



Creating A Single Global Electronic Market

Message Service Specification

ebXML Transport, Routing & Packaging

Version 0.99

20 April 2001

1 Status of this Document

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

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

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

This version

http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.99.pdf

Latest version

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

Previous version

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

2 ebXML Participants

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

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

3 Table of Contents

1	Status of this Document	2
2	ebXML Participants	3
3	Table of Contents	4
4	Introduction	8
4.1	Summary of Contents of Document	8
4.2	Document Conventions	9
4.3	Audience	9
4.4	Caveats and Assumptions	9
4.5	Related Documents	9
5	Design Objectives	11
6	System Overview	12
6.1	What the Message Service does	12
6.2	Message Service Overview	12
6.3	Use of version attribute	13
7	Packaging Specification	14
7.1	Introduction	14
7.1.1	SOAP Structural Conformance	15
7.2	Message Package	15
7.3	Header Container	15
7.3.1	Content-Type	15
7.3.2	Header Container Example	16
7.4	Payload Container	16
7.4.1	Example of a Payload Container	16
7.5	Additional MIME Parameters	16
7.6	Reporting MIME Errors	17
8	ebXML SOAP Extensions	18
8.1	XML Prolog	18
8.1.1	XML Declaration	18
8.1.2	Encoding Declaration	18
8.2	ebXML SOAP Envelope extensions	18
8.2.1	Namespace pseudo attribute	19
8.2.2	xsi:schemaLocation attribute	19
8.2.3	ebXML SOAP Extensions	20
8.2.4	#wildcard element content	20
8.2.5	id attributes	20
8.3	SOAP Header element	21
8.4	MessageHeader element	21
8.4.1	From and To elements	21
8.4.2	CPAId element	22
8.4.3	ConversationId element	22
8.4.4	Service element	23
8.4.5	Action element	23
8.4.6	MessageData element	23

- 8.4.7 QualityOfServiceInfo element 24
- 8.4.8 SequenceNumber element 25
- 8.4.9 Description element 26
- 8.4.10 version attribute 26
- 8.4.11 SOAP mustUnderstand attribute 26
- 8.4.12 MessageHeader sample 27
- 8.5 TraceHeaderList element 27
 - 8.5.1 SOAP mustUnderstand attribute 27
 - 8.5.2 SOAP actor attribute 27
 - 8.5.3 version attribute 27
 - 8.5.4 TraceHeader element 28
 - 8.5.5 Single Hop TraceHeader Sample 29
 - 8.5.6 Multi-hop TraceHeader Sample 30
- 8.6 Acknowledgment Element 31
 - 8.6.1 Timestamp element 31
 - 8.6.2 From element 31
 - 8.6.3 [XMLDSIG] Reference element 32
 - 8.6.4 SOAP mustUnderstand attribute 32
 - 8.6.5 SOAP actor attribute 32
 - 8.6.6 version attribute 32
 - 8.6.7 Acknowledgement sample 32
- 8.7 Via element 32
 - 8.7.1 SOAP mustUnderstand attribute 33
 - 8.7.2 SOAP actor attribute 33
 - 8.7.3 version attribute 33
 - 8.7.4 syncReply attribute 33
 - 8.7.5 reliableMessagingMethod attribute 33
 - 8.7.6 ackRequested attribute 33
 - 8.7.7 CPAlid element 33
 - 8.7.8 Service and Action elements 34
 - 8.7.9 Sample Via element 34
- 8.8 ErrorList element 34
 - 8.8.1 id attribute 34
 - 8.8.2 SOAP mustUnderstand attribute 34
 - 8.8.3 version attribute 34
 - 8.8.4 highestSeverity attribute 35
 - 8.8.5 Error element 35
 - 8.8.6 Examples 36
 - 8.8.7 errorCode values 36
- 8.9 Signature element 37
- 8.10 SOAP Body Extensions 37
- 8.11 Manifest element 38
 - 8.11.1 id attribute 38
 - 8.11.2 version attribute 38
 - 8.11.3 #wildcard element 38
 - 8.11.4 Reference element 38
 - 8.11.5 References included in a Manifest 39
 - 8.11.6 Manifest Validation 39
 - 8.11.7 Manifest sample 39
- 8.12 StatusRequest Element 40
 - 8.12.1 StatusRequest Sample 40
- 8.13 StatusResponse element 40
 - 8.13.1 RefToMessageId element 40
 - 8.13.2 Timestamp element 40

- 8.13.3 version attribute 40
- 8.13.4 messageStatus attribute 40
- 8.13.5 StatusResponse sample 41
- 8.14 DeliveryReceipt element 41
 - 8.14.1 Timestamp element 41
 - 8.14.2 DigestValue element 41
 - 8.14.3 version attribute 41
 - 8.14.4 DeliveryReceipt sample 41
- 8.15 Combining ebXML SOAP Extension Elements 42
 - 8.15.1 Manifest element 42
 - 8.15.2 MessageHeader element 42
 - 8.15.3 TraceHeaderList element 42
 - 8.15.4 StatusRequest element 42
 - 8.15.5 StatusResponse element 42
 - 8.15.6 ErrorList element 42
 - 8.15.7 Acknowledgment element 42
 - 8.15.8 Signature element 42
 - 8.15.9 Via element 42
- 9 Message Service Handler Services 43**
 - 9.1 Message Status Request Service 43
 - 9.1.1 Message Status Request Message 43
 - 9.1.2 Message Status Response Message 44
 - 9.1.3 Security Considerations 44
 - 9.2 Message Service Handler Ping Service 44
 - 9.2.1 Message Service Handler Ping Message 44
 - 9.2.2 Message Service Handler Pong Message 45
 - 9.2.3 Security Considerations 45
- 10 Reliable Messaging 46**
 - 10.1.1 Persistent Storage and System Failure 46
 - 10.1.2 Methods of Implementing Reliable Messaging 46
 - 10.2 Reliable Messaging Parameters 46
 - 10.2.1 Delivery Semantics 46
 - 10.2.2 mshTimeAccuracy 47
 - 10.2.3 Time To Live 47
 - 10.2.4 reliableMessagingMethod 47
 - 10.2.5 ackRequested 47
 - 10.2.6 retries 48
 - 10.2.7 rretryInterval 48
 - 10.2.8 persistDuration 48
 - 10.3 ebXML Reliable Messaging Protocol 48
 - 10.4 Failed Message Delivery 52
- 11 Error Reporting and Handling 53**
 - 11.1 Definitions 53
 - 11.2 Types of Errors 53
 - 11.3 When to generate Error Messages 53
 - 11.3.1 Security Considerations 54
 - 11.4 Identifying the Error Reporting Location 54
 - 11.5 Service and Action Element Values 54
- 12 Security 55**
 - 12.1 Security and Management 55

