXML SOAP Envelope extensions	15
58	2.3.1 Namespace pseudo attribute	15
59	2.3.2 xsi:schemaLocation attribute	15
60	2.3.3 SOAP Header Element	16
61	2.3.4 SOAP Body Element	16
62	2.3.5 ebXML SOAP Extensions	16
63	2.3.6 #wildcard Element Content	17
64	2.3.7 id attribute	17
65	2.3.8 version attribute	17
66	2.3.9 SOAP mustUnderstand attribute	18
67	2.3.10 ebXML "Next MSH" actor URI	18
68	2.3.11 ebXML "To Party MSH" actor URI	18
69	3 Core Extension Elements	18
70	3.1 MessageHeader Element	18
71	3.1.1 From and To Elements	19
72	3.1.2 CPAlid Element	19
73	3.1.3 ConversationId Element	20
74	3.1.4 Service Element	20
75	3.1.5 Action Element	21
76	3.1.6 MessageData Element	21
77	3.1.7 DuplicateElimination Element	22
78	3.1.8 Description Element	22
79	3.1.9 MessageHeader Sample	22
80	3.2 Manifest Element	22
81	3.2.1 Reference Element	23
82	3.2.2 Manifest Validation	23
83	3.2.3 Manifest Sample	24
84	4 Core Modules	24
85	4.1 Security Module	24
86	4.1.1 Signature Element	24
87	4.1.2 Security and Management	25
88	4.1.3 Signature Generation	25
89	4.1.4 Countermeasure Technologies	27

90	4.1.5	Security Considerations	28
91	4.2	Error Handling Module	29
92	4.2.2	Types of Errors	29
93	4.2.3	ErrorList Element	30
94	4.2.4	Implementing Error Reporting and Handling	32
95	4.3	SyncReply Module	33
96	4.3.1	SyncReply Element	33
97	5	Combining ebXML SOAP Extension Elements	33
98	5.1.1	MessageHeader Element Interaction	34
99	5.1.2	Manifest Element Interaction	34
100	5.1.3	Signature Element Interaction	34
101	5.1.4	ErrorList Element Interaction	34
102	5.1.5	SyncReply Element Interaction	34
103		Part II. Additional Features	35
104	6	Reliable Messaging Module	35
105	6.1	Persistent Storage and System Failure	35
106	6.2	Methods of Implementing Reliable Messaging	35
107	6.3	Reliable Messaging SOAP Header Extensions	36
108	6.3.1	AckRequested Element	36
109	6.3.2	Acknowledgment Element	37
110	6.4	Reliable Messaging Parameters	38
111	6.4.1	DuplicateElimination	38
112	6.4.2	AckRequested	39
113	6.4.3	Retries	39
114	6.4.4	RetryInterval	39
115	6.4.5	TimeToLive	39
116	6.4.6	PersistDuration	39
117	6.4.7	syncReplyMode	39
118	6.5	ebXML Reliable Messaging Protocol	40
119	6.5.1	Sending Message Behavior	40
120	6.5.2	Receiving Message Behavior	40
121	6.5.3	Generating an Acknowledgment Message	41
122	6.5.4	Resending Lost Application Messages	41
123	6.5.5	Resending Acknowledgments	42
124	6.5.6	Duplicate Message Handling	43
125	6.5.7	Failed Message Delivery	43
126	6.6	Reliable Messaging Combinations	44
127	7	Message Status Service	44
128	7.1	Message Status Messages	45
129	7.1.1	Message Status Request Message	45
130	7.1.2	Message Status Response Message	45
131	7.1.3	Security Considerations	45
132	7.2	StatusRequest Element	45
133	7.2.1	RefToMessageId Element	46
134	7.2.2	StatusRequest Sample	46
135	7.2.3	StatusRequest Element Interaction	46
136	7.3	StatusResponse Element	46
137	7.3.1	RefToMessageId Element	46
138	7.3.2	Timestamp Element	46
139	7.3.3	messageStatus attribute	46
140	7.3.4	StatusResponse Sample	47
141	7.3.5	StatusResponse Element Interaction	47
142	8	Message Service Handler Ping Service	47
143	8.1	Message Service Handler Ping Message	47
144	8.2	Message Service Handler Pong Message	48
145	8.3	Security Considerations	49
146	9	MessageOrder Module	49
147	9.1	MessageOrder Element	49
148	9.1.1	SequenceNumber Element	49
149	9.1.2	MessageOrder Sample	50
150	9.2	MessageOrder Element Interaction	50
151	10	Multi-Hop Module	50
152	10.1	Multi-hop Reliable Messaging	51
153	10.1.1	AckRequested Sample	51

154	10.1.2	Acknowledgment Sample.....	51
155	10.1.3	Multi-Hop Acknowledgments.....	51
156	10.1.4	Signing Multi-Hop Acknowledgments.....	52
157	10.1.5	Multi-Hop Security Considerations.....	52
158	10.2	Message Ordering and Multi-Hop.....	52
159	Part III. Normative Appendices.....		53
160	Appendix A	The ebXML SOAP Extension Elements Schema.....	53
161	Appendix B	Communications Protocol Bindings.....	58
162	B.1	Introduction.....	58
163	B.2	HTTP.....	58
164	B.2.1	Minimum level of HTTP protocol.....	58
165	B.2.2	Sending ebXML Service messages over HTTP.....	58
166	B.2.3	HTTP Response Codes.....	59
167	B.2.4	SOAP Error conditions and Synchronous Exchanges.....	60
168	B.2.5	Synchronous vs. Asynchronous.....	60
169	B.2.6	Access Control.....	60
170	B.2.7	Confidentiality and Transport Protocol Level Security.....	60
171	B.3	SMTP.....	61
172	B.3.1	Minimum Level of Supported Protocols.....	61
173	B.3.2	Sending ebXML Messages over SMTP.....	61
174	B.3.3	Response Messages.....	63
175	B.3.4	Access Control.....	63
176	B.3.5	Confidentiality and Transport Protocol Level Security.....	63
177	B.3.6	SMTP Model.....	63
178	B.4	Communication Errors during Reliable Messaging.....	64
179	Appendix C	Supported Security Services.....	65
180	References.....		67
181	Normative References.....		67
182	Non-Normative References.....		68
183	Contact Information.....		69
184	Acknowledgments.....		70
185	Disclaimer.....		70
186	Copyright Statement.....		70
187			

187 Introduction

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

192 This specification focuses on defining a communications-protocol neutral method for exchanging
193 electronic business messages. It defines specific enveloping constructs supporting reliable, secure
194 delivery of business information. Furthermore, the specification defines a flexible enveloping technique,
195 permitting messages to contain payloads of any format type. This versatility ensures legacy electronic
196 business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or HL7) can leverage the
197 advantages of the ebXML infrastructure along with users of emerging technologies.

198 1 Summary of Contents of this Document

199 This specification defines the *ebXML Message Service Protocol* enabling the secure and reliable
200 exchange of messages between two parties. It includes descriptions of:

- 201 • the ebXML Message structure used to package payload data for transport between parties,
- 202 • the behavior of the Message Service Handler sending and receiving those messages over a data
203 communications protocol.

204 This specification is independent of both the payload and the communications protocol used. Appendices
205 to this specification describe how to use this specification with HTTP [RFC2616] and SMTP [RFC2821].

206 This specification is organized around the following topics:

207 Core Functionality

- 208 • **Packaging Specification** – A description of how to package an ebXML Message and its associated parts
209 into a form that can be sent using a communications protocol such as HTTP or SMTP (section 2.1),
- 210 • **ebXML SOAP Envelope Extensions** – A specification of the structure and composition of the information
211 necessary for an *ebXML Message Service* to generate or process an ebXML Message (section 2.3),
- 212 • **Error Handling** – A description of how one *ebXML Message Service* reports errors it detects to another
213 ebXML Message Service Handler (section 4.2),
- 214 • **Security** – Provides a specification of the security semantics for ebXML Messages (section 4.1),
- 215 • **SyncReply** – Indicates to the Next MSH whether or not replies are to be returned synchronously (section
216 4.3).

217 Additional Features

- 218 • **Reliable Messaging** – The Reliable Messaging function defines an interoperable protocol where any two
219 Message Service implementations can reliably exchange messages sent using once-and-only-once delivery
220 semantics (section 6),
- 221 • **Message Status Service** – A description of services enabling one service to discover the status of another
222 Message Service Handler (MSH) or an individual message (section 7 and 8),
- 223 • **Message Order** – The Order of message receipt by the *To Party MSH* can be guaranteed (section 9),
- 224 • **Multi-Hop** – Messages may be sent through intermediary MSH nodes (section 10).

225 Appendices to this specification cover the following:

- 226 • **Appendix A Schema** – This normative appendix contains XML schema definition [XMLSchema] for the
227 ebXML SOAP *Header* and *Body* Extensions,
- 228 • **Appendix B Communications Protocol Envelope Mappings** – This normative appendix describes how to
229 transport *ebXML Message Service* compliant messages over HTTP and SMTP,
- 230 • **Appendix C Security Profiles** – a discussion concerning Security Service Profiles.

231 **1.1.1 Document Conventions**

232 Terms in *Italics* are defined in the ebXML Glossary of Terms [ebGLOSS]. Terms listed in **Bold Italics**
233 represent the element and/or attribute content. Terms listed in `Courier` font relate to MIME
234 components. Notes are listed in Times New Roman font and are informative (non-normative). Attribute
235 names begin with lowercase. Element names begin with Uppercase.

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

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

257 **1.1.2 Audience**

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

260 **1.1.3 Caveats and Assumptions**

261 It is assumed the reader has an understanding of communications protocols, MIME, XML, SOAP, SOAP
262 Messages with Attachments and security technologies.

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

265 It is strongly RECOMMENDED implementors read and understand the Collaboration Protocol Profile/
266 Agreement [ebCPP] specification and its implications prior to implementation.

267 **1.1.4 Related Documents**

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

- 270 • **ebXML Technical Architecture Specification [ebTA]** – defines the overall technical architecture for ebXML
- 271 • **ebXML Technical Architecture Risk Assessment Technical Report [secRISK]** – defines the security
272 mechanisms necessary to negate anticipated, selected threats
- 273 • **ebXML Collaboration Protocol Profile and Agreement Specification [ebCPP]** – defines how one party
274 can discover and/or agree upon the information the party needs to know about another party prior to sending
275 them a message that complies with this specification
- 276 • **ebXML Registry/Repository Services Specification [ebRS]** – defines a registry service for the ebXML
277 environment

278 1.2 Concept of Operation

279 1.2.1 Scope

280 The ebXML Message Service (ebMS) defines the message enveloping and header document schema
281 used to transfer ebXML messages over a communications protocol such as HTTP or SMTP and the
282 behavior of software sending and receiving ebXML messages. The ebMS is defined as a set of layered
283 extensions to the base Simple Object Access Protocol [SOAP] and SOAP Messages with Attachments
284 [SOAPAttach] specifications. This document provides security and reliability features necessary to
285 support international electronic business. These security and reliability features are not provided in the
286 SOAP or SOAP with Attachments specifications.

287 The ebXML infrastructure is composed of several independent, but related, components. Specifications
288 for the individual components are fashioned as stand-alone documents. The specifications are totally
289 self-contained; nevertheless, design decisions within one document can and do impact the other
290 documents. Considering this, the ebMS is a closely coordinated definition for an ebXML message service
291 handler (MSH).

292 The ebMS provides the message packaging, routing and transport facilities for the ebXML infrastructure.
293 The ebMS is not defined as a physical component, but rather as an abstraction of a process. An
294 implementation of this specification could be delivered as a wholly independent software application or an
295 integrated component of some larger business process.

296 1.2.2 Background and Objectives

297 Traditional business information exchanges have conformed to a variety of standards-based syntaxes.
298 These exchanges were largely based on electronic data interchange (EDI) standards born out of
299 mainframe and batch processing. Some of the standards defined bindings to specific communications
300 protocols. These EDI techniques worked well; however, they were difficult and expensive to implement.
301 Therefore, use of these systems was normally limited to large enterprises possessing mature information
302 technology capabilities.

303 The proliferation of XML-based business interchanges served as the catalyst for defining a new global
304 paradigm that ensured all business activities, regardless of size, could engage in electronic business
305 activities. The prime objective of ebMS is to facilitate the exchange of electronic business messages
306 within an XML framework. Business messages, identified as the 'payloads' of the ebXML messages, are
307 not necessarily expressed in XML. XML-based messages, as well as traditional EDI formats, are
308 transported by the ebMS. Actually, the ebMS payload can take any digital form—XML, ASC X12, HL7,
309 AIAG E5, database tables, binary image files, etc.

310 The ebXML architecture requires that the ebXML Message Service protocol be capable of being carried
311 over any available communications protocol. Therefore, this document does not mandate use of a
312 specific communications protocol. This version of the specification provides bindings to HTTP and SMTP,
313 but other protocols can, and reasonably will, be used.

314 The ebXML Requirements Specification [ebREQ] mandates the need for secure, reliable
315 communications. The ebXML work focuses on leveraging existing and emerging technology—attempts to
316 create new protocols are discouraged. Therefore, this document defines security within the context of
317 existing security standards and protocols. Those requirements satisfied with existing standards are
318 specified in the ebMS, others must be deferred until new technologies or standards are available, for
319 example encryption of individual message header elements.

320 Reliability requirements defined in the ebREQ relate to delivery of ebXML messages over the
321 communications channels. The ebMS provides mechanisms to satisfy the ebREQ requirements. The
322 reliable messaging elements of the ebMS supply reliability to the communications layer; they are not
323 intended as business-level acknowledgments to the applications supported by the ebMS. This is an
324 important distinction. Business processes often anticipate responses to messages they generate. The
325 responses may take the form of a simple acknowledgment of message receipt by the application
326 receiving the message or a companion message reflecting action on the original message. Those
327 messages are outside of the MSH scope. The acknowledgment defined in this specification does not

328 indicate the payload of the ebXML message was syntactically correct. It does not acknowledge the
329 accuracy of the payload information. It does not indicate business acceptance of the information or
330 agreement with the content of the payload. The ebMS is designed to provide the sender with the
331 confidence the receiving MSH has received the message securely and intact.

332 The underlying architecture of the MSH assumes messages are exchanged between two ebMS-
333 compliant MSH nodes. This pair of MSH nodes provides a hop-to-hop model extended as required to
334 support a multi-hop environment. The multi-hop environment allows the next destination of the message
335 to be an intermediary MSH other than the 'receiving MSH' identified by the original sending MSH. The
336 ebMS architecture assumes the sender of the message MAY be unaware of the specific path used to
337 deliver a message. However, it MUST be assumed the original sender has knowledge of the final
338 recipient of the message and the first of one or more intermediary hops.

339 The MSH supports the concept of 'quality of service.' The degree of service quality is controlled by an
340 agreement existing between the parties directly involved in the message exchange. In practice, multiple
341 agreements may be required between the two parties. The agreements might be tailored to the particular
342 needs of the business exchanges. For instance, business partners may have a contract defining the
343 message exchanges related to buying products from a domestic facility and another defining the
344 message exchanges for buying from an overseas facility. Alternatively, the partners might agree to follow
345 the agreements developed by their trade association. Multiple agreements may also exist between the
346 various parties handling the message from the original sender to the final recipient. These agreements
347 could include:

- 348 • an agreement between the MSH at the message origination site and the MSH at the final destination; and
- 349 • agreement between the MSH at the message origination site and the MSH acting as an intermediary; and
- 350 • an agreement between the MSH at the final destination and the MSH acting as an intermediary. There
351 would, of course, be agreements between any additional intermediaries; however, the originating site MSH
352 and final destination MSH MAY have no knowledge of these agreements.

353 An ebMS-compliant MSH shall respect the in-force agreements between itself and any other ebMS-
354 compliant MSH with which it communicates. In broad terms, these agreements are expressed as
355 Collaboration Protocol Agreements (CPA). This specification identifies the information that must be
356 agreed. It does not specify the method or form used to create and maintain these agreements. It is
357 assumed, in practice, the actual content of the contracts may be contained in initialization/configuration
358 files, databases, or XML documents complying with the ebXML Collaboration Protocol Profile and
359 Agreement Specification [ebCPP].

360 **1.2.3 Operational Policies and Constraints**

361 The ebMS is a service logically positioned between one or more business applications and a
362 communications service. This requires the definition of an abstract service interface between the
363 business applications and the MSH. This document acknowledges the interface, but does not provide a
364 definition for the interface. Future versions of the ebMS MAY define the service interface structure.

365 Bindings to two communications protocols are defined in this document; however, the MSH is specified
366 independent of any communications protocols. While early work focuses on HTTP for transport, no
367 preference is being provided to this protocol. Other protocols may be used and future versions of the
368 specification may provide details related to those protocols.

369 The ebMS relies on external configuration information. This information is determined either through
370 defined business processes or trading partner agreements. These data are captured for use within a
371 Collaboration Protocol Profile (CPP) or Collaboration Protocol Agreement (CPA). The ebXML
372 Collaboration Protocol Profile and Agreement Specification [ebCPP] provides definitions for the
373 information constituting the agreements. The ebXML architecture defines the relationship between this
374 component of the infrastructure and the ebMS. As regards the MSH, the information composing a
375 CPP/CPA must be available to support normal operation. However, the method used by a specific
376 implementation of the MSH does not mandate the existence of a discrete instance of a CPA. The CPA is
377 expressed as an XML document. Some implementations may elect to populate a database with the
378 information from the CPA and then use the database. This specification does not prescribe how the CPA

379 information is derived, stored, or used: it only states specific information items must be available for the
380 MSH to achieve successful operations.

381 1.2.4 Modes of Operation

382 This specification does not mandate how the MSH will be installed within the overall ebXML framework. It
383 is assumed some MSH implementations will not implement all functionality defined in this specification.
384 For instance, a set of trading partners may not require reliable messaging services; therefore, no reliable
385 messaging capabilities exist within their MSH. But, all MSH implementations shall comply with the
386 specification with regard to the functions supported in the specific implementation and provide error
387 notifications for functionality requested but not supported. Documentation for a MSH implementation
388 SHALL identify all ebMS features not satisfied in the implementation.

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

392 *Figure 1* depicts a logical arrangement of the functional
393 modules existing within one possible implementation of the
394 *ebXML Message Services* architecture. These modules are
395 arranged in a manner to indicate their inter-relationships
396 and dependencies.

397 **Header Processing** – the creation of the ebXML Header
398 elements for the *ebXML Message* uses input from the
399 application, passed through the Message Service Interface,
400 information from the *Collaboration Protocol Agreement*
401 governing the message, and generated information such as
402 digital signature, timestamps and unique identifiers.

403 **Header Parsing** – extracting or transforming information
404 from a received ebXML Header element into a form suitable
405 for processing by the MSH implementation.

406 **Security Services** – digital signature creation and
407 verification, encryption, authentication and authorization.
408 These services MAY be used by other components of the
409 MSH including the Header Processing and Header Parsing
410 components.

411 **Reliable Messaging Services** – handles the delivery and
412 acknowledgment of ebXML Messages. The service
413 includes handling for persistence, retry, error notification
414 and acknowledgment of messages requiring reliable
415 delivery.

416 **Message Packaging** – the final enveloping of an *ebXML*
417 *Message* (ebXML header elements and payload) into its
418 SOAP Messages with Attachments [SOAPAttach] container.

419 **Error Handling** – this component handles the reporting of
420 errors encountered during MSH or Application processing of
421 a message.

422 **Message Service Interface** – an abstract service interface
423 applications use to interact with the MSH to send and
424 receive messages and which the MSH uses to interface
425 with applications handling received messages (Delivery
426 Module).

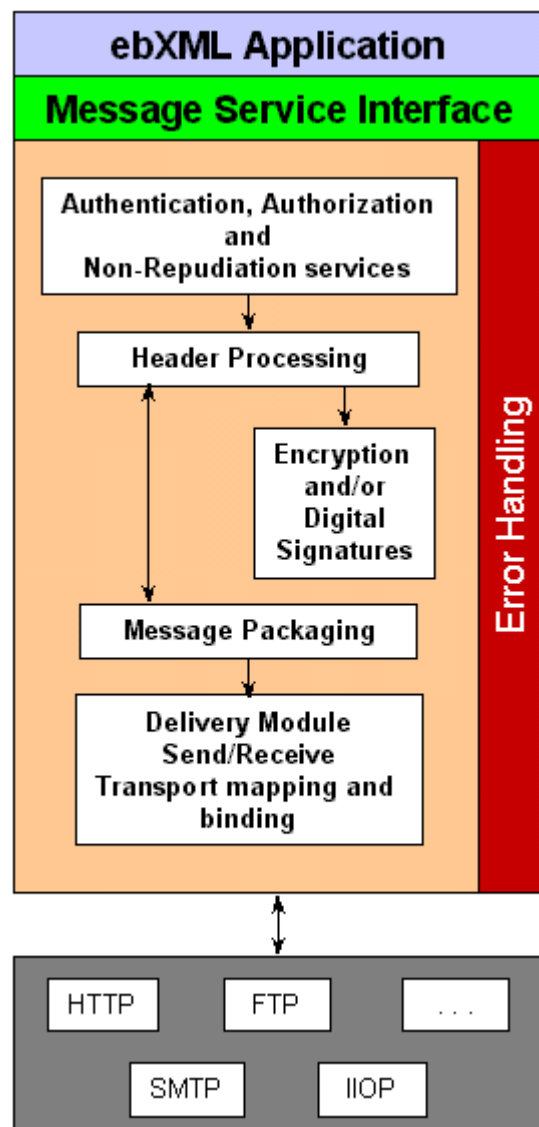


Figure 1.1 Typical Relationship between ebXML Message Service Handler Components

427 **1.3 Minimal Requirements for Conformance**

428 An implementation of this specification **MUST** satisfy **ALL** of the following conditions to be considered a
429 conforming implementation:

- 430 • It supports all the mandatory syntax, features and behavior (as identified by the [RFC2119] key words
431 **MUST**, **MUST NOT**, **REQUIRED**, **SHALL** and **SHALL NOT**) defined in Part I – Core Functionality.
- 432 • It supports all the mandatory syntax, features and behavior defined for each of the additional module(s),
433 defined in Part II – Additional Features, the implementation has chosen to implement.
- 434 • It complies with the following interpretation of the keywords **OPTIONAL** and **MAY**: When these keywords
435 apply to the behavior of the implementation, the implementation is free to support these behaviors or not, as
436 meant in [RFC2119]. When these keywords apply to message contents relevant to a module of features, a
437 conforming implementation of such a module **MUST** be capable of processing these optional message
438 contents according to the described ebXML semantics.
- 439 • If it has implemented optional syntax, features and/or behavior defined in this specification, it **MUST** be
440 capable of interoperating with another implementation that has not implemented the optional syntax,
441 features and/or behavior. It **MUST** be capable of processing the prescribed failure mechanism for those
442 optional features it has chosen to implement.
- 443 • It is capable of interoperating with another implementation that has chosen to implement optional syntax,
444 features and/or behavior, defined in this specification, it has chosen not to implement. Handling of
445 unsupported features **SHALL** be implemented in accordance with the prescribed failure mechanism defined
446 for the feature.

