



Creating A Single Global Electronic Market

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

Message Service Specification

ebXML Transport, Routing & Packaging

Version 0.8

10 November 2000

25 **1 Status of this Document**

26

27 This document specifies an ebXML DRAFT for the eBusiness community.

28

29 Distribution of this document is unlimited.

30

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

33

34 ***This version:***35 http://www.ebxml.org/working/project_teams/...

36

37 ***Latest version:***38 <http://www.ebxml.org/...>

39

40 ***Previous version:***41 <http://www.ebxml.org/.....>

42

43

44 **2 ebXML participants**

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

48
49 Ralph Berwanger – bTrade.com
50 Jonathan Borden - Author of XMTP
51 Jon Bosak – Sun Microsystems
52 Marc Breissinger - webMethods
53 Dick Brooks – Group 8760
54 Doug Bunting - Ariba
55 David Burdett - Commerce One
56 Len Callaway – Drummond Group, Inc.
57 David Craft – VerticalNet
58 Philippe De Smedt - Viquity
59 Lawrence Ding - WorldSpan
60 Rik Drummond - Drummond Group, Inc. (Representing XML Solutions)
61 Christopher Ferris – Sun Microsystems
62 Maryann Hondo - IBM
63 Jim Hughes - Fujitsu
64 John Ibbotson - IBM
65 Ian Jones – British Telecommunications
66 Ravi Kacker – Kraft Foods
67 Nick Kassem – Sun Microsystems
68 Henry Lowe - OMG
69 Jim McCarthy - webXI
70 Bob Miller - GSX
71 Dale Moberg - Sterling Commerce
72 Joel Munter – Intel
73 Farrukh Najmi – Sun Microsystems
74 Akira Ochi – Fujitsu
75 Masayoshi Shimamura – Fujitsu
76 Kathy Spector – Extricity
77 Nikola Stojanovic - Columbine JDS Systems
78 Gordon Van Huizen – Process Software
79 Martha Warfelt - DaimlerChrysler
80 Prasad Yendluri – Vitria

81
82
83
84
85
86
87

88 **3 Table of Contents**

89 1 Status of this Document 2

90 2 ebXML participants 3

91 3 Table of Contents 4

92 4 Introduction **68**

93 4.1 Summary of Contents of Document..... **68**

94 4.2 Document Conventions..... **79**

95 4.3 Audience..... **79**

96 4.4 Related Documents..... **79**

97 5 Design Objectives **840**

98 5.1 Goals/Objectives/Requirements/Problem Description **840**

99 5.2 Caveats and Assumptions..... **840**

100 6 System Overview **840**

101 6.1 What ebXML Message Services does **840**

102 6.2 Where ebXML Message Services May Be Implemented..... **840**

103 7 Definition and Scope **840**

104 7.1 Packaging Specification **840**

105 7.1.1 ebXML Message Structure..... **840**

106 7.1.2 ebXML Header Envelope and Payload Envelope **944**

107 7.1.3 MIME usage Conventions **944**

108 7.2 ebXML Message Envelope..... **1042**

109 7.2.1 Content-Type..... **1042**

110 7.2.2 Content-Length..... **1042**

111 7.2.3 ebXML Message Envelope Example..... **1143**

112 7.3 ebXML Header Container..... **1143**

113 7.3.1 Content-ID..... **1143**

114 7.3.2 Content-Length..... **1143**

115 7.3.3 Content-Type..... **1143**

116 7.3.4 ebXML Header Envelope Example **1244**

117 7.4 ebXML Payload Container **1244**

118 7.4.1 Content-ID..... **1345**

119 7.4.2 Content-Length..... **1345**

120 7.4.3 Content-Type..... **1345**

121 7.4.4 Example of an ebXML MIME Payload Container **1345**

122 7.5 ebXML Header Document..... **1345**

123 7.6 XML Prolog..... **1446**

124 7.7 ebXMLHeader Element..... **1446**

125 7.7.1 ebXMLHeader attributes **1446**

126 7.7.2 ebXMLHeader elements **1547**

127 7.7.3 ebXMLHeader sample **1547**

128 7.8 XML Manifest..... **1547**

129 7.8.1 XML DocumentReference..... **1547**

130 7.8.2 Manifest sample **1648**

131 7.9 XML Header..... **1648**

132 7.9.1 From and To..... **1648**

133 7.9.2 TPAInfo **1749**

134 7.9.3 MessageData **1749**

135 7.9.4 ReliableMessagingInfo..... **1820**

136 </ReliableMessagingInfo>..... **1820**

137 7.9.5 XML Header sample **1820**

138 7.10 XML Routing Header **1924**

139 7.11 Reliable Messaging Flow **1924**

140 7.12 Reliable Messaging Recovery Procedures **2123**

141	7.12.1 Messaging Service Parameters	2123
142	7.12.2 Recovery Sequence for Lost Messages	2224
143	7.12.3 Maximum Number of Retries and Retry Interval	2224
144	7.13 ebXML Error Reporting	2426
145	7.13.1 Definitions.....	2426
146	7.13.2 Types of Errors.....	2426
147	7.13.3 When to generate Error Messages.....	2426
148	7.13.4 Identifying the Error Reporting Location	2426
149	7.13.5 ebXML Error Message	2527
150	7.14 Security	2934
151	8 References	2934
152	8.1 Normative References.....	2934
153	8.2 Non-Normative References	2934
154	9 Disclaimer	2934
155	10 Contact Information.....	3032
156	Appendix A DTD Definitions	3234
157	A.1 ebXML Header DTD.....	3234
158	A.2 ebXML Error DTD	3335
159	Appendix B Examples	3436
160	B.1 Complete Example of an ebXML Message Envelope using multipart/related Content- Type sent via HTTP POST	3436
161	B.2 Complete Example of an ebXML Message Envelope using multipart/related Content- Type sent via SMTP	3638
162	Appendix C Candidate Packaging Technologies and Selection Process.....	3840
163	C.1 Selection Process	3840
164	C.2 MIME.....	3944
165	C.3 XML.....	3944
166	C.4 Conclusion.....	3944
167	Appendix D MIME Type discussion.....	4042
168	Appendix E Communication Protocol Interfaces.....	4143
169	E.1 HTTP [RFC 2068]	4143
170	E.2 SMTP [RFC 821].....	4244
171	E.3 FTP [RFC 959].....	4345
172	E.4 Communication Protocol Errors during Reliable Messaging.....	4345
173	Appendix F Detailed list of the Message Services Requirement Phases	4547
174	Copyright Statement.....	4648
175		
176		
177		

178 4 Introduction

179 This is a draft standard for trial implementation. The specification is the first in a series of phased
180 deliverables. This version of the specification does not address complete message security,
181 extensibility, service interface, reliability, and versioning. These are being developed as separate
182 documents and will be included in later versions of this document or as additional service
183 specifications to the ebXML Message Services Specification.
184

185 4.1 Summary of Contents of Document

186 This specification defines the ebXML Message Service protocol which enables the secure and
187 reliable exchange of messages between two parties. It includes descriptions of:

- 188
- 189 • the *ebXML Message* structure used to package ebXML Messages for transport between
190 parties, and
- 191
- 192 • the behavior of the Message service that sends or receives those messages.
193

194 No assumption or dependency is made relative to communication protocol or type of payload.
195 The specifications contained here are both payload and communication protocol neutral.
196

197 This specification is organized around the following topics:

- 198 • **Packaging Specification** - A description of how to package an *ebXML Message* and
199 associated parts. This section includes specifications for the various structures and
200 containers. The Packaging Specification is a standard MIME multipart/related structure
201 with two parts: XML Message Headers and Payload. The payload may be any type of
202 data that MIME RFC 2045 and related IETF MIME extensions may support. The XML
203 based Message Header elements and their structure were chosen after reviewing several
204 current transports, both proprietary and non-proprietary, to ensure that the appropriate
205 header elements were included in the specification
- 206 • **Message Headers** - A specification of the structure and composition of the information
207 necessary for an ebXML Message Service to successfully generate or process an ebXML
208 compliant message.
- 209 • **Reliable Messaging** - The Reliable Messaging function defines an interoperable protocol
210 such that any two Message Service implementations can “reliably” exchange messages
211 that are sent using “reliable messaging” semantics. Please see Section 7.11.
- 212 • **Error Handling** - This section describes how one ebXML Message Service reports errors
213 it detects to another ebXML Message Service.
- 214 • **Security** - This version of the specification supports limited security services that is those
215 security services that can be supported within the payload. The *multipart/related*
216 payload may be encrypted using cryptographic techniques suitable for the payload type.
217

218 Appendices to this specification cover the following:
219

- 220 • Appendix A Schemas and DTD Definitions
- 221 • Appendix B Examples
- 222 • Appendix C Candidate Packaging Technologies and Selection Process
- 223 • Appendix D MIME Type discussion

- 224 • Appendix E Communication Protocol Envelope Mappings
- 225 • Appendix F Detailed list of the Message Services Requirement Phases
- 226

227 4.2 Document Conventions

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

231
232 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
233 RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be
234 interpreted as described in RFC 2119 [Bra97].

235
236 Note that the force of these words is modified by the requirement level of the document in which
237 they are used.

238
239 MUST: This word, or the terms “REQUIRED” or “SHALL”, means that the definition is an
240 absolute requirement of the specification.

241
242 MUST NOT: This phrase, or the phrase “SHALL NOT”, means that the definition is an
243 absolute prohibition of the specification.

244
245 SHOULD: This word, or the adjective “RECOMMENDED”, means that there may exist
246 valid reasons in particular circumstances to ignore a particular item, but the full
247 implications must be understood and carefully weighed before choosing a different
248 course.

249
250 SHOULD NOT: This phrase, or the phrase “NOT RECOMMENDED”, means that there
251 may exist valid reasons in particular circumstances when the particular behavior is
252 acceptable or even useful, but the full implications should be understood and the case
253 carefully weighed before implementing any behavior described with this label.

254

255 4.3 Audience

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

258

259 It is assumed that the reader has an understanding of transports, MIME and XML.

260

261 4.4 Related Documents

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

- 263 • **ebXML Trading Partner Specification** (under development) - defines how one party can
264 discover and/or agree upon the information that party needs to know about another party
265 prior to sending them a message that complies with this specification
- 266 • **ebXML Message Service Interface Specification** (to be developed) - defines an
267 interface that may be used by software to interact with an ebXML Message Service
- 268 • **ebXML Message Services Security Specification** (under development) – defines the
269 security mechanisms necessary to negate anticipated, selected threats
- 270 • **ebXML Message Services Requirements Specification** – defines the requirements of
271 the Message Services

272 5 Design Objectives

273 5.1 Goals/Objectives/Requirements/Problem Description

274 The design objectives and goals are to define a Message Service (MS) to support XML based
275 electronic business between small, medium and large enterprises. This specification is intended
276 to enable a low cost solution, while preserving a vendor's ability to add unique value through
277 added robustness and superior performance. It is the intention of the Transport, Routing and
278 Packaging Project Team to keep this specification as simple and succinct as possible. Every item
279 in this specification is being prototyped by the ebXML Proof of Concept Team in order to ensure
280 the clarity and succinctness of this specification.

281 5.2 Caveats and Assumptions

282

283 6 System Overview

284 This document defines the enveloping and *ebXML Message* header structure used to transfer
285 *ebXML Messages* over a data communication mechanism. This document provides sufficient
286 detail to develop software for the packaging, exchange and processing of *ebXML Messages*.

287 6.1 What ebXML Message Services does

288 ebXML Message Services (MS) defines, robust yet basic functionality necessary to transfer
289 messages between two ebXML Message Services using various existing communication
290 protocols. The ebXML Message Service will perform in a manner which will allow for reliability,
291 persistence of messages, security, and extensibility.

292 6.2 Where ebXML Message Services May Be Implemented

293 The ebXML Message Services is expected to be implemented in environments requiring a robust,
294 low cost solution to enable electronic business.