- 12.2 Collaboration Protocol Agreement 55
- 12.3 Countermeasure Technologies 55
 - 12.3.1 Persistent Digital Signature 55
 - 12.3.2 Persistent Signed Receipt 57
 - 12.3.3 Non-persistent Authentication..... 57
 - 12.3.4 Non-persistent Integrity 57
 - 12.3.5 Persistent Confidentiality 58
 - 12.3.6 Non-persistent Confidentiality..... 58
 - 12.3.7 Persistent Authorization..... 58
 - 12.3.8 Non-persistent Authorization 58
 - 12.3.9 Trusted Timestamp..... 58
 - 12.3.10 Supported Security Services 58
- 13 References..... 61
 - 13.1 Normative References..... 61
 - 13.2 Non-Normative References 62
- 14 Disclaimer 63
- 15 Contact Information..... 64
- Appendix A ebXML SOAP Extension Elements Schema 66
- Appendix B Communication Protocol Bindings..... 72
 - B.1 Introduction 72
 - B.2 HTTP 72
 - B.2.1 Minimum level of HTTP protocol 72
 - B.2.2 Sending ebXML Service messages over HTTP 72
 - B.2.3 HTTP Response Codes..... 74
 - B.2.4 SOAP Error conditions and Synchronous Exchanges 74
 - B.2.5 Synchronous vs. Asynchronous 74
 - B.2.6 Access Control..... 74
 - B.2.7 Confidentiality and Communication Protocol Level Security 75
 - B.3 SMTP..... 75
 - B.3.1 Minimum level of supported protocols 76
 - B.3.2 Sending ebXML Messages over SMTP 76
 - B.3.3 Response Messages 77
 - B.3.4 Access Control..... 78
 - B.3.5 Confidentiality and Communication Protocol Level Security 78
 - B.3.6 SMTP Model 78
 - B.4 Communication Errors during Reliable Messaging 79
- Copyright Statement 80

1 4 Introduction

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

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

13 4.1 Summary of Contents of Document

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

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

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

21 This specification is organized around the following topics:

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

38 Appendices to this specification cover the following:

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

4.2 Document Conventions

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

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

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

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

4.3 Audience

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

4.4 Caveats and Assumptions

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

4.5 Related Documents

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

- **ebXML Message Services Requirements Specification**[ebMSREQ] – defines the requirements of these Message Services
- **ebXML Technical Architecture**[ebTA] – defines the overall technical architecture for ebXML
- **ebXML Technical Architecture Security Specification**[ebTASEC] – defines the security mechanisms necessary to negate anticipated, selected threats
- **ebXML Collaboration Protocol Profile and Agreement Specification**[ebCPP] - defines how one party can discover and/or agree upon the information that party needs to know about another party prior to sending them a message that complies with this specification

- 88 • **ebXML Registry/Repository Services Specification**[ebRS] – defines a registry service for
- 89 the ebXML environment
- 90

91 **5 Design Objectives**

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

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

103 6 System Overview

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

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

115 6.1 What the Message Service does

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

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

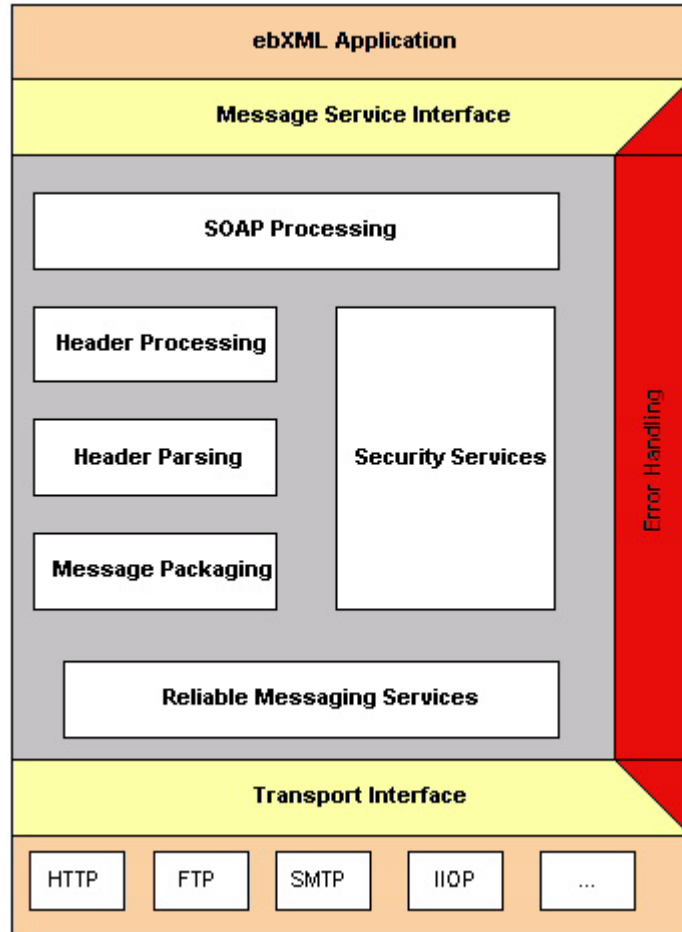
123 6.2 Message Service Overview

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

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

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

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



151
 152 **Figure 6-1 Typical Relationship between ebXML Message Service Handler Components**

153 **6.3 Use of version attribute**

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

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

161

162 7 Packaging Specification

163 7.1 Introduction

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

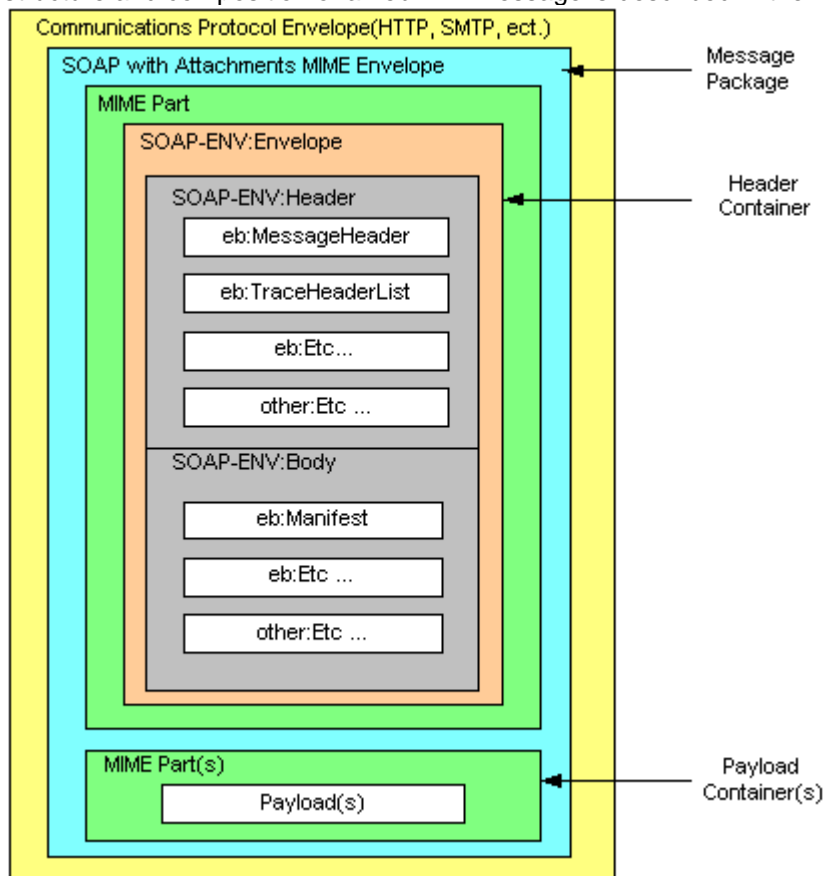
167 There are two logical MIME parts within the *Message Package*:

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

173 The *SOAP Message* is an XML document that consists of the SOAP Envelope element. This is the
 174 root element of the XML document representing the *SOAP Message*. The SOAP Envelope element
 175 consists of the following:

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

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



181
 182 **Figure 7-1 ebXML Message Structure**

183 7.1.1 SOAP Structural Conformance

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

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

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

190 7.2 Message Package

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

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

200
201
202
203
204
205
206

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

207 7.3 Header Container

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

211 7.3.1 Content-Type

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

215
216

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

217 7.3.1.1 charset Attribute

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

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

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

227
228

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

229 **7.3.2 Header Container Example**

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

231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247

```

Content-ID: messagepackage-123@example.com      --- | Header
Content-Type: text/xml;
          charset="UTF-8"
<SOAP-ENV:Envelope                             -- | SOAP Message
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" >
  <SOAP-ENV:Header>
  ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
  ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>                          -- |
---boundaryValue                               --- |
    
```

248 **7.4 Payload Container**

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

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

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

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

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

262 **7.4.1 Example of a Payload Container**

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

264
265
266
267
268
269
270
271
272

```

Content-ID: <domainname.example.com> ----- | ebXML MIME
Content-Type: application/xml               ----- |
<Invoice>                                  ----- | Payload
  <Invoicedata>                             ----- | Payload Container
  ...
  </Invoicedata>
</Invoice>
    
```

273

274 **7.5 Additional MIME Parameters**

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

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

280 **7.6 Reporting MIME Errors**

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

283 8 ebXML SOAP Extensions

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

- 287 • different software components are likely to be used to generate ebXML SOAP extension
288 elements,
- 289 • an ebXML SOAP extension element is not always present or,
- 290 • the data contained in the ebXML SOAP extension element MAY be digitally signed
291 separately from the other ebXML SOAP extension elements.

292 8.1 XML Prolog

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

```
296 Content-Type: text/xml; charset="UTF-8"  
297 <?xml version="1.0" encoding="UTF-8"?>  
298  
299
```

300 8.1.1 XML Declaration

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

305 8.1.2 Encoding Declaration

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

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

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

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

317 8.2 ebXML SOAP Envelope extensions

318 In conformance with the [SOAP] specification, all extension element content MUST be namespace
319 qualified. All of the ebXML SOAP extension element content defined in this specification MUST be
320 namespace qualified to the ebXML SOAP **Envelope** extensions namespace as defined in section
321 8.2.1.

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

325 8.2.1 Namespace pseudo attribute

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

328 8.2.2 xsi:schemaLocation attribute

329 The SOAP namespace:

330
331

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

332 resolves to a schema that conforms to an early Working Draft version of the W3C XMLSchema,
333 specifically identified by the following URI:

334
335

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

336 The W3C XML Schema specification[XMLSchema] has since gone to Candidate Recommendation
337 status, effective October 24, 2000 and more recently to Proposed Recommendation effective March
338 30, 2001. Many, if not most, tool support for schema validation and validating XML parsers available
339 at the time that this specification was written have been designed to support the Candidate
340 Recommendation draft of the XML Schema specification[XMLSchema]. In addition, the ebXML SOAP
341 extension element schema has been defined using the Candidate Recommendation draft of the XML
342 Schema specification[XMLSchema] (see Appendix A).

343 In order to enable validating parsers and various schema validating tools to correctly process and
344 parse ebXML SOAP messages, it has been necessary that the ebXML TR&P team adopt an
345 equivalent, but updated version of the SOAP schema that conforms to the W3C Candidate
346 Recommendation draft of the XML Schema specification[XMLSchema]. ebXML MSH
347 implementations are strongly RECOMMENDED to include the XMLSchema-instance namespace
348 qualified **schemaLocation** attribute in the SOAP Envelope element to indicate to validating parsers
349 the location of the schema document that should be used to validate the document. Failure to include
350 the **schemaLocation** attribute will possibly preclude receiving MSH implementations from being able
351 to validate messages received.

352 For example:

353
354
355
356
357

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

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

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

367
368
369
370
371
372
373
374
375
376
377
378
379

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
    http://ebxml.org/project_teams/transport/envelope.xsd" ...>
  <SOAP-ENV:Header xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
    xsi:schemaLocation="http://www.ebxml.org/namespaces/messageHeader
      http://ebxml.org/project_teams/transport/messageHeaderv0_99.xsd" ...>
    <eb:MessageHeader ...> ...
  </eb:MessageHeader>
</SOAP-ENV:Header>
  <SOAP-ENV:Body xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
    xsi:schemaLocation="http://www.ebxml.org/namespaces/messageHeader
```

```

380     http://ebxml.org/project_teams/transport/messageHeaderv0_99.xsd" ...>
381     <eb:Manifest ...> ...
382     </eb:Manifest>
383     </SOAP-ENV:Body>
384     </SOAP-ENV:Envelope>

```

385 8.2.3 ebXML SOAP Extensions

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

- 387 • SOAP **Header** extensions:
- 388 - **MessageHeader** – a REQUIRED element that contains routing information for the
 - 389 message (To/From, etc.) as well as other context information about the message.
 - 390 - **TraceHeaderList** – an element that contains entries that identifies the Message Service
 - 391 Handler(s) that sent and should receive the message. This element MAY be omitted.
 - 392 - **ErrorList** – an element that contains a list of the errors that are being reported against a
 - 393 previous message. The **ErrorList** element is only used if reporting an error on a
 - 394 previous message. This element MAY be omitted.
 - 395 - **Signature** – an element that contains a digital signature that conforms to [XMLDSIG]
 - 396 that signs data associated with the message. This element MAY be omitted.
 - 397 - **Acknowledgment** – an element that is used by a receiving MSH to acknowledge to the
 - 398 sending MSH that a previous message has been received. This element MAY be
 - 399 omitted.
 - 400 - **Via** – an element that is used to convey information to the next ebXML Message Service
 - 401 Handler that receives the message. This element MAY be omitted.
- 402 • SOAP **Body** extensions:
- 403 - **Manifest** – an element that points to any data present either in the *Payload Container* or
 - 404 elsewhere, e.g. on the web. This element MAY be omitted.
 - 405 - **DeliveryReceipt** – an element that is used by the *To Party* that received a message, to
 - 406 let the *From Party* that sent the message know that the message was received. This
 - 407 element MAY be omitted.
 - 408 - **StatusRequest** – an element that is used to identify a message whose status is being
 - 409 requested. This element MAY be omitted.
 - 410 - **StatusResponse** – an element that is used by a MSH when responding to a request on
 - 411 the status of a message that was previously received. This element MAY be omitted.

412 8.2.4 #wildcard element content

413 Some ebXML SOAP extension elements allow for foreign namespace-qualified element content to be

414 added to provide for extensibility. The extension element content MUST be namespace-qualified in

415 accordance with [XMLNamespaces] and MUST belong to a foreign namespace. A foreign

416 namespace is one that is NOT <http://www.ebxml.org/namespaces/messageHeader>.

417 Any foreign namespace-qualified element added SHOULD include the SOAP **mustUnderstand**

418 attribute. If the SOAP **mustUnderstand** attribute is NOT present, the default value implied is '0'

419 (false). If an implementation of the MSH does not recognize the namespace of the element and the

420 value of the SOAP **mustUnderstand** attribute is '1' (true), the MSH SHALL report an error (see

421 section 11) with **errorCode** set to **NotSupported** and **severity** set to **error**. If the value of the

422 **mustUnderstand** attribute is '0' or if the **mustUnderstand** attribute is not present, then an

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

424 8.2.5 id attributes

425 Each of the ebXML SOAP extension elements listed above has an optional **id** attribute which is an

426 XML ID that MAY be added to provide for the ability to uniquely identify the element within the *SOAP*

427 *Message*. This MAY be used when applying a digital signature to the ebXML *SOAP Message* as

428 individual ebXML SOAP extension elements can be targeted for inclusion or exclusion by specifying a

429 URI of "#<idvalue>" in the **Reference** element.

430 8.3 SOAP Header element

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