447 More details on Conformance to this specification – conformance levels or profiles and on their
448 recommended implementation – are described in a companion document, "*Message Service
449 Implementation Guidelines*" from the OASIS ebXML Implementation, Interoperability and Conformance
450 (IIC) Technical Committee.

451

Part I. Core Functionality

452

2 ebXML with SOAP

453 The ebXML Message Service Specification defines a set of namespace-qualified SOAP **Header** and
 454 **Body** element extensions within the SOAP **Envelope**. These are packaged within a MIME multipart to
 455 allow payloads or attachments to be included with the SOAP extension elements. In general, separate
 456 ebXML SOAP extension elements are used where:

- 457 • different software components may be used to generate ebXML SOAP extension elements,
- 458 • an ebXML SOAP extension element is not always present or,
- 459 • the data contained in the ebXML SOAP extension element MAY be digitally signed separately from the other
 460 ebXML SOAP extension elements.

2.1 Packaging Specification

462 An ebXML Message is a communications protocol independent MIME/Multipart message envelope,
 463 structured in compliance with the SOAP Messages with Attachments [SOAPAttach] specification, referred
 464 to as a *Message Package*.

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

- 466 • The first MIME part, referred to as the *Header Container*, containing one SOAP 1.1 compliant
 467 message. This XML document is referred to as a *SOAP Message* for the remainder of this
 468 specification,
 469
 470
- 471 • zero or more additional MIME parts, referred to as *Payload Containers*, containing application
 472 level payloads.
 473

474 The general structure and composition of an ebXML
 475 Message is described in the following figure (2.1).

476

477 The *SOAP Message* is an XML document consisting
 478 of a SOAP **Envelope** element. This is the root
 479 element of the XML document representing a *SOAP
 480 Message*. The SOAP **Envelope** element consists of:

- 481 • One SOAP **Header** element. This is a generic
 482 mechanism for adding features to a *SOAP
 483 Message*, including ebXML specific header
 484 elements.
- 485 • One SOAP **Body** element. This is a container for
 486 message service handler control data and
 487 information related to the payload parts of the
 488 message.

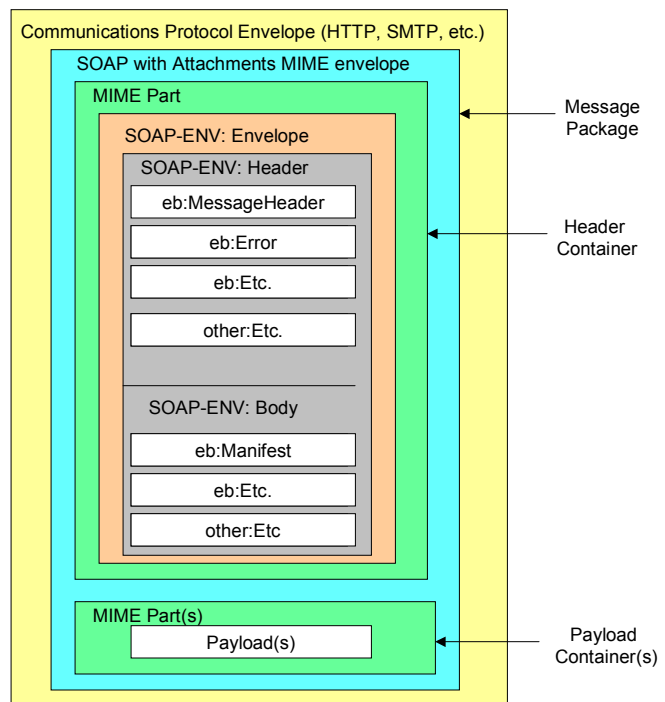


Figure 2.1 ebXML Message Structure

489 2.1.1 SOAP Structural Conformance

490 The *ebXML Message* packaging complies with the following specifications:

- 491 • Simple Object Access Protocol (SOAP) 1.1 [SOAP]
- 492 • SOAP Messages with Attachments [SOAPAttach]

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

495 2.1.2 Message Package

496 All MIME header elements of the *Message Package* are in conformance with the SOAP Messages with
497 Attachments [SOAPAttach] specification. In addition, the `Content-Type` MIME header in the *Message*
498 *Package* contain a `type` attribute matching the MIME media type of the MIME body part containing the
499 *SOAP Message* document. In accordance with the [SOAP] specification, the MIME media type of the
500 *SOAP Message* has the value "text/xml".

501 It is strongly RECOMMENDED the initial headers contain a `Content-ID` MIME header structured in
502 accordance with MIME [RFC2045], and in addition to the required parameters for the Multipart/Related
503 media type, the `start` parameter (OPTIONAL in MIME Multipart/Related [RFC2387]) always be present.
504 This permits more robust error detection. The following fragment is an example of the MIME headers for
505 the multipart/related *Message Package*:

```
506 Content-Type: multipart/related; type="text/xml"; boundary="boundaryValue";  
507 start=messagepackage-123@example.com  
508  
509 --boundaryValue  
510 Content-ID: <messagepackage-123@example.com>
```

511 Implementations MUST support non-multipart messages, which may occur when there are no ebXML
512 payloads. An ebXML message with no payload may be sent either as a plain SOAP message or as a
513 [SOAPAttach] multipart message with only one body part.

514 2.1.3 Header Container

515 The root body part of the *Message Package* is referred to in this specification as the *Header Container*.
516 The *Header Container* is a MIME body part consisting of one *SOAP Message* as defined in the SOAP
517 Messages with Attachments [SOAPAttach] specification.

518 2.1.3.1 Content-Type

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

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

523 2.1.3.2 charset attribute

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

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

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

533 2.1.3.3 Header Container Example

534 The following fragment represents an example of a *Header Container*:

```

535 Content-ID: <messagepackage-123@example.com> --- | Header
536 Content-Type: text/xml; charset="UTF-8"
537
538 <SOAP:Envelope -- | SOAP Message
539   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
540   <SOAP:Header>
541     ...
542   </SOAP:Header>
543   <SOAP:Body>
544     ...
545   </SOAP:Body>
546 </SOAP:Envelope> -- |
547
548 --boundaryValue --- |

```

549 2.1.4 Payload Container

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

552 If the *Message Package* contains an application payload, it SHOULD be enclosed within a *Payload*
 553 *Container*.

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

556 The contents of each *Payload Container* MUST be identified in the ebXML Message **Manifest** element
 557 within the SOAP **Body** (see section 3.2).

558 The ebXML Message Service Specification makes no provision, nor limits in any way, the structure or
 559 content of application payloads. Payloads MAY be simple-plain-text objects or complex nested multipart
 560 objects. The specification of the structure and composition of payload objects is the prerogative of the
 561 organization defining the business process or information exchange using the *ebXML Message Service*.

562 2.1.4.1 Example of a Payload Container

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

```

564 Content-ID: <domainname.example.com> ----- | ebXML MIME
565 Content-Type: application/xml ----- |
566
567 <Invoice> ----- |
568   <Invoicedata> ----- | Payload
569     ...
570   </Invoicedata> ----- |
571 </Invoice> ----- |

```

572 Note: It might be noticed the content-type used in the preceding example (application/XML) is different than the
 573 content-type in the example SOAP envelope in section 2.1.2 above (text/XML). The SOAP 1.1 specification states
 574 the content-type used for the SOAP envelope MUST be 'text/xml'. However, many MIME experts disagree with
 575 the choice of the primary media type designation of 'text/*' for XML documents as most XML is not "human
 576 readable" in the sense the MIME designation of 'text' was meant to infer. They believe XML documents should be
 577 classified as 'application/XML'.

578 2.1.5 Additional MIME Parameters

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

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

584 **2.1.6 Reporting MIME Errors**

585 If a MIME error is detected in the *Message Package* then it MUST be reported as specified in SOAP with
586 Attachments [SOAPAttach].

587 **2.2 XML Prolog**

588 The SOAP *Message*'s XML Prolog, if present, MAY contain an XML declaration. This specification has
589 defined no additional comments or processing instructions appearing in the XML prolog. For example:

```
590 Content-Type: text/xml; charset="UTF-8"  
591  
592 <?xml version="1.0" encoding="UTF-8"?>
```

593 **2.2.1 XML Declaration**

594 The XML declaration MAY be present in a SOAP *Message*. If present, it MUST contain the version
595 specification required by the XML Recommendation [XML] and MAY contain an encoding declaration.
596 The semantics described below MUST be implemented by a compliant *ebXML Message Service*.

597 **2.2.2 Encoding Declaration**

598 If both the encoding declaration and the *Header Container* MIME charset are present, the XML prolog for
599 the SOAP *Message* SHALL contain the encoding declaration SHALL be equivalent to the `charset`
600 attribute of the MIME `Content-Type` of the *Header Container* (see section 2.1.3).

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

603 If the character encoding cannot be determined by an XML processor using the rules specified in section
604 4.3.3 of XML [XML], the XML declaration and its contained encoding declaration SHALL be provided in
605 the ebXML SOAP *Header* Document.

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

607 **2.3 ebXML SOAP Envelope extensions**

608 In conformance with the [SOAP] specification, all extension element content is namespace qualified. All of
609 the ebXML SOAP extension element content defined in this specification is namespace qualified to the
610 ebXML SOAP *Envelope* extensions namespace as defined in section 2.2.2.

611 Namespace declarations (`xmlns` pseudo attributes) for the ebXML SOAP extensions may be included in
612 the SOAP *Envelope*, *Header* or *Body* elements, or directly in each of the ebXML SOAP extension
613 elements.

614 **2.3.1 Namespace pseudo attribute**

615 The namespace declaration for the ebXML SOAP *Envelope* extensions (`xmlns` pseudo attribute) (see
616 [XMLNS]) has a REQUIRED value of:

```
617 http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
```

618 **2.3.2 xsi:schemaLocation attribute**

619 The SOAP namespace:

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

621 resolves to a W3C XML Schema specification. The ebXML OASIS ebXML Messaging TC has provided
622 an equivalent version of the SOAP schema conforming to the W3C Recommendation version of the XML
623 Schema specification [XMLSchema].

```
624 http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
```

625 All ebXML MSH implementations are strongly RECOMMENDED to include the XMLSchema-instance
 626 namespace qualified **schemaLocation** attribute in the SOAP **Envelope** element to indicate to validating
 627 parsers a location of the schema document that should be used to validate the document. Failure to
 628 include the **schemaLocation** attribute could prevent XML schema validation of received messages.

629 For example:

```
630 <SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
631             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
632             xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
633             http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
```

634 In addition, ebXML SOAP **Header** and **Body** extension element content may be similarly qualified so as
 635 to identify the location where validating parsers can find the schema document containing the ebXML
 636 namespace qualified SOAP extension element definitions. The ebXML SOAP extension element schema
 637 has been defined using the W3C Recommendation version of the XML Schema specification
 638 [XMLSchema] (see Appendix A). The XMLSchema-instance namespace qualified **schemaLocation**
 639 attribute should include a mapping of the ebXML SOAP **Envelope** extensions namespace to its schema
 640 document in the same element that declares the ebXML SOAP **Envelope** extensions namespace.

641 The **schemaLocation** for the namespace described above in section 2.3.1 is:

```
642 http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
```

643 Separate **schemaLocation** attribute are RECOMMENDED so tools, which may not correctly use the
 644 **schemaLocation** attribute to resolve schema for more than one namespace, will still be capable of
 645 validating an ebXML SOAP *message*. For example:

```
646 <SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
647             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
648             xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
649             http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
650   <SOAP:Header
651     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
652     xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
653     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
654     <eb:MessageHeader ...>
655       ...
656     </eb:MessageHeader>
657   </SOAP:Header>
658   <SOAP:Body
659     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
660     xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
661     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
662     <eb:Manifest eb:version="2.0">
663       ...
664     </eb:Manifest>
665   </SOAP:Body>
666 </SOAP:Envelope>
```

667 2.3.3 SOAP Header Element

668 The SOAP **Header** element is the first child element of the SOAP **Envelope** element. It MUST have a
 669 namespace qualifier that matches the SOAP **Envelope** namespace declaration for the namespace
 670 "http://schemas.xmlsoap.org/soap/envelope/".

671 2.3.4 SOAP Body Element

672 The SOAP **Body** element is the second child element of the SOAP **Envelope** element. It MUST have a
 673 namespace qualifier that matches the SOAP **Envelope** namespace declaration for the namespace
 674 "http://schemas.xmlsoap.org/soap/envelope/".

675 2.3.5 ebXML SOAP Extensions

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

677 2.3.5.1 SOAP Header extensions:

- 678 • **MessageHeader** – a REQUIRED element containing routing information for the message (To/From, etc.) as
679 well as other context information about the message.
- 680 • **SyncReply** – an element indicating the required transport state to the next SOAP node.

681 2.3.5.2 SOAP Body extension:

- 682 • **Manifest** – an element pointing to any data present either in the *Payload Container(s)* or elsewhere, e.g. on
683 the web. This element MAY be omitted.

684 2.3.5.3 Core ebXML Modules:

- 685 • Error Handling Module
 - 686 - **ErrorList** – a SOAP Header element containing a list of the errors being reported against a previous
687 message. The **ErrorList** element is only used if reporting an error or warning on a previous message.
688 This element MAY be omitted.
- 689 • Security Module
 - 690 - **Signature** – an element that contains a digital signature that conforms to [XMLDSIG] that signs data
691 associated with the message. This element MAY be omitted.

692 2.3.6 #wildcard Element Content

693 Some ebXML SOAP extension elements, as indicated in the schema, allow for foreign namespace-
694 qualified element content to be added for extensibility. The extension element content MUST be
695 namespace-qualified in accordance with XMLNS [XMLNS] and MUST belong to a foreign namespace. A
696 foreign namespace is one that is NOT [http://www.oasis-open.org/committees/ebxml-](http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd)
697 [msg/schema/msg-header-2_0.xsd](http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd). The wildcard elements are provided wherever extensions might be
698 required for private extensions or future expansions to the protocol.

699 An implementation of the MSH MAY ignore the namespace-qualified element and its content.

700 2.3.7 id attribute

701 Each of the ebXML SOAP extension elements defined in this specification has an **id** attribute which is an
702 XML ID that MAY be added to provide for the ability to uniquely identify the element within the SOAP
703 *Message*. This MAY be used when applying a digital signature to the ebXML SOAP *Message* as
704 individual ebXML SOAP extension elements can be targeted for inclusion or exclusion by specifying a
705 URI of "#<idvalue>" in the **Reference** element.

706 2.3.8 version attribute

707 The REQUIRED **version** attribute indicates the version of the ebXML Message Service Header
708 Specification to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide future
709 versioning capabilities. For conformance to this specification, all of the version attributes on any SOAP
710 extension elements defined in this specification MUST have a value of "2.0". An ebXML message MAY
711 contain SOAP header extension elements that have a value other than "2.0". An implementation
712 conforming to this specification that receives a message with ebXML SOAP extensions qualified with a
713 version other than "2.0" MAY process the message if it recognizes the version identified and is capable of
714 processing it. It MUST respond with an error (details TBD) if it does not recognize the identified version.
715 The **version** attribute MUST be namespace qualified for the ebXML SOAP **Envelope** extensions
716 namespace defined above.

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

721 **2.3.9 SOAP mustUnderstand attribute**

722 The REQUIRED SOAP *mustUnderstand* attribute on SOAP *Header* extensions, namespace qualified to
723 the SOAP namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates whether the contents of
724 the element MUST be understood by a receiving process or else the message MUST be rejected in
725 accordance with SOAP [SOAP]. This attribute with a value of '1' (true) indicates the element MUST be
726 understood or rejected. This attribute with a value of '0' (false), the default, indicates the element may be
727 ignored if not understood.

728 **2.3.10 ebXML "Next MSH" actor URI**

729 The URI *urn:oasis:names:tc:ebxml-msg:actor:nextMSH* when used in the context of the SOAP *actor*
730 attribute value SHALL be interpreted to mean an entity that acts in the role of an instance of the ebXML
731 MSH conforming to this specification.

732 This *actor* URI has been established to allow for the possibility that SOAP nodes that are NOT ebXML
733 MSH nodes MAY participate in the message path of an *ebXML Message*. An example might be a SOAP
734 node that digitally signs or encrypts a message.

735 All ebXML MSH nodes MUST act in this role.

736 **2.3.11 ebXML "To Party MSH" actor URI**

737 The URI *urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH* when used in the context of the SOAP
738 *actor* attribute value SHALL be interpreted to mean an instance of an ebXML MSH node, conforming to
739 this specification, acting in the role of the Party identified in the *MessageHeader/To/PartyId* element of
740 the same message. An ebXML MSH MAY be configured to act in this role. How this is done is outside
741 the scope of this specification.

742 The MSH that is the ultimate destination of ebXML messages MUST act in the role of the *To Party MSH*
743 actor URI in addition to acting in the default actor as defined by SOAP.

744 **3 Core Extension Elements**

745 **3.1 MessageHeader Element**

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

748 The *MessageHeader* element is a composite element comprised of the following subordinate elements:

- 749 • an *id* attribute (see section 2.3.7 for details)
- 750 • a *version* attribute (see section 2.3.8 for details)
- 751 • a SOAP *mustUnderstand* attribute with a value of "1" (see section 2.3.9 for details)
- 752 • *From* element
- 753 • *To* element
- 754 • *CPAId* element
- 755 • *ConversationId* element
- 756 • *Service* element
- 757 • *Action* element
- 758 • *MessageData* element
- 759 • *DuplicateElimination* element
- 760 • *Description* element

761 3.1.1 From and To Elements

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

766 The **From** and the **To** elements each contains:

- 767 • **PartyId** elements – occurs one or more times
- 768 • **Role** element – occurs zero or one times.

769 If either the **From** or **To** elements contains multiple **PartyId** elements, all members of the list MUST
770 identify the same organization. Unless a single **type** value refers to multiple identification systems, the
771 value of any given **type** attribute MUST be unique within the list of **PartyId** elements contained within
772 either the From or To element.

773 Note: This mechanism is particularly useful when transport of a message between the parties may involve multiple
774 intermediaries. More generally, the *From Party* should provide identification in all domains it knows in support of
775 intermediaries and destinations that may give preference to particular identification systems.

776 The **From** and **To** elements contain zero or one **Role** child element that, if present, SHALL immediately
777 follow the last **PartyId** child element.

778 3.1.1.1 PartyId Element

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

784 If the **PartyId type** attribute is not present, the content of the **PartyId** element MUST be a URI
785 [RFC2396], otherwise the *Receiving MSH* SHOULD report an error (see section 4.1.5) with **errorCode**
786 set to **Inconsistent** and **severity** set to **Error**. It is strongly RECOMMENDED that the content of the
787 **PartyId** element be a URI.

788 3.1.1.2 Role Element

789 The **Role** element identifies the authorized role (**fromAuthorizedRole** or **toAuthorizedRole**) of the *Party*
790 sending (when present as a child of the **From** element) and/or receiving (when present as a child of the
791 **To** element) the message. The value of the **Role** element is a non-empty string, which is specified in the
792 *CPA*.

793 Note: Role is better defined as a URI – e.g. <http://rosettanet.org/roles/buyer>.

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

```
795 <eb:From>
796   <eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
797   <eb:PartyId eb:type="SCAC">RDWY</PartyId>
798   <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
799 </eb:From>
800 <eb:To>
801   <eb:PartyId>mailto:joe@example.com</eb:PartyId>
802   <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
803 </eb:To>
```

804 3.1.2 CPAId Element

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

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

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

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

815 The messaging parameters are determined by the appropriate elements from the **CPA**, as identified by
 816 the **CPAId** element.

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

820 If a *Receiving MSH* detects an inconsistency, then it MUST report it with an **errorCode** of **Inconsistent**
 821 and a **severity** of **Error**. If the **CPAId** is not recognized, then it MUST report it with an **errorCode** of
 822 **NotRecognized** and a **severity** of **Error**.

823 3.1.3 ConversationId Element

824 The REQUIRED **ConversationId** element is a string identifying the set of related messages that make up
 825 a conversation between two *Parties*. It MUST be unique within the context of the specified **CPAId**. The
 826 *Party* initiating a conversation determines the value of the **ConversationId** element that SHALL be
 827 reflected in all messages pertaining to that conversation.

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

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

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

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

837 3.1.4 Service Element

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

- 840 • a standards organization, or
- 841 • an individual or enterprise

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

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

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

847 Note: URIs in the **Service** element that start with the namespace **urn:oasis:names:tc:ebxml-msg:service** are
 848 reserved for use by this specification.