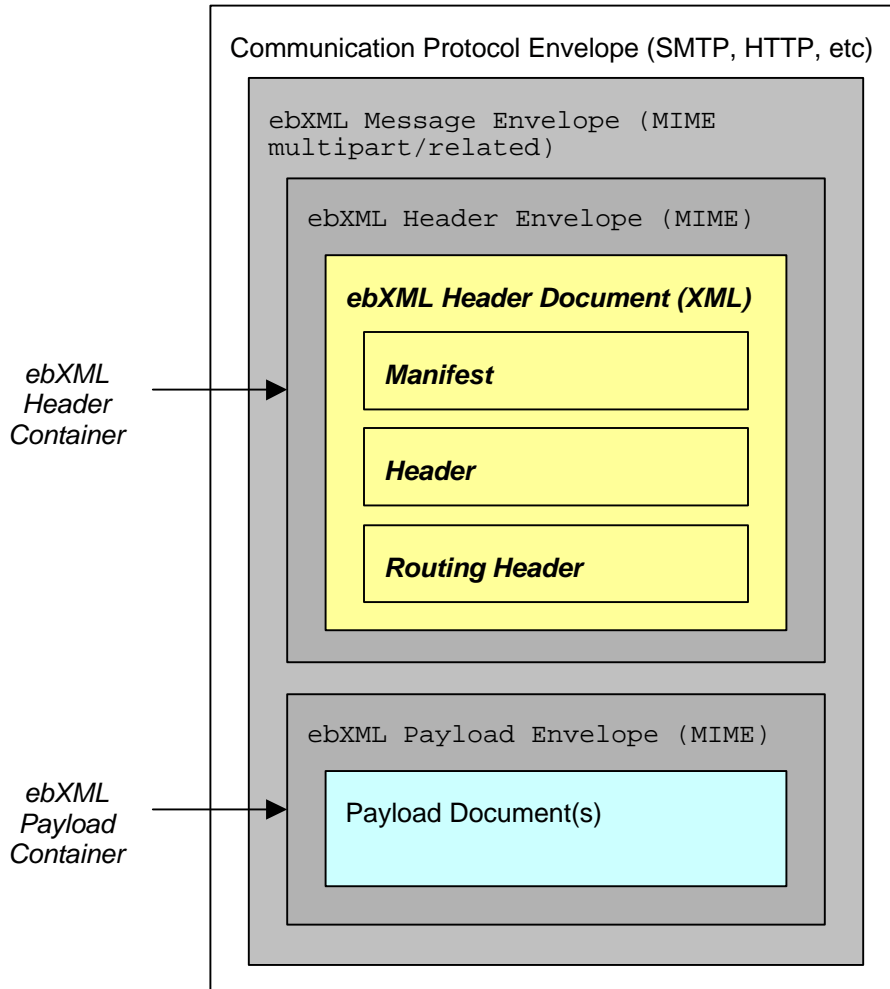
295 7 Definition and Scope

296 7.1 Packaging Specification

297 7.1.1 ebXML Message Structure

298 An *ebXML Message* consists of:

- 299 • an outer Communication Protocol Envelope, such as HTTP or SMTP,
- 300 • an inner communication "protocol independent" *ebXML Message Envelope*, specified
301 using MIME multipart/related, that contains the two main parts of the Message:
 - 302 - an ebXML Header Container that is used to envelope one ebXML Header Document, and
 - 303 - an optional, single *ebXML Payload Container* that MUST be used to envelope the actual
304 payload (transferred data) of the Message



Note: The Courier font is used to represent MIME components. Items shown in **bold italics** represent XML items.

Figure 7-1 ebXML Message Structure

305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325

7.1.2 ebXML Header Envelope and Payload Envelope

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

An *ebXML Header (or Payload) Document* is the content of the standard MIME part and is:

- an XML document in an **ebXML Header**, or
- an XML or some other document for the ebXML Payload

Any special considerations for the usage of the *ebXML Message Envelope* in TCP/IP, HTTP and SMTP transports are described in Appendix E.

7.1.3 MIME usage Conventions

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

326 7.2 ebXML Message Envelope

327 The MIME structured *ebXML Message Envelope* is used to identify the message as an ebXML
328 compliant structure and encapsulates the header and payload in MIME body parts. It MUST
329 conform to [RFC2045] and MUST contain two MIME headers:

- 330 • Content-Type
- 331 • Content-Length

332

333 7.2.1 Content-Type

334 The MIME Content-Type MUST be set to `multipart/related` for all *ebXML Message*
335 *Envelopes*. See Appendix C for selection rationale. For example:

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

337

338 The MIME Content-Type header contains three attributes:

- 339 • type
- 340 • boundary
- 341 • version

342 7.2.1.1 type Attribute

343 The MIME `type` attribute is used to identify the *ebXML Message Envelope* as an ebXML
344 compliant structure. It conforms to a MIME XML Media Type [XMLMedia] and MUST be set to
345 "`application/vnd.eb+xml`". For example:

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

347 7.2.1.2 boundary Attribute

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

```
352 boundary:="-----8760"
```

353 7.2.1.3 version Attribute

354 The MIME `version` attribute is used to identify the particular version of *ebXML Message*
355 *Envelope* being used. All message headers SHOULD USE "0.8". For example:

```
356 version="0.8"
```

357 7.2.2 Content-Length

358 The MIME Content-Length header is a decimal value used to identify the total number of
359 OCTETS contained in all constituent message body parts, including body part boundaries.

360

361 The value of the Content-Length MIME header is computed by counting the total number of
362 OCTETS starting with the first OCTET after the CRLF following the first MIME header and ending
363 with the OCTET immediately before the MIME object's last boundary string.

364

365 Example:

366 `Content-Length: 9841`

367 **7.2.3 ebXML Message Envelope Example**

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

369 `Content-Type: multipart/related; type="application/vnd.eb+xml"; boundary:="-----8760";`
 370 `Content-Length: 9841`

371 **7.3 ebXML Header Container**

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

- 373 • one XML based ebXML Header Envelope, and
- 374 • one XML **ebXML Header Document** (described in section 8 of this document)

375

376 The following rules apply:

- 377 • the *ebXML Header Container* MUST be the first MIME body part in the *ebXML Message*.
- 378 • there MUST be one and only one *XML ebXML Header Document* in each *ebXML*
- 379 *Message*. However, an *ebXML Payload Container* may be a completely encapsulated
- 380 *ebXML Message*.

381

382 The MIME based *ebXML Header Envelope* conforms to [RFC 2045] and MUST consist of three

383 MIME headers:

- 384 • `Content-ID`
- 385 • `Content-Length`
- 386 • `Content-Type`

387

388 The *ebXML Header Document* within the content portion of the MIME container MAY be

389 enhanced during transport, provided it has not been digitally signed. Any change in the size of the

390 *ebXML Header Document* MUST be reflected in `Content-Length` attribute of the *ebXML*

391 *Message Envelope* and *ebXML Header Envelope*.

392 **7.3.1 Content-ID**

393 The `Content-ID` MIME header identifies this instance of an ebXML Message header body part.

394 The value for `Content-ID` SHOULD be a unique identifier, in accordance with RFC 2045. For

395 example:

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

397 **7.3.2 Content-Length**

398 The MIME `Content-Length` header is a decimal value used to identify the total number of

399 OCTETS contained in the *ebXML Header Container* MIME body part. For example:

400 `Content-Length: 4208`

401 **7.3.3 Content-Type**

402 The MIME `Content-Type` for an ebXML header is identified with the value

403 "application/vnd.eb+xml". `Content-Type` MUST contain two attributes:

- 404 • `version`, and
- 405 • `charset`

406 **7.3.3.1 version Attribute**

- 407 • The MIME `version` attribute indicates the version of the ebXML Message Service
408 Specification to which the *ebXML Header Document* conforms. For example:

```
409 version="0.8";
```

410 **7.3.3.2 charset Attribute**

411 The MIME `charset` attribute identifies the character set used to create the ebXML Header
412 Document. The list of valid values can be found at <http://www.iana.org/>.

413
414 The MIME `charset` attribute SHALL be equivalent to the encoding attribute of the *ebXML
415 Header Document* (see section 7.6). For maximum interoperability it is RECOMMENDED that
416 [UTF-8] be used. Note: this is not the default for MIME. For example:

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

419 **7.3.4 ebXML Header Envelope Example**

420 The following represents an example of an *ebXML Header Envelope* and *ebXML Header
421 Document*:

422	Content-ID: ebxmlheader-123@ebxmlhost.realm --		
423	Content-Length: 2048	MIME ebXML	
424	Content-Type: application/vnd.eb+xml;	Header Envelope	
425	version="0.8"; charset="UTF-8" --		ebXML
426			Header
427	<ebXMLHeader> -----		Container
428	<Manifest>.....	XML ebXML Header	
429	</Manifest>	Document	
430	<Header>.....		
431	</Header>		
432	<Routing Header>.....		
433	</Routing Header>		
434	</ebXMLHeader> -----		

435 A complete example of an *ebXML Header Container* is presented in Appendix B.

436 **7.4 ebXML Payload Container**

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

439
440 If there is no payload within the *ebXML Message* then the *ebXML Payload Container* MUST not
441 be present.

442
443 The contents of the *ebXML Payload Container* MUST be identified by the *Message Manifest*
444 element within the *ebXML Header Document* (see section 7.8).

445
446 If the *Message Manifest* is an empty XML element, the *ebXML Payload Container* MUST NOT be
447 present in the *ebXML Message*.

448
449 If an *ebXML Payload Container* is present, it MUST conform to MIME [RFC2045] and MUST
450 consist of:

- 451 • a MIME header portion - the *ebXML Payload Envelope*, and
- 452 • a content portion - the payload itself which may be of any valid MIME type.

453 The *ebXML MIME Payload Envelope*, MUST consist of three MIME headers:

- 455 • Content-ID
- 456 • Content-Length
- 457 • Content-Type

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

462 **7.4.1 Content-ID**

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

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

467 **7.4.2 Content-Length**

468 The MIME Content-Length header is a decimal value used to identify the total number of
 469 OCTETS contained in the content portion of the *ebXML Payload Container*. For example:

```
470 Content-Length: 5012
```

471 **7.4.3 Content-Type**

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

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

476 **7.4.4 Example of an ebXML MIME Payload Container**

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

```
478 Content-ID: ebxmlpayload-123@ebxmlhost.realm --| |
479 Content-Length: 4096 | ebXML MIME |
480 Content-Type: application/xml -----| Payload Envelope | ebXML
481 | | | Payload
482 <Invoice> -----| | Container
483 <Invoicedata>..... | Payload |
484 </Invoicedata> | |
485 </Invoice> -----| |
```

486 A complete example of the ebXML Payload Container is presented in Appendix B.

489 **7.5 ebXML Header Document**

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

- 492 • different software is likely to be used to generate that header-element,

- 493 • the structure of the header element might vary independently of the other header-
494 elements, or
- 495 • the data contained in the header-element MAY need to be digitally signed separately
496 from the other header-elements.
497

498 7.6 XML Prolog

499
500 The XML prolog for the *ebXML Header Document* SHALL contain the encoding attribute which
501 SHALL be equivalent to the `charset` attribute of the MIME `Content-Type` of the ebXML
502 Message Header Container (see section 7.3.3.2). It is RECOMMENDED that UTF-8 be used
503 explicitly although this is one of the default values assumed if none is specified.
504

505 NOTE: The encoding attribute is OPTIONAL in the XML version 1.0 specification [XML], however,
506 it is mandatory for the ebXML message header to ensure no conflicts occur with the `charset`
507 attribute of the MIME `Content-Type` of the container and to ensure maximum interoperability. For
508 example:

509
510

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

511 7.7 ebXMLHeader Element

512 The root element of the *XML ebXML Header Document* is named **ebXMLHeader**. It is comprised
513 of three XML attributes and two subordinate elements.

514 7.7.1 ebXMLHeader attributes

515

516 There are three attributes associated with the **ebXMLHeader**, which are:

- 517 • Namespace (`xmlns`)
518 • Version
519 • `MessageType`
520

521 7.7.1.1 Namespace

522 The namespace declaration (**xmlns**) (see [XML Namespace]) has a REQUIRED value of
523 "`http://www.ebxml.org/namespaces/messageHeader`".
524

525 7.7.1.2 Version

526 The **Version** attribute is required. Its purpose is to provide for future versioning capabilities. It has
527 a default value of '0.8'.
528

529 7.7.1.3 MessageType

530 The purpose of the **MessageType** attribute is to enable ebXML-aware software to distinguish
531 between normal and communication protocol-specific messages, such as acknowledgment and
532 error messages. The **MessageType** is an enumeration consisting of three possible values:

- 533 • **Normal** – the ebXML Payload Container contains data that has been provided to the
534 ebXML Message Service by the software that called it
- 535 • **Acknowledgment** – a ebXML Message Service-specific acknowledgment message.
- 536 • **Error** – an ebXML Message Service-specific error message.

537

538 **7.7.2 ebXMLHeader elements**539 The **ebXMLHeader** element MUST contain the following two elements:

- 540 • Manifest
- 541 • Header

542

543 **7.7.2.1 Manifest**

544 The **Manifest** is a REQUIRED element that contains a list of references to the other parts of the
 545 Message. This includes references to the documents, which comprise the *Payload* of the
 546 Message.