```
434
435 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" ...>
436   <SOAP-ENV:Header>...</SOAP-ENV:Header>
437   <SOAP-ENV:Body>...</SOAP-ENV:Body>
438 </SOAP-ENV:Envelope>
```

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

441 8.4 MessageHeader element

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

444 The **MessageHeader** element is a composite element comprised of the following ten subordinate
 445 elements:

- 446 • **From**
- 447 • **To**
- 448 • **CPAId**
- 449 • **ConversationId**
- 450 • **Service**
- 451 • **Action**
- 452 • **MessageData**
- 453 • **QualityOfServiceInfo**
- 454 • **SequenceNumber**
- 455 • **Description**

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

- 457 • SOAP **mustUnderstand**
- 458 • **Version**

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

460 8.4.1 From and To elements

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

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

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

469 This mechanism is particularly useful when transport of a message between the parties may involve
 470 multiple intermediaries (see Sections 8.5.6, Multi-hop TraceHeader Sample and 10.3, ebXML
 471 Reliable Messaging Protocol). More generally, the *From Party* should provide identification in all
 472 domains it knows in support of intermediaries and destinations that may give preference to particular
 473 identification systems.

474 8.4.1.1 PartyID element

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

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

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

```
485
486 <eb:From>
487   <eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
488   <eb:PartyId eb:type="SCAC">RDWY</eb:PartyId>
489 </eb:From>
490 <eb:To>
491   <eb:PartyId>mailto:joe@example.com</eb:PartyId>
492 </eb:To>
```

493 8.4.2 CPAId element

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

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

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

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

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

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

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

513 8.4.3 ConversationId element

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

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

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

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

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

527 8.4.4 Service element

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

- 530 • a standards organization, or
- 531 • an individual or enterprise

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

533 An example of the *Service* element follows:

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

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

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

538 8.4.4.1 type attribute

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

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

545 8.4.5 Action element

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

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

550 8.4.6 MessageData element

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

- 553 • *MessageId*
- 554 • *Timestamp*
- 555 • *RefToMessageId*
- 556 • *TimeToLive*

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

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

563 8.4.6.1 MessageId element

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

566 8.4.6.2 Timestamp element

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

569 8.4.6.3 RefToMessageId element

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

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

575 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value MUST
576 be the **MessageId** value of the ebXML Message being acknowledged. See also sections 8.13.5 and
577 10.

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

580 8.4.6.4 TimeToLive element

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

584 8.4.7 QualityOfServiceInfo element

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

- 587 • **deliverySemantics**
- 588 • **messageOrderSemantics**
- 589 • **deliveryReceiptRequested**

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

594 8.4.7.1 deliveryReceiptRequested attribute

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

598 The **deliveryReceiptRequested** element indicates that the *To Party* has received the message. This
599 is separate from a Reliable Messaging acknowledgment message, which only indicates that a
600 receiving MSH has successfully received a message.

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

603 Valid values for **deliveryReceiptRequested** are:

- 604 • **Unsigned** - requests that an unsigned Delivery Receipt is requested
- 605 • **Signed** - requests that a signed Delivery Receipt is requested, or
- 606 • **None** - indicates that no Delivery Receipt is requested.

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

608 When a *To Party* receives a message with **deliveryReceiptRequested** attribute set to **Signed** or
 609 **Unsigned** then it should verify that it is able to support the type of Delivery Receipt requested.
 610 If the *To Party* can produce the Delivery Receipt of the type requested, then it MUST return to the
 611 *From Party* a message containing a **DeliveryReceipt** element.
 612 If the *To Party* cannot return a Delivery Receipt of the type requested then it MUST report the error to
 613 the *From Party* using an **errorCode** of **NotSupported** and a **severity** of **Error**.

614 An example of **deliveryReceiptRequested** follows:

```
615
616 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
617     eb:messageOrderSemantics="Guaranteed"
618     eb:deliveryReceiptRequested="Unsigned" />
```

619 **8.4.7.2 messageOrderSemantics attribute**

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

- 622 • **Guaranteed**. The messages are passed to the receiving application in the order that the
 623 sending application specified.
- 624 • **NotGuaranteed** The messages may be passed to the receiving application in different order
 625 from the order the sending application specified.

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

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

633 If **deliverySemantics** is not **OnceAndOnlyOnce** and **messageOrderSemantics** is set to
 634 **NotGuaranteed** then report the error to the *From Party* with an **errorCode** of **Inconsistent** and a
 635 **severity** of **Error** (see sections 10 and 11).

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

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

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

```
644
645 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
646     eb:messageOrderSemantics="Guaranteed" />
```

647 **8.4.8 SequenceNumber element**

648 The **SequenceNumber** element indicates the sequence in which messages MUST be processed by
 649 a receiving MSH. The **SequenceNumber** is unique within the **ConversationId** and MSH. The *From*
 650 *Party* MSH and the *To Party* MSH each set an independent **SequenceNumber** as the sending MSH
 651 within the **ConversationID**. It is set to zero on the first message from that MSH for a conversation
 652 and then incremented by one for each subsequent message sent.

653 The **SequenceNumber** element MUST appear only when **deliverySemantics** has a value of
 654 **OnceAndOnlyOnce** and **messageOrderSemantics** has a value of **Guaranteed**. If this criterion is

655 not met, an error MUST be reported to the From Party MSH with an **errorCode** of **Inconsistent** and
656 a **severity** of **Error**.

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

661 If the implementation defined limit for saved out-of-sequence messages is reached, then the receiving
662 MSH MUST indicate a delivery failure to the sending MSH with **errorCode** set to **DeliveryFailure** and
663 **severity** set to **Error** (see section 11).

664 The **SequenceNumber** element is an integer value that is incremented by the sending MSH (e.g. 0,
665 1, 2, 3, 4...) for each application-prepared message sent by that MSH within the **ConversationId**. The
666 next value of 99999999 in the increment is "0". The value of **SequenceNumber** consists of ASCII
667 numerals in the range 0-99999999. In following cases, **SequenceNumber** takes the value "0":

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

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

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

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

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

682
683

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

684 8.4.9 Description element

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

690 8.4.10 version attribute

691 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Header*
692 Specification to which the ebXML SOAP extensions conform. Its purpose is to provide future
693 versioning capabilities. The value of the **version** attribute MUST be "1.0". Future versions of this
694 specification SHALL require other values of this attribute. The **version** attribute MUST be
695 namespace qualified for the ebXML SOAP **Envelope** extensions namespace defined above in
696 section 8.2.1.

697 8.4.11 SOAP mustUnderstand attribute

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

702 8.4.12 MessageHeader sample

703 The following fragment demonstrates the structure of the *MessageHeader* element within the SOAP
704 Header:

```
705 <eb:MessageHeader id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
706   <eb:From><eb:PartyId>uri:example.com</eb:PartyId></eb:From>
707   <eb:To eb:type="someType">
708     <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
709   </eb:To>
710   <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
711   <eb:ConversationId>987654321</eb:ConversationId>
712   <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
713   <eb:Action>NewPurchaseOrder</eb:Action>
714   <eb:MessageData>
715     <eb:MessageId>mid:UUID-2</eb:MessageId>
716     <eb:Timestamp>2000-07-25T12:19:05Z</eb:Timestamp>
717     <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
718   </eb:MessageData>
719   <eb:QualityOfServiceInfo
720     eb:deliverySemantics="OnceAndOnlyOnce"
721     eb:deliveryReceiptRequested="Signed" />
722 </eb:MessageHeader>
```

724 8.5 TraceHeaderList element

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

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

- 729 • the message is being sent over a single hop (see section 8.5.5), and
- 730 • the message is not being sent reliably (see section 10)

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

- 732 • SOAP *mustUnderstand*
- 733 • SOAP *actor* attribute with the value "<http://schemas.xmlsoap.org/soap/actor/next>"
- 734 • *Version*

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

736 8.5.1 SOAP mustUnderstand attribute

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

741 8.5.2 SOAP actor attribute

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

749 8.5.3 version attribute

750 The REQUIRED *version* attribute indicates the version of the *ebXML Message Service Header*
751 *Specification* to which the ebXML *SOAP Header* extensions conform. Its purpose is to provide future
752 versioning capabilities. The value of the *version* attribute MUST be "0.99". Future versions of this

753 specification SHALL require other values of this attribute. The version attribute MUST be namespace
754 qualified for the ebXML SOAP **Envelope** extensionsnamespace defined above.

755 **8.5.4 TraceHeader element**

756 The **TraceHeader** element contains information about a single transmission of a message between
757 two instances of a MSH. If a message traverses multiple hops by passing through one or more
758 intermediate MSH nodes as it travels between the *From Party* MSH and the *To Party* MSH, then each
759 transmission over each successive “hop” results in the addition of a new **TraceHeader** element by
760 the sending MSH.

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

- 762 • **Sender**
- 763 • **Receiver**
- 764 • **Timestamp**
- 765 • **#wildcard**

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

767 **8.5.4.1 Sender element**

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

- 769 • **PartyId**
- 770 • **Location**

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

774 **8.5.4.1.1 PartyId element**

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

778 **8.5.4.1.2 Location element**

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

- 782 • responds to an earlier message
- 783 • acknowledges an earlier message
- 784 • reports an error in an earlier message.

785 **8.5.4.2 Receiver element**

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

- 787 • **PartyId**
- 788 • **Location**

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

792 The descendant elements of the **Receiver** element (**PartyId** and **Location**) are implemented in the
793 same manner as the Sender element (see sections 8.5.4.1.1 and 8.5.4.1.2).

794 **8.5.4.3 Timestamp element**

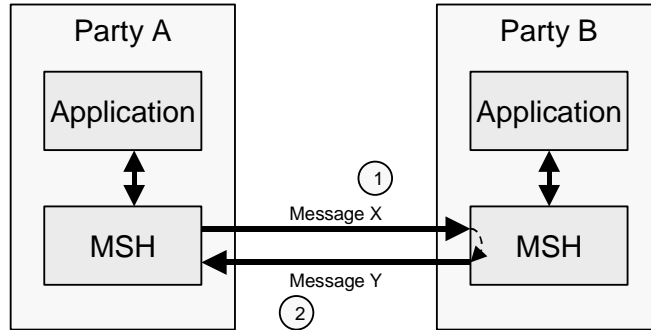
795 The **Timestamp** element is the time the individual **TraceHeader** was created. It is in the same format
 796 as in the **Timestamp** element in the **MessageData** element (section 8.4.6.2).

797 **8.5.4.4 #wildcard element**

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

799 **8.5.5 Single Hop TraceHeader Sample**

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



801

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

803 The content of the corresponding messages could include:

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

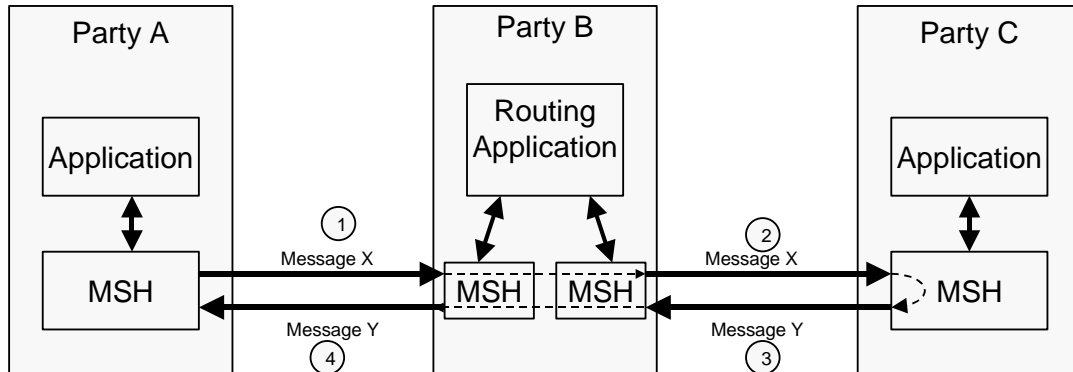
805

```

806 <eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
807   <eb:From>
808     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
809   </eb:From>
810   <eb:To>
811     <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
812   </eb:To>
813   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
814   ...
815   <eb:MessageData>
816     <eb:MessageId>29dmridj103kvna</eb:MessageId>
817     ...
818   </eb:MessageData>
819   ...
820 </eb:MessageHeader>
821
822 <eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
823   <eb:TraceHeader>
824     <eb:Sender>
825       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
826       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
827     </eb:Sender>
828     <eb:Receiver>
829       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
830       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
831     </eb:Receiver>
832     <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
833   </eb:TraceHeader>
834 </eb:TraceHeaderList>
    
```

835 **8.5.6 Multi-hop TraceHeader Sample**

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



838

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

840 The content of the corresponding messages could include:

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

842

```
843 <eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
844   <eb:From>
845     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
846   </eb:From>
847   <eb:To>
848     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
849   </eb:To>
850   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
851   ...
852   <eb:MessageData>
853     <eb:MessageId>29dmridj103kvna</eb:MessageId>
854     ...
855   </eb:MessageData>
856   ...
857 </eb:MessageHeader>
858
859 <eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1"
860   SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
861   <eb:TraceHeader>
862     <eb:Sender>
863       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
864       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
865     </eb:Sender>
866     <eb:Receiver>
867       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
868     </eb:Receiver>
869     <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
870   </eb:TraceHeader>
871 </eb:TraceHeaderList>
```

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