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

850 3.1.4.1 type attribute

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

854 If the **type** attribute is not present, the content of the **Service** element MUST be a URI [RFC2396]. If it is
855 not a URI then report an error with **errorCode** of **Inconsistent** and **severity** of **Error** (see section 4.1.5).

856 3.1.5 Action Element

857 The REQUIRED **Action** element identifies a process within a **Service** that processes the Message.
858 **Action** SHALL be unique within the **Service** in which it is defined. The value of the **Action** element is
859 specified by the designer of the **service**. An example of the **Action** element follows:

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

861 If the value of either the **Service** or **Action** element are unrecognized by the *Receiving MSH*, then it
862 MUST report the error with an **errorCode** of **NotRecognized** and a **severity** of **Error**.

863 3.1.6 MessageData Element

864 The REQUIRED **MessageData** element provides a means of uniquely identifying an ebXML Message. It
865 contains the following:

- 866 • **MessageId** element
- 867 • **Timestamp** element
- 868 • **RefToMessageId** element
- 869 • **TimeToLive** element

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

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

876 3.1.6.1 MessageId Element

877 The REQUIRED element **MessageId** is a globally unique identifier for each message conforming to
878 MessageId [RFC2822].

879 Note: In the Message-Id and Content-Id MIME headers, values are always surrounded by angle brackets. However
880 references in mid: or cid: scheme URI's and the MessageId and RefToMessageId elements MUST NOT include
881 these delimiters.

882 3.1.6.2 Timestamp Element

883 The REQUIRED **Timestamp** is a value representing the time that the message header was created
884 conforming to a dateTime [XMLSchema] and MUST be expressed as UTC. Indicating UTC in the
885 **Timestamp** element by including the 'Z' identifier is optional.

886 3.1.6.3 RefToMessageId Element

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

890 For Error messages, the **RefToMessageId** element is REQUIRED and its value MUST be the
891 **MessageId** value of the message in error (as defined in section 4.2).

892 3.1.6.4 TimeToLive Element

893 If the **TimeToLive** element is present, it MUST be used to indicate the time, expressed as UTC, by which
894 a message should be delivered to the *To Party MSH*. It MUST conform to an XML Schema dateTime.

895 In this context, the **TimeToLive** has expired if the time of the internal clock, adjusted for UTC, of the
896 *Receiving MSH* is greater than the value of **TimeToLive** for the message.

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

901 The **TimeToLive** element is discussed further under Reliable Messaging in section 6.4.5.

902 3.1.7 DuplicateElimination Element

903 The **DuplicateElimination** element, if present, identifies a request by the sender for the receiving MSH to
904 check for duplicate messages (see section 6.4.1 for more details).

905 Valid values for **DuplicateElimination**:

- 906 • **DuplicateElimination** present – duplicate messages SHOULD be eliminated.
- 907 • **DuplicateElimination** not present – this results in a delivery behavior of Best-Effort.

908 The **DuplicateElimination** element MUST NOT be present if the CPA has **duplicateElimination** set to
909 **never** (see section 6.4.1 and section 6.6 for more details).

910 3.1.8 Description Element

911 The **Description** element may be present zero or more times. Its purpose is to provide a human
912 readable description of the purpose or intent of the message. The language of the description is defined
913 by a required **xml:lang** attribute. The **xml:lang** attribute MUST comply with the rules for identifying
914 languages specified in XML [XML]. Each occurrence SHOULD have a different value for **xml:lang**.

915 3.1.9 MessageHeader Sample

916 The following fragment demonstrates the structure of the **MessageHeader** element within the SOAP
917 **Header**:

```
918 <eb:MessageHeader eb:id="..." eb:version="2.0" SOAP:mustUnderstand="1">
919   <eb:From>
920     <eb:PartyId>uri:example.com</eb:PartyId>
921     <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
922   </eb:From>
923   <eb:To>
924     <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
925     <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
926   </eb:To>
927   <eb:CPAId>http://www.oasis-open.org/cpa/123456</eb:CPAId>
928   <eb:ConversationId>987654321</eb:ConversationId>
929   <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
930   <eb:Action>NewPurchaseOrder</eb:Action>
931   <eb:MessageData>
932     <eb:MessageId>UUID-2</eb:MessageId>
933     <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp>
934     <eb:RefToMessageId>UUID-1</eb:RefToMessageId>
935   </eb:MessageData>
936   <eb:DuplicateElimination/>
937 </eb:MessageHeader>
```

938 3.2 Manifest Element

939 The **Manifest** element MAY be present as a child of the SOAP **Body** element. The **Manifest** element is
940 a composite element consisting of one or more **Reference** elements. Each **Reference** element identifies

941 payload data associated with the message, whether included as part of the message as payload
 942 document(s) contained in a *Payload Container*, or remote resources accessible via a URL. It is
 943 RECOMMENDED that no payload data be present in the SOAP **Body**. The purpose of the **Manifest** is:

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

946 The **Manifest** element is comprised of the following:

- 947 • an **id** attribute (see section 2.3.7 for details)
- 948 • a **version** attribute (see section 2.3.8 for details)
- 949 • one or more **Reference** elements

950 3.2.1 Reference Element

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

- 952 • zero or more **Schema** elements – information about the schema(s) that define the instance document
 953 identified in the parent **Reference** element
- 954 • zero or more **Description** elements – a textual description of the payload object referenced by the parent
 955 **Reference** element

956 The **Reference** element itself is a simple link [XLINK]. It should be noted that the use of XLINK in this
 957 context is chosen solely for the purpose of providing a concise vocabulary for describing an association.
 958 Use of an XLINK processor or engine is NOT REQUIRED, but may prove useful in certain
 959 implementations.

960 The **Reference** element has the following attribute content in addition to the element content described
 961 above:

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

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

972 3.2.1.1 Schema Element

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

- 977 • **location** – the REQUIRED URI of the schema
- 978 • **version** – a version identifier of the schema

979 3.2.1.2 Description Element

980 See section 3.1.8 for more details. An example of a **Description** element follows.

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

982 3.2.2 Manifest Validation

983 If an **xlink:href** attribute contains a URI that is a content id (URI scheme "cid") then a MIME part with
 984 that `content-id` MUST be present in the corresponding *Payload Container* of the message. If it is not,

985 then the error SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem** and a **severity**
986 of **Error**.

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

990 Note: If a payload exists, which is not referenced by the *Manifest*, that payload SHOULD be discarded.

991 3.2.3 Manifest Sample

992 The following fragment demonstrates a typical *Manifest* for a single payload MIME body part:

```
993 <eb:Manifest eb:id="Manifest" eb:version="2.0">
994   <eb:Reference eb:id="pay01"
995     xlink:href="cid:payload-1"
996     xlink:role="http://regrep.org/gci/purchaseOrder">
997     <eb:Schema eb:location="http://regrep.org/gci/purchaseOrder/po.xsd" eb:version="2.0"/>
998     <eb:Description xml:lang="en-US">Purchase Order for 100,000 widgets</eb:Description>
999   </eb:Reference>
1000 </eb:Manifest>
```

1001 4 Core Modules

1002 4.1 Security Module

1003 The *ebXML Message Service*, by its very nature, presents certain security risks. A Message Service may
1004 be at risk by means of:

- 1005 • Unauthorized access
- 1006 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1007 • Denial-of-Service and spoofing

1008 Each security risk is described in detail in the ebXML Technical Architecture Risk Assessment Technical
1009 Report [secRISK].

1010 Each of these security risks may be addressed in whole, or in part, by the application of one, or a
1011 combination, of the countermeasures described in this section. This specification describes a set of
1012 profiles, or combinations of selected countermeasures, selected to address key risks based upon
1013 commonly available technologies. Each of the specified profiles includes a description of the risks that
1014 are not addressed. See Appendix C for a table of security profiles.

1015 Application of countermeasures SHOULD be balanced against an assessment of the inherent risks and
1016 the value of the asset(s) that might be placed at risk. For this specification, a *Signed Message* is any
1017 message containing a **Signature** element.

1018 4.1.1 Signature Element

1019 An ebXML Message MAY be digitally signed to provide security countermeasures. Zero or more
1020 **Signature** elements, belonging to the XML Signature [XMLDSIG] defined namespace, MAY be present
1021 as a child of the SOAP **Header**. The **Signature** element MUST be namespace qualified in accordance
1022 with XML Signature [XMLDSIG]. The structure and content of the **Signature** element MUST conform to
1023 the XML Signature [XMLDSIG] specification. If there is more than one **Signature** element contained
1024 within the SOAP **Header**, the first MUST represent the digital signature of the ebXML Message as signed
1025 by the *From Party MSH* in conformance with section 4.1. Additional **Signature** elements MAY be
1026 present, but their purpose is undefined by this specification.

1027 Refer to section 4.1.3 for a detailed discussion on how to construct the **Signature** element when digitally
1028 signing an ebXML Message.

1029 4.1.2 Security and Management

1030 No technology, regardless of how advanced it might be, is an adequate substitute to the effective
1031 application of security management policies and practices.

1032 It is strongly RECOMMENDED that the site manager of an *ebXML Message Service* apply due diligence
1033 to the support and maintenance of its security mechanisms, site (or physical) security procedures,
1034 cryptographic protocols, update implementations and apply fixes as appropriate. (See
1035 <http://www.cert.org/> and <http://ciac.llnl.gov/>)

1036 4.1.2.1 Collaboration Protocol Agreement

1037 The configuration of Security for MSHs is specified in the *CPA*. Two areas of the *CPA* have security
1038 definitions as follows:

- 1039 • The Document Exchange section addresses security to be applied to the payload of the message. The
1040 MSH is not responsible for any security specified at this level but may offer these services to the message
1041 sender.
- 1042 • The Transport section addresses security applied to the entire ebXML Document, which includes the header
1043 and the payload(s).

1044 4.1.3 Signature Generation

1045 An ebXML Message is signed using [XMLDSIG] following these steps:

- 1046 1) Create a **SignedInfo** element with **SignatureMethod**, **CanonicalizationMethod** and **Reference**
1047 elements for the SOAP **Envelope** and any required payload objects, as prescribed by XML
1048 Signature [XMLDSIG].
- 1049 2) Canonicalize and then calculate the **SignatureValue** over **SignedInfo** based on algorithms
1050 specified in **SignedInfo** as specified in XML Signature [XMLDSIG].
- 1051 3) Construct the **Signature** element that includes the **SignedInfo**, **KeyInfo** (RECOMMENDED) and
1052 **SignatureValue** elements as specified in XML Signature [XMLDSIG].
- 1053 4) Include the namespace qualified **Signature** element in the SOAP **Header** just signed.

1054 The **SignedInfo** element SHALL have a **CanonicalizationMethod** element, a **SignatureMethod** element
1055 and one or more **Reference** elements, as defined in XML Signature [XMLDSIG].

1056 The RECOMMENDED canonicalization method applied to the data to be signed is

```
1057 <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

1058 described in [XMLC14N]. This algorithm excludes comments.

1059 The **SignatureMethod** element SHALL be present and SHALL have an **Algorithm** attribute. The
1060 RECOMMENDED value for the **Algorithm** attribute is:

```
1061 <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlsig#dsa-sha1"/>
```

1062 This RECOMMENDED value SHALL be supported by all compliant *ebXML Message Service* software
1063 implementations.

1064 The [XMLDSIG] **Reference** element for the SOAP **Envelope** document SHALL have a URI attribute
1065 value of "" to provide for the signature to be applied to the document that contains the **Signature** element.

1066 The [XMLDSIG] **Reference** element for the SOAP **Envelope** MAY include a **Type** attribute that has a
1067 value "http://www.w3.org/2000/09/xmlsig#Object" in accordance with XML Signature [XMLDSIG]. This
1068 attribute is purely informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be
1069 prepared to handle either case. The **Reference** element MAY include the **id** attribute.

1070 The [XMLDSIG] **Reference** element for the SOAP **Envelope** SHALL include a child **Transforms**
1071 element. The **Transforms** element SHALL include the following **Transform** child elements.

1072 The first **Transform** element has an **Algorithm** attribute with a value of:

```
1073 <Transform Algorithm="http://www.w3.org/2000/09/xmlsig#enveloped-signature"/>
```

1074 The result of this statement excludes the parent **Signature** element and all its descendants.

1075 The second **Transform** element has a child **XPath** element that has a value of:

```
1076 <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
1077   <XPath> not (ancestor-or-self::() [@SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"] |
1078     ancestor-or-self::() [@SOAP:actor="http://schemas.xmlsoap.org/soap/actor/next" ] )
1079   </XPath>
1080 </Transform>
```

1081 The result of this [XPath] statement excludes all elements within the SOAP **Envelope** which contain a SOAP:**actor** attribute targeting the **nextMSH**, and all their descendants. It also excludes all elements with **actor** attributes targeting the element at the next node (which may change en route). Any intermediate node or MSH MUST NOT change, format or in any way modify any element not targeted to the intermediary. Intermediate nodes MUST NOT add or delete white space. Any such change may invalidate the signature.

1087 The last **Transform** element SHOULD have an **Algorithm** attribute with a value of:

```
1088 <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

1089 The result of this algorithm is to canonicalize the SOAP **Envelope** XML and exclude comments.

1090 Note: These transforms are intended for the SOAP Envelope and its contents. These transforms are NOT intended for the payload objects. The determination of appropriate transforms for each payload is left to the implementation.

1092 Each payload object requiring signing SHALL be represented by a [XMLDSIG] **Reference** element that SHALL have a **URI** attribute resolving to the payload object. This can be either the Content-Id URI of the MIME body part of the payload object, or a URI matching the Content-Location of the MIME body part of the payload object, or a URI that resolves to a payload object external to the Message Package. It is strongly RECOMMENDED that the URI attribute value match the xlink:href URI value of the corresponding **Manifest/Reference** element for the payload object.

1098 Note: When a transfer encoding (e.g. base64) specified by a Content-Transfer-Encoding MIME header is used for the SOAP Envelope or payload objects, the signature generation MUST be executed before the encoding.

1100 Example of digitally signed ebXML SOAP Message:

```
1101 <?xml version="1.0" encoding="utf-8"?>
1102 <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
1103   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
1104   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
1105   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1106   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
1107     http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
1108     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
1109     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
1110   <SOAP:Header>
1111     <eb:MessageHeader eb:id="..." eb:version="2.0" SOAP:mustUnderstand="1">
1112       ...
1113     </eb:MessageHeader>
1114     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
1115       <SignedInfo>
1116         <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
1117         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
1118         <Reference URI="">
1119           <Transforms>
1120             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
1121             <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
1122               <XPath> not (ancestor-or-self::() [@SOAP:actor=
1123                 &quot;urn:oasis:names:tc:ebxml-msg:actor:nextMSH&quot;]
1124                 | ancestor-or-self::() [@SOAP:actor=
1125                 &quot;http://schemas.xmlsoap.org/soap/actor/next&quot;] )
1126               </XPath>
1127             </Transform>
1128             <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
1129           </Transforms>
1130           <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
```

```

1131     <DigestValue>...</DigestValue>
1132   </Reference>
1133   <Reference URI="cid://blahblahblah/">
1134     <DigestMethod Algorithm="http://www.w3.org/2000/09/xmlsig#sha1"/>
1135     <DigestValue>...</DigestValue>
1136   </Reference>
1137 </SignedInfo>
1138 <SignatureValue>...</SignatureValue>
1139 <KeyInfo>...</KeyInfo>
1140 </Signature>
1141 </SOAP:Header>
1142 <SOAP:Body>
1143   <eb:Manifest eb:id="Mani01" eb:version="2.0">
1144     <eb:Reference xlink:href="cid://blahblahblah/" xlink:role="http://ebxml.org/gci/invoice">
1145       <eb:Schema eb:version="2.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
1146     </eb:Reference>
1147   </eb:Manifest>
1148 </SOAP:Body>
1149 </SOAP:Envelope>

```

1150 4.1.4 Countermeasure Technologies

1151 4.1.4.1 Persistent Digital Signature

1152 The only available technology that can be applied to the purpose of digitally signing an ebXML Message
 1153 (the ebXML SOAP **Header** and **Body** and its associated payload objects) is provided by technology that
 1154 conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature conforming
 1155 to this specification can selectively sign portions of an XML document(s), permitting the documents to be
 1156 augmented (new element content added) while preserving the validity of the signature(s).

1157 If signatures are being used to digitally sign an ebXML Message then XML Signature [DSIG] MUST be
 1158 used to bind the ebXML SOAP **Header** and **Body** to the ebXML Payload Container(s) or data elsewhere
 1159 on the web that relate to the message.

1160 An ebXML Message requiring a digital signature SHALL be signed following the process defined in this
 1161 section of the specification and SHALL be in full compliance with XML Signature [XMLDSIG].

1162 4.1.4.2 Persistent Signed Receipt

1163 An *ebXML Message* that has been digitally signed MAY be acknowledged with an *Acknowledgment*
 1164 *Message* that itself is digitally signed in the manner described in the previous section. The
 1165 *Acknowledgment Message* MUST contain a [XMLDSIG] **Reference** element list consistent with those
 1166 contained in the [XMLDSIG] **Signature** element of the original message.

1167 4.1.4.3 Non-persistent Authentication

1168 Non-persistent authentication is provided by the communications channel used to transport the *ebXML*
 1169 *Message*. This authentication MAY be either in one direction or bi-directional. The specific method will be
 1170 determined by the communications protocol used. For instance, the use of a secure network protocol,
 1171 such as TLS [RFC2246] or IPSEC [RFC2402] provides the sender of an *ebXML Message* with a way to
 1172 authenticate the destination for the TCP/IP environment.

1173 4.1.4.4 Non-persistent Integrity

1174 A secure network protocol such as TLS [RFC2246] or IPSEC [RFC2402] MAY be configured to provide
 1175 for digests and comparisons of the packets transmitted via the network connection.

1176 4.1.4.5 Persistent Confidentiality

1177 XML Encryption is a W3C/IETF joint activity actively engaged in the drafting of a specification for the
 1178 selective encryption of an XML document(s). It is anticipated that this specification will be completed
 1179 within the next year. The ebXML Transport, Routing and Packaging team for v1.0 of this specification
 1180 has identified this technology as the only viable means of providing persistent, selective confidentiality of
 1181 elements within an *ebXML Message* including the SOAP **Header**.

1182 Confidentiality for ebXML Payload Containers MAY be provided by functionality possessed by a MSH.
1183 Payload confidentiality MAY be provided by using XML Encryption (when available) or some other
1184 cryptographic process (such as S/MIME [S/MIME], [S/MIMEV3], or PGP MIME [PGP/MIME]) bilaterally
1185 agreed upon by the parties involved. The XML Encryption standard shall be the default encryption
1186 method when XML Encryption has achieved W3C Recommendation status.

1187 Note: When both signature and encryption are required of the MSH, sign first and then encrypt.

1188 **4.1.4.6 Non-persistent Confidentiality**

1189 A secure network protocol, such as TLS [RFC2246] or IPSEC [RFC2402], provides transient
1190 confidentiality of a message as it is transferred between two ebXML adjacent MSH nodes.

1191 **4.1.4.7 Persistent Authorization**

1192 The OASIS Security Services Technical Committee (TC) is actively engaged in the definition of a
1193 specification that provides for the exchange of security credentials, including Name Assertion and
1194 Entitlements, based on Security Assertion Markup Language [SAML]. Use of technology based on this
1195 anticipated specification may provide persistent authorization for an *ebXML Message* once it becomes
1196 available.

1197 **4.1.4.8 Non-persistent Authorization**

1198 A secure network protocol such as TLS [RFC2246] or IPSEC [RFC2402] MAY be configured to provide
1199 for bilateral authentication of certificates prior to establishing a session. This provides for the ability for an
1200 ebXML MSH to authenticate the source of a connection and to recognize the source as an authorized
1201 source of *ebXML Messages*.

1202 **4.1.4.9 Trusted Timestamp**

1203 At the time of this specification, services offering trusted timestamp capabilities are becoming available.
1204 Once these become more widely available, and a standard has been defined for their use and
1205 expression, these standards, technologies and services will be evaluated and considered for use in later
1206 versions of this specification.

1207 **4.1.5 Security Considerations**

1208 Implementors should take note, there is a vulnerability present even when an XML Digital Signature is
1209 used to protect to protect the integrity and origin of ebXML messages. The significance of the
1210 vulnerability necessarily depends on the deployed environment and the transport used to exchange
1211 ebXML messages.

1212 The vulnerability is present because ebXML messaging is an integration of both XML and MIME
1213 technologies. Whenever two or more technologies are conjoined there are always additional (sometimes
1214 unique) security issues to be addressed. In this case, MIME is used as the framework for the message
1215 package, containing the SOAP *Envelope* and any payload containers. Various elements of the SOAP
1216 *Envelope* make reference to the payloads, identified via MIME mechanisms. In addition, various labels
1217 are duplicated in both the SOAP *Envelope* and the MIME framework, for example, the type of the content
1218 in the payload. The issue is how and when all of this information is used.

1219 Specifically, the MIME Content-ID: header is used to specify a unique, identifying label for each payload.
1220 The label is used in the SOAP *Envelope* to identify the payload whenever it is needed. The MIME
1221 Content-Type: header is used to identify the type of content carried in the payload; some content types
1222 may contain additional parameters serving to further qualify the actual type. This information is available
1223 in the SOAP *Envelope*.

1224 The MIME headers are not protected, even when an XML-based digital signature is applied. Although
1225 XML Encryption is not currently available and thus not currently used, its application is developing
1226 similarly to XML digital signatures. Insofar as its application is the same as that of XML digital signatures,
1227 its use will not protect the MIME headers. Thus, an ebXML message may be at risk depending on how

1228 the information in the MIME headers is processed as compared to the information in the SOAP
1229 **Envelope**.