547 **7.7.2.2 Header**

548 The **Header** is a REQUIRED element that contains the information REQUIRED by the recipient to
 549 process the message. The message originator creates this information to which additional
 550 information MAY be added.

551

552 **7.7.3 ebXMLHeader sample**553 The following is a sample **ebXMLHeader** document fragment demonstrating the overall structure:

```
554 <?xml version="1.0" encoding="UTF-8"?>
555 <ebXMLHeader xmlns="http://www.ebxml.org/namespaces/messageHeader"
556     Version="0.8" MessageType="Normal">
557     <Manifest>...</Manifest>
558     <Header>...</Header>
559 </ebXMLHeader>
```

560 **7.8 XML Manifest**

561 The required **Manifest** element is a composite element consisting of zero or more
 562 **DocumentReference** elements. Each **DocumentReference** element identifies data associated
 563 with the message, whether included as part of the message, or remote resources accessible via a
 564 URL. The **Manifest** SHALL be the first subordinate element in the **ebXMLHeader**. It identifies
 565 the payload document(s) contained in the *ebXML Message Container*. The purpose of the
 566 **Manifest** is to make it easier to directly extract a particular document associated with the
 567 Message.

568 **7.8.1 XML DocumentReference**569 The **DocumentReference** element is a composite element consisting of three subordinate
570 elements as follows:

- 571 • **DocumentDescription**
- 572 • **DocumentLabel**
- 573 • **DocumentId**

574

575 **7.8.1.1 DocumentDescription**

576 The **DocumentDescription** is an OPTIONAL textual description of the document/resource.
 577 The language of the description is defined by a required **xml:lang** attribute. The **xml:lang**
 578 attribute must comply with the rules for identifying languages specified in [XML].

579

580

581 **7.8.1.2 DocumentLabel**

582 The **DocumentLabel** is a code that enables the purpose of the referenced document to be
 583 determined without retrieving the referenced document.
 584

585 **7.8.1.3 DocumentId**

586 The **DocumentId** is the URL of the `Content-ID` of a MIME body part, as defined in [RFC2392],
 587 representing payload data, or a remote URL to some external resource.
 588

589 **7.8.2 Manifest sample**

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

```
592 <Manifest>
593   <DocumentReference>
594     <DocumentLabel>PurchaseOrder</DocumentLabel>
595     <DocumentId>cid:0987654321</DocumentId>
596   </DocumentReference>
597 </Manifest>
```

598 **7.9 XML Header**

599 The **Header** element immediately follows the **Manifest** element. It is required in all
 600 **ebXMLHeader** documents. The **Header** element is a composite element comprised of the
 601 following required subordinate elements:

- 602 • **From**
- 603 • **To**
- 604 • **TPAInfo**
- 605 • **MessageData**
- 606 • **ReliableMessagingInfo**

607

608 **7.9.1 From and To**

609 The **From** element identifies the **Party** which originated the message. It is a logical identifier,
 610 which MAY take the form of a URN. An example of this would be a DUNS number. The **From**
 611 element consists of a **PartyId** element.

612

613 The **To** element identifies the intended recipient of the message. As with **From**, it is a logical
 614 identifier which is comprised of a **PartyId** element.

615

616 The **PartyId** element has a single attribute; **context** and a text value. The purpose of the context
 617 attribute is to provide a context for the text value of the **PartyId** element. The following fragment
 618 demonstrates usage of the **From** and **To** elements of the **ebXMLHeader**.

```
619 <From>
620   <PartyId context="DUNS">1234567890123</PartyId>
621 </From>
622 <To>
623   <PartyId context="DUNS">3210987654321</PartyId>
```


624 `</To>`

625 7.9.2 TPAInfo

626 The **TPAInfo** element follows the **From** and **To** elements in the **Header** element structure. The
 627 **TPAInfo** element is a composite set of information that relates to the *Trading Partner Agreement*
 628 under which the message is governed. The **TPAInfo** element has four subordinate elements as
 629 follows:

- 630 • **TPAId**
- 631 • **ConversationId**
- 632 • **ServiceInterface**
- 633 • **Action**

634 7.9.2.1 TPAId

635 The **TPAId** is a URI which identifies the *Trading Partner Agreement* which governs the
 636 processing of the message.

637 7.9.2.2 ConversationId

638 The **ConversationId** is a URI which identifies the set of related messages that make up a
 639 conversation between two **Parties**.

640 7.9.2.3 ServiceInterface

641 The **ServiceInterface** identifies the Service Interface that SHOULD act on the payload in the
 642 message. It is unique within the domain of the **Party** to which the message is being sent. URN's
 643 MAY be considered suitable for the element content.

644 7.9.2.4 Action

645 The **Action** identifies a process within a Service Interface, which processes the Message.
 646 **Action** SHALL be unique within the Service Interface in which it is defined.

647 7.9.2.5 TPAInfo sample

648 The following example fragment demonstrates the usage of the **TPAInfo** element.

```
649 <TPAInfo>
650     <TPAId context = "tpadb">12345678</TPAId>
651     <ConversationId>987654321</ConversationId>
652     <ServiceInterface>QuoteToCollect</ServiceInterface>
653     <Action>NewPurchaseOrder</Action>
654 </TPAInfo>
```

655 7.9.3 MessageData

656 The required **MessageData** element follows the **TPAInfo** element. The purpose of the
 657 **MessageData** element is to provide a means of identifying an *ebXML Message*. It is a composite
 658 element that contains the following three elements:

- 659 • **MessageID**
- 660 • **TimeStamp**
- 661 • **RefToMessageID**

662 7.9.3.1 MessageId

663 The **MessageId** is a unique identifier for the message conforming to [RFC2392]. The "local part"
 664 of the identifier is implementation dependent.

665 7.9.3.2 TimeStamp

666 The **TimeStamp** is a value representing the time that the message header was created
 667 conforming to [ISO-8601]. The format of CCYYMMDDTHHMMSS.SSSZ is used. This time
 668 format is Coordinated Universal Time (UTC).

669 7.9.3.3 RefToMessageId

670 For **Normal** Messages, the **RefToMessageId** is an optional reference containing the **MessageID**
 671 of an earlier ebXML Message. If there is no earlier message, the element **MUST** be empty. For
 672 **Error** messages, the **RefToMessageId** reference is mandatory and its value **MUST** be the
 673 **MessageId** of the message in error (as defined in section 7.13.1).
 674 For **Acknowledgment** Messages, the **RefToMessageId** reference is mandatory, and its value
 675 **MUST** be the **MessageId** of the ebXML Message being acknowledged.

676 7.9.3.4 MessageId sample

677 The following example demonstrates the usage of the **MessageData** element.

```
678 <MessageData>
679   <MessageId>UUID-2</MessageId>
680   <TimeStamp>20000725T121905.000Z</TimeStamp>
681   <RefToMessageId>UUID-1</RefToMessageId>
682 </MessageData>
```

683 7.9.4 ReliableMessagingInfo

684 The last element of the **ebXMLHeader** is the **ReliableMessagingInfo** element. This element
 685 identifies the degree of reliability with which the message will be delivered. This element has a
 686 single attribute, **DeliverySemantics**. This attribute is an enumeration, which may have one of the
 687 following values:
 688

- 689 • "OnceAndOnlyOnce" – reliable messaging semantics: the receiving Service Interface
 690 handler will receive a given message no more than once, the sending Message Service
 691 will execute retry procedures in the event of failure and the sending Service Interface
 692 handler will be notified in the event of failure.
- 693 • "BestEffort" – reliable delivery semantics are not specified: the Sending Service Interface
 694 handler is not notified of failure to deliver the message, duplicate messages might be
 695 delivered and persistent storages are not required.

```
696 <ReliableMessagingInfo DeliverySemantics="OnceAndOnlyOnce">
```

```
697 </ReliableMessagingInfo>
```

698 7.9.5 XML Header sample

699 The following fragment demonstrates the structure of the **Header** element of the **ebXMLHeader**
 700 document:
 701

```
702 <Header>
703   <From>...</From>
704   <To>...</To>
705   <TPAInfo>...</TPAInfo>
706   <MessageData>...</MessageData>
707   <ReliableMessagingInfo>...</ReliableMessagingInfo>
708 </Header>
```

709 **7.10 XML Routing Header**

710 One **RoutingHeader** element immediately follows the **Header** element. It is required in all
 711 **ebXMLHeader** documents. The **RoutingHeader** element is a composite element comprised of at
 712 least the following four required subordinate elements:

- 713 • **SenderURI** – the Sender’s Messaging Service Handler URI.
- 714 • **ReceiverURI** – the Receiver’s Messaging Service Handler URI.
- 715 • **ErrorURI** – URI designated by the Sender for reporting errors.
- 716 • **Timestamp** – timestamp of the **RoutingHeader** creation, in the same format used for
 717 **Timestamp** in the **XML Header MessageData** element.

718
 719 When the **RoutingHeader** is used for a message sent with Reliable Messaging functions
 720 (**DeliverySemantics** is set to “OnceAndOnlyOnce” in the **XML Header ReliableMessagingInfo**
 721 element), the Sender SHALL add one additional **RoutingHeader** element to the **RoutingHeader**:

- 722 • **SequenceNumber** – Integer value that is incremented (e.g. 1, 2, 3, 4...) for each Sender-
 723 prepared message sent to the Receiver. The Sequence Number consists of ASCII
 724 numerals in the range 1-999,999,999. In following cases, the Sequence Number takes
 725 the value “1”:
- 726 - First message from the Sender to a particular Receiver
- 727 - First message after wraparound (next value after 999,999,999)
- 728 - First message after removing Sequence Number information in the Sender (Sender MAY
 729 remove Sequence Number information when it has no messages which were sent to
 730 the Receiver for long time).

731
 732 The following fragment demonstrates the structure of the **RoutingHeader** element of the
 733 **ebXMLHeader** document when Reliable Messaging is used:

```

734 <RoutingHeader>
735   <SenderURI>...</SenderURI>
736   <ReceiverURI>...</ReceiverURI>
737   <ErrorURI>...</ErrorURI>
738   <Timestamp>...</Timestamp>
739   <SequenceNumber>...</SequenceNumber>
740 </RoutingHeader>
    
```

741 **7.11 Reliable Messaging Flow**

742 The Reliable Messaging function defines an interoperable protocol such that any two Messaging
 743 Service implementations can “reliably” exchange messages that are sent using “reliable
 744 messaging” semantics.

745
 746 Reliably exchanging messages means that, with respect to Sending and Receiving Message
 747 Service implementations:

- 748 • For any given message provided to the Sending Messaging Service, the Receiving
 749 Messaging Service will deliver at most one copy of the message to the Receiver.
- 750 • A positive acknowledgement will be sent from the Receiving Messaging Service to the
 751 Sending Messaging Service to indicate receipt and storage in persistent storage, and if this
 752 acknowledgement is not received the Sending Messaging Service will notify the original
 753 Sending Party

- 754 • Both the Sending and Receiving Messaging Services will use persistent storage for recovery

755

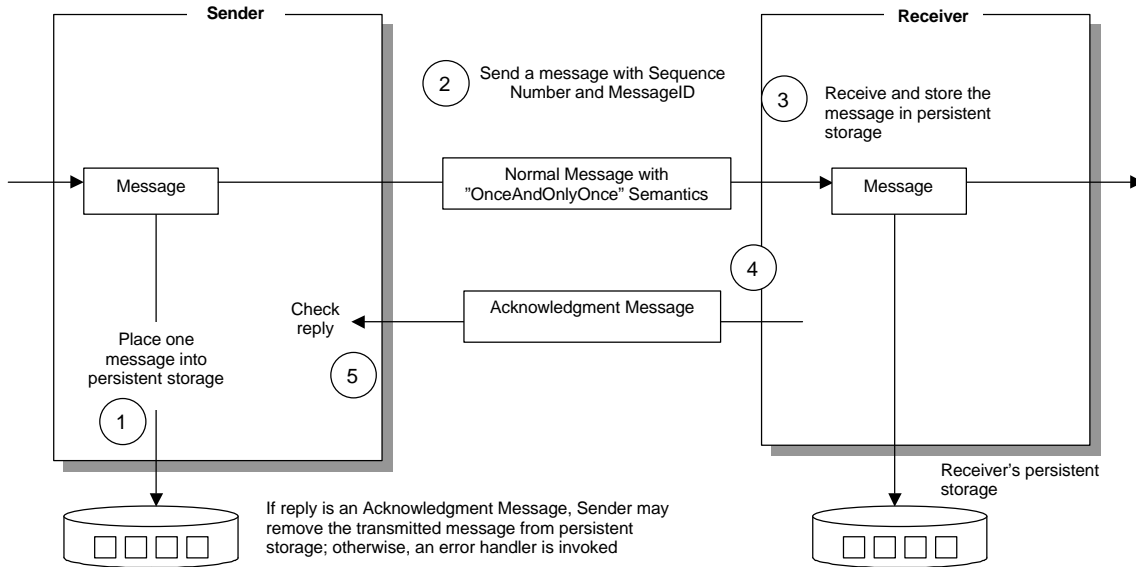
756 Reliable Messaging is defined only for direct connections between Messaging Service
 757 implementations. At a later time, networks consisting of intermediate Messaging Service
 758 implementations will be supported.