```
873 <eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
874   <eb:From>
875     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
876   </eb:From>
877   <eb:To>
878     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
879   </eb:To>
880   <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
881   ...
882   <eb:MessageData>
883     <eb:MessageId>29dmridj103kvna</eb:MessageId>
884     ...
```

```

885 </eb:MessageData>
886   ...
887 </eb:MessageHeader>
888
889 <eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1"
890   SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
891   <eb:TraceHeader>
892     <eb:Sender>
893       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
894       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
895     </eb:Sender>
896     <eb:Receiver>
897       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
898       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
899     </eb:Receiver>
900     <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
901   </eb:TraceHeader>
902   <eb:TraceHeader>
903     <eb:Sender>
904       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
905       <eb:Location>http://PartyB.com/PartyAMsh</eb:Location>
906     </eb:Sender>
907     <eb:Receiver>
908       <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
909       <eb:Location>http://PartyC.com/PartyBMsh</eb:Location>
910     </eb:Receiver>
911     <eb:Timestamp>2000-12-16T21:19:45Z</eb:Timestamp>
912   </eb:TraceHeader>
913 </eb:TraceHeaderList>

```

914 8.6 Acknowledgment Element

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

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

- 920 • a **Timestamp** element
- 921 • a **From** element
- 922 • zero or more [XMLDSIG] **Reference** element(s)
- 923 • a REQUIRED SOAP **mustUnderstand** attribute
- 924 • a REQUIRED SOAP **actor** attribute
- 925 • a REQUIRED **version** attribute
- 926 • an **id** attribute (See section 8.2.5 for details)

927 8.6.1 Timestamp element

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

931 8.6.2 From element

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

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

937 **8.6.3 [XMLDSIG] Reference element**

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

943 **8.6.4 SOAP mustUnderstand attribute**

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

948 **8.6.5 SOAP actor attribute**

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

953 **8.6.6 version attribute**

954 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service*
 955 *HeaderSpecification* to which the ebXML **SOAP Header** extensions conform. Its purpose is to provide
 956 future versioning capabilities. The value of the **version** attribute MUST be "1.0". Future versions of
 957 this specification SHALL require other values of this attribute. The **version** attribute MUST be
 958 namespace qualified for the ebXML **SOAP Envelope** extensions namespace defined above.

959 **8.6.7 Acknowledgement sample**

960 An example of the **Acknowledgment** element is given below:

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

969 **8.7 Via element**

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

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

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

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

- 976 • **syncReply** attribute
- 977 • **reliableMessagingMethod** attribute
- 978 • **ackRequested** attribute
- 979 • **CPAId** element

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

- 981 • **Service** element
- 982 • **Action** element

983 8.7.1 SOAP *mustUnderstand* attribute

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

990 8.7.2 SOAP *actor* attribute

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

995 8.7.3 *version* attribute

996 The REQUIRED *version* attribute indicates the version of the *ebXML Message Service* Header
997 Specification to which the ebXML SOAP *Header* extensions conform. Its purpose is to provide future
998 versioning capabilities. The value of the *version* attribute MUST be "1.0". Future versions of this
999 specification SHALL require other values of this attribute. The *version* attribute MUST be namespace
1000 qualified for the ebXML SOAP *Envelope* extensions namespace defined above.

1001 8.7.4 *syncReply* attribute

1002 The *syncReply* attribute is used only if the data communication protocol is *synchronous* (e.g. HTTP).
1003 It is an [XML Schema] boolean. If the communication protocol is not synchronous, then the value of
1004 *syncReply* is ignored. If the *syncReply* attribute is not present, it is semantically equivalent to its
1005 presence with a value of "false". If the *syncReply* attribute is present with a value of *true*, the MSH
1006 must return the response from the application or business process in the payload of the synchronous
1007 reply message. See also the description of *syncReply* in the [ebCPP] specification.

1008 8.7.5 *reliableMessagingMethod* attribute

1009 The *reliableMessagingMethod* attribute is an enumeration that SHALL have one of the following
1010 values:

- 1011 • *ebXML*
- 1012 • *Transport*

1013 The default implied value for this attribute is *ebXML*.

1014 8.7.6 *ackRequested* attribute

1015 The *ackRequested* attribute is an enumeration that SHALL have one of the following values:

- 1016 • *Signed*
- 1017 • *UnSigned*
- 1018 • *None*

1019 The default implied value for this attribute is *None*. This attribute is used to indicate to the receiving
1020 MSH whether an acknowledgment message is expected, and if so, whether the acknowledgment
1021 message should be signed by the receiving MSH. Refer to section 10.2.5 for a complete discussion
1022 as to the use of this attribute.

1023 8.7.7 *CPAId* element

1024 The *CPAId* element is a string that identifies the parameters that govern the exchange of messages
1025 between two MSH instances. It has the same meaning as the *CPAId* in the *MessageHeader* except
1026 that the parameters identified by the *CPAId* apply just to the exchange of messages between the two
1027 MSH instances rather than between the *Parties* identified in the *To* and *From* elements of the

1028 **MessageHeader** (section 8.4.2). This allows different parameters, transport protocols, etc, to be used
1029 on different hops when a message is passed through intermediaries.

1030 If the **CPAId** element is present, the identified parameter values SHOULD be used instead of the
1031 values identified by the **CPAId** in the **MessageHeader** element.

1032 8.7.8 Service and Action elements

1033 The **Service** and **Action** elements have the same meaning as the **Service** and **Action** elements in
1034 the **MessageHeader** element (see sections 8.4.4 and 8.4.5) except that they are interpreted and
1035 acted on by the next MSH whether or not the MSH is operated by the *To Party*.

1036 The designer of the service or business process that is using the *ebXML Message Service* defines
1037 the values used for **Service** and **Action**.

1038 The **Service** and **Action** elements are OPTIONAL. However, if the **Service** element is present then
1039 the **Action** element MUST also be present and vice versa.

1040 8.7.9 Sample Via element

1041 The following is a sample **Via** element.