1230 The Content-ID: MIME header is critical. An adversary could easily mount a denial-of-service attack by
1231 mixing and matching payloads with the Content-ID: headers. As with most denial-of-service attacks, no
1232 specific protection is offered for this vulnerability. However, it should be detected since the digest
1233 calculated for the actual payload will not match the digest included in the SOAP **Envelope** when the
1234 digital signature is validated.

1235 The presence of the content type in both the MIME headers and SOAP **Envelope** is a problem. Ordinary
1236 security practices discourage duplicating information in two places. When information is duplicated,
1237 ordinary security practices require the information in both places to be compared to ensure they are
1238 equal. It would be considered a security violation if both sets of information fail to match.

1239 An adversary could change the MIME headers while a message is en route from its origin to its
1240 destination and this would not be detected when the security services are validated. This threat is less
1241 significant in a peer-to-peer transport environment as compared to a multi-hop transport environment. All
1242 implementations are at risk if the ebXML message is ever recorded in a long-term storage area since a
1243 compromise of that area puts the message at risk for modification.

1244 The actual risk depends on how an implementation uses each of the duplicate sets of information. If any
1245 processing beyond the MIME parsing for body part identification and separation is dependent on the
1246 information in the MIME headers, then the implementation is at risk of being directed to take unintended
1247 or undesirable actions. How this might be exploited is best compared to the common programming
1248 mistake of permitting buffer overflows: it depends on the creativity and persistence of the adversary.

1249 Thus, an implementation could reduce the risk by ensuring that the unprotected information in the MIME
1250 headers is never used except by the MIME parser for the minimum purpose of identifying and separating
1251 the body parts. This version of the specification makes no recommendation regarding whether or not an
1252 implementation should compare the duplicate sets of information nor what action to take based on the
1253 results of the comparison.

1254 4.2 Error Handling Module

1255 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects in an
1256 ebXML Message to another MSH. The *ebXML Message Service* error reporting and handling module is
1257 to be considered as a layer of processing above the SOAP processor layer. This means the ebXML MSH
1258 is essentially an application-level handler of a *SOAP Message* from the perspective of the SOAP
1259 Processor. The SOAP processor MAY generate a SOAP **Fault** message if it is unable to process the
1260 message. A *Sending MSH* MUST be prepared to accept and process these SOAP **Fault** values.

1261 It is possible for the ebXML MSH software to cause a SOAP **Fault** to be generated and returned to the
1262 sender of a *SOAP Message*. In this event, the returned message MUST conform to the [SOAP]
1263 specification processing guidelines for SOAP **Fault** values.

1264 An ebXML *SOAP Message* reporting an error with a **highestSeverity** of **Warning** SHALL NOT be
1265 reported or returned as a SOAP **Fault**.

1266 4.2.1.1 Definitions:

1267 For clarity, two phrases are defined for use in this section:

- 1268 • "message in error" – A *message* containing or causing an error or warning of some kind
- 1269 • "message reporting the error" – A *message* containing an ebXML **ErrorList** element that describes the
1270 warning(s) and/or error(s) found in a message in error (also referred to as an *Error Message* elsewhere in
1271 this document).

1272 4.2.2 Types of Errors

1273 One MSH needs to report errors to another MSH. For example, errors associated with:

- 1274 • ebXML namespace qualified content of the *SOAP Message* document (see section 2.3.1)
- 1275 • reliable messaging failures (see section 6.5.7)
- 1276 • security (see section 4.1)

1277 Unless specified to the contrary, all references to "an error" in the remainder of this specification imply
1278 any or all of the types of errors listed above or defined elsewhere.

1279 Errors associated with data communications protocols are detected and reported using the standard
1280 mechanisms supported by that data communications protocol and do not use the error reporting
1281 mechanism described here.

1282 4.2.3 ErrorList Element

1283 The existence of an **ErrorList** extension element within the SOAP **Header** element indicates the
1284 message identified by the **RefToMessageId** in the **MessageHeader** element has an error.

1285 The **ErrorList** element consists of:

- 1286 • **id** attribute (see section 2.3.7 for details)
- 1287 • a **version** attribute (see section 2.3.8 for details)
- 1288 • a SOAP **mustUnderstand** attribute with a value of "1" (see section 2.3.9 for details)
- 1289 • **highestSeverity** attribute
- 1290 • one or more **Error** elements

1291 If there are no errors to be reported then the **ErrorList** element MUST NOT be present.

1292 4.2.3.1 highestSeverity attribute

1293 The **highestSeverity** attribute contains the highest severity of any of the **Error** elements. Specifically, if
1294 any of the **Error** elements have a **severity** of **Error**, **highestSeverity** MUST be set to **Error**, otherwise,
1295 **highestSeverity** MUST be set to **Warning**.

1296 4.2.3.2 Error Element

1297 An **Error** element consists of:

- 1298 • **id** attribute (see section 2.3.7 for details)
- 1299 • **codeContext** attribute
- 1300 • **errorCode** attribute
- 1301 • **severity** attribute
- 1302 • **location** attribute
- 1303 • **Description** element

1304 4.2.3.2.1 id attribute

1305 If the error is a part of an ebXML element, the **id** of the element MAY be provided for error tracking.

1306 4.2.3.2.2 codeContext attribute

1307 The **codeContext** attribute identifies the namespace or scheme for the **errorCodes**. It MUST be a URI.
1308 Its default value is **urn:oasis:names:tc:ebxml-msg:service:errors**. If it does not have the default value,
1309 then it indicates an implementation of this specification has used its own **errorCode** attribute values.

1310 Use of a **codeContext** attribute value other than the default is NOT RECOMMENDED. In addition, an
1311 implementation of this specification should not use its own **errorCode** attribute values if an existing
1312 **errorCode** as defined in this section has the same or very similar meaning.

1313 4.2.3.2.3 **errorCode attribute**

1314 The REQUIRED **errorCode** attribute indicates the nature of the error in the message in error. Valid
1315 values for the **errorCode** and a description of the code's meaning are given in the next section.

1316 4.2.3.2.4 **severity attribute**

1317 The REQUIRED **severity** attribute indicates the severity of the error. Valid values are:

- 1318 • **Warning** – This indicates other messages in the conversation could be generated in the normal way in spite
1319 of this problem.
- 1320 • **Error** – This indicates there is an unrecoverable error in the message and no further message processing
1321 should occur. Appropriate failure conditions should be communicated to the Application.

1322 4.2.3.2.5 **location attribute**

1323 The **location** attribute points to the part of the message containing the error.

1324 If an error exists in an ebXML element and the containing document is "well formed" (see XML [XML]),
1325 then the content of the **location** attribute MUST be an XPointer [XPointer].

1326 If the error is associated with an ebXML Payload Container, then **location** contains the `content-id` of
1327 the MIME part in error, using URI scheme "cid".

1328 4.2.3.2.6 **Description Element**

1329 The content of the **Description** element provides a narrative description of the error in the language
1330 defined by the **xml:lang** attribute. The XML parser or other software validating the message typically
1331 generates the message. The content is defined by the vendor/developer of the software that generated
1332 the **Error** element. (See section 3.1.8)

1333 4.2.3.3 **ErrorList Sample**

1334 An example of an **ErrorList** element is given below.

```
1335 <eb:ErrorList eb:id="3490sdo", eb:highestSeverity="error" eb:version="2.0" SOAP:mustUnderstand="1">
1336   <eb:Error eb:errorCode="SecurityFailure" eb:severity="Error" eb:location="URI_of_ds:Signature">
1337     <eb:Description xml:lang="en-US">Validation of signature failed<eb:Description>
1338   </eb:Error>
1339   <eb:Error ...> ... </eb:Error>
1340 </eb:ErrorList>
```

1341 4.2.3.4 **errorCode values**

1342 This section describes the values for the **errorCode** attribute used in a *message reporting an error*. They
1343 are described in a table with three headings:

- 1344 • the first column contains the value to be used as an **errorCode**, e.g. **SecurityFailure**
- 1345 • the second column contains a "Short Description" of the **errorCode**. This narrative MUST NOT be used in
1346 the content of the **Error** element.
- 1347 • the third column contains a "Long Description" that provides an explanation of the meaning of the error and
1348 provides guidance on when the particular **errorCode** should be used.

1349 4.2.3.4.1 **Reporting Errors in the ebXML Elements**

1350 The following list contains error codes that can be associated with ebXML elements:

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	Although the document is well formed and valid, a module is

	supported	present 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.

1351 **4.2.3.4.2 Non-XML Document Errors**

1352 The following are error codes that identify errors not associated with the ebXML elements:

Error Code	Short Description	Long Description
DeliveryFailure	Message Delivery Failure	A message has been received that either probably or definitely could not be sent to its next destination. Note: if <i>severity</i> is set to Warning then there is a small probability that the message was delivered.
TimeToLiveExpired	Message Time To Live Expired	A message has been received that arrived after the time specified in the TimeToLive element of the MessageHeader element.
SecurityFailure	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
MimeProblem	URI resolve error	If an xlink:href attribute contains a URI, not a content id (URI scheme "cid"), and the URI cannot be resolved, then it is an implementation decision whether to report the error.
Unknown	Unknown Error	Indicates that an error has occurred 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.

1353 **4.2.4 Implementing Error Reporting and Handling**1354 **4.2.4.1 When to Generate Error Messages**1355 When a MSH detects an error in a message it is strongly RECOMMENDED the error is reported to the
1356 MSH that sent the message in error. This is possible when:

- 1357 • the Error Reporting Location (see section 4.2.4.2) to which the message reporting the error should be sent
1358 can be determined
- 1359 • the message in error does not have an **ErrorList** element with **highestSeverity** set to **Error**.

1360 If the Error Reporting Location cannot be found or the message in error has an **ErrorList** element with
1361 **highestSeverity** set to **Error**, it is RECOMMENDED:

- 1362 • the error is logged
- 1363 • the problem is resolved by other means
- 1364 • no further action is taken.

1365 **4.2.4.2 Identifying the Error Reporting Location**1366 The Error Reporting Location is a URI specified by the sender of the message in error that indicates
1367 where to send a *message reporting the error*.

1368 The **ErrorURI** implied by the **CPA**, identified by the **CPAId** on the message, SHOULD be used.
 1369 Otherwise, the recipient MAY resolve an **ErrorURI** using the **From** element of the message in error. If
 1370 neither is possible, no error will be reported to the sending **Party**.

1371 Even if the message in error cannot be successfully analyzed, MSH implementers MAY try to determine
 1372 the Error Reporting Location by other means. How this is done is an implementation decision.

1373 4.2.4.3 Service and Action Element Values

1374 An **ErrorList** element can be included in a SOAP **Header** that is part of a *message* being sent as a result
 1375 of processing of an earlier message. In this case, the values for the **Service** and **Action** elements are
 1376 set by the designer of the Service. This method MUST NOT be used if the **highestSeverity** is **Error**.

1377 An **ErrorList** element can also be included in an independent *message*. In this case the values of the
 1378 **Service** and **Action** elements MUST be set as follows:

- 1379 • The **Service** element MUST be set to: `urn:oasis:names:tc:ebxml-msg:service`
- 1380 • The **Action** element MUST be set to **MessageError**.

1381 4.3 SyncReply Module

1382 It may be necessary for the sender of a message, using a synchronous communications protocol, such as
 1383 HTTP, to receive the associated response message over the same connection the request message was
 1384 delivered. In the case of HTTP, the sender of the HTTP request message containing an ebXML message
 1385 needs to have the response ebXML message delivered to it on the same HTTP connection.

1386 If there are intermediary nodes (either ebXML MSH nodes or possibly other SOAP nodes) involved in the
 1387 message path, it is necessary to provide some means by which the sender of a message can indicate it is
 1388 expecting a response so the intermediary nodes can keep the connection open.

1389 The **SyncReply** ebXML SOAP extension element is provided for this purpose.

1390 4.3.1 SyncReply Element

1391 The **SyncReply** element MAY be present as a direct child descendant of the SOAP **Header** element. It
 1392 consists of:

- 1393 • an **id** attribute (see section 2.3.7 for details)
- 1394 • a **version** attribute (see section 2.3.8 for details)
- 1395 • a SOAP **actor** attribute with the REQUIRED value of "`http://schemas.xmlsoap.org/soap/actor/next`"
- 1396 • a SOAP **mustUnderstand** attribute with a value of "1" (see section 2.3.9 for details)

1397 If present, this element indicates to the receiving SOAP or ebXML MSH node the connection over which
 1398 the message was received SHOULD be kept open in expectation of a response message to be returned
 1399 via the same connection.

1400 This element MUST NOT be used to override the value of **syncReplyMode** in the CPA. If the value of
 1401 **syncReplyMode** is **none** and a **SyncReply** element is present, the *Receiving MSH* should issue an error
 1402 with **errorCode** of **Inconsistent** and a **severity** of **Error** (see section 4.1.5).

1403 An example of a **SyncReply** element:

```
1404 <eb:SyncReply eb:id="3833kkj9" eb:version="2.0" SOAP:mustUnderstand="1"  
1405 SOAP:actor="http://schemas.xmlsoap.org/soap/actor/next"/>
```

1406 5 Combining ebXML SOAP Extension Elements

1407 This section describes how the various ebXML SOAP extension elements may be used in combination.

1408 5.1.1 MessageHeader Element Interaction

1409 The *MessageHeader* element MUST be present in every message.

1410 5.1.2 Manifest Element Interaction

1411 The *Manifest* element MUST be present if there is any data associated with the message not present in
1412 the *Header Container*. This applies specifically to data in the *Payload Container(s)* or elsewhere, e.g. on
1413 the web.

1414 5.1.3 Signature Element Interaction

1415 One or more XML Signature [XMLDSIG] *Signature* elements MAY be present on any message.

1416 5.1.4 ErrorList Element Interaction

1417 If the *highestSeverity* attribute on the *ErrorList* is set to *Warning*, then this element MAY be present
1418 with any element.

1419 If the *highestSeverity* attribute on the *ErrorList* is set to *Error*, then this element MUST NOT be present
1420 with the *Manifest* element

1421 5.1.5 SyncReply Element Interaction

1422 The *SyncReply* element MAY be present on any outbound message sent using synchronous
1423 communication protocol.

1424

Part II. Additional Features

1425

6 Reliable Messaging Module

1426

1427

1428

1429

Reliable Messaging defines an interoperable protocol such that two Message Service Handlers (MSH) can reliably exchange messages, using acknowledgment, retry and duplicate detection and elimination mechanisms, resulting in the *To Party* receiving the message Once-And-Only-Once. The protocol is flexible, allowing for both store-and-forward and end-to-end reliable messaging.

1430

1431

1432

1433

1434

Reliability is achieved by a *Receiving MSH* responding to a message with an *Acknowledgment Message*. An *Acknowledgment Message* is any ebXML message containing an **Acknowledgment** element. Failure to receive an *Acknowledgment Message* by a *Sending MSH* MAY trigger successive retries until such time as an *Acknowledgment Message* is received or the predetermined number of retries has been exceeded at which time the *From Party* MUST be notified of the probable delivery failure.

1435

1436

Whenever an identical message may be received more than once, some method of duplicate detection and elimination is indicated, usually through the mechanism of a *persistent store*.

1437

6.1 Persistent Storage and System Failure

1438

1439

1440

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

1441

1442

1443

1444

1445

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

1446

1447

1448

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

1449

1450

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

1451

1452

1453

1454

1455

- the complete message, at least until the information in the message has been passed to the application or other process needing to process it,
- the time the message was received, so the information can be used to generate the response to a *Message Status Request* (see section 7.1.1),
- the complete response message.

1456

6.2 Methods of Implementing Reliable Messaging

1457

Support for Reliable Messaging is implemented in one of the following ways:

1458

1459

1460

1461

1462

- using the ebXML Reliable Messaging protocol,
- using ebXML SOAP structures together with commercial software products that are designed to provide reliable delivery of messages using alternative protocols,
- user application support for some features, especially duplicate elimination, or
- some mixture of the above options on a per-feature basis.

1463 6.3 Reliable Messaging SOAP Header Extensions

1464 6.3.1 AckRequested Element

1465 The **AckRequested** element is an OPTIONAL extension to the SOAP **Header** used by the *Sending MSH*
1466 to request a *Receiving MSH*, acting in the role of the actor URI identified in the SOAP **actor** attribute,
1467 returns an *Acknowledgment Message*.

1468 The **AckRequested** element contains the following:

- 1469 • a **id** attribute (see section 2.3.7 for details)
- 1470 • a **version** attribute (see section 2.3.8 for details)
- 1471 • a SOAP **mustUnderstand** attribute with a value of "1" (see section 2.3.9 for details)
- 1472 • a SOAP **actor** attribute
- 1473 • a **signed** attribute

1474 This element is used to indicate to a *Receiving MSH*, acting in the role identified by the SOAP **actor**
1475 attribute, whether an *Acknowledgment Message* is expected, and if so, whether the message should be
1476 signed by the *Receiving MSH*.

1477 An *ebXML Message* MAY have zero, one, or two instances of an **AckRequested** element. A single MSH
1478 node SHOULD only insert one **AckRequested** element. If there are two **AckRequested** elements
1479 present, they MUST have different values for their respective SOAP **actor** attributes. At most one
1480 **AckRequested** element can be targeted at the **actor** URI meaning *Next MSH* (see section 2.3.10) and at
1481 most one **AckRequested** element can be targeted at the **actor** URI meaning *To Party MSH* (see section
1482 2.3.11) for any given message.

1483 6.3.1.1 SOAP actor attribute

1484 The **AckRequested** element MUST be targeted at either the *Next MSH* or the *To Party MSH* (these are
1485 equivalent for single-hop routing). This is accomplished by including a SOAP **actor** with a URN value
1486 with one of the two ebXML **actor** URNs defined in sections 2.3.10 and 2.3.11 or by leaving this attribute
1487 out. The default **actor** targets the *To Party MSH*.

1488 6.3.1.2 signed attribute

1489 The REQUIRED **signed** attribute is used by a *From Party* to indicate whether or not a message received
1490 by the *To Party MSH* should result in the *To Party* returning a signed *Acknowledgment Message* –
1491 containing a [XMLDSIG] **Signature** element as described in section 4.1. Valid values for **signed** are:

- 1492 • **true** - a signed *Acknowledgment Message* is requested, or
- 1493 • **false** - an unsigned *Acknowledgment Message* is requested.

1494 Before setting the value of the **signed** attribute in **AckRequested**, the *Sending MSH* SHOULD check if
1495 the *Receiving MSH* supports *Acknowledgment Messages* of the type requested (see also [ebCPP]).

1496 When a *Receiving MSH* receives a message with **signed** attribute set to **true** or **false** then it should verify
1497 it is able to support the type of *Acknowledgment Message* requested.

- 1498 • If the *Receiving MSH* can produce the *Acknowledgment Message* of the type requested, then it MUST
1499 return to the *Sending MSH* a message containing an **Acknowledgment** element.
- 1500 • If the *Receiving MSH* cannot return an *Acknowledgment Message* as requested it MUST report the error to
1501 the *Sending MSH* using an **errorCode** of **Inconsistent** and a **severity** of either **Error** if inconsistent with the
1502 CPA, or **Warning** if not supported..

1503 6.3.1.3 AckRequested Sample

1504 In the following example, an *Acknowledgment Message* is requested of a MSH node acting in the role of
1505 the *To Party* (see section 2.3.11). The **Acknowledgment** element generated MUST be targeted to the

1506 ebXML MSH node acting in the role of the *From Party* along the reverse message path (end-to-end
1507 acknowledgment).

1508 `<eb:AckRequested SOAP:mustUnderstand="1" eb:version="2.0" eb:signed="false"/>`

1509 **6.3.1.4 AckRequested Element Interaction**

1510 An **AckRequested** element MUST NOT be included on a message with only an **Acknowledgment**
1511 element (no payload). This restriction is imposed to avoid endless loops of *Acknowledgment Messages*.
1512 An *Error Message* MUST NOT contain an **AckRequested** element.

1513 **6.3.2 Acknowledgment Element**

1514 The **Acknowledgment** element is an OPTIONAL extension to the SOAP **Header** used by one Message
1515 Service Handler to indicate to another Message Service Handler that it has received a message. The
1516 **RefToMessageId** element in an **Acknowledgment** element is used to identify the message being
1517 acknowledged by its **MessageId**.

1518 The **Acknowledgment** element consists of the following elements and attributes:

- 1519 • an **id** attribute (see section 2.3.7 for details)
- 1520 • a **version** attribute (see section 2.3.8 for details)
- 1521 • a SOAP **mustUnderstand** attribute with a value of "1" (see section 2.3.9 for details)
- 1522 • a SOAP **actor** attribute
- 1523 • a **Timestamp** element
- 1524 • a **RefToMessageId** element
- 1525 • a **From** element
- 1526 • zero or more [XMLDSIG] **Reference** element(s)

1527 **6.3.2.1 SOAP actor attribute**

1528 The SOAP **actor** attribute of the **Acknowledgment** element SHALL have a value corresponding to the
1529 **AckRequested** element of the message being acknowledged. If there is no SOAP **actor** attribute
1530 present on an **Acknowledgment** element, the default target is the *To Party MSH* (see section for 10.1.3).

1531 **6.3.2.2 Timestamp Element**

1532 The REQUIRED **Timestamp** element is a value representing the time that the message being
1533 acknowledged was received by the *MSH* generating the acknowledgment message. It must conform to a
1534 dateTime [XMLSchema] and is expressed as UTC (section 3.1.6.2).

1535 **6.3.2.3 RefToMessageId Element**

1536 The REQUIRED **RefToMessageId** element contains the **MessageId** of the message whose delivery is
1537 being reported.