759

760 All ebXML Message Service implementations SHALL support the Reliable Messaging function.
 761 With respect to a particular Sender and Receiver pair, transmission of one reliable message
 762 SHALL be completed before another reliable message may be sent.

763

764 The following figure shows the reliable messaging flow:



765
 766

Figure 7-2: Reliable Message Transfer Sequence

767 Reliable Messaging processing is shown in the following sequence:

768

769 (1) Message preparation

770

771 Sender initially stores messages passed from the ebXML "From-Party" in persistent storage,
 772 and then prepare the stored message for message transfer.

773

774 (2) Sending message

775

776 A Reliable Message has **DeliverySemantics** = "OnceAndOnlyOnce", and receipt of a
 777 message with this value notifies the Receiver of Reliable Messaging semantics.

778

779 (3) Receiving, checking and storing message

780

781 The Receiver receives the reliable message and, if the message is not a duplicate message,
 782 stores the message in persistent storage and processes the message appropriately.

783

784 (4) Acknowledgment by Receiver

785

786 The Receiver returns an Acknowledgment Message to the Sender for every received reliable
 787 message, even if it is a duplicate message.

788

789 (5) Sender checks the acknowledgement and removes transferred message
 790
 791 Sender checks the Acknowledgement Message from the Receiver. If the reply is an
 792 appropriate Acknowledgement Message for the transferred message, Sender may remove
 793 the transferred message from Sender's persistent storage if the message is no longer
 794 needed for some other messaging service function or later failure recovery.
 795

796 The Receiver's Messaging Service sends an Acknowledgement Message to the Sender's
 797 Messaging Service for every Normal Reliable Messaging message received. There is no reply to
 798 the Acknowledgement message from the Sender's Messaging Service.
 799

800 In the Acknowledgement Message:

- 801 • The **MessageType** SHALL be "Acknowledgement"
- 802 • There is no Payload and no business level response information.
- 803 • **From** SHALL be the **ReceiverURI** as shown in the Routing Header Document
- 804 • **To** SHALL be the **SenderURI** as shown in the Routing Header Document
- 805 • **TPAId** and **ConversationID** as shown in the Header Document
- 806 • **ServiceInterface** and **Action** SHALL be empty
- 807 • **RefToMessageId** SHALL be the **MessageId** of the reliable message
- 808 • **DeliverySemantics** SHALL be "BestEffort"

809 **7.12 Reliable Messaging Recovery Procedures**

810 **7.12.1 Messaging Service Parameters**

811
 812 In Reliable Messaging, the sending messaging service uses the following Messaging Service
 813 parameters during recovery procedures.
 814

815 This information may be determined in a number of ways, such as the TPA or some other
 816 method.

817 **Table 7-1 Messaging Service Parameters used in Recovery**

Argument	Outline Description
Timeout	Wait time for any response from the Receiver. <ul style="list-style-type: none"> • Integer value specifying a number of seconds • After sending a Normal Message, the Sender SHALL wait for any response (MS Acknowledgement or Error Message) for the specified time before start of retry
Retries	Maximum number of retries. <ul style="list-style-type: none"> • Integer value specifying the number of retries • The Sender SHALL repeat retries the specified number of times until the Sender receives an MS Acknowledgement Message • If the Sender does not receive an MS Acknowledgement Message after the maximum number of retries, the Sender SHALL notify the incident to the higher level (application and/or system admin)

<i>RetryInterval</i>	Wait time between retries, if an Acknowledgement Message is not received <ul style="list-style-type: none"> • Integer value specifying a number of seconds • After a retry, the Sender SHALL wait for a response (MS Acknowledgement or Error Message) for specified time before start of the next retry
-----------------------------	--

818 **7.12.2 Recovery Sequence for Lost Messages**

819

820 When the Sender detects a timeout while waiting for an Acknowledgement Message from the last
 821 sent message, the appropriate recovery handler in the Sender executes a Messaging Service
 822 recovery sequence.

823

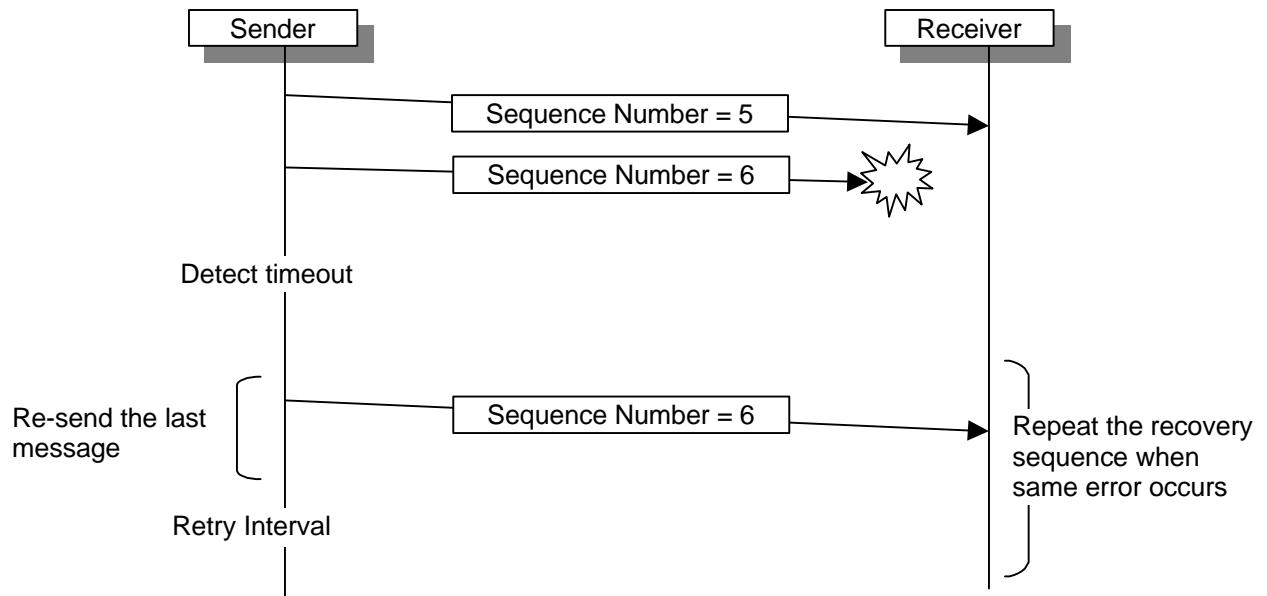
824 The timeout value period is defined as **Timeout**. The recovery sequence SHALL re-send the final
 825 message to the Receiver and SHALL use a retry interval, **RetryInterval**, between attempts The
 826 retry sequence SHALL be attempted a **Retries** number of times.

827

828 The content of the re-sent message is exactly the same as the original message. In the recovery
 829 sequence or after the recovery sequence,

830

- 831 • If the Sender does not receive any error message or Acknowledgment Message in the
 832 retry interval, the recovery handler repeats the recovery sequence the **Retries** number
 833 of times.
- 834
- 835 • If the Sender detects or receives another Error Message, the recovery handler executes
 836 the appropriate recovery sequence for the error.
- 837
- 838 • If the Sender receives an Acknowledgment Message during the recovery sequence, the
 839 message transmission is completed.



840

841

Figure 7-3 Recovery Sequence for Timeout

842

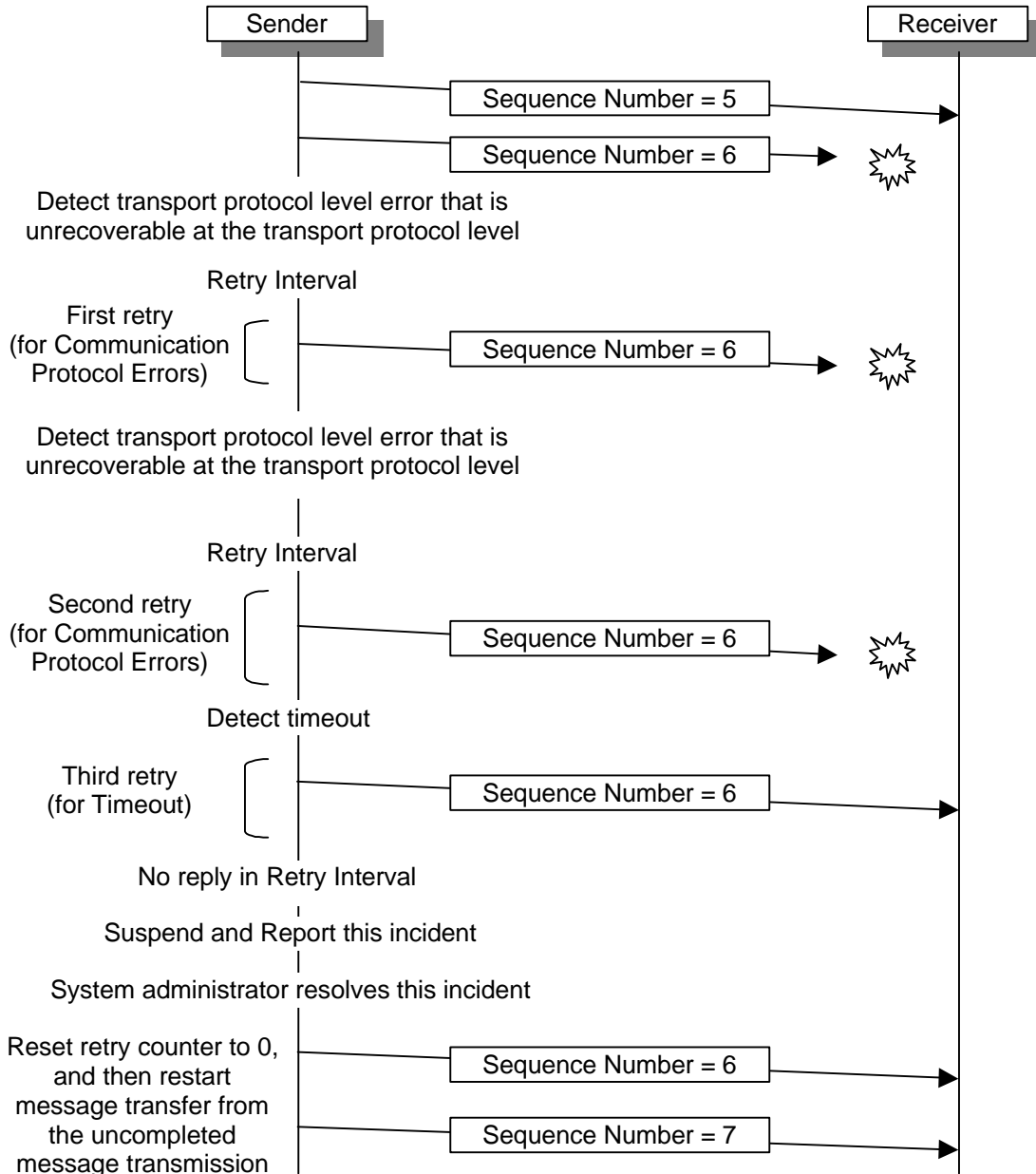
843

7.12.3 Maximum Number of Retries and Retry Interval

844 The retry interval is defined as **RetryInterval**. When the total number of retries in a reliable
 845 message transmission reaches a maximum number, defined as **Retries**, and the last error is still
 846 not resolved, the recovery handler will:

- 847
- 848 (1) Suspend sending messages to the Receiver
- 849
- 850 (2) Report this incident to a higher-level so that a system administrator can resolve this incident
- 851

852 When the system administrator resolves the incident, the recovery handler will reset the retry
 853 counter to zero and then re-start message transfer sequence from the uncompleted reliable
 854 message transmission.