```
1042
1043 <eb:Via SOAP-ENV:mustUnderstand="1"
1044     SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
1045     eb:version="1.0"
1046     eb:syncReply="false">
1047     <eb:CPAId>yaddaydda</eb:CPAId>
1048     <eb:Service>Proxy</eb:Service>
1049     <eb:Action>LogActivity</eb:Action>
1050 </eb:Via>
```

1051 8.8 ErrorList element

1052 The existence of an **ErrorList** element within the SOAP **Header** element indicates that the message
1053 that is identified by the **RefToMessageld** in the **MessageHeader** element has an error.

1054 The **ErrorList** element consists of one or more **Error** elements and the following attributes:

- 1055 • **id** attribute
- 1056 • SOAP **mustUnderstand** attribute
- 1057 • **version** attribute
- 1058 • **highestSeverity** attribute

1059 If there are no errors to be reported then the **ErrorList** element MUST NOT be present.

1060 8.8.1 id attribute

1061 The **id** attribute uniquely identifies the **ErrorList** element within the document.

1062 8.8.2 SOAP mustUnderstand attribute

1063 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP namespace
1064 (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the **ErrorList** element
1065 MUST be understood by a receiving process or else the message MUST be rejected in accordance
1066 with [SOAP]. This attribute MUST have a value of '1' (true).

1067 8.8.3 version attribute

1068 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Specification*
1069 to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide for future versioning
1070 capabilities. The value of the **version** attribute MUST be "1.0". Future versions of this specification
1071 SHALL require other values of this attribute. The version attribute MUST be namespace qualified for
1072 the ebXML SOAP **Envelope** extensions namespace defined above.

1073 8.8.4 highestSeverity attribute

1074 The **highestSeverity** attribute contains the highest severity of any of the **Error** elements. Specifically,
1075 if any of the **Error** elements have a **severity** of **Error** then **highestSeverity** must be set to **Error**,
1076 otherwise set **highestSeverity** to **Warning**.

1077 8.8.5 Error element

1078 An **Error** element consists of the following attributes:

- 1079 • **codeContext**
- 1080 • **errorCode**
- 1081 • **severity**
- 1082 • **location**
- 1083 • **xml:lang**
- 1084 • **id** (See section 8.2.5 for details)

1085 The content of the **Error** element contains an error message.

1086 8.8.5.1 codeContext attribute

1087 The REQUIRED **codeContext** attribute identifies the namespace or scheme for the **errorCodes**. It
1088 MUST be a URI. Its default value is **http://www.ebxml.org/messageServiceErrors**. If it does not
1089 have the default value, then it indicates that an implementation of this specification has used its own
1090 **errorCodes**.

1091 Use of non-ebXML values for **errorCodes** is NOT RECOMMENDED. In addition, an implementation
1092 of this specification MUST NOT use its own **errorCodes** if an existing **errorCode** as defined in this
1093 section has the same or very similar meaning.

1094 8.8.5.2 errorCode attribute

1095 The REQUIRED **errorCode** attribute indicates the nature of the error in the message in error. Valid
1096 values for the **errorCode** and a description of the code's meaning are given in sections 8.8.7.1 and
1097 8.8.7.2

1098 8.8.5.3 severity attribute

1099 The REQUIRED **severity** attribute indicates the severity of the error. Valid values are:

- 1100 • **Warning** - This indicates that although there is an error, other messages in the conversation
1101 will still be generated in the normal way.
- 1102 • **Error** - This indicates that there is an unrecoverable error in the message and no further
1103 messages will be generated as part of the conversation.

1104 8.8.5.4 location attribute

1105 The **location** attribute points to the part of the message that is in error.

1106 If an error exists in an ebXML element and the element is "well formed" (see [XML]), then the content
1107 of the **location** attribute MUST be an [XPointer].

1108 If the error is associated with the MIME envelope that wraps the SOAP envelope and the ebXML
1109 Payload, then **location** contains the `content-id` of the MIME part that is in error, in the format
1110 `cid:23912480wsr`, where the text after the ":" is the value of the MIME part's `content-id`.

1111 8.8.5.5 Error element Content

1112 The content of the error message provides a narrative description of the error in the language defined
1113 by the **xml:lang** attribute. Typically, it will be the message generated by the XML parser or other

- 1114 software that is validating the message. This means that the content is defined by the
- 1115 vendor/developer of the software that generated the **Error** element.
- 1116 The *xml:lang* attribute must comply with the rules for identifying languages specified in [XML].
- 1117 The content of the **Error** element can be empty.

1118 **8.8.6 Examples**

1119 An example of an **ErrorList** element is given below.
1120

```

1121 <eb:ErrorList eb:id='3490sdo9', eb:highestSeverity="error" eb:version="1.0"
1122 SOAP-ENV:mustUnderstand="1">
1123   <eb:Error eb:errorCode='SecurityFailure' eb:severity="Error"
1124     eb:location='URI_of_ds:Signature_goes_here' xml:lang="us-en">
1125     Validation of signature failed </eb:Error>
1126   <eb:Error ... </eb:Error>
1127 </eb:ErrorList>
    