1538 **6.3.2.4 From Element**

1539 This is the same element as the **From** element within **MessageHeader** element (see section 3.1.1).
1540 However, when used in the context of an **Acknowledgment** element, it contains the identifier of the *Party*
1541 generating the *Acknowledgment Message*.

1542 If the **From** element is omitted then the *Party* sending the element is identified by the **From** element in
1543 the **MessageHeader** element.

1544 **6.3.2.5 [XMLDSIG] Reference Element**

1545 An *Acknowledgment Message* MAY be used to enable non-repudiation of receipt by a MSH by including
1546 one or more **Reference** elements, from the XML Signature [XMLDSIG] namespace, derived from the
1547 *message being acknowledged* (see section 4.1.3 for details). The **Reference** element(s) MUST be

1548 namespace qualified to the aforementioned namespace and MUST conform to the XML Signature
 1549 [XMLDSIG] specification. If the *message being acknowledged* contains an **AckRequested** element with
 1550 a **signed** attribute set to **true**, then the [XMLDSIG] **Reference** list is REQUIRED.

1551 Receipt of an *Acknowledgment Message*, indicates the original message reached its destination. Receipt
 1552 of a signed *Acknowledgment Message* validates the sender of the *Acknowledgment Message*. However,
 1553 a signed *Acknowledgment Message* does not indicate whether the message arrived intact. Including a
 1554 digest (see [XMLDSIG] section 4.3.3) of the original message in the *Acknowledgment Message* indicates
 1555 to the original sender what was received by the recipient of the message being acknowledged. The
 1556 digest contained in the *Acknowledgment Message* may be compared to a digest of the original message.
 1557 If the digests match, the message arrived intact. Such a digest already exists in the original message, if it
 1558 is signed, contained within the [XMLDSIG] **Signature / Reference** element(s).

1559 If the original message is signed, the [XMLDSIG] **Signature / Reference** element(s) of the original
 1560 message will be identical to the **Acknowledgment / Reference** element(s) in the
 1561 *Acknowledgment Message*. If the original message is not signed, the [XMLDSIG] **Reference** element
 1562 must be derived from the original message (see section 4.1.3).

1563 Upon receipt of an end-to-end *Acknowledgment Message*, the *From Party MSH* MAY notify the
 1564 application of successful delivery for the referenced message. This MSH SHOULD ignore subsequent
 1565 *Error* or *Acknowledgment Messages* with the same **RefToMessageId** value.

1566 6.3.2.6 Acknowledgment Sample

1567 An example *Acknowledgment* element targeted at the *To Party MSH*:

```
1568 <eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0">
1569   <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
1570   <eb:RefToMessageId>323210:e52151ec74:7ffc@xtacy</eb:RefToMessageId>
1571   <eb:From> <eb:PartyId>uri:www.example.com</eb:PartyId> </eb:From>
1572 </eb:Acknowledgment>
```

1573 6.3.2.7 Sending an Acknowledgment Message by Itself

1574 If there are no errors in the message received and an *Acknowledgment Message* is being sent on its own,
 1575 not as a message containing payload data, then the **Service** and **Action** MUST be set as follows:

- 1576 • the **Service** element MUST be set to **urn:oasis:names:tc:ebxml-msg:service**
- 1577 • the **Action** element MUST be set to **Acknowledgment**

1578 6.3.2.8 Acknowledgment Element Interaction

1579 An *Acknowledgment* element MAY be present on any message, except as noted in section 6.3.1.4. An
 1580 *Acknowledgment Message* MUST NOT be returned for an *Error Message*.

1581 6.4 Reliable Messaging Parameters

1582 This section describes the parameters required to control reliable messaging. Many of these parameters
 1583 can be obtained from a CPA.

1584 6.4.1 DuplicateElimination

1585 The **DuplicateElimination** element MUST be used by the *From Party MSH* to indicate whether the
 1586 *Receiving MSH* MUST eliminate duplicates (see section 6.6 for Reliable Messaging behaviors). If the
 1587 value of **DuplicateElimination** in the CPA is **never**, **DuplicateElimination** MUST NOT be present.

- 1588 • If **DuplicateElimination** is present – The *To Party MSH* must persist messages in a persistent store so
 1589 duplicate messages will be presented to the *To Party Application At-Most-Once*, or
- 1590 • If **DuplicateElimination** is not present – The *To Party MSH* is not required to maintain the message in
 1591 persistent store and is not required to check for duplicates.

1592 If **DuplicateElimination** is present, the *To Party MSH* must adopt a reliable messaging behavior (see
 1593 section 6.6) causing duplicate messages to be ignored.

1594 If **DuplicateElimination** is not present, a *Receiving MSH* is not required to check for duplicate message
1595 delivery. Duplicate messages might be delivered to an application and persistent storage of messages is
1596 not required – although elimination of duplicates is still allowed.

1597 If the *To Party* is unable to support the requested functionality, or if the value of **duplicateElimination** in
1598 the CPA does not match the implied value of the element, the *To Party* SHOULD report the error to the
1599 *From Party* using an **errorCode** of **Inconsistent** and a **Severity** of **Error**.

1600 **6.4.2 AckRequested**

1601 The **AckRequested** parameter is used by the *Sending MSH* to request a *Receiving MSH*, acting in the
1602 role of the actor URI identified in the SOAP **actor** attribute, return an *Acknowledgment Message*
1603 containing an **Acknowledgment** element (see section 6.3.1).

1604 **6.4.3 Retries**

1605 The **Retries** parameter, from a CPA, is an integer value specifying the maximum number of times a
1606 *Sending MSH* SHOULD attempt to redeliver an unacknowledged *message* using the same
1607 communications protocol.

1608 **6.4.4 RetryInterval**

1609 The **RetryInterval** parameter, from a CPA, is a time value, expressed as a duration in accordance with
1610 the **duration** [XMLSchema] data type. This value specifies the minimum time a *Sending MSH* SHOULD
1611 wait between **Retries**, if an *Acknowledgment Message* is not received or if a communications error was
1612 detected during an attempt to send the message. **RetryInterval** applies to the time between sending of
1613 the original message and the first retry as well as the time between retries.

1614 **6.4.5 TimeToLive**

1615 **TimeToLive** is defined in section 3.1.6.4.

1616 For a reliably delivered message, **TimeToLive** MUST conform to:

1617
$$\mathbf{TimeToLive} > \mathbf{Timestamp} + ((\mathbf{Retries} + 1) * \mathbf{RetryInterval}).$$

1618 where **TimeStamp** comes from **MessageData**.

1619 **6.4.6 PersistDuration**

1620 The **PersistDuration** parameter, from a CPA, is the minimum length of time, expressed as a **duration**
1621 [XMLSchema], data from a reliably sent *Message*, is kept in *Persistent Storage* by a *Receiving MSH*.

1622 If the **PersistDuration** has passed since the message was first sent, a *Sending MSH* SHOULD NOT
1623 resend a message with the same **MessageId**.

1624 If a message cannot be sent successfully before **PersistDuration** has passed, then the *Sending MSH*
1625 should report a delivery failure (see section 6.5.7).

1626 **TimeStamp** for a reliably sent message (found in the message header), plus its **PersistDuration** (found
1627 in the CPA), must be greater than its **TimeToLive** (found in the message header).

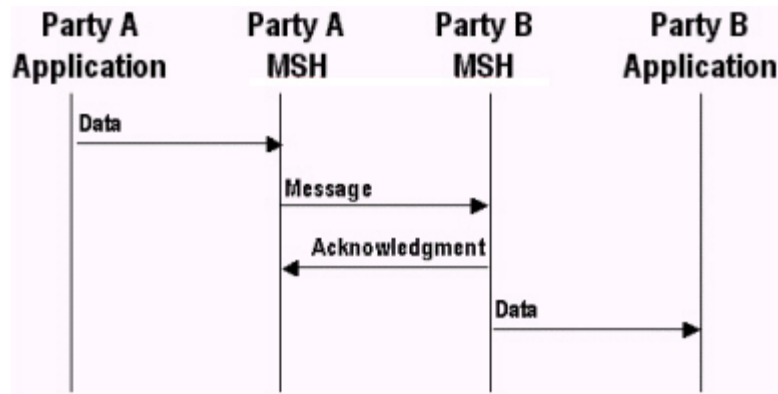
1628 **6.4.7 syncReplyMode**

1629 The **syncReplyMode** parameter from the CPA is used only if the data communications protocol is
1630 *synchronous* (e.g. HTTP). If the communications protocol is not *synchronous*, then the value of
1631 **syncReplyMode** is ignored. If the **syncReplyMode** attribute is not present, it is semantically equivalent
1632 to its presence with a value of **none**. If the **syncReplyMode** parameter is not **none**, a **SyncReply**
1633 element MUST be present and the MSH must return any response from the application or business
1634 process in the payload of the *synchronous* reply message, as specified in the CPA. Valid values of
1635 **syncReplyMode** are **mshSignalsOnly**, **signalsOnly**, **signalsAndResponse**, **responseOnly**, and **none**.
1636 See also the description of **syncReplyMode** in the CPPA [ebCPP] specification.

1637 If the value of *syncReplyMode* is *none* and a *SyncReply* element is present, the *Receiving MSH* should
 1638 issue an error with *errorCode* of *Inconsistent* and a *severity* of *Error* (see section 4.1.5).

1639 6.5 ebXML Reliable Messaging Protocol

1640 The ebXML Reliable Messaging Protocol is illustrated by the following figure.



1641

1642 **Figure 6-1 Indicating a message has been received**

1643 The receipt of the *Acknowledgment Message* indicates the message being acknowledged has been
 1644 successfully received and either processed or persisted by the *Receiving MSH*.

1645 An *Acknowledgment Message* MUST contain an *Acknowledgment* element as described in section 6.3.1
 1646 with a *RefToMessageld* containing the same value as the *MessageId* element in the *message being*
 1647 *acknowledged*.

1648 6.5.1 Sending Message Behavior

1649 If a MSH is given data by an application needing to be sent reliably, the MSH MUST do the following:

- 1650 1. Create a message from components received from the application.
- 1651 2. Insert an *AckRequested* element as defined in section 6.3.1.
- 1652 3. Save the message in *persistent storage* (see section 6.1).
- 1653 4. Send the message to the *Receiving MSH*.
- 1654 5. Wait for the return of an *Acknowledgment Message* acknowledging receipt of this specific
 1655 message and, if it does not arrive before *RetryInterval* has elapsed, or if a communications
 1656 protocol error is encountered, then take the appropriate action as described in section 6.5.4.

1657 6.5.2 Receiving Message Behavior

1658 If this is an *Acknowledgment Message* as defined in section 6 then:

- 1659 1 Look for a message in *persistent storage* with a *MessageId* the same as the value of
 1660 *RefToMessageld* on the received Message.
- 1661 2 If a message is found in *persistent storage* then mark the persisted message as delivered.

1662 If the *Receiving MSH* is NOT the *To Party MSH* (as defined in section 2.3.10 and 2.3.11), then see
 1663 section 10.1.3 for the behavior of the *AckRequested* element.

1664 If an *AckRequested* element is present (not an *Acknowledgment Message*) then:

- 1665 1 If the message is a duplicate (i.e. there is a *MessageId* held in persistent storage containing the
 1666 same value as the *MessageId* in the received message), generate an *Acknowledgment Message*
 1667 (see section 6.5.3). Follow the procedure in section 6.5.5 for resending lost *Acknowledgment*

1668 *Messages*. The *Receiving MSH* MUST NOT deliver the message to the application interface.
 1669 Note: The check for duplicates is only performed when **DuplicateElimination** is present.

- 1670 2 If the message is not a duplicate or (there is no **MessageId** held in persistent storage
 1671 corresponding to the **MessageId** in the received message) then:
- 1672 a If there is a **DuplicateElimination** element, save the **MessageId** of the received message in
 1673 persistent storage. As an implementation decision, the whole message MAY be stored.
- 1674 b Generate an *Acknowledgment Message* in response (this may be as part of another
 1675 message). The *Receiving MSH* MUST NOT send an *Acknowledgment Message* until the
 1676 message has been safely stored in *persistent storage* or delivered to the application
 1677 interface. Delivery of an *Acknowledgment Message* constitutes an obligation by the
 1678 *Receiving MSH* to deliver the message to the application or forward to the next MSH in the
 1679 message path as appropriate.

1680 If there is no **AckRequested** element then do the following:

- 1681 1 If there is a **DuplicateElimination** element, and the message is a duplicate, then do nothing.
 1682 2 Otherwise, deliver the message to the application interface

1683 If the *Receiving MSH* node is operating as an intermediary along the message's message path, then it
 1684 MAY use store-and-forward behavior. However, it MUST NOT filter out perceived duplicate messages
 1685 from their normal processing at that node.

1686 If an *Acknowledgment Message* is received unexpectedly, it should be ignored. No error should be sent.

1687 **6.5.3 Generating an Acknowledgment Message**

1688 An *Acknowledgment Message* MUST be generated whenever a message is received with an
 1689 **AckRequested** element having a SOAP **actor** URI targeting the *Receiving MSH* node.

1690 As a minimum, it MUST contain an **Acknowledgment** element with a **RefToMessageId** containing the
 1691 same value as the **MessageId** element in the message being acknowledged. This message MUST be
 1692 placed in persistent storage with the same **PersistDuration** as the original message.

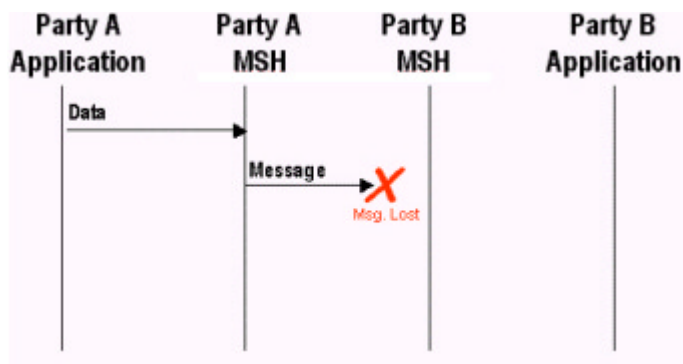
1693 The *Acknowledgment Message* can be sent at the same time as the response to the received message.
 1694 In this case, the values for the **MessageHeader** elements of the *Acknowledgment Message* are
 1695 determined by the **Service** and **Action** associated with the business response.

1696 If an *Acknowledgment Message* is being sent on its own, then the value of the **MessageHeader** elements
 1697 MUST be set as follows:

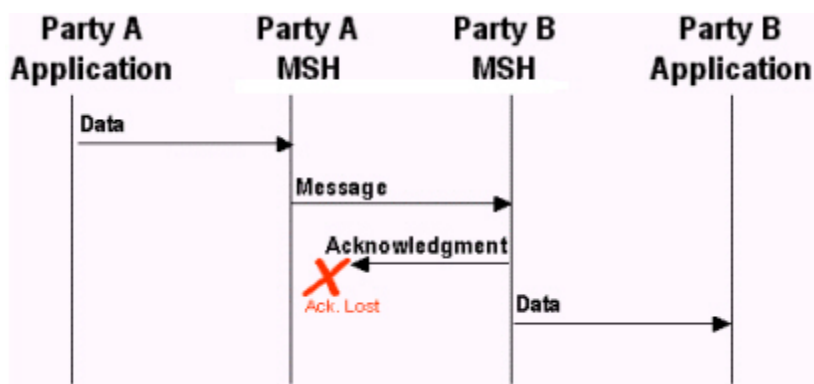
- 1698 • The **Service** element MUST be set to: **urn:oasis:names:tc:ebxml-msg:service**
- 1699 • The **Action** element MUST be set to **Acknowledgment**.
- 1700 • The **From** element MAY be populated with the **To** element extracted from the message received and all
 1701 child elements from the **To** element received SHOULD be included in this **From** element.
- 1702 • The **To** element MAY be populated with the **From** element extracted from the message received and all
 1703 child elements from the **From** element received SHOULD be included in this **To** element.
- 1704 • The **RefToMessageId** element MUST be set to the **MessageId** of the message received.

1705 **6.5.4 Resending Lost Application Messages**

1706 This section describes the behavior required by the sender and receiver of a message in order to handle
 1707 lost messages. A message is "lost" when a *Sending MSH* does not receive a positive acknowledgment to
 1708 a message. For example, it is possible a *message* was lost:



1709

1710 **Figure 6-2 Undelivered Message**1711 It is also possible the *Acknowledgment Message* was lost, for example:

1712

1713 **Figure 6-3 Lost Acknowledgment Message**1714 Note: *Acknowledgment Messages* are never acknowledged.1715 The rules applying to the non-receipt of an anticipated *Acknowledgment Message* due to the loss of either the
1716 application message or the *Acknowledgment Message* are as follows:

- 1717 • The *Sending MSH* MUST resend the original message if an *Acknowledgment Message* has been requested
1718 but has not been received and the following are true:
 - 1719 • At least the time specified in the **RetryInterval** parameter has passed since the message was last sent,
 - 1720 • The message has been resent less than the number of times specified in the **Retries** parameter.
- 1721 • If the *Sending MSH* does not receive an *Acknowledgment Message* after the maximum number of retries,
1722 the *Sending MSH* SHALL notify the application and/or system administrator function of the failure to receive
1723 an *Acknowledgment Message* (see also section 4.2.3.2.4 concerning treatment of errors).
- 1724 • If the *Sending MSH* detects a communications protocol error, the *Sending MSH* MUST resend the message
1725 using the same algorithm as if it has not received an *Acknowledgment Message*.

1726 **6.5.5 Resending Acknowledgments**1727 If the *Receiving MSH* receives a message it discovers to be a duplicate, it should resend the original
1728 *Acknowledgment Message* if the message is stored in *persistent store*. In this case, do the following:1729 Look in persistent storage for the first response to the received message (i.e. it contains a
1730 **RefToMessageId** that matches the **MessageId** of the received message).1731 If a response message was found in *persistent storage* then resend the persisted message back to the
1732 MSH that sent the received message. If no response message was found in *persistent storage*, then:

- 1733 (1) If **syncReplyMode** is not set to **none** and if the CPA indicates an application response is
1734 included, then it must be the case that the application has not finished processing the earlier

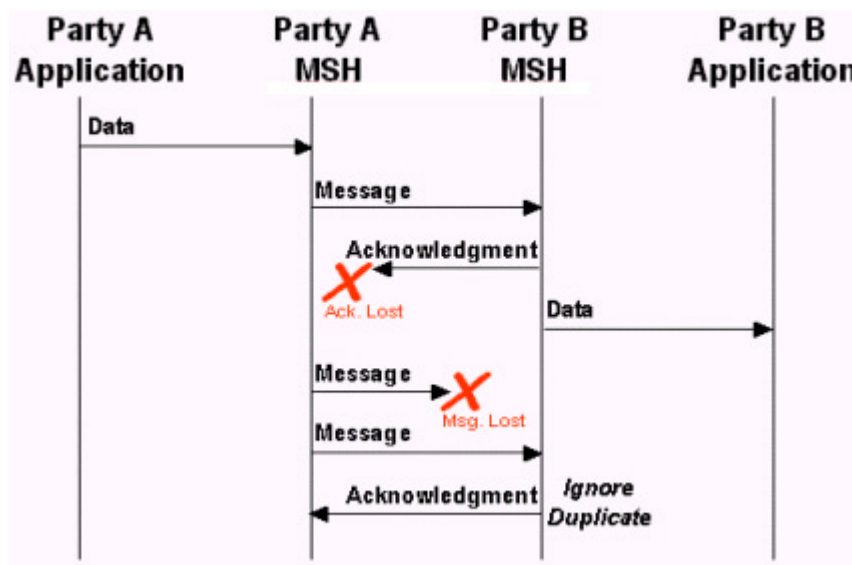
1735 copy of the same message. Therefore, wait for the response from the application and then
 1736 return that response synchronously over the same connection that was used for the
 1737 retransmission.

1738 (2) Otherwise, generate an *Acknowledgment Message*.

1739 6.5.6 Duplicate Message Handling

1740 In the context of this specification:

- 1741 • an "identical message" – a *message* containing the same ebXML SOAP **Header**, **Body** and ebXML Payload
 1742 Container(s) as the earlier sent *message*.
- 1743 • a "duplicate message" – a *message* containing the same **MessageId** as a previously received message.
- 1744 • the "first response message" – the message with the earliest **Timestamp** in the **MessageData** element
 1745 having the same **RefToMessageId** as the duplicate message.



1746

1747

Figure 6-4 Resending Unacknowledged Messages

1748 The diagram above shows the behavior to be followed by the *Sending* and *Receiving* MSH for messages
 1749 sent with an **AckRequested** element and a **DuplicateElimination** element. Specifically:

- 1750 1) The sender of the *message* (e.g. Party A MSH) MUST resend the "identical message" if no
 1751 *Acknowledgment Message* is received.
- 1752 2) When the recipient (Party B MSH) of the *message* receives a "duplicate message", it MUST resend to
 1753 the sender (Party A MSH) an *Acknowledgment Message* identical to the *first response message* sent
 1754 to the sender Party A MSH).
- 1755 3) The recipient of the *message* (Party B MSH) MUST NOT forward the message a second time to the
 1756 application/process.

1757 6.5.7 Failed Message Delivery

1758 If a message sent with an **AckRequested** element cannot be delivered, the MSH or process handling the
 1759 message (as in the case of a routing intermediary) SHALL send a delivery failure notification to the *From*
 1760 *Party*. The delivery failure notification message is an *Error Message* with **errorCode** of **DeliveryFailure**
 1761 and a **severity** of:

- 1762 • **Error** if the party who detected the problem could not transmit the message (e.g. the communications
 1763 transport was not available)
- 1764 • **Warning** if the message was transmitted, but an *Acknowledgment Message* was not received. This means
 1765 the message probably was not delivered.