855

856 **Figure 7-4 Repeat of Recovery Sequence (maximum number of retries specified is 3)**

857

858 7.13 ebXML Error Reporting

859 This section describes how one ebXML Message Service reports errors it detects to another
860 ebXML Message Service.

861 7.13.1 Definitions

862 For clarity two phrases are defined which are used in this section:

- 863 • *message in error*. A message which contains or causes an error of some kind
- 864 • *message reporting the error*. A message that contains an ebXML Error Document that
865 describes the error(s) found in a *message in error*.

866 7.13.2 Types of Errors

867 One ebXML Message Service needs to report to another ebXML Message Service errors in
868 *message in error* that are associated with:

- 869 • the structure or content of the *Message Envelope* (e.g. MIME),
- 870 • the ebXML Message Header document,
- 871 • security, or
- 872 • reliable messaging failures.

873

874 Unless specified to the contrary, all references to "an error" in the remainder of this specification
875 imply any of the types of errors described above.

876

877 Errors associated with Data Communication protocols are detected and managed in an
878 implementation specific way and are not part of this error reporting mechanism

879 7.13.3 When to generate Error Messages

880 When an ebXML Message Service detects an error in a *message in error*, a *message reporting*
881 *the error* MUST be generated and delivered to the ebXML Message Service which sent the
882 *message in error* for a normal ebXML message if:

- 883 • the Error Reporting Location (see section 7.13.4) to which the *message reporting the*
884 *error* should be sent can be determined, and
- 885 • the message in error does not have a **MessageType** of **Error**.

886

887 If the Error Reporting Location cannot be found or the *message in error* has a **MessageType** of
888 **Error**, it is recommended that:

- 889 • the error is logged,
- 890 • the problem is resolved by other means, and
- 891 • no further action is taken.

892 7.13.4 Identifying the Error Reporting Location

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

- 895 • by reference, for example by using the **TPAId** to identify the Party Agreement which
896 contains the Error Reporting Location, or
- 897 • by value, for example by using the **ErrorURI** contained within the Routing Header
898 element.

899 If a *message* contains both an **ErrorURI** and other means of identifying the Error Reporting
 900 Location then the **ErrorURI** MUST be used.

901
 902 If an **ErrorURI** is not used then the method used to determine the Error Reporting Location is
 903 outside of the scope of this version of the specification.

904
 905 Even if the *message in error* cannot be successfully analyzed or parsed, ebXML Message
 906 Service implementers SHOULD try to determine the Error Reporting Location by other means.
 907 How this is done is an implementation decision.

908 **7.13.5 ebXML Error Message**

909 This section defines the structure and content of an ebXML Error Message that is contained
 910 within a *message reporting an error*.

911 **7.13.5.1 Message Structure**

912 An ebXML Error Message is created using the rules for creating an ebXML Message contained
 913 within this specification. In addition:

- 914 • the **MessageType** in the ebXML header is set to **Error**
- 915 • the payload consists of a single ebXML Error Document

916 **7.13.5.2 ebXML Error Document**

917 An ebXML Error Document has a root element that consists of:

- 918 • an **ErrorHeader** element that identifies the nature and severity of the error, and
- 919 • zero or more **ErrorLocation** elements, that identify the part(s) of the message(s) that are
 920 in error.

921 The structure of an ebXML Error Document is illustrated below.

922
 923
 924
 925
 926
 927
 928
 929
 930

```
<?xml version="1.0" encoding="UTF-8"?>
<ebXMLError xmlns="http://www.ebxml.org/namespaces/error"
  Version="0.8">
  <ErrorHeader>...</ErrorHeader>
  <ErrorLocation>...</ErrorLocation>
  <ErrorLocation>...</ErrorLocation>
  ...
</ebXMLError>
```

931 Later versions of this specification may define how to report more than one error within an ebXML
 932 Error Document.

933 **7.13.5.3 Error Header Element**

934 The **ErrorHeader** element identifies the nature and severity of the error. It consists of the
 935 following attributes/elements.

936 **7.13.5.3.1 ID attribute**

937 The optional **ID** attribute uniquely identifies the **ErrorHeader** Element within the document.

938 **7.13.5.3.2 ErrorCode element**

939 The required **ErrorCode** element indicates the nature of the error in the *message in error*. Valid
 940 values for the **ErrorCode** are given in section 7.13.5.7.

941 **7.13.5.3.3 Severity element**

942 The required **Severity** element indicates the severity of the error. Valid values are:

- 943 • **Warning** - This indicates that although there is a message in error other messages in the
 944 conversation will still be generated in the normal way.
- 945 • **Error** - This indicates that there is an unrecoverable error in the message in error and no
 946 further messages will be generated as part of the conversation.

947 **7.13.5.3.4 Description element**

948 The optional **Description** element provides a narrative description of the error in the language
 949 defined by the *xml:lang* attribute on the **Description** element. The content of this attribute is
 950 defined by the vendor/developer of the software, which generated the ebXML Error Document.
 951 *xml:lang* must comply with the rules for identifying languages specified in [XML].

952 **7.13.5.3.5 SoftwareDetails element**

953 The optional **SoftwareDetails** element contains a value that is set by the vendor/developer of the
 954 software, which generated the ebXML Error Document. It SHOULD contain data that enables the
 955 vendor/developer to identify the precise location in their software and the set of circumstances
 956 that caused the software to generate a *message reporting the error*. It is RECOMMENDED that
 957 this element include plain text to identify:

- 958 • the name of the software vendor;
- 959 • the name, version and release number of the software that generated the ebXML Error
 960 Document
- 961 • the part of the software that caused the error to be generated which can be used by the
 962 Software Vendor to identify the circumstances that caused the error

963 **7.13.5.4 Examples**

964 Two examples of an **ErrorHeader** element are given below.

```
965 <ErrorHeader ID='ab184832' >
966 <ErrorCode>UnableToParse</ErrorCode>
967 <Severity>Error</Severity>
968 <Description xml:lang='en-uk'>The "MessageManifest" element is not well formed.</Description>
969 <SoftwareDetails>Software Development Corp.; ebXML Connector!!; v2.7, build 2.7313; Ref
970 HA</SoftwareDetails>
971 </ErrorHeader>
```

```
974 <ErrorHeader ID='sdj2309823' >
975 <ErrorCode>NotSupported</ErrorCode>
976 <Severity>Error</Severity>
977 <Description xml:lang='en-us'>Delivery Semantics of "OnceAndOnlyOnce" are not
978 supported.</Description>
979 <SoftwareDetails>Unreliable Software Development Corp.; ebXML Message Handler !!; v23.5, build 5751;
980 Ref: xapowekxd</SoftwareDetails>
981 </ErrorHeader>
```

982 **7.13.5.5 Error Location Element**

983 The **ErrorLocation** Element identifies the location of an error either within a message or
 984 elsewhere.

985 Frequently a single **ErrorLocation** element will be all that is required within an ebXML Error
 986 document. For example, an **ErrorCode** of **ValueNotRecognized** is likely to reference an element
 987 or attribute and no other **ErrorLocation** element will be needed.

989

990 Sometimes though, multiple **ErrorLocation** elements will be required to define where the
 991 problem is. For example, an error code with a value of **Inconsistent** would frequently have two or
 992 more **ErrorLocation** elements that point to the various items that are inconsistent.
 993 The number of **ErrorLocation** elements included in an ebXML Error document is an
 994 implementation decision.

995
 996 An **ErrorLocation** element consists of the following attributes/elements:

- 997 • **ID** attribute
- 998 • **RefToMessageld** element
- 999 • **Href** element

1000 7.13.5.5.1 ID Attribute

1001 The optional **ID** attribute uniquely identifies the **ErrorLocation** element within the ebXML Error
 1002 document.

1003 7.13.5.5.2 RefToMessageld element

1004 The optional **RefToMessageld** element contains the **Messageld** from the ebXML Header
 1005 Document of the *message in error*. This must be present if a **Messageld** can be identified within
 1006 the *message in error*.

1007 7.13.5.5.3 Href element

1008 The **Href** URI identifies either some other location within the *message in error*, or elsewhere, that
 1009 helps identify the location of the error.

1010 7.13.5.6 Examples

1011 Two examples of an **ErrorLocation** element are given below. The first example is indicating that
 1012 the referenced message is inconsistent with a previously agreed Party Agreement.

```
1013
1014 <ErrorLocation ID='4982hw'>
1015 <RefToMessageld>ab131982387123</RefToMessageld>
1016 <Href>url:example.com/tpa/471839<Href>
1017 </ErrorLocation>
```

1018 The second example is pointing to an error in an ebXML Header Document.

```
1019
1020 <ErrorLocation ID='120938uqwe'>
1021 <RefToMessageld>ac198327123098</RefToMessageld>
1022 <Href>cid:-----8760<Href>
1023 </ErrorLocation>
```

1025 7.13.5.7 Error Codes

1026 This section describes the **ErrorCodes** (see section 7.13.5.3.2) that are used in a *message*
 1027 *reporting an error*. They are described as a list of bullet points. The following describes how to
 1028 interpret this list:

- 1029 • the first word is the actual **ErrorCode**, e.g. **UnableToParse**
- 1030 • the single sentence that immediately follows the error code is a "narrative" that describes
 1031 the **ErrorCode**, for example "XML not well formed or invalid".
- 1032 • the sentence(s) that follow the narrative, are the explanation of the meaning of the error
 1033 and provide guidance on when the particular **ErrorCode** should be used.
- 1034 • It is RECOMMENDED that implementers:
 - 1035 - use both the **ErrorCode** and the "narrative" to explain an error to, for example, a user

- 1036 - translate the "narrative" into the preferred language of the recipient of *the message in*
1037 *error* if this is known

1038 **7.13.5.8 Reporting Errors in the ebXML Header Document**

1039 The following list contains error codes that can be associated with XML documents, for example
1040 the ebXML Header Document:

- 1041 • **UnableToParse** - XML not well formed or invalid. The XML document is not well formed
1042 or not valid and cannot be successfully parsed. See [XML] for the meaning of "well
1043 formed" and "not valid".
- 1044 • **ValueNotRecognized** - Element content or attribute value not recognized. Although the
1045 document is well formed and valid, the element/attribute contains a value which could not
1046 recognized and therefore could not be used by the ebXML Message Service
- 1047 • **NotSupported** - Element or attribute not supported. Although the document is well
1048 formed and valid, an element or attribute is present that:
 - 1049 - is consistent with the rules and constraints contained in this specification, but
 - 1050 - is not supported by the ebXML Message Service that is processing the message.
- 1051 • **Inconsistent** - Element content or attribute value inconsistent with other elements or
1052 attributes. Although the document is well formed and valid, according to the rules and
1053 constraints contained in this specification the content of an element or attribute is
1054 inconsistent with the content of other elements or their attributes.
- 1055 • **OtherXml** - Other error in an element content or attribute value. Although the document
1056 is well formed and valid, the element content or attribute value contains values which do
1057 not conform to the rules and constraints contained in this specification and is not covered
1058 by other error codes. The **Description** element should be used to indicate the nature of
1059 the problem.

1060 **7.13.5.9 Non-XML Document Errors**

1061 The following are error codes that identify errors that are not associated with an XML Document:

- 1062 • **MessageTooLarge** - Message too large. The message is too large to be processed by
1063 the ebXML Message Service.
- 1064 • **MimeProblem** - A MIME error has occurred. An error has been detected in the structure
1065 or format of a MIME part of the message. For example:
 - 1066 - Missing MIME Part. Although the MIME message is correctly structured, a MIME part is
1067 missing that should have been present if the rules and constraints contained in this
1068 specification are followed
 - 1069 - Unexpected MIME Part. Unexpected MIME part. Although the MIME message is correctly
1070 structured, a MIME part is present that is not expected in the particular context
1071 according to the rules and constraints contained in this specification
- 1072 • **Unknown** - Unknown Error. Indicates that an error has occurred that is not covered
1073 explicitly by any of the other errors. The **Description** element should be used to indicate
1074 the nature of the problem.

1075 Note this list will be expanded in future versions of this specification, for example to report errors
1076 on security.
1077

1078 **7.14 Security**

1079 This version of the specification supports limited security services, that is those security services
1080 that can be supported within the payload. The `multipart/related` payload may be
1081 encrypted using cryptographic techniques suitable for the payload type. Expanded definition of
1082 security will be addressed in the Phase 2.