1766 It is possible an error message with an **Error** element having an **errorCode** set to **DeliveryFailure**
 1767 cannot be delivered successfully for some reason. If this occurs, then the *From Party*, the ultimate
 1768 destination for the *Error Message*, MUST be informed of the problem by other means. How this is done is
 1769 outside the scope of this specification

1770 Note: If the *From Party MSH* receives an *Acknowledgment Message* from the *To Party MSH*, it should ignore all
 1771 other **DeliveryFailure** or *Acknowledgment Messages*.

1772 6.6 Reliable Messaging Combinations

	Duplicate- Elimination [§]	AckRequested ToPartyMSH	AckRequested NextMSH	Comment
1	Y	Y	Y	Once-And-Only-Once Reliable Messaging at the End-To-End and At-Least-Once to the Intermediate. Intermediate and To Party can issue Delivery Failure Notifications if they cannot deliver.
2	Y	Y	N	Once-And-Only-Once Reliable Message at the End-To-End level only based upon end-to-end retransmission
3	Y	N	Y	At-Least-Once Reliable Messaging at the Intermediate Level – Once-And-Only-Once end-to-end if all Intermediates are Reliable. No End-to-End notification.
4	Y	N	N	At-Most-Once Duplicate Elimination only at the To Party. No retries at the Intermediate or the End.
5	N	Y	Y	At-Least-Once Reliable Messaging with duplicates possible at the Intermediate and the To Party.
6	N	Y	N	At-Least-Once Reliable Messaging duplicates possible at the Intermediate and the To Party.
7	N	N	Y	At-Least-Once Reliable Messaging to the Intermediate and at the End. No End-to-End notification.
8	N	N	N	Best Effort

1773 [§]Duplicate Elimination is only performed at the To Party MSH, not at the Intermediate Level.

1774 7 Message Status Service

1775 The Message Status Request Service consists of the following:

- 1776 • A Message Status Request message containing details regarding a message previously sent is sent to a
 1777 Message Service Handler (MSH)
- 1778 • The Message Service Handler receiving the request responds with a Message Status Response message.

1779 A Message Service Handler SHOULD respond to Message Status Requests for messages that have
 1780 been sent reliably and the **MessageId** in the **RefToMessageId** is present in *persistent storage* (see
 1781 section 6.1).

1782 A Message Service Handler MAY respond to Message Status Requests for messages that have not been
 1783 sent reliably.

1784 A Message Service SHOULD NOT use the Message Status Request Service to implement Reliable
 1785 Messaging.

1786 If a *Receiving MSH* does not support the service requested, it SHOULD return an *Error Message* with an
 1787 **errorCode** of **NotSupported** and a **highestSeverity** attribute set to **Error**. Each service is described
 1788 below.

1789 7.1 Message Status Messages

1790 7.1.1 Message Status Request Message

1791 A Message Status Request message consists of an *ebXML Message* with no ebXML Payload Container
1792 and the following:

- 1793 • a **MessageHeader** element containing:
 - 1794 • a **From** element identifying the *Party* that created the Message Status Request message
 - 1795 • a **To** element identifying a *Party* who should receive the message.
 - 1796 • a **Service** element that contains: *urn:oasis:names:tc:ebxml-msg:service*
 - 1797 • an **Action** element that contains **StatusRequest**
 - 1798 • a **MessageData** element
- 1799 • a **StatusRequest** element containing:
 - 1800 • a **RefToMessageld** element in **StatusRequest** element containing the **Messageld** of the message
1801 whose status is being queried.
- 1802 • an [XMLDSIG] **Signature** element (see section 4.1 for more details)

1803 The message is then sent to the *To Party*.

1804 7.1.2 Message Status Response Message

1805 Once the *To Party* receives the Message Status Request message, they SHOULD generate a Message
1806 Status Response message with no ebXML Payload Container consisting of the following:

- 1807 • a **MessageHeader** element containing:
 - 1808 ▪ a **From** element that identifies the sender of the Message Status Response message
 - 1809 ▪ a **To** element set to the value of the **From** element in the Message Status Request message
 - 1810 ▪ a **Service** element that contains *urn:oasis:names:tc:ebxml-msg:service*
 - 1811 ▪ an **Action** element that contains **StatusResponse**
 - 1812 ▪ a **MessageData** element containing:
 - 1813 • a **RefToMessageld** that identifies the Message Status Request message.
- 1814 • **StatusResponse** element (see section 7.2.3)
- 1815 • an [XMLDSIG] **Signature** element (see section 4.1 for more details)

1816 The message is then sent to the *To Party*.

1817 7.1.3 Security Considerations

1818 Parties who receive a Message Status Request message SHOULD always respond to the message.
1819 However, they MAY ignore the message instead of responding with **messageStatus** set to
1820 **Unauthorized** if they consider the sender of the message to be unauthorized. The decision process
1821 resulting in this course of action is implementation dependent.

1822 7.2 StatusRequest Element

1823 The OPTIONAL **StatusRequest** element is an immediate child of a SOAP **Body** and is used to identify
1824 an earlier message whose status is being requested (see section 7.3.5).

1825 The **StatusRequest** element consists of the following:

- 1826 • an **id** attribute (see section 2.3.7 for details)
- 1827 • a **version** attribute (see section 2.3.8 for details)
- 1828 • a **RefToMessageld** element

1829 7.2.1 RefToMessageId Element

1830 A REQUIRED *RefToMessageId* element contains the *MessageId* of the message whose status is being
1831 requested.

1832 7.2.2 StatusRequest Sample

1833 An example of the *StatusRequest* element is given below:

```
1834 <eb:StatusRequest eb:version="2.0" >
1835   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1836 </eb:StatusRequest>
```

1837 7.2.3 StatusRequest Element Interaction

1838 A *StatusRequest* element MUST NOT be present with the following elements:

- 1839 • a *Manifest* element
- 1840 • a *StatusResponse* element
- 1841 • an *ErrorList* element

1842 7.3 StatusResponse Element

1843 The OPTIONAL *StatusResponse* element is an immediate child of a SOAP *Body* and is used by one
1844 MSH to describe the status of processing of a message.

1845 The *StatusResponse* element consists of the following elements and attributes:

- 1846 • an *id* attribute (see section 2.3.7 for details)
- 1847 • a *version* attribute (see section 2.3.8 for details)
- 1848 • a *RefToMessageId* element
- 1849 • a *Timestamp* element
- 1850 • a *messageStatus* attribute

1851 7.3.1 RefToMessageId Element

1852 A REQUIRED *RefToMessageId* element contains the *MessageId* of the message whose status is being
1853 reported. *RefToMessageId* element child of the *MessageData* element of a message containing a
1854 *StatusResponse* element SHALL have the *MessageId* of the message containing the *StatusRequest*
1855 element to which the *StatusResponse* element applies. The *RefToMessageId* child element of the
1856 *StatusRequest* or *StatusResponse* element SHALL contain the *MessageId* of the message whose
1857 status is being queried.

1858 7.3.2 Timestamp Element

1859 The *Timestamp* element contains the time the message, whose status is being reported, was received
1860 (section 3.1.6.2.). This MUST be omitted if the message, whose status is being reported, is
1861 *NotRecognized* or the request was *Unauthorized*.

1862 7.3.3 messageStatus attribute

1863 The REQUIRED *messageStatus* attribute identifies the status of the message identified by the
1864 *RefToMessageId* element. It SHALL be set to one of the following values:

- 1865 • *Unauthorized* – the Message Status Request is not authorized or accepted
- 1866 • *NotRecognized* – the message identified by the *RefToMessageId* element in the *StatusResponse*
1867 element is not recognized
- 1868 • *Received* – the message identified by the *RefToMessageId* element in the *StatusResponse* element has
1869 been received by the MSH
- 1870 • *Processed* – the message identified by the *RefToMessageId* element in the *StatusResponse* element has
1871 been processed by the MSH

- 1872 • **Forwarded** – the message identified by the **RefToMessageId** element in the **StatusResponse** element has
1873 been forwarded by the MSH to another MSH

1874 Note: if a Message Status Request is sent after the elapsed time indicated by **PersistDuration** has passed since the
1875 message being queried was sent, the Message Status Response may indicate the **MessageId** was **NotRecognized** –
1876 the **MessageId** is no longer in persistent storage.

1877 7.3.4 StatusResponse Sample

1878 An example of the **StatusResponse** element is given below:

```
1879 <eb:StatusResponse eb:version="2.0" eb:messageStatus="Received">
1880   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1881   <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
1882 </eb:StatusResponse>
```

1883 7.3.5 StatusResponse Element Interaction

1884 This element MUST NOT be present with the following elements:

- 1885 • a **Manifest** element
1886 • a **StatusRequest** element
1887 • an **ErrorList** element with a **highestSeverity** attribute set to **Error**

1888 8 Message Service Handler Ping Service

1889 The OPTIONAL Message Service Handler Ping Service enables one MSH to determine if another MSH is
1890 operating. It consists of:

- 1891 • one MSH sending a Message Service Handler Ping message to a MSH, and
1892 • another MSH, receiving the Ping, responding with a Message Service Handler Pong message.

1893 If a *Receiving MSH* does not support the service requested, it SHOULD return an *Error Message* with an
1894 **errorCode** of **NotSupported** and a **highestSeverity** attribute set to **Error**.

1895 8.1 Message Service Handler Ping Message

1896 A Message Service Handler Ping (MSH Ping) message consists of an *ebXML Message* containing no
1897 ebXML Payload Container and the following:

- 1898 • a **MessageHeader** element containing the following:
- 1899 • a **From** element identifying the *Party* creating the MSH Ping message
 - 1900 • a **To** element identifying the *Party* being sent the MSH Ping message
 - 1901 • a **CPAId** element
 - 1902 • a **ConversationId** element
 - 1903 • a **Service** element containing: **urn:oasis:names:tc:ebxml-msg:service**
 - 1904 • an **Action** element containing **Ping**
 - 1905 • a **MessageData** element
- 1906 • an [XMLDSIG] **Signature** element (see section 4.1 for details).

1907 The message is then sent to the *To Party*.

1908 An example Ping:

```
1909 . . .Transport Headers
1910 SOAPAction: "ebXML"
1911 Content-type: multipart/related; boundary="ebXMLBoundary"
1912
1913 --ebXMLBoundary
1914 Content-Type: text/xml
1915
```

```

1916 <?xml version="1.0" encoding="UTF-8"?>
1917 <SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1918   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
1919   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
1920     http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
1921 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
1922   xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
1923     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
1924   <eb:MessageHeader version="2.0" SOAP:mustUnderstand="1"
1925     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
1926     xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
1927       http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
1928     <eb:From> <eb:PartyId>urn:duns:123456789</eb:PartyId> </eb:From>
1929     <eb:To> <eb:PartyId>urn:duns:912345678</eb:PartyId> </eb:To>
1930     <eb:CPAId>20001209-133003-28572</eb:CPAId>
1931     <eb:ConversationId>20010215-111213-28572</eb:ConversationId>
1932     <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
1933     <eb:Action>Ping</eb:Action>
1934     <eb:MessageData>
1935       <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
1936       <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
1937     </eb:MessageData>
1938   </eb:MessageHeader>
1939 </SOAP:Header>
1940 <SOAP:Body/>
1941 </SOAP:Envelope>
1942
1943 --ebXMLBoundary--

```

1944 Note: The above example shows a Multipart/Related MIME structure with only one bodypart.

1945 8.2 Message Service Handler Pong Message

1946 Once the *To Party* receives the MSH Ping message, they MAY generate a Message Service Handler
 1947 Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML Payload Container
 1948 and the following:

- 1949 • a **MessageHeader** element containing the following:
 - 1950 • a **From** element identifying the creator of the MSH Pong message
 - 1951 • a **To** element identifying a *Party* that generated the MSH Ping message
 - 1952 • a **CPAId** element
 - 1953 • a **ConversationId** element
 - 1954 • a **Service** element containing the value: *urn:oasis:names:tc:ebxml-msg:service*
 - 1955 • an **Action** element containing the value **Pong**
 - 1956 • a **MessageData** element containing:
 - 1957 • a **RefToMessageId** identifying the MSH Ping message.
- 1958 • an [XMLDSIG] **Signature** element (see section 4.1.1 for details).

1959 An example Pong:

```

1960 . . .Transport Headers
1961 SOAPAction: "ebXML"
1962 Content-Type: text/xml
1963
1964 <?xml version="1.0" encoding="UTF-8"?>
1965 <SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1966   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
1967   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
1968     http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
1969 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
1970   xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
1971     http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
1972   <eb:MessageHeader eb:version="2.0" SOAP:mustUnderstand="1">
1973     <eb:From> <eb:PartyId>urn:duns:912345678</eb:PartyId> </eb:From>
1974     <eb:To> <eb:PartyId>urn:duns:123456789</eb:PartyId> </eb:To>

```



```

1975 <eb:CPAId>20001209-133003-28572</eb:CPAId>
1976 <eb:ConversationId>20010215-111213-28572</eb:ConversationId>
1977 <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
1978 <eb:Action>Pong</eb:Action>
1979 <eb:MessageData>
1980 <eb:MessageId>20010215-111213-395884@example2.com</eb:MessageId>
1981 <eb:Timestamp>2001-02-15T11:12:13</eb:Timestamp>
1982 <eb:RefToMessageId>20010215-111212-28572@example.com</eb:RefToMessageId>
1983 </eb:MessageData>
1984 </eb:MessageHeader>
1985 </SOAP:Header>
1986 <SOAP:Body/>
1987 </SOAP:Envelope>

```

1988 Note: This example shows a non-multipart MIME structure.

1989 8.3 Security Considerations

1990 Parties who receive a MSH Ping message SHOULD always respond to the message. However, there is
 1991 a risk some parties might use the MSH Ping message to determine the existence of a Message Service
 1992 Handler as part of a security attack on that MSH. Therefore, recipients of a MSH Ping MAY ignore the
 1993 message if they consider that the sender of the message received is unauthorized or part of some attack.
 1994 The decision process that results in this course of action is implementation dependent.

1995 9 MessageOrder Module

1996 The **MessageOrder** module allows messages to be presented to the *To Party* in a particular order. This
 1997 is accomplished through the use of the **MessageOrder** element. Reliable Messaging MUST be used
 1998 when a **MessageOrder** element is present.

1999 **MessageOrder** module MUST only be used in conjunction with the ebXML Reliable Messaging Module
 2000 (section 6) with a scheme of Once-And-Only-Once (sections 6.6). If a sequence is sent and one
 2001 message fails to arrive at the *To Party MSH*, all subsequent messages will also fail to be presented to the
 2002 *To Party Application* (see **status** attribute section 9.1.1).

2003 9.1 MessageOrder Element

2004 The **MessageOrder** element is an OPTIONAL extension to the SOAP **Header** requesting the
 2005 preservation of message order in this conversation.

2006 The **MessageOrder** element contains the following:

- 2007 • a **id** attribute (see section 2.3.7)
- 2008 • a **version** attribute (see section 2.3.8 for details)
- 2009 • a SOAP **mustUnderstand** attribute with a value of "1" (see section 2.3.9 for details)
- 2010 • a **SequenceNumber** element

2011 When the **MessageOrder** element is present, **DuplicateElimination** MUST also be present and
 2012 **SyncReply** MUST NOT be present.

2013 9.1.1 SequenceNumber Element

2014 The REQUIRED **SequenceNumber** element indicates the sequence a *Receiving MSH* MUST process
 2015 messages. The **SequenceNumber** is unique within the **ConversationId** and MSH. The *From Party MSH*
 2016 and the *To Party MSH* each set an independent **SequenceNumber** as the *Sending MSH* within the
 2017 **ConversationId**. It is set to zero on the first message from that MSH within a conversation and then
 2018 incremented by one for each subsequent message sent.

2019 A MSH that receives a message with a **SequenceNumber** element MUST NOT pass the message to an
 2020 application until all the messages with a lower **SequenceNumber** have been passed to the application.

2021 If the implementation defined limit for saved out-of-sequence messages is reached, then the *Receiving*
 2022 *MSH* MUST indicate a delivery failure to the *Sending MSH* with **errorCode** set to **DeliveryFailure** and
 2023 **severity** set to **Error** (see section 4.1.5).

2024 The **SequenceNumber** element is an integer value incremented by the *Sending MSH* (e.g. 0, 1, 2, 3, 4...)
 2025 for each application-prepared message sent by that MSH within the **ConversationId**. The next value after
 2026 99999999 in the increment is "0". The value of **SequenceNumber** consists of ASCII numerals in the
 2027 range 0-99999999. In following cases, **SequenceNumber** takes the value "0":

- 2028 1. First message from the *Sending MSH* within the conversation
- 2029 2. First message after resetting **SequenceNumber** information by the *Sending MSH*
- 2030 3. First message after wraparound (next value after 99999999)

2031 The **SequenceNumber** element has a single attribute, **status**. This attribute is an enumeration, which
 2032 SHALL have one of the following values:

- 2033 • **Reset** – the **SequenceNumber** is reset as shown in 1 or 2 above
- 2034 • **Continue** – the **SequenceNumber** continues sequentially (including 3 above)

2035 When the **SequenceNumber** is set to "0" because of 1 or 2 above, the *Sending MSH* MUST set the
 2036 **status** attribute of the message to **Reset**. In all other cases, including 3 above, the **status** attribute
 2037 MUST be set to **Continue**. The default value of the **status** attribute is **Continue**.

2038 A *Sending MSH* MUST wait before resetting the **SequenceNumber** of a conversation until it has received
 2039 confirmation of all the messages previously sent for the conversation. Only when all the sent Messages
 2040 are accounted for, can the *Sending MSH* reset the **SequenceNumber**.

2041 9.1.2 MessageOrder Sample

2042 An example of the **MessageOrder** element is given below:

```
2043 <eb:MessageOrder eb:version="2.0" SOAP:mustUnderstand="1">
2044   <eb:SequenceNumber>00000010</eb:SequenceNumber>
2045 </eb:MessageOrder>
```

2046 9.2 MessageOrder Element Interaction

2047 For this version of the ebXML Messaging Specification, the **MessageOrder** element MUST NOT be
 2048 present with the **SyncReply** element. If these two elements are received in the same message, the
 2049 *Receiving MSH* SHOULD report an error (see section 4.1.5) with **errorCode** set to **Inconsistent** and
 2050 **severity** set to **Error**.

2051 10 Multi-Hop Module

2052 Multi-hop is the process of passing the message through one or more intermediary nodes or MSH's. An
 2053 Intermediary is any node or MSH where the message is received, but is not the *Sending* or *Receiving*
 2054 *MSH*. This node is called an Intermediary.

2055 Intermediaries may be for the purpose of Store-and-Forward or may be involved in some processing
 2056 activity such as a trusted third-party timestamp service. For the purposes of this version of this
 2057 specification, Intermediaries are considered only as Store-and-Forward entities.

2058 Intermediaries MAY be involved in removing and adding SOAP extension elements or modules targeted
 2059 either to the **Next** SOAP node or the **NextMSH**. SOAP rules specify, the receiving node must remove
 2060 any element or module targeted to the **Next** SOAP node. If the element or module needs to continue to
 2061 appear on the SOAP message destined to the **Next** SOAP node, or in this specification the **NextMSH**, it
 2062 must be reapplied. This deleting and adding of elements or modules poses potential difficulties for signed
 2063 ebXML messages. Any Intermediary node or MSH MUST NOT change, format or in any way modify any
 2064 element not targeted to the Intermediary. Any such change may invalidate the signature.

2065 10.1 Multi-hop Reliable Messaging

2066 Multi-hop (hop-to-hop) Reliable Messaging is accomplished using the **AckRequested** element (section
2067 6.3.1) and an **Acknowledgment Message** containing an **Acknowledgment** element (section 6.3.1.4) each
2068 with a SOAP **actor** of **Next MSH** (section 2.3.10) between the **Sending MSH** and the **Receiving MSH**.
2069 This MAY be used in store-and-forward multi-hop situations.

2070 The use of the duplicate elimination is not required for Intermediate nodes. Since duplicate elimination by
2071 an intermediate MSH can interfere with End-to-End Reliable Messaging Retries, the intermediate MSH
2072 MUST know it is an intermediate and MUST NOT perform duplicate elimination tasks.

2073 At this time, the values of **Retry** and **RetryInterval** between Intermediate MSHs remains implementation
2074 specific. See section 6.4 for more detail on Reliable Messaging.

2075 10.1.1 AckRequested Sample

2076 An example of the **AckRequested** element targeted at the **NextMSH** is given below:

```
2077 <eb:AckRequested SOAP:mustUnderstand="1" eb:version="2.0" eb:signed="false"  
2078 SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
```

2079 In the preceding example, an **Acknowledgment Message** is requested of the next ebXML MSH node (see
2080 section 2.3.10) in the message. The **Acknowledgment** element generated MUST be targeted at the next
2081 ebXML MSH node along the reverse message path (the **Sending MSH**) using the SOAP **actor** with a
2082 value of **NextMSH** (section 2.3.10).

2083 Any Intermediary receiving an **AckRequested** with SOAP **actor** of **NextMSH** MUST remove the
2084 **AckRequested** element before forwarding to the next MSH. Any Intermediary MAY insert a single
2085 **AckRequested** element into the SOAP **Header** with a SOAP **actor** of **NextMSH**. There SHALL NOT be
2086 two **AckRequested** elements targeted at the next MSH.

2087 When the **SyncReply** element is present, an **AckRequested** element with SOAP **actor** of **NextMSH**
2088 MUST NOT be present. If the **SyncReply** element is not present, the Intermediary MAY return the
2089 Intermediate **Acknowledgment Message** synchronously with a synchronous transport protocol. If these
2090 two elements are received in the same message, the **Receiving MSH** SHOULD report an error (see
2091 section 4.1.5) with **errorCode** set to **Inconsistent** and **severity** set to **Error**.

2092 10.1.2 Acknowledgment Sample

2093 An example of the **Acknowledgment** element targeted at the **NextMSH** is given below:

```
2094 <eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0"  
2095 SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH">  
2096 <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>  
2097 <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>  
2098 <eb:From> <eb:PartyId>uri:www.example.com</eb:PartyId> </eb:From>  
2099 </eb:Acknowledgment>
```

2100 10.1.3 Multi-Hop Acknowledgments

2101 There MAY be two **AckRequested** elements on the same message. An **Acknowledgment** MUST be
2102 sent for each **AckRequested** using an identical SOAP **actor** attribute as the **AckRequested** element.

2103 If the **Receiving MSH** is the **To Party MSH**, then see section 6.5.2. If the **Receiving MSH** is the **To Party**
2104 **MSH** and there is an **AckRequested** element targeting the Next MSH (the **To Party MSH** is acting in both
2105 roles), then perform both procedures (this section and section 6.5.2) for generating **Acknowledgment**
2106 **Messages**. This MAY require sending two **Acknowledgment** elements, possibly on the same message,
2107 one targeted for the **Next MSH** and one targeted for the **To Party MSH**.

2108 There MAY be multiple **Acknowledgements** elements, on the same message or on different messages,
2109 returning from either the Next MSH or from the **To Party MSH**. A MSH supporting Multi-hop MUST
2110 differentiate, based upon the **actor**, which **Acknowledgment** is being returned and act accordingly.

2111 If this is an **Acknowledgment Message** as defined in section 6 then:

- 2112 1 Look for a message in *persistent storage* with a **MessageId** the same as the value of
2113 **RefToMessageId** on the received Message.
- 2114 2 If a message is found in *persistent storage* then mark the persisted message as delivered.

2115 If an **AckRequested** element is present (not an *Acknowledgment Message*) then generate an
2116 *Acknowledgment Message* in response (this may be as part of another message). The *Receiving MSH*
2117 MUST NOT send an *Acknowledgment Message* until the message has been persisted or delivered to the
2118 *Next MSH*.

2119 **10.1.4 Signing Multi-Hop Acknowledgments**

2120 When a signed Intermediate *Acknowledgment Message* is requested (i.e. a signed *Acknowledgment*
2121 *Message* with a SOAP **actor** of *NextMSH*), it MUST be sent by itself and not bundled with any other
2122 message. The XML Signature [XMLDSIG] **Signature** element with **Transforms**, as described in section
2123 4.1.3, will exclude this **Acknowledgment** element. To send a signed *Acknowledgment Message* with
2124 SOAP **actor** of *NextMSH*, create a message with no payloads, including a single **Acknowledgment**
2125 element (see section 6.3.2.6), and a [XMLDSIG] **Signature** element with the following **Transforms**:

```
2126           <Transforms>  
2127             <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>  
2128             <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>  
2129           </Transforms>
```

2130 **10.1.5 Multi-Hop Security Considerations**

2131 SOAP messaging allows intermediaries to add or remove elements targeted to the intermediary node.
2132 This has potential conflicts with end-to-end signatures since the slightest change in any character of the
2133 SOAP **Envelope** or to a payload will invalidate the **ds:Signature** by changing the calculated digest.
2134 Intermediaries MUST NOT add or remove elements unless they contain a SOAP **actor** of *next* or
2135 *nextMSH*. Intermediaries MUST NOT disturb white space – line terminators (CR/LF), tabs, spaces, etc. –
2136 outside those elements being added or removed.

2137 **10.2 Message Ordering and Multi-Hop**

2138 Intermediary MSH nodes MUST NOT participate in Message Order processing as specified in section 9.

2139

Part III. Normative Appendices

2140

Appendix A The ebXML SOAP Extension Elements Schema

2141 The OASIS ebXML Messaging Technical Committee has provided a version of the SOAP 1.1 envelope
2142 schema specified using the schema vocabulary that conforms to the W3C XML Schema
2143 Recommendation specification [XMLSchema].

2144 SOAP1.1- <http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd>

2145 It was necessary to craft a schema for the XLINK [XLINK] attribute vocabulary to conform to the W3C
2146 XML Schema Recommendation [XMLSchema]. This schema is referenced from the ebXML SOAP
2147 extension elements schema and is available from the following URL:

2148 Xlink - <http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd>

```

2149 <?xml version="1.0" encoding="UTF-8"?>
2150 <!-- Some parsers may require explicit declaration of xmlns:xml="http://www.w3.org/XML/1998/namespace".
2151      In that case, a copy of this schema augmented with the above declaration should be cached and used
2152      for the purpose of schema validation on ebXML messages. -->
2153 <schema targetNamespace="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
2154   xmlns:tns="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
2155   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2156   xmlns:xlink="http://www.w3.org/1999/xlink"
2157   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
2158   xmlns="http://www.w3.org/2001/XMLSchema"
2159   elementFormDefault="qualified"
2160   attributeFormDefault="qualified"
2161   version="1.0">
2162   <import namespace="http://www.w3.org/2000/09/xmldsig#"
2163     schemaLocation="http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
2164   <import namespace="http://www.w3.org/1999/xlink"
2165     schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd"/>
2166   <import namespace="http://schemas.xmlsoap.org/soap/envelope/"
2167     schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd"/>
2168   <import namespace="http://www.w3.org/XML/1998/namespace"
2169     schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
2170   <!-- MANIFEST, for use in soap:Body element -->
2171   <element name="Manifest">
2172     <complexType>
2173       <sequence>
2174         <element ref="tns:Reference" maxOccurs="unbounded"/>
2175         <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2176       </sequence>
2177       <attributeGroup ref="tns:bodyExtension.grp"/>
2178     </complexType>
2179   </element>
2180   <element name="Reference">
2181     <complexType>
2182       <sequence>
2183         <element ref="tns:Schema" minOccurs="0" maxOccurs="unbounded"/>
2184         <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
2185         <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2186       </sequence>
2187       <attribute ref="tns:id"/>
2188       <attribute ref="xlink:type" fixed="simple"/>
2189       <attribute ref="xlink:href" use="required"/>
2190       <attribute ref="xlink:role"/>
2191       <anyAttribute namespace="##other" processContents="lax"/>
2192     </complexType>
2193   </element>
2194   <element name="Schema">
2195     <complexType>
2196       <attribute name="location" type="anyURI" use="required"/>
2197       <attribute name="version" type="tns:non-empty-string"/>
2198     </complexType>

```

```

2199 </element>
2200 <!-- MESSAGEHEADER, for use in soap:Header element -->
2201 <element name="MessageHeader">
2202   <complexType>
2203     <sequence>
2204       <element ref="tns:From"/>
2205       <element ref="tns:To"/>
2206       <element ref="tns:CPAId"/>
2207       <element ref="tns:ConversationId"/>
2208       <element ref="tns:Service"/>
2209       <element ref="tns:Action"/>
2210       <element ref="tns:MessageData"/>
2211       <element ref="tns:DuplicateElimination" minOccurs="0"/>
2212       <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
2213       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2214     </sequence>
2215     <attributeGroup ref="tns:headerExtension.grp"/>
2216   </complexType>
2217 </element>
2218 <element name="CPAId" type="tns:non-empty-string"/>
2219 <element name="ConversationId" type="tns:non-empty-string"/>
2220 <element name="Service">
2221   <complexType>
2222     <simpleContent>
2223       <extension base="tns:non-empty-string">
2224         <attribute name="type" type="tns:non-empty-string"/>
2225       </extension>
2226     </simpleContent>
2227   </complexType>
2228 </element>
2229 <element name="Action" type="tns:non-empty-string"/>
2230 <element name="MessageData">
2231   <complexType>
2232     <sequence>
2233       <element ref="tns:MessageId"/>
2234       <element ref="tns:Timestamp"/>
2235       <element ref="tns:RefToMessageId" minOccurs="0"/>
2236       <element ref="tns:TimeToLive" minOccurs="0"/>
2237     </sequence>
2238   </complexType>
2239 </element>
2240 <element name="MessageId" type="tns:non-empty-string"/>
2241 <element name="TimeToLive" type="dateTime"/>
2242 <element name="DuplicateElimination">
2243 </element>
2244 <!-- SYNC REPLY, for use in soap:Header element -->
2245 <element name="SyncReply">
2246   <complexType>
2247     <sequence>
2248       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2249     </sequence>
2250     <attributeGroup ref="tns:headerExtension.grp"/>
2251     <attribute ref="soap:actor" use="required"/>
2252   </complexType>
2253 </element>
2254 <!-- MESSAGE ORDER, for use in soap:Header element -->
2255 <element name="MessageOrder">
2256   <complexType>
2257     <sequence>
2258       <element ref="tns:SequenceNumber"/>
2259       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2260     </sequence>
2261     <attributeGroup ref="tns:headerExtension.grp"/>
2262   </complexType>
2263 </element>
2264 <element name="SequenceNumber" type="tns:sequenceNumber.type"/>
2265 <!-- ACK REQUESTED, for use in soap:Header element -->
2266 <element name="AckRequested">
2267   <complexType>
2268     <sequence>
2269       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>

```

```

2270     </sequence>
2271     <attributeGroup ref="tns:headerExtension.grp"/>
2272     <attribute ref="soap:actor"/>
2273     <attribute name="signed" type="boolean" use="required"/>
2274     </complexType>
2275 </element>
2276 <!-- ACKNOWLEDGMENT, for use in soap:Header element -->
2277 <element name="Acknowledgment">
2278     <complexType>
2279         <sequence>
2280             <element ref="tns:Timestamp"/>
2281             <element ref="tns:RefToMessageId"/>
2282             <element ref="tns:From" minOccurs="0"/>
2283             <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
2284             <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2285         </sequence>
2286         <attributeGroup ref="tns:headerExtension.grp"/>
2287         <attribute ref="soap:actor"/>
2288     </complexType>
2289 </element>
2290 <!-- ERROR LIST, for use in soap:Header element -->
2291 <element name="ErrorList">
2292     <complexType>
2293         <sequence>
2294             <element ref="tns:Error" maxOccurs="unbounded"/>
2295             <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2296         </sequence>
2297         <attributeGroup ref="tns:headerExtension.grp"/>
2298         <attribute name="highestSeverity" type="tns:severity.type" use="required"/>
2299     </complexType>
2300 </element>
2301 <element name="Error">
2302     <complexType>
2303         <sequence>
2304             <element ref="tns:Description" minOccurs="0"/>
2305             <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2306         </sequence>
2307         <attribute ref="tns:id"/>
2308         <attribute name="codeContext" type="anyURI"
2309             default="urn:oasis:names:tc:ebxml-msg:service:errors"/>
2310         <attribute name="errorCode" type="tns:non-empty-string" use="required"/>
2311         <attribute name="severity" type="tns:severity.type" use="required"/>
2312         <attribute name="location" type="tns:non-empty-string"/>
2313         <anyAttribute namespace="##other" processContents="lax"/>
2314     </complexType>
2315 </element>
2316 <!-- STATUS RESPONSE, for use in soap:Body element -->
2317 <element name="StatusResponse">
2318     <complexType>
2319         <sequence>
2320             <element ref="tns:RefToMessageId"/>
2321             <element ref="tns:Timestamp" minOccurs="0"/>
2322             <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2323         </sequence>
2324         <attributeGroup ref="tns:bodyExtension.grp"/>
2325         <attribute name="messageStatus" type="tns:messageStatus.type" use="required"/>
2326     </complexType>
2327 </element>
2328 <!-- STATUS REQUEST, for use in soap:Body element -->
2329 <element name="StatusRequest">
2330     <complexType>
2331         <sequence>
2332             <element ref="tns:RefToMessageId"/>
2333             <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2334         </sequence>
2335         <attributeGroup ref="tns:bodyExtension.grp"/>
2336     </complexType>
2337 </element>
2338 <!-- COMMON TYPES -->
2339 <complexType name="sequenceNumber.type">
2340     <simpleContent>

```

```

2341     <extension base="nonNegativeInteger">
2342         <attribute name="status" type="tns:status.type" default="Continue"/>
2343     </extension>
2344 </simpleContent>
2345 </complexType>
2346 <simpleType name="status.type">
2347     <restriction base="NMTOKEN">
2348         <enumeration value="Reset"/>
2349         <enumeration value="Continue"/>
2350     </restriction>
2351 </simpleType>
2352 <simpleType name="messageStatus.type">
2353     <restriction base="NMTOKEN">
2354         <enumeration value="Unauthorized"/>
2355         <enumeration value="NotRecognized"/>
2356         <enumeration value="Received"/>
2357         <enumeration value="Processed"/>
2358         <enumeration value="Forwarded"/>
2359     </restriction>
2360 </simpleType>
2361 <simpleType name="non-empty-string">
2362     <restriction base="string">
2363         <minLength value="1"/>
2364     </restriction>
2365 </simpleType>
2366 <simpleType name="severity.type">
2367     <restriction base="NMTOKEN">
2368         <enumeration value="Warning"/>
2369         <enumeration value="Error"/>
2370     </restriction>
2371 </simpleType>
2372 <!-- COMMON ATTRIBUTES and ATTRIBUTE GROUPS -->
2373 <attribute name="id" type="ID"/>
2374 <attribute name="version" type="tns:non-empty-string"/>
2375 <attributeGroup name="headerExtension.grp">
2376     <attribute ref="tns:id"/>
2377     <attribute ref="tns:version" use="required"/>
2378     <attribute ref="soap:mustUnderstand" use="required"/>
2379     <anyAttribute namespace="##other" processContents="lax"/>
2380 </attributeGroup>
2381 <attributeGroup name="bodyExtension.grp">
2382     <attribute ref="tns:id"/>
2383     <attribute ref="tns:version" use="required"/>
2384     <anyAttribute namespace="##other" processContents="lax"/>
2385 </attributeGroup>
2386 <!-- COMMON ELEMENTS -->
2387 <element name="PartyId">
2388     <complexType>
2389         <simpleContent>
2390             <extension base="tns:non-empty-string">
2391                 <attribute name="type" type="tns:non-empty-string"/>
2392             </extension>
2393         </simpleContent>
2394     </complexType>
2395 </element>
2396 <element name="To">
2397     <complexType>
2398         <sequence>
2399             <element ref="tns:PartyId" maxOccurs="unbounded"/>
2400             <element name="Role" type="tns:non-empty-string" minOccurs="0"/>
2401         </sequence>
2402     </complexType>
2403 </element>
2404 <element name="From">
2405     <complexType>
2406         <sequence>
2407             <element ref="tns:PartyId" maxOccurs="unbounded"/>
2408             <element name="Role" type="tns:non-empty-string" minOccurs="0"/>
2409         </sequence>
2410     </complexType>
2411 </element>

```



```
2412 <element name="Description">
2413   <complexType>
2414     <simpleContent>
2415       <extension base="tns:non-empty-string">
2416         <attribute ref="xml:lang" use="required"/>
2417       </extension>
2418     </simpleContent>
2419   </complexType>
2420 </element>
2421 <element name="RefToMessageId" type="tns:non-empty-string"/>
2422 <element name="Timestamp" type="dateTime"/>
2423 </schema>
```

2424

2424 Appendix B Communications Protocol Bindings

2425 B.1 Introduction

2426 One of the goals of this specification is to design a message handling service usable over a variety of
2427 network and application level transport protocols. These protocols serve as the "carrier" of ebXML
2428 Messages and provide the underlying services necessary to carry out a complete ebXML Message
2429 exchange between two parties. HTTP, FTP, Java Message Service (JMS) and SMTP are examples of
2430 application level transport protocols. TCP and SNA/LU6.2 are examples of network transport protocols.
2431 Transport protocols vary in their support for data content, processing behavior and error handling and
2432 reporting. For example, it is customary to send binary data in raw form over HTTP. However, in the case
2433 of SMTP it is customary to "encode" binary data into a 7-bit representation. HTTP is equally capable of
2434 carrying out *synchronous* or *asynchronous* message exchanges whereas it is likely that message
2435 exchanges occurring over SMTP will be *asynchronous*. This section describes the technical details
2436 needed to implement this abstract ebXML Message Handling Service over particular transport protocols.

2437 This section specifies communications protocol bindings and technical details for carrying *ebXML*
2438 *Message Service* messages for the following communications protocols:

- 2439 • Hypertext Transfer Protocol [RFC2616], in both *asynchronous* and *synchronous* forms of transfer.
- 2440 • Simple Mail Transfer Protocol [RFC2821], in *asynchronous* form of transfer only.

2441 B.2 HTTP

2442 B.2.1 Minimum level of HTTP protocol

2443 Hypertext Transfer Protocol Version 1.1 [RFC2616] is the minimum level of protocol that **MUST** be used.

2444 B.2.2 Sending ebXML Service messages over HTTP

2445 Even though several HTTP request methods are available, this specification only defines the use of HTTP
2446 POST requests for sending *ebXML Message Service* messages over HTTP. The identity of the ebXML
2447 MSH (e.g. `ebxmlhandler`) may be part of the HTTP POST request:

```
2448 POST /ebxmlhandler HTTP/1.1
```

2449 Prior to sending over HTTP, an ebXML Message **MUST** be formatted according to ebXML Message
2450 Service Specification. Additionally, the messages **MUST** conform to the HTTP specific MIME canonical
2451 form constraints specified in section 19.4 of RFC 2616 [RFC2616] specification.

2452 HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is **OPTIONAL** for such
2453 parts in an ebXML Service Message prior to sending over HTTP. However, content-transfer-encoding of
2454 such parts (e.g. using base64 encoding scheme) is not precluded by this specification.

2455 The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- 2456 • The **Content-Type: Multipart/Related** MIME header with the associated parameters, from the
2457 ebXML Service Message Envelope **MUST** appear as an HTTP header.
- 2458 • All other MIME headers that constitute the ebXML Message Envelope **MUST** also become part of the HTTP
2459 header.
- 2460 • The mandatory `SOAPAction` HTTP header field must also be included in the HTTP header and **MAY** have
2461 a value of "ebXML"

2462 `SOAPAction: "ebXML"`

- 2463 • Other headers with semantics defined by MIME specifications, such as Content-Transfer-Encoding, **SHALL**
2464 **NOT** appear as HTTP headers. Specifically, the "MIME-Version: 1.0" header **MUST NOT** appear as an
2465 HTTP header. However, HTTP-specific MIME-like headers defined by HTTP 1.1 **MAY** be used with the
2466 semantic defined in the HTTP specification.

- All ebXML Service Message parts that follow the ebXML Message Envelope, including the MIME boundary string, constitute the HTTP entity body. This encompasses the SOAP **Envelope** and the constituent ebXML parts and attachments including the trailing MIME boundary strings.

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

```

2471 POST /servlet/ebXMLhandler HTTP/1.1
2472 Host: www.example2.com
2473 SOAPAction: "ebXML"
2474 Content-type: multipart/related; boundary="BoundaryY"; type="text/xml";
2475         start="<ebxmhheader111@example.com>"
2476
2477 --BoundaryY
2478 Content-ID: <ebxmhheader111@example.com>
2479 Content-Type: text/xml
2480
2481 <?xml version="1.0" encoding="UTF-8"?>
2482 <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
2483     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2484     xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
2485     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
2486     xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
2487         http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
2488         http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
2489         http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
2490 <SOAP:Header>
2491     <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
2492         <eb:From>
2493             <eb:PartyId>urn:duns:123456789</eb:PartyId>
2494         </eb:From>
2495         <eb:To>
2496             <eb:PartyId>urn:duns:912345678</eb:PartyId>
2497         </eb:To>
2498         <eb:CPAId>20001209-133003-28572</eb:CPAId>
2499         <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2500         <eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
2501         <eb:Action>NewOrder</eb:Action>
2502         <eb:MessageData>
2503             <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
2504             <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
2505         </eb:MessageData>
2506     </eb:MessageHeader>
2507 </SOAP:Header>
2508 <SOAP:Body>
2509     <eb:Manifest eb:version="2.0">
2510         <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2511             xlink:role="XLinkRole" xlink:type="simple">
2512             <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
2513         </eb:Reference>
2514     </eb:Manifest>
2515 </SOAP:Body>
2516 </SOAP:Envelope>
2517
2518 --BoundaryY--
2519 Content-ID: <ebxmlpayload111@example.com>
2520 Content-Type: text/xml
2521
2522 <?xml version="1.0" encoding="UTF-8"?>
2523 <purchase_order>
2524     <po_number>1</po_number>
2525     <part_number>123</part_number>
2526     <price currency="USD">500.00</price>
2527 </purchase_order>
2528
2529 --BoundaryY--

```

2530 B.2.3 HTTP Response Codes

2531 In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be followed, for
 2532 returning the HTTP level response codes. A 2xx code MUST be returned when the HTTP Posted

2533 message is successfully received by the receiving HTTP entity. However, see exception for SOAP error
2534 conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx range MAY be returned for conditions
2535 corresponding to them. However, error conditions encountered while processing an ebXML Service
2536 Message MUST be reported using the error mechanism defined by the ebXML Message Service
2537 Specification (see section 4.1.5).

2538 **B.2.4 SOAP Error conditions and Synchronous Exchanges**

2539 The SOAP 1.1 specification states:

2540 *"In case of a SOAP error while processing the request, the SOAP HTTP server MUST issue an HTTP*
2541 *500 "Internal Server Error" response and include a SOAP message in the response containing a SOAP*
2542 *Fault element indicating the SOAP processing error. "*

2543 However, the scope of the SOAP 1.1 specification is limited to *synchronous* mode of message exchange
2544 over HTTP, whereas the ebXML Message Service Specification specifies both *synchronous* and
2545 *asynchronous* modes of message exchange over HTTP. Hence, the SOAP 1.1 specification MUST be
2546 followed for *synchronous* mode of message exchange, where the SOAP *Message* containing a SOAP
2547 **Fault** element indicating the SOAP processing error MUST be returned in the HTTP response with a
2548 response code of "HTTP 500 Internal Server Error". When *asynchronous* mode of message exchange is
2549 being used, a HTTP response code in the range 2xx MUST be returned when the message is received
2550 successfully and any error conditions (including SOAP errors) must be returned via separate HTTP Post.