1083 **8 References**

1084 **8.1 Normative References**

- 1085 [Glossary] ebXML Glossary, see ebXML Project Team Home Page
- 1086 [ISO 8601] International Standards Organization Ref. ISO 8601 Second Edition, Published 1997
- 1087 [RFC 2392] IETF Request For Comments 2392. Content-ID and Message-ID Uniform Resource
1088 Locators. E. Levinson, Published August 1998
- 1089 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of
1090 Internet Message Bodies, N Freed & N Borenstein, Published November 1996
- 1091 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage
1092 conventions.
- 1093 [XML Namespace] Recommendation for Namespaces in XML, World Wide Web Consortium, 14
1094 January 1999, <http://www.w3.org/TR/REC-xml-names>
- 1095 [XML] Extensible Mark Up Language. A W3C recommendation. See
1096 <http://www.w3.org/TR/1998/REC-xml-19980210> for the 10 February 1998 version.
1097

1098 **8.2 Non-Normative References**

- 1099 [XMTP] XMTP - Extensible Mail Transport Protocol
1100 <http://www.openhealth.org/documents/xmtp.htm>
- 1101 [TRPREQ] ebXML Transport, Routing and Packaging: Overview and Requirements, Version
1102 0.96, Published 25 May 2000
- 1103 [XMLMedia] IETF Internet Draft on XML Media Types. See [http://www.imc.org/draft-murata-xml-](http://www.imc.org/draft-murata-xml-08)
1104 08. Note. It is anticipated that this Internet Draft will soon become a RFC. Final
1105 versions of this specification will refer to the equivalent RFC.
- 1106
1107

1108 **9 Disclaimer**

1109 The views and specification expressed in this document are those of the authors and are not
1110 necessarily those of their employers. The authors and their employers specifically disclaim
1111 responsibility for any problems arising from correct or incorrect implementation or use of this
1112 design.

1113 **10 Contact Information**

1114

1115 **Team Leader**

1116 Name Rik Drummond
 1117 Company Drummond Group, Inc.
 1118 Street 5008 Bentwood Crt.
 1119 City, State, Postal Code Fort Worth, Texas 76132
 1120 Country USA
 1121 Phone: (817) 294-7339
 1122 EMail: rik@drummondgroup.com

1123

1124 **Vice Team Leader**

1125 Name Chris Ferris
 1126 Company Sun Microsystems
 1127 Street One Network Drive
 1128 City, State, Postal Code Burlington, MA 01803-0903
 1129 Country USA
 1130 Phone: (781) 442-3063
 1131 EMail: chris.ferris@sun.com

1132

1133 **Team Editor**

1134 Name David Burnett
 1135 Company CommerceOne
 1136 Street 4400 Rosewood Drive 3rd Fl, Bldg 4
 1137 City, State, Postal Code Pleasanton, CA 94588
 1138 Country USA
 1139 Phone: (925) 520-4422 or (650) 623-2888
 1140 EMail: david.burdett@commerceone.com

1141

1142 **Authors**

1143 Name Dick Brooks
 1144 Company Group 8760
 1145 Street 110 12th Street North, Suite F103
 1146 City, State, Postal Code Birmingham, Alabama 35203
 1147 Phone: (205) 250-8053
 1148 E-mail: dick@8760.com

1149

1150 Name David Burdett
 1151 Company CommerceOne
 1152 Street 4400 Rosewood Drive 3rd Fl, Bldg 4
 1153 City, State, Postal Code Pleasanton, CA 94588
 1154 Country USA
 1155 Phone: (925) 520-4422 or (650) 623-2888
 1156 EMail: david.burdett@commerceone.com

1157

1158 Name Chris Ferris
 1159 Company Sun Microsystems
 1160 Street One Network Drive
 1161 City, State, Postal Code Burlington, MA 01803-0903
 1162 Country USA
 1163 Phone: (781) 442-3063
 1164 EMail: chris.ferris@east.sun.com

1165

1166 Name John Ibbotson
 1167 Company IBM UK Ltd
 1168 Street Hursley Park
 1169 City, State, Postal Code Winchester SO21 2JN
 1170 Country United Kingdom
 1171 Phone: +44 (1962) 815188
 1172 Email: john_ibbotson@uk.ibm.com
 1173
 1174 Name Nicholas Kassem
 1175 Company Java Software, Sun Microsystems
 1176 Street 901 San Antonio Road, MS CUP02-201
 1177 City, State, Postal Code Palo Alto, CA 94303-4900
 1178 Phone: (408) 863-3535
 1179 E-mail: Nick.Kassem@eng.sun.com
 1180
 1181 Name Masayoshi Shimamura
 1182 Company Fujitsu Limited
 1183 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
 1184 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan
 1185 Phone: +81-45-476-4590
 1186 E-mail: shima@rp.open.cs.fujitsu.co.jp
 1187
 1188 **Document Editing Team**
 1189 Name Ralph Berwanger
 1190 Company bTrade.com
 1191 Street 2324 Gateway Drive
 1192 City, State, Postal Code Irving, TX 75063
 1193 Country USA
 1194 Phone: (972) 580-2900
 1195 EMail: rberwanger@btrade.com
 1196
 1197 Name Ian Jones
 1198 Company British Telecommunications
 1199 Street Enterprise House, 84-85 Adam Street
 1200 City, State, Postal Code Cardiff, CF24 2XF
 1201 Country UK
 1202 Phone: +44 29 2072 4063
 1203 EMail: ian.c.jones@bt.com
 1204
 1205 Name Martha Warfelt
 1206 Company DaimlerChrysler Corporation
 1207 Street 800 Chrysler Drive
 1208 City, State, Postal Code Auburn Hills, MI
 1209 Country USA
 1210 Phone: (248) 944-5481
 1211 EMail: maw2@daimlerchrysler.com
 1212
 1213
 1214

1215 Appendix A DTD Definitions

1216 The following are definitions for validation of the *ebXML Message* header structure.

1217 A.1 ebXML Header DTD

```

1218 <?xml version="1.0" encoding="UTF-8"?>
1219 <!ELEMENT ebXMLHeader (Manifest , Header , RoutingHeader )>
1220 <!ATTLIST ebXMLHeader xmlns CDATA #FIXED "http://ebxml.org/namespaces/messageHeader"
1221     Version CDATA #FIXED '0.8'
1222     MessageType (Normal | Error | Acknowledgment ) #REQUIRED >
1223 <!ELEMENT Manifest (DocumentReference )*>
1224 <!ELEMENT DocumentReference (DocumentLabel , DocumentId , DocumentDescription? )>
1225 <!ELEMENT DocumentLabel (#PCDATA )>
1226 <!ATTLIST DocumentLabel e-dtype NMTOKEN #FIXED 'string' >
1227
1228 <!ELEMENT DocumentId (#PCDATA )>
1229 <!ATTLIST DocumentId e-dtype NMTOKEN #FIXED 'uri' >
1230
1231 <!ELEMENT DocumentDescription (#PCDATA )>
1232 <!ATTLIST DocumentDescription xml:lang NMTOKEN #REQUIRED>
1233
1234 <!ELEMENT Header (From , To , TPAInfo , MessageData , ReliableMessagingInfo )>
1235 <!ELEMENT ServiceInterface (#PCDATA )>
1236 <!ATTLIST ServiceInterface e-dtype NMTOKEN #FIXED 'string' >
1237
1238 <!ELEMENT Action (#PCDATA )>
1239 <!ATTLIST Action e-dtype NMTOKEN #FIXED 'string' >
1240
1241 <!ELEMENT TPAId (#PCDATA )>
1242 <!ATTLIST TPAId e-dtype NMTOKEN #FIXED 'uri' >
1243
1244 <!ELEMENT ConversationId (#PCDATA )>
1245 <!ATTLIST ConversationId e-dtype NMTOKEN #FIXED 'uri' >
1246
1247 <!ELEMENT MessageData (MessageId , Timestamp , RefToMessageId? )>
1248
1249 <!ELEMENT RefToMessageId (#PCDATA )>
1250 <!ATTLIST RefToMessageId e-dtype NMTOKEN #FIXED 'string' >
1251
1252 <!ELEMENT Timestamp (#PCDATA )>
1253 <!ATTLIST Timestamp e-dtype NMTOKEN #FIXED 'dateTime' >
1254
1255 <!ELEMENT MessageId (#PCDATA )>
1256 <!ATTLIST MessageId e-dtype NMTOKEN #FIXED 'string' >

```



```

1257 <!ELEMENT From (PartyId )>
1258
1259 <!ELEMENT To (PartyId )>
1260
1261 <!ELEMENT PartyId (#PCDATA )>
1262 <!ATTLIST PartyId context CDATA #IMPLIED
1263         e-dtype NMTOKEN #FIXED 'uri' >
1264 <!ELEMENT ReliableMessagingInfo EMPTY>
1265 <!ATTLIST ReliableMessagingInfo DeliverySemantics (OnceAndOnlyOnce | BestEffort ) #REQUIRED >
1266
1267 <!ELEMENT TPAInfo (TPAId , ConversationId , ServiceInterface , Action )>
1268 <!ELEMENT RoutingHeader (SenderURI , ReceiverURI , ErrorURI , Timestamp , SequenceNumber? )>
1269 <!ELEMENT SenderURI (#PCDATA )>
1270 <!ELEMENT ReceiverURI (#PCDATA )>
1271 <!ELEMENT SequenceNumber (#PCDATA )>
1272 <!ELEMENT ErrorURI (#PCDATA )>

```

1273

1274 A.2 ebXML Error DTD

```

1275 <?xml version ="1.0"?>
1276 <!ELEMENT ebXMLError (ErrorHeader, ErrorLocation*) >
1277 <!ELEMENT ErrorHeader (ErrorCode, Severity, Description?, SoftwareDetails>
1278 <!ATTLIST ErrorHeader
1279     ID          NMTOKEN #IMPLIED
1280 <!ELEMENT ErrorCode (#PCDATA) > <-- string max 20 char -->
1281 <!ELEMENT Severity (#PCDATA) > <-- Either 'Warning' or 'Error' -->
1282 <!ELEMENT Description (#PCDATA) > <-- string max 1024 (?) char -->
1283 <!ATTLIST Description
1284     xml:lang    NMTOKEN #REQUIRED >
1285 <!ELEMENT SoftwareDetails(#PCDATA) ?> <-- string max 16k (?) chars -->
1286 <!ELEMENT ErrorLocation (RefToMessageId?, Href ) >
1287 <!ATTLIST ErrorLocation
1288     ID          NMTOKEN #IMPLIED >
1289 <!ELEMENT RefToMessageId (#PCDATA) >
1290 <!ELEMENT Href (#PCDATA) >
1291

```

1292 **Appendix B Examples**

1293 The following are complete examples of *ebXML Messages* showing the structure as defined in
 1294 this specification.

1295 **B.1 Complete Example of an ebXML Message Envelope using**
 1296 **multipart/related Content-Type sent via HTTP POST**

```

1297 POST http://127.0.0.1:9090/servlet/AS2Appl HTTP/1.0
1298 Connection: Keep-Alive
1299 User-Agent: Group 8760 Java MultiPost
1300 Content-type: multipart/related; type="application/vnd.eb+xml"; version="0.8"; boundary=-----8760567890----
1301 Content-Length: 2717
1302
1303 -----8760567890----
1304 Content-ID: ebxmlheader-8760-901543739
1305 Content-Length: 1357
1306 Content-type: application/vnd.eb+xml; version="0.8"; charset="UTF-8"
1307
1308 <?xml version = "1.0" encoding="UTF-8"?>
1309 <!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
1310 <ebXMLHeader
1311     xmlns = "http://www.ebxml.org/namespaces/messageHeader"
1312     Version = "0.8"
1313     MessageType = "Normal">
1314   <Manifest>
1315     <DocumentReference>
1316       <DocumentLabel>Purchase Order Request Action</DocumentLabel>
1317       <DocumentId>cid:8760.com901543736</DocumentId>
1318       <DocumentDescription xml:lang="en-us">GCI Purchase Order</DocumentDescription>
1319     </DocumentReference>
1320   </Manifest>
1321   <Header>
1322     <From>
1323       <PartyId context = "DUNS">2059397184</PartyId>
1324     </From>
1325     <To>
1326       <PartyId context = "DUNS">943561654</PartyId>
1327     </To>
1328     <TPAInfo>
1329       <TPAId>/2059397184/943561654GCIPO-20000202</TPAId>
1330       <ConversationId>8760.com901543737</ConversationId>
1331       <ServiceInterface>OrderProcessing</ServiceInterface>
1332       <Action>ProcessNewOrder</Action>
1333     </TPAInfo>
    
```

```

1334 <MessageData>
1335   <MessageId>8760.com901543738</MessageId>
1336   <Timestamp>20001014175625510.000Z</Timestamp>
1337   <RefToMessageId>Not Applicable</RefToMessageId>
1338 </MessageData>
1339 <ReliableMessagingInfo DeliverySemantics ="OnceAndOnlyOnce"/>
1340 </Header>
1341 <RoutingHeader>
1342   <SenderURI>http://dcx.com/ebXMLHandler</SenderURI>
1343   <ReceiverURI>http://gm.com/ebXMLHandler</ReceiverURI>
1344   <ErrorURI>mailto:ebxmlerrors@ford.com</ErrorURI>
1345   <Timestamp>20001014175625510.000Z</Timestamp>
1346   <SequenceNumber>00001</SequenceNumber>
1347 </RoutingHeader>
1348 </ebXMLHeader>
1349 -----8760567890----
1350 Content-Length: 1043
1351 Content-ID: 8760.com901543736
1352 Content-type: application/xml
1353
1354 <?xml version="1.0" encoding="UTF-8"?>
1355 <!DOCTYPE Order SYSTEM "OrderV0.1072400.dtd">
1356 <Order actionRequestStatusIndicator="Create">
1357   <Document id="" status="COPY" type="" language="EN">
1358     <creationDate date="2000-02-02"/>
1359   </Document>
1360   <buyer>
1361     <PartyIdentification>
1362       <GlobalLocationNumber>4325335000003</GlobalLocationNumber>
1363     </PartyIdentification>
1364     <BuyerAlignmentIdentification>Buyer</BuyerAlignmentIdentification>
1365   </buyer>
1366   <seller>
1367     <PartyIdentification>
1368       <GlobalLocationNumber/>
1369     </PartyIdentification>
1370     <SellerAlignmentIdentification/>
1371   </seller>
1372   <shipTo>
1373     <PartyIdentification>
1374       <GlobalLocationNumber/>
1375     </PartyIdentification>
1376     <ShipToAlignmentIdentification/>
1377   </shipTo>
1378   <requestedMovementType movement="requestedDeliveryDate">
1379     <movementDate date="2000-03-03"/>