2551 **B.2.5 Synchronous vs. Asynchronous**

2552 When a synchronous transport is in use, the MSH response message(s) SHOULD be returned on the
2553 same HTTP connection as the inbound request, with an appropriate HTTP response code, as described
2554 above. When the **syncReplyMode** parameter is set to values other than **none**, the application response
2555 messages, if any, are also returned on the same HTTP connection as the inbound request, rather than
2556 using an independent HTTP Post request. If the **syncReplyMode** has a value of **none**, an HTTP
2557 response with a response code as defined in section B.2.3 above and with an empty HTTP body MUST
2558 be returned in response to the HTTP Post.

2559 **B.2.6 Access Control**

2560 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the
2561 use of an access control mechanism. The HTTP access authentication process described in "HTTP
2562 Authentication: Basic and Digest Access Authentication" [RFC2617] defines the access control
2563 mechanisms allowed to protect an ebXML Message Service Handler from unauthorized access.

2564 Implementers MAY support all of the access control schemes defined in [RFC2617] including support of
2565 the Basic Authentication mechanism, as described in [RFC2617] section 2, when Access Control is used.

2566 Implementers that use basic authentication for access control SHOULD also use communications
2567 protocol level security, as specified in the section titled "Confidentiality and Transport Protocol Level
2568 Security" in this document.

2569 **B.2.7 Confidentiality and Transport Protocol Level Security**

2570 An ebXML Message Service Handler MAY use transport layer encryption to protect the confidentiality of
2571 ebXML Messages and HTTP transport headers. The IETF Transport Layer Security specification TLS
2572 [RFC2246] provides the specific technical details and list of allowable options, which may be used by
2573 ebXML Message Service Handlers. ebXML Message Service Handlers MUST be capable of operating in
2574 backwards compatibility mode with SSL [SSL3], as defined in Appendix E of TLS [RFC2246].

2575 ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key sizes
2576 specified within TLS [RFC2246]. At a minimum ebXML Message Service Handlers MUST support the key
2577 sizes and algorithms necessary for backward compatibility with [SSL3].

2578 The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that stronger
2579 encryption keys/algorithms SHOULD be used.

2580 Both TLS [RFC2246] and SSL [SSL3] require the use of server side digital certificates. Client side
2581 certificate based authentication is also permitted. All ebXML Message Service handlers MUST support
2582 hierarchical and peer-to-peer or direct-trust trust models.

2583 **B.3 SMTP**

2584 The Simple Mail Transfer Protocol (SMTP) [RFC2821] specification is commonly referred to as Internet
2585 Electronic Mail. This specifications has been augmented over the years by other specifications, which
2586 define additional functionality "layered on top" of this baseline specifications. These include:

2587 Multipurpose Internet Mail Extensions (MIME) [RFC2045], [RFC2046], [RFC2387]

2588 SMTP Service Extension for Authentication [RFC2554]

2589 SMTP Service Extension for Secure SMTP over TLS [RFC2487]

2590 Typically, Internet Electronic Mail Implementations consist of two "agent" types:

2591 Message Transfer Agent (MTA): Programs that send and receive mail messages with other MTA's on
2592 behalf of MUA's. Microsoft Exchange Server is an example of a MTA

2593 Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail messages and
2594 communicate with an MTA to send/retrieve mail messages. Microsoft Outlook is an example of a MUA.

2595 MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

2596 MUA's are responsible for constructing electronic mail messages in accordance with the Internet
2597 Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML
2598 compliant message for transport via eMail from the perspective of a MUA. No attempt is made to define
2599 the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.

2600 **B.3.1 Minimum Level of Supported Protocols**

2601 Simple Mail Transfer Protocol [RFC2821]

2602 MIME [RFC2045] and [RFC2046]

2603 Multipart/Related MIME [RFC2387]

2604 **B.3.2 Sending ebXML Messages over SMTP**

2605 Prior to sending messages over SMTP an ebXML Message MUST be formatted according to the ebXML
2606 Message Service Specification. Additionally the messages must also conform to the syntax, format and
2607 encoding rules specified by MIME [RFC2045], [RFC2046] and [RFC2387].

2608 Many types of data that a party might desire to transport via email are represented as 8bit characters or
2609 binary data. Such data cannot be transmitted over SMTP [RFC2821], which restricts mail messages to
2610 7bit US-ASCII data with lines no longer than 1000 characters including any trailing CRLF line separator. If
2611 a sending Message Service Handler knows that a receiving MTA, or ANY intermediary MTA's, are
2612 restricted to handling 7-bit data then any document part that uses 8 bit (or binary) representation must be
2613 "transformed" according to the encoding rules specified in section 6 of MIME [RFC2045]. In cases where
2614 a Message Service Handler knows that a receiving MTA and ALL intermediary MTA's are capable of
2615 handling 8-bit data then no transformation is needed on any part of the ebXML Message.

2616 The rules for forming an ebXML Message for transport via SMTP are as follows:

- 2617 • If using SMTP [RFC2821] restricted transport paths, apply transfer encoding to all 8-bit data that will be
2618 transported in an ebXML message, according to the encoding rules defined in section 6 of MIME
2619 [RFC2045]. The Content-Transfer-Encoding MIME header MUST be included in the MIME envelope portion
2620 of any body part that has been transformed (encoded).

- 2621 • The Content-Type: Multipart/Related MIME header with the associated parameters, from the
2622 ebXML Message Envelope MUST appear as an eMail MIME header.
- 2623 • All other MIME headers that constitute the ebXML Message Envelope MUST also become part of the eMail
2624 MIME header.
- 2625 • The SOAPAction MIME header field must also be included in the eMail MIME header and MAY have the
2626 value of ebXML:
2627 SOAPAction: "ebXML"
- 2628 • The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- 2629 • The eMail header "To:" MUST contain the SMTP [RFC2821] compliant eMail address of the ebXML
2630 Message Service Handler.
- 2631 • The eMail header "From:" MUST contain the SMTP [RFC2821] compliant eMail address of the senders
2632 ebXML Message Service Handler.
- 2633 • Construct a "Date:" eMail header in accordance with SMTP [RFC2821]
- 2634 • Other headers MAY occur within the eMail message header in accordance with SMTP [RFC2821] and
2635 MIME [RFC2045], however ebXML Message Service Handlers MAY choose to ignore them.

2636 The example below shows a minimal example of an eMail message containing an ebXML Message:

```

2637 From: ebXMLhandler@example.com
2638 To: ebXMLhandler@example2.com
2639 Date: Thu, 08 Feb 2001 19:32:11 CST
2640 MIME-Version: 1.0
2641 SOAPAction: "ebXML"
2642 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2643 start="<ebxhmheader111@example.com>"
2644
2645 This is an ebXML SMTP Example
2646
2647 --Boundary
2648 Content-ID: <ebxhmheader111@example.com>
2649 Content-Type: text/xml
2650
2651 <?xml version="1.0" encoding="UTF-8"?>
2652 <SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
2653 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2654 xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
2655 xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
2656 http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
2657 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
2658 xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
2659 http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
2660 <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
2661 <eb:From>
2662 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2663 </eb:From>
2664 <eb:To>
2665 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2666 </eb:To>
2667 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2668 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2669 <eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
2670 <eb:Action>NewOrder</eb:Action>
2671 <eb:MessageData>
2672 <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
2673 <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
2674 </eb:MessageData>
2675 <eb:DuplicateElimination/>
2676 </eb:MessageHeader>
2677 </SOAP:Header>
2678 <SOAP:Body xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
2679 xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
2680 http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
2681 <eb:Manifest eb:version="2.0">
2682 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2683 xlink:role="XLinkRole"
2684 xlink:type="simple">

```

```

2685     <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
2686     </eb:Reference>
2687     </eb:Manifest>
2688 </SOAP:Body>
2689 </SOAP:Envelope>
2690
2691 --Boundary
2692 Content-ID: <ebxhmheader111@example.com>
2693 Content-Type: text/xml
2694
2695 <?xml version="1.0" encoding="UTF-8"?>
2696 <purchase_order>
2697   <po_number>1</po_number>
2698   <part_number>123</part_number>
2699   <price currency="USD">500.00</price>
2700 </purchase_order>
2701
2702 --Boundary--

```

2703 **B.3.3 Response Messages**

2704 All ebXML response messages, including errors and acknowledgments, are delivered *asynchronously*
 2705 between ebXML Message Service Handlers. Each response message MUST be constructed in
 2706 accordance with the rules specified in the section B.3.2.

2707 All ebXML Message Service Handlers MUST be capable of receiving a delivery failure notification
 2708 message sent by an MTA. A MSH that receives a delivery failure notification message SHOULD examine
 2709 the message to determine which ebXML message, sent by the MSH, resulted in a message delivery
 2710 failure. The MSH SHOULD attempt to identify the application responsible for sending the offending
 2711 message causing the failure. The MSH SHOULD attempt to notify the application that a message
 2712 delivery failure has occurred. If the MSH is unable to determine the source of the offending message the
 2713 MSH administrator should be notified.

2714 MSH's which cannot identify a received message as a valid ebXML message or a message delivery
 2715 failure SHOULD retain the unidentified message in a "dead letter" folder.

2716 A MSH SHOULD place an entry in an audit log indicating the disposition of each received message.

2717 **B.3.4 Access Control**

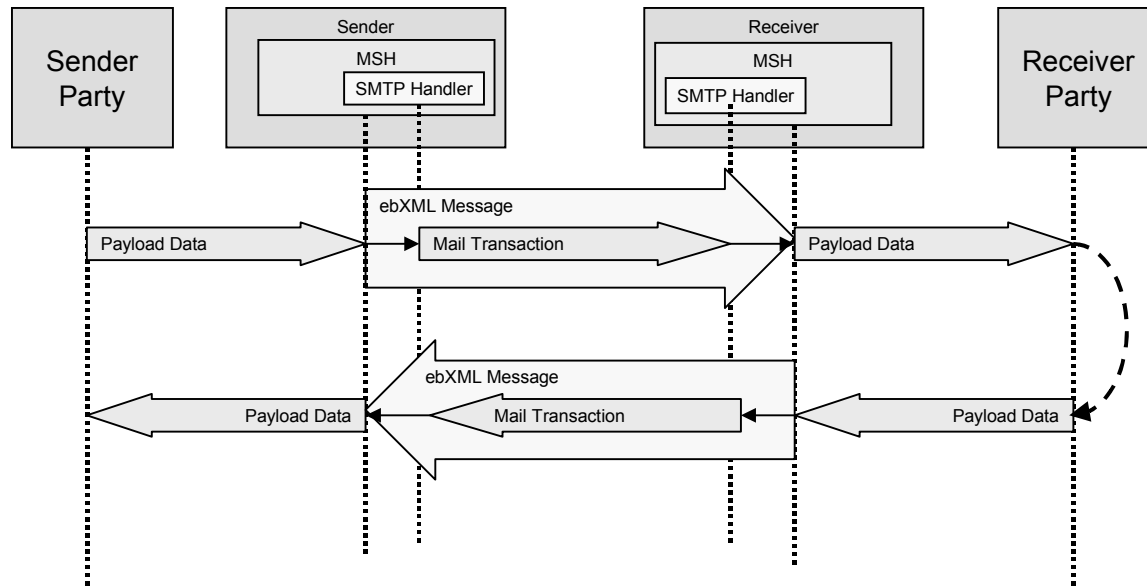
2718 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the
 2719 use of an access control mechanism. The SMTP access authentication process described in "SMTP
 2720 Service Extension for Authentication" [RFC2554] defines the ebXML recommended access control
 2721 mechanism to protect a SMTP based ebXML Message Service Handler from unauthorized access.

2722 **B.3.5 Confidentiality and Transport Protocol Level Security**

2723 An ebXML Message Service Handler MAY use transport layer encryption to protect the confidentiality of
 2724 ebXML messages. The IETF "SMTP Service Extension for Secure SMTP over TLS" specification
 2725 [RFC2487] provides the specific technical details and list of allowable options, which may be used.

2726 **B.3.6 SMTP Model**

2727 All *ebXML Message Service* messages carried as mail in an SMTP [RFC2821] Mail Transaction as
 2728 shown in Figure B1.



2729

2730 **Figure B-1 SMTP Mail Depiction**

2731 **B.4 Communication Errors during Reliable Messaging**

2732 When the Sender or the Receiver detects a communications protocol level error (such as an HTTP,
 2733 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport recovery
 2734 handler will execute a recovery sequence. Only if the error is unrecoverable, does Reliable Messaging
 2735 recovery take place (see section 6).

2736

2736

Appendix C Supported Security Services

2737 The general architecture of the ebXML Message Service Specification is intended to support all the
 2738 security services required for electronic business. The following table combines the security services of
 2739 the *Message Service Handler* into a set of security profiles. These profiles, or combinations of these
 2740 profiles, support the specific security policy of the ebXML user community. Due to the immature state of
 2741 XML security specifications, this version of the specification requires support for profiles 0 and 1 only.
 2742 This does not preclude users from employing additional security features to protect ebXML exchanges;
 2743 however, interoperability between parties using any profiles other than 0 and 1 cannot be guaranteed.

2744

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

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 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 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)

2745

2746 **References**2747 **Normative References**

- 2748 [\[RFC2119\]](#) Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task Force, March 1997
- 2749
- 2750 [\[RFC2045\]](#) Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein, Published November 1996
- 2751
- 2752 [\[RFC2046\]](#) Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N. Borenstein. November 1996.
- 2753
- 2754 [\[RFC2246\]](#) Dierks, T. and C. Allen, "The TLS Protocol", January 1999.
- 2755 [\[RFC2387\]](#) The MIME Multipart/Related Content-type. E. Levinson. August 1998.
- 2756 [\[RFC2392\]](#) Content-ID and Message-ID Uniform Resource Locators. E. Levinson, August 1998
- 2757 [\[RFC2396\]](#) Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, August 1998
- 2758 [\[RFC2402\]](#) IP Authentication Header. S. Kent, R. Atkinson. November 1998. RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November 1998.
- 2759
- 2760 [\[RFC2487\]](#) SMTP Service Extension for Secure SMTP over TLS. P. Hoffman, January 1999.
- 2761 [\[RFC2554\]](#) SMTP Service Extension for Authentication. J. Myers. March 1999.
- 2762 [\[RFC2821\]](#) Simple Mail Transfer Protocol, J. Klensin, Editor, April 2001 Obsoletes RFC 821
- 2763 [\[RFC2616\]](#) Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", June 1999.
- 2764
- 2765 [\[RFC2617\]](#) Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", June 1999.
- 2766
- 2767
- 2768 [\[RFC2817\]](#) Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", May 2000.
- 2769 [\[RFC2818\]](#) Rescorla, E., "HTTP Over TLS", May 2000 [SOAP] Simple Object Access Protocol
- 2770 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08 May 2000,
- 2771 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 2772
- 2773
- 2774
- 2775 [SOAPAttach] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000
- 2776 <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211>
- 2777
- 2778 [SSL3] A. Frier, P. Karlton and P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996.
- 2779
- 2780 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage conventions.
- 2781 [XLINK] W3C XML Linking Recommendation, <http://www.w3.org/TR/2001/REC-xlink-20010627/>
- 2782 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition), October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 2783
- 2784 [XMLC14N] W3C Recommendation Canonical XML 1.0,
- 2785 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

- 2786 [XMLNS] W3C Recommendation for Namespaces in XML, World Wide Web Consortium, 14
2787 January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- 2788 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,
2789 <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>.
- 2790 [XMLMedia] [RFC 3023](#), XML Media Types. M. Murata, S. St. Laurent, January 2001
- 2791 [XPointer] XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11
2792 September 2001, <http://www.w3.org/TR/2001/CR-xptr-20010911/>
- 2793
- 2794 **Non-Normative References**
- 2795 [ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 1.0,
2796 published 10 May, 2001, <http://www.ebxml.org/specs/ebCCP.doc>
- 2797 [ebBPSS] ebXML Business Process Specification Schema, version 1.0, published 27 April 2001,
2798 <http://www.ebxml.org/specs/ebBPSS.pdf>.
- 2799 [ebTA] ebXML Technical Architecture, version 1.04 published 16 February, 2001,
2800 <http://www.ebxml.org/specs/ebTA.doc>
- 2801 [ebRS] ebXML Registry Services Specification, version 2.0, published 6 December 2001
2802 <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf>,
2803 published, 5 December 2001.
2804 <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRim.pdf>
- 2805 [ebREQ] ebXML Requirements Specification, <http://www.ebxml.org/specs/ebREQ.pdf>,
2806 published 8 May 2001.
- 2807 [ebGLOSS] ebXML Glossary, <http://www.ebxml.org/specs/ebGLOSS.doc>, published 11 May, 2001.
- 2808 [PGP/MIME] [RFC2015](#), "MIME Security with Pretty Good Privacy (PGP)", M. Elkins. October 1996.
- 2809 [SAML] Security Assertion Markup Language,
2810 <http://www.oasis-open.org/committees/security/docs/draft-sstc-use-strawman-03.html>
- 2811 [S/MIME] [RFC 2311](#), "S/MIME Version 2 Message Specification", S. Dusse, P. Hoffman, B.
2812 Ramsdell, L. Lundblade, L. Repka. March 1998.
- 2813 [S/MIMECH] [RFC 2312](#), "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman, B. Ramsdell,
2814 J. Weinstein. March 1998.
- 2815 [S/MIMEV3] [RFC 2633](#) S/MIME Version 3 Message Specification. B. Ramsdell, Ed June 1999.
- 2816 [secRISK] ebXML Technical Architecture Risk Assessment Technical Report, version 0.36
2817 published 20 April 2001
- 2818 [XMLSchema] W3C XML Schema Recommendation,
2819 <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>
2820 <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
2821 <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>
- 2822 [XMTP] XMTP - Extensible Mail Transport Protocol
2823 <http://www.openhealth.org/documents/xmtp.htm>

2824 **Contact Information**2825 **Team Leader**

Name Ian Jones
Company British Telecommunications
Address Enterprise House, 84-85 Adam Street
Cardiff, CF24 2XF United Kingdom
Phone: +44 29 2072 4063
EMail: ian.c.jones@bt.com

2826 **Vice Team Leader**

Name Brian Gibb
Company Sterling Commerce
Address 750 W. John Carpenter Freeway
Irving, Texas 75039 USA
Phone: +1 (469) 524.2628
EMail: brian_gibb@stercomm.com

2827 **Team Editor**

Name David Fischer
Company Drummond Group, Inc
Address P.O. Box 101567
Fort Worth, Texas 76105 USA
Phone +1 (817) 294-7339
EMail david@drummondgroup.com

2828 Acknowledgments

2829 The OASIS ebXML-MS Technical Committee would like to thank the members of the original joint
2830 UN/CEFACT-OASIS ebXML Messaging Team for their work to produce v1.0 of this specification.

2831

Ralph Berwanger	bTrade.com	Ravi Kacker	Kraft Foods
Jonathan Borden	Author of XMTP	Henry Lowe	OMG
Jon Bosak	Sun Microsystems	Jim McCarthy	webXI
Marc Breissinger	webMethods	Bob Miller	GXS
Dick Brooks	Group 8760	Dale Moberg	Sterling Commerce
Doug Bunting	Ariba	Joel Munter	Intel
David Burdett	Commerce One	Shumpei Nakagaki	NEC Corporation
David Craft	VerticalNet	Farrukh Najmi	Sun Microsystems
Philippe De Smedt	Viquity	Akira Ochi	Fujitsu
Lawrence Ding	WorldSpan	Martin Sachs	IBM
Rik Drummond	Drummond Group	Saikat Saha	Commerce One
Andrew Eisenberg	Progress Software	Masayoshi Shimamura	Fujitsu
Colleen Evans	Sonic Software	Prakash Sinha	Netfish Technologies
David Fischer	Drummond Group	Rich Salz	Zolera Systems
Christopher Ferris	Sun Microsystems	Tae Joon Song	eSum Technologies, Inc.
Robert Fox	Softshare	Kathy Spector	Extricity
Brian Gibb	Sterling Commerce	Nikola Stojanovic	Encoda Systems, Inc.
Maryann Hondo	IBM	David Turner	Microsoft
Jim Hughes	Fujitsu	Gordon Van Huizen	Progress Software
John Ibbotson	IBM	Martha Warfelt	DaimlerChrysler Corporation
Ian Jones	British Telecommunications	Prasad Yendluri	Web Methods

2832 Disclaimer

2832

2833 The views and specification expressed in this document are those of the authors and are not necessarily
2834 those of their employers. The authors and their employers specifically disclaim responsibility for any
2835 problems arising from correct or incorrect implementation or use of this design.

2836 Copyright Statement

2837 *Copyright (C) The Organization for the Advancement of Structured Information Standards [OASIS]*
2838 *January 2002. All Rights Reserved.*

2839 *This document and translations of it may be copied and furnished to others, and derivative works that*
2840 *comment on or otherwise explain it or assist in its implementation may be prepared, copied, published*
2841 *and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice*
2842 *and this paragraph are included on all such copies and derivative works. However, this document itself*
2843 *may not be modified in any way, such as by removing the copyright notice or references to OASIS,*
2844 *except as needed for the purpose of developing OASIS specifications, in which case the procedures for*
2845 *copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to*
2846 *translate it into languages other than English.*

2847 *The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors*
2848 *or assigns.*

2849 *This document and the information contained herein is provided on an "AS IS" basis and OASIS*
2850 *DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY*
2851 *WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR*
2852 *ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE."*