```

```

1380     </requestedMovementType>
1381     <lineltem lineltemNumber="1">
1382         <itemld>
1383             <GlobalTradeltemNumber/>
1384         </itemld>
1385         <requestedQuantity value="100"/>
1386         <price netPrice="1000.00" currencyOfNetPrice=""/>
1387     </lineltem>
1388 </Order>
1389
1390 -----8760567890-----
    
```

1391 **B.2 Complete Example of an ebXML Message Envelope using** 1392 **multipart/related Content-Type sent via SMTP**

1393 The default Content-transfer-encoding type of 7BIT is being used in this message.
 1394

```

1395 From dick@8760.com Sun May 7 17:01:14 2000
1396 Received: from granger.mail.mindspring.net by alpha2000.tech-comm.com; (8.8.5/1.1.8.2/05Jun95-1217PM)
1397     id RAA32702; Sun, 7 May 2000 17:01:13 -0500 (CDT)
1398 Received: from gamma (user-33qt10l.dialup.mindspring.com [199.174.132.21])
1399     by granger.mail.mindspring.net (8.9.3/8.8.5) with SMTP id SAA11942
1400     for <ebxmlhandler@8760.com>; Sun, 7 May 2000 18:11:14 -0400 (EDT)
1401 From: "Dick Brooks (E)" <dick@8760.com>
1402 To: <ebxmlhandler@8760.com>
1403 Subject: OTA Commission Event
1404 Date: Sun, 7 May 2000 17:07:38 -0500
1405 Message-ID: <NDBBIOBMLCDOHCHIKMGKKEIDAAA.dick@8760.com>
1406 MIME-Version: 1.0
1407 X-Priority: 3 (Normal)
1408 X-MSMail-Priority: Normal
1409 X-Mailer: Microsoft Outlook IMO, Build 9.0.2416 (9.0.2910.0)
1410 Importance: Normal
1411 X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300
1412 Content-Length: 8081
1413 Content-Type: multipart/related; type="application/vnd.eb+xml"; version="0.8"; boundary="---
1414 =_NextPart_000_0005_01BFB846.BF7FABA0"
1415
1416 -----=_NextPart_000_0005_01BFB846.BF7FABA0
1417 Content-Type: application/vnd.eb+xml; charset="UTF-8"
1418 Content-ID: ebxmlheader-9000
1419 Content-Length: 272
1420
1421 <?xml version="1.0" encoding="UTF-8"?>
1422 <ebXMLHeader xmlns = "http://www.ebxml.org/namespaces/messageHeader"
1423     Version = "0.8"
    
```

```

1424     MessageType = "Normal">
1425     <Manifest>
1426     <DocumentReference>
1427     <DocumentLabel>Purchase Order Request Action</DocumentLabel>
1428     <DocumentId>
1429     cid:ebxmlpayload-9000
1430     </DocumentId>
1431     </DocumentReference>
1432     </Manifest>
1433     <Header>
1434     <From>
1435     <PartyId context = "DUNS">requester-DUNS-number</PartyId>
1436     </From>
1437     <To>
1438     <PartyId context = "DUNS">responder-DUNS-number</PartyId>
1439     </To>
1440     <TPAInfo>
1441     <TPAId context = "tpadb">
1442     /requester-DUNS-number/responder-DUNS-number/PIP3A4/1.1
1443     </TPAId>
1444     <ConversationId context = "CreatePurchaseOrder">
1445     uid@requester-domain
1446     </ConversationId>
1447     <BusinessServiceInterface>
1448     Seller Service
1449     </BusinessServiceInterface>
1450     <Action version="1.1">Purchase Order Request Action</Action>
1451     </TPAInfo>
1452     <MessageData>
1453     <MessageId>uid@requester-domain</MessageId>
1454     <TimeStamp>CCYYMMDDThhmmss.sssZ</TimeStamp>
1455     <RefToMessageId>Not Applicable</RefToMessageId>
1456     </MessageData>
1457     <ReliableMessagingInfo DeliverySemantics = "BestEffort"/>
1458     </Header>
1459     </ebXMLHeader>
1460     -----=_NextPart_000_0005_01BFB846.BF7FABA0
1461     Content-Type: text/xml
1462     Content-ID: ebxmlpayload-9000
1463     Content-Length: 7515
1464
1465     <?xml version="1.0" encoding="UTF-8"?>
1466     <HITISMessage xmlns="" Version="1.0">
1467     <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
1468     <FromURI>http://www.pms.com/HITISInterface</FromURI>
1469     <ToURI>http://www.crs.com/HITISInterface</ToURI>

```

```

1470      <ReplyToURI>http://www.pms.com/HITISInterface</ReplyToURI>
1471      <MessageID>1234567890</MessageID>
1472      <OriginalMessageID>1234567890</OriginalMessageID>
1473      <TimeStamp>1999-11-10T10:23:44</TimeStamp>
1474      <Token>1234-567-8901</Token>
1475      </Header>
1476      <Body>
1477          <HITISOperation OperationName="CommissionEventsUpdate">
1478
1479
1480              <BodyPartstuffgoeshere/>
1481
1482
1483          </HITISOperation>
1484      </Body>
1485 </HITISMessage>
1486 -----=_NextPart_000_0005_01BFB846.BF7FABA0--
    
```

1487 **Appendix C Candidate Packaging Technologies and** 1488 **Selection Process**

1489 The packaging sub-group began its investigation of packaging technologies by identifying the
 1490 technologies currently used for business-to-business message exchange or were being
 1491 developed for this purpose. The following packaging technologies were identified:

- 1492 • MIME - currently in use by companies exchanging business transactions using E-mail
 1493 and HTTP
- 1494 • XML - currently used by Microsoft (BizTalk and SOAP) and others

1495 **C.1 Selection Process**

1496 Each candidate technology was evaluated based on its ability to meet the requirements listed in
 1497 the section titled "Packaging and other Requirements" in this document. When necessary,
 1498 specific parties were contacted to provide details describing how a technology was being used to
 1499 meet specific requirements. The following parties were contacted to provide expert insight:

- 1500 • Microsoft - David Turner, regarding use of XML packaging in BizTalk
- 1501 • Develop Mentor - Don Box, regarding use of XML packaging in SOAP
- 1502 • Vitria - Prasad Yendluri, regarding use of XML packaging in RosettaNet
- 1503 • Jonathan Borden - author of [XMTP], an XML to MIME transformation tool
 1504

1505 The packaging sub-group considered the inputs of people from the ebXML Transport mailing list
 1506 as well as the parties listed above, before making a selection.

1507 **C.2 MIME**

1508 Multipurpose Internet Mail Extensions (MIME) is an international standard created by the Internet
 1509 Engineering Task Force. It has been implemented by numerous software vendors across the
 1510 globe and has been used to exchange mixed type payloads, including XML, for several years.
 1511 MIME was designed purely as a packaging (enveloping) solution to allow the transport of mixed
 1512 payloads using Internet E-mail (SMTP). MIME is also being used by other transport technologies
 1513 as a packaging technology, most notably HTTP.

1514 **C.3 XML**

1515 eXtensible Markup Language (XML) version 1.0 is a technical specification holding a
 1516 RECOMMENDED status created by the World Wide Web Consortium. It has been implemented
 1517 by numerous software vendors across the globe and has been used to describe a broad
 1518 spectrum of document structures from very simple to very complex. XML is a very flexible
 1519 markup language that can be used to represent virtually any type of document. XML can be used
 1520 solely for packaging (enveloping) documents of any type, providing the data can be "transformed"
 1521 into "legal" XML.

1522
 1523 In some cases, XML documents MUST be placed into transport specific "envelopes" before being
 1524 transported. For example, XML data MUST be placed in a MIME envelope when being
 1525 transported via SMTP or HTTP.

1526 **C.4 Conclusion**

1527 The packaging sub-group examined the capabilities of both XML and MIME relative to the list of
 1528 packaging requirements above. It's important to note that neither technology met all of the ebXML
 1529 requirements and in the end it was the packaging sub-groups assessment of which technology
 1530 came closest to meeting ALL of the ebXML requirements that determined which technology
 1531 SHOULD be used.

1532
 1533 MIME was chosen to serve as the ebXML packaging technology, over XML, based on the
 1534 information contained in the following table:

Reason	Requirement(s) Satisfied
There is no formal packaging recommendation within IETF or W3C, based on XML. If ebXML were to choose XML as a packaging technology it would be required to define an XML packaging specification and submit this to IETF or W3C for adoption as a formal standard.	to not reinvent the wheel - re-use where possible [TRPREQ]
XML requires that binary and other types of payload data including XML documents be base64 encoded in order to be encapsulated within a XML root document. Base64 encoding ensures that no illegal XML characters exist within a document and recursive XML documents are "hidden". Base64 encoding imposes a significant processing overhead and results in larger messages, which affect both transmission and processing times. Base64 encoding of binary data is required of MIME content when being transported by SMTP, but this is a transport level requirement, not a requirement imposed by MIME. Binary data can be packaged and transported without alteration when using MIME over HTTP	Minimize intrusion to payload (special encoding or alteration) Low processing overhead

At the time of defining this specification there is no industry standard way to package an encrypted message, or portion of a message, using XML.	All or part of the documents in a message MAY be encrypted prior to sending [TRPREQ]
MIME could be used in conformance within existing IETF recommendations, no additions or changes are initially required to produce a functional envelope.	to not reinvent the wheel - re-use where possible [TRPREQ]

1535 **Appendix D MIME Type discussion**

1536 Three MIME media types were considered to serve as Content-Type for the *ebXML Message*
 1537 *Envelope*:

- 1538 • Multipart/related
- 1539 • Multipart/Mixed
- 1540 • Multipart/form-data
- 1541 •

1542 **The group selected the multipart/related media type to serve as the preferred message**
 1543 **envelope Content-Type.**

1544 *Note:*
 1545 *There was some discussion over the similarities of multipart/related and multipart/mixed, both of*
 1546 *which appear to offer similar capabilities and both could meet stated requirements. However, the*
 1547 *group converged on multipart/related, believing it to be more semantically appropriate for ebXML.*
 1548 *There was significant discussion over whether to support multipart/form-data as an alternate*
 1549 *Content-Type for message-envelope, due to the large installed base of web browsers that*
 1550 *support this Content-Type.*

1551
 1552
 1553 *It was determined that multipart/related was a more generic Content-Type than multipart/form-*
 1554 *data and the multipart/related Content-Type is the preferred Content-Type for ebXML Message*
 1555 *Envelopes. Multipart/form-data Content-Type is typically associated with HTTP/HTML web forms,*
 1556 *whereas multipart/related can be associated with any type of data.*

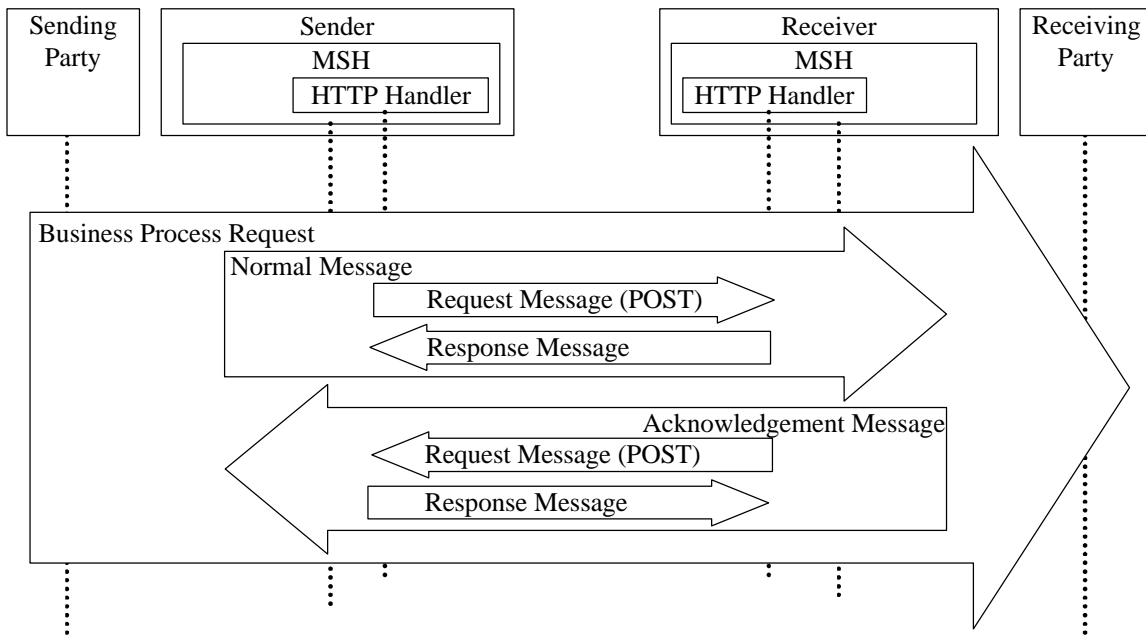
1557
 1558 Additionally, due to limitations in their handling of multipart ebXML payloads it was determined
 1559 that existing web browsers are unable to support the full breadth of functions needed to package
 1560 complex *ebXML Messages* containing multipart payloads. Therefore browser vendors are
 1561 encouraged to add support for the ebXML enveloping standard as specified in this document.

1562 **Appendix E Communication Protocol Interfaces**

1563 The ebXML Message Service messages are carried by Transport Protocols as shown in the
 1564 following sections.

1565 **E.1 HTTP [RFC 2068]**

1566 All ebXML Message Service messages are carried by an HTTP Request Message (POST
 1567 method). The HTTP Response Message to an HTTP Request Message has no entity body.
 1568 The following Figure E.1 shows how a Normal Message and its corresponding Acknowledgement
 1569 Message (when Reliable Messaging is used) are carried using HTTP:
 1570



1571
 1572
 1573
 1574

Figure E.1 HTTP Flow

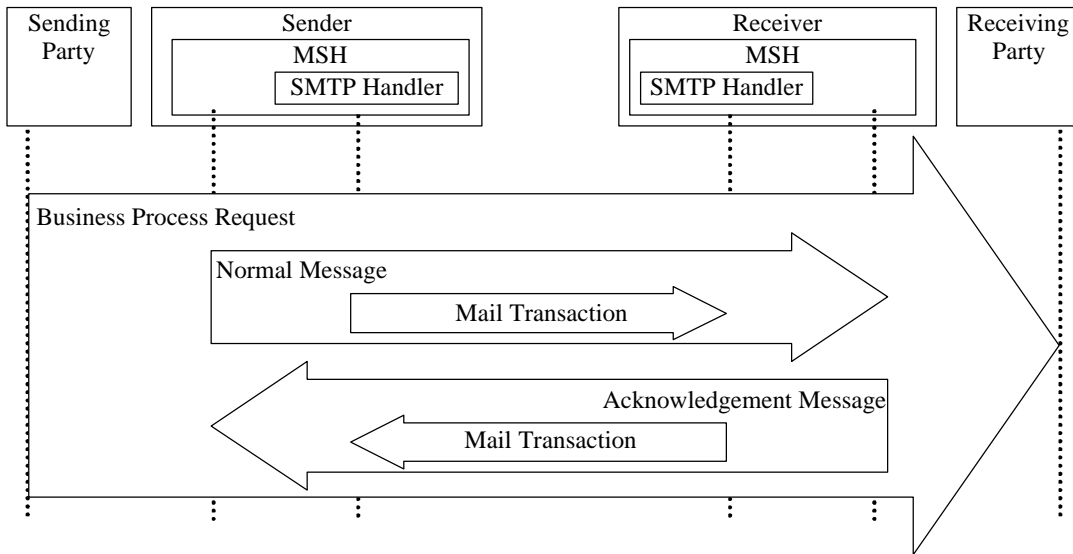
Table E.1 Relationship with HTTP

<i>ebXML Message Service message</i>	<i>HTTP</i>
Normal Message	Request Message (POST method) from Sender to Receiver Response Message to the Request Message has no entity body
Acknowledgement Message	Request Message (POST method) from Receiver to Sender Response Message to the Request Message has no entity body

Error Message	Request Message (POST method) from Receiver to Sender Response Message to the Request Message has no entity body
---------------	---

1575 **E.2 SMTP [RFC 821]**

1576 All ebXML Message Service messages are carried as mail in an SMTP Mail Transaction as
1577 shown in the following Figures.
1578



1579
1580 **Figure E.2 SMTP Flow**

1581 The Mail Transaction follows RFC 821, "SIMPLE MAIL TRANSFER PROTOCOL", as shown in
1582 the following Figure:
1583
1584

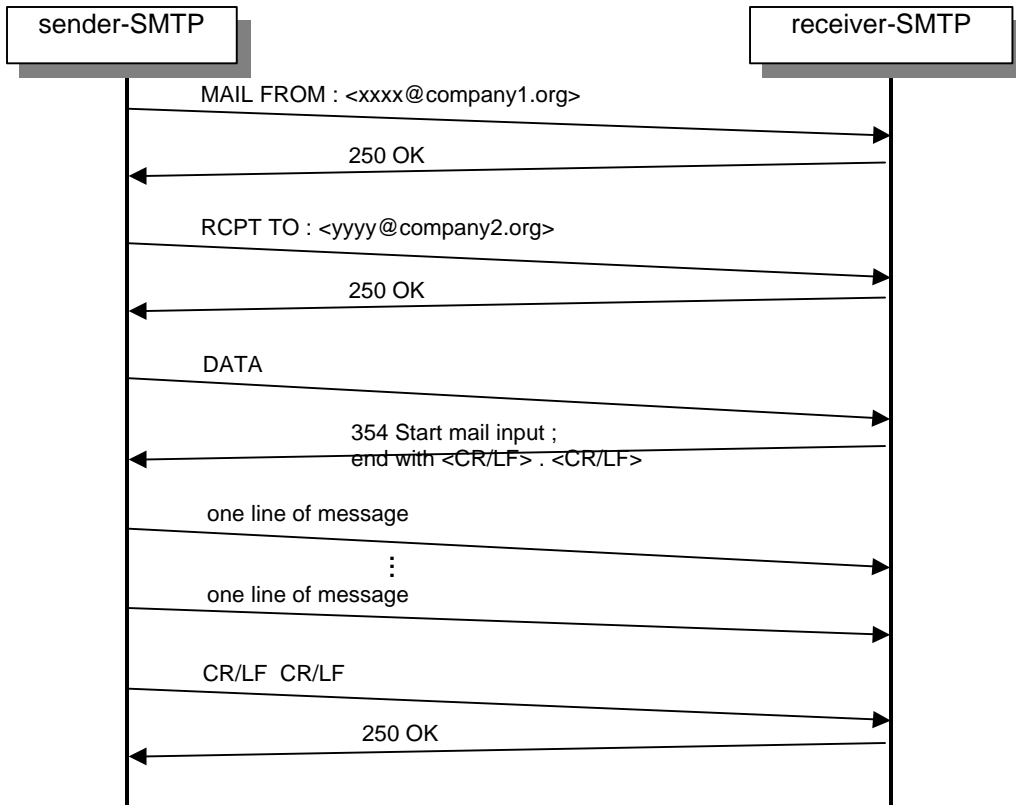


Figure E.3 SMTP Sequence

Table E.2 Relationship with SMTP

<i>ebXML Message Service message</i>	<i>SMTP</i>
Normal Message	Mail Transaction from Sender to Receiver
Acknowledgement Message	Mail Transaction from Receiver to Sender
Error Message	Mail Transaction from Receiver to Sender

E.3 FTP [RFC 959]

This section to be added.

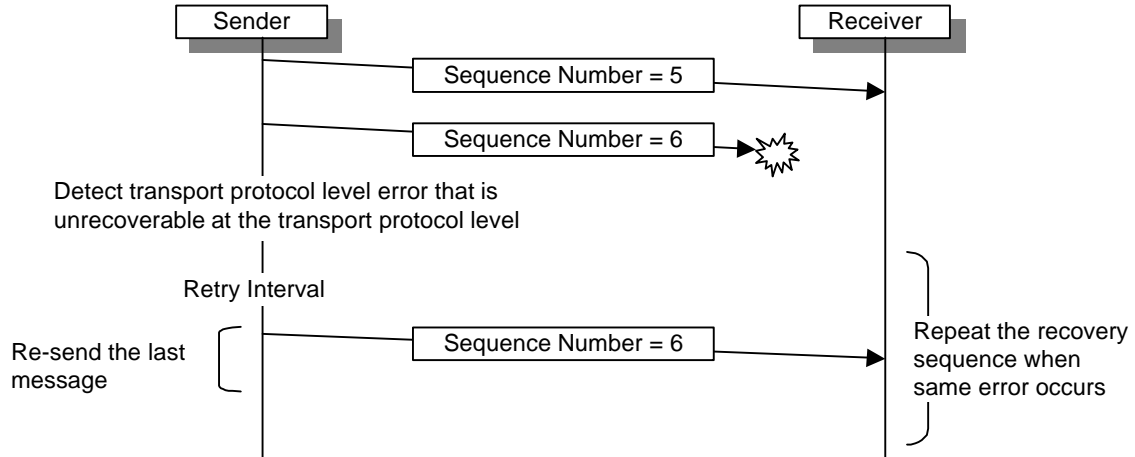
E.4 Communication Protocol Errors during Reliable Messaging

When the Sender or the Receiver detects a transport protocol level error (such as an HTTP, SMTP or FTP error), the appropriate transport recovery handler will execute a recovery sequence. No Reliable Messaging functions are involved in this recovery sequence, since it happens at a lower level.

However, if the Sender detects a transport protocol level error that is unrecoverable at the transport protocol level, the appropriate recovery handler in the Sender will execute a Messaging Message Service Specification v0.8

1599 Service recovery sequence. This recovery sequence SHALL use a retry interval and SHALL re-
1600 send the last message to the Receiver. The format of the re-sent message is exactly the same as
1601 the original message. In the recovery sequence or after the recovery sequence:

- 1602 - If the Sender detects a transport protocol level error again, which is unrecoverable at the
1603 transport protocol level, the recovery handler repeats the recovery sequence for an
1604 implementation-defined number of times.
- 1605 - If the Sender detects or receives another error, the recovery handler executes an
1606 appropriate recovery sequence for the error.
- 1607 - If the Sender receives an Acknowledgment Message, the message transmission is
1608 completed.



1609 **Figure E.4 Recovery Sequence for Communication Protocol Errors**
1610

1611 **Appendix F Detailed list of the Message Services**
1612 **Requirement Phases**

1613
1614 (This section to be added.)
1615

1616 Copyright Statement

1617 Copyright © ebXML 2000. All Rights Reserved.

1618

1619 This document and translations of it may be copied and furnished to others, and derivative works
1620 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
1621 published and distributed, in whole or in part, without restriction of any kind, provided that the
1622 above copyright notice and this paragraph are included on all such copies and derivative works.
1623 However, this document itself may not be modified in any way, such as by removing the copyright
1624 notice or references to the Internet Society or other Internet organizations, except as needed for
1625 the purpose of developing Internet standards in which case the procedures for copyrights defined
1626 in the Internet Standards process must be followed, or as required to translate it into languages
1627 other than English.

1628

1629 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
1630 successors or assigns.

1631

1632 This document and the information contained herein is provided on an
1633 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
1634 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
1635 HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
1636 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

1637