```

1128 **8.8.7 errorCode values**

1129 This section describes the values for the **errorCode** element (see section 8.8.5.2) used in a *message*
1130 *reporting an error*. They are described in a table with three headings:

- 1131 • the first column contains the value to be used as an **errorCode**, e.g. **SecurityFailure**
- 1132 • the second column contains a "Short Description" of the **errorCode**. Note: this narrative MUST
1133 NOT be used in the content of the **Error** element.
- 1134 • the third column contains a "Long Description" that provides an explanation of the meaning of
1135 the error and provides guidance on when the particular **errorCode** should be used.

1136 **8.8.7.1 Reporting Errors in the ebXML Elements**

1137 The following list contains error codes that can be associated with ebXML elements:
1138

Error Code	Short Description	Long Description
ValueNotRecognized	Element content or attribute value not recognized.	Although the document is well formed and valid, the element/attribute contains a value that could not be recognized and therefore could not be used by the <i>ebXML Message Service</i> .
NotSupported	Element or attribute not supported	Although the document is well formed and valid, an element or attribute is present that is consistent with the rules and constraints contained in this specification, but is not supported by the <i>ebXML Message Service</i> processing the message.
Inconsistent	Element content or attribute value inconsistent with other elements or attributes.	Although the document is well formed and valid, according to the rules and constraints contained in this specification the content of an element or attribute is inconsistent with the content of other elements or their attributes.
OtherXml	Other error in an element content or attribute value.	Although the document is well formed and valid, the element content or attribute value contains values that do not conform to the rules and constraints contained in this specification and is not covered by other error codes. The content of the Error element should be used to indicate the nature of the problem.

1139 **8.8.7.2 Non-XML Document Errors**

1140 The following are error codes that identify errors not associated with the ebXML elements:

1141

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

1142 **8.9 Signature element**

1143 An ebXML Message may be digitally signed to provide security countermeasures. Zero or more
 1144 **Signature** elements, belonging to the [XMLDSIG] defined namespace MAY be present in the **SOAP**
 1145 **Header**. The **Signature** element MUST be namespace qualified in accordance with [XMLDSIG]. The
 1146 structure and content of the **Signature** element MUST conform to the [XMLDSIG] specification. If
 1147 there is more than one **Signature** element contained within the **SOAP Header**, the first MUST
 1148 represent the digital signature of the ebXML Message as signed by the *From Party* MSH in
 1149 conformance with section 12. Additional **Signature** elements MAY be present, but their purpose is
 1150 undefined by this specification.

1151 Refer to section 12 for a detailed discussion on how to construct the **Signature** element when
 1152 digitally signing an ebXML Message.

1153 **8.10 SOAP Body Extensions**

1154 The **SOAP Body** element is the second child element of the **SOAP Envelope** element. It MUST have
 1155 a namespace qualifier that matches the **SOAP Envelope** namespace declaration for the namespace
 1156 "<http://schemas.xmlsoap.org/soap/envelope/>". For example:

1157
 1158
 1159
 1160
 1161

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

1162 The **SOAP Body** element contains the ebXML **SOAP Body** extension element content as follows:

- 1163 • **Manifest** element
- 1164 • **StatusResponse** element
- 1165 • **DeliveryReceipt** element
- 1166 • **StatusRequest** element

1167 Each is defined in the following sections.

1168 8.11 Manifest element

1169 The **Manifest** element is a composite element consisting of one or more **Reference** elements. Each
1170 **Reference** element identifies data associated with the message, whether included as part of the
1171 message as payload document(s) contained in a *Payload Container*, or remote resources accessible
1172 via a URL. It is RECOMMENDED that no payload data be present in the SOAP-ENV:Body. The
1173 purpose of the **Manifest** is as follows:

- 1174 • to make it easier to directly extract a particular payload associated with this ebXML Message,
- 1175 • to allow an application to determine whether it can process the payload without having to
1176 parse it.

1177 The **Manifest** element is comprised of the following attributes and elements, each of which is
1178 described below:

- 1179 • an **id** attribute
- 1180 • a REQUIRED **version** attribute
- 1181 • one or more **Reference** elements
- 1182 • **#wildcard**

1183 8.11.1 id attribute

1184 The **Manifest** element MUST have an **id** attribute that is an XML ID.

1185 8.11.2 version attribute

1186 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Specification*
1187 to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide future versioning
1188 capabilities. The value of the **version** attribute MUST be "1.0". Future versions of this specification
1189 SHALL require other values of this attribute. The version attribute MUST be namespace qualified for
1190 the ebXML SOAP **Envelope** extensions namespace defined above.

1191 8.11.3 #wildcard element

1192 Refer to section 8.2.4 for discussion of #wildcard element handling.

1193 8.11.4 Reference element

1194 The **Reference** element is a composite element consisting of the following subordinate elements:

- 1195 • **Schema** - information about the schema(s) that define the instance document identified in the
1196 parent **Reference** element
- 1197 • **Description** - a textual description of the payload object referenced by the parent **Reference**
1198 element
- 1199 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

1200 The **Reference** element itself is an [XLINK] simple link. XLINK is presently a Candidate
1201 Recommendation (CR) of the W3C. It should be noted that the use of XLINK in this context is chosen
1202 solely for the purpose of providing a concise vocabulary for describing an association. Use of an
1203 XLINK processor or engine is NOT REQUIRED, but MAY prove useful in certain implementations.

1204 The **Reference** element has the following attribute content in addition to the element content
1205 described above:

- 1206 • **id** - an XML ID for the **Reference** element,
- 1207 • **xlink:type** - this attribute defines the element as being an XLINK simple link. It has a fixed
1208 value of 'simple',
- 1209 • **xlink:href** - this REQUIRED attribute has a value that is the URI of the payload object
1210 referenced. It SHALL conform to the [XLINK] specification criteria for a simple link.
- 1211 • **xlink:role** - this attribute identifies some resource that describes the payload object or its
1212 purpose. If present, then it SHALL have a value that is a valid URI in accordance with the
1213 [XLINK] specification,

- 1214 • Any other namespace-qualified attribute MAY be present. A receiving MSH MAY choose to
1215 ignore any foreign namespace attributes other than those defined above.

1216 8.11.4.1 Schema element

1217 If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema, DTD,
1218 or a database schema), then the **Schema** element SHOULD be present as a child of the **Reference**
1219 element. It provides a means of identifying the schema and its version defining the payload object
1220 identified by the parent **Reference** element. The **Schema** element contains the following attributes:

- 1221 • **location** - the REQUIRED URI of the schema
1222 • **version** – a version identifier of the schema

1223 8.11.4.2 Description element

1224 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a
1225 textual description of the payload object referenced by the parent **Reference** element. The language
1226 of the description is defined by a REQUIRED **xml:lang** attribute. The **xml:lang** attribute MUST
1227 comply with the rules for identifying languages specified in [XML]. This element is provided to allow a
1228 human readable description of the payload object identified by the parent **Reference** element. If
1229 multiple **Description** elements are present, each SHOULD have a unique **xml:lang** attribute value.
1230 An example of a **Description** element follows.

1231
1232

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

1233 8.11.4.3 #wildcard element

1234 Refer to section 8.2.4 for discussion of #wildcard element handling.

1235 8.11.5 References included in a Manifest

1236 The designer of the business process or information exchange that is using ebXML Messaging
1237 decides what payload data is referenced by the *Manifest* and the values to be used for **xlink:role**.

1238 8.11.6 Manifest Validation

1239 If an **xlink:href** attribute contains a URI that is a content id (URI scheme "cid") then a MIME part
1240 with that `content-id` MUST be present in the *Payload Container* of the message. If it is not, then
1241 the error SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem** and a **severity**
1242 of **Error**.

1243 If an **xlink:href** attribute contains a URI that is not a content id (URI scheme "cid"), and that URI
1244 cannot be resolved, then it is an implementation decision on whether to report the error. If the error is
1245 to be reported, then it SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem** and
1246 a **severity** of **Error**.

1247 8.11.7 Manifest sample

1248 The following fragment demonstrates a typical **Manifest** for a message with a single payload MIME
1249 body part:

1250
1251

```
<eb:Manifest eb:id="Manifest" eb:version="1.0">
```


1252

```
  <eb:Reference eb:id="pay01"
```


1253

```
    xlink:href="cid:payload-1"
```


1254

```
    xlink:role="http://regrep.org/gci/purchaseOrder">
```


1255

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


1256

```
    <eb:Schema eb:location="http://regrep.org/gci/purchaseOrder/po.xsd"
```


1257

```
      eb:version="1.0"/>
```


1258

```
  </eb:Reference>
```


1259

```
</eb:Manifest>
```

1260 8.12 StatusRequest Element

1261 The **StatusRequest** element is an immediate child of a SOAP **Body** and is used to identify an earlier
1262 message whose status is being requested (see section 9.1).

1263 The **StatusRequest** element consists of the following elements and attributes:

- 1264 • a REQUIRED **RefToMessageId** element
- 1265 • a REQUIRED **version** attribute
- 1266 • an **id** attribute

1267 8.12.1 StatusRequest Sample

1268 An example of the **StatusRequest** element is given below:

```
1269 <eb:StatusRequest eb:version="1.0" >
1270   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1271 </eb:StatusRequest>
```

1273 8.13 StatusResponse element

1274 The **StatusResponse** element is used by one MSH to respond to a request on the status of the
1275 processing of a message that was previously sent (see also section 9.1).

1276 The **StatusResponse** element consists of the following elements and attributes:

- 1277 • a REQUIRED **RefToMessageId** element
- 1278 • a **Timestamp** element
- 1279 • a REQUIRED **version** attribute
- 1280 • a **messageStatus** attribute
- 1281 • an **id** attribute (See section 8.2.5 for details)

1282 8.13.1 RefToMessageId element

1283 A REQUIRED **RefToMessageId** element that contains the **MessageId** of the message whose status
1284 is being reported.

1285 8.13.2 Timestamp element

1286 The **Timestamp** element contains the time that the message, whose status is being reported, was
1287 received (section 8.4.6.2.). This **MUST** be omitted if the message whose status is being reported is
1288 **NotRecognized** or the request was **Unauthorized**.

1289 8.13.3 version attribute

1290 The REQUIRED **version** attribute indicates the version of the *ebXML Message Service Specification*
1291 to which the ebXML *SOAP Header* extensions conform. Its purpose is to provide future versioning
1292 capabilities. The value of the **version** attribute **MUST** be "1.0". Future versions of this specification
1293 **SHALL** require other values of this attribute. The **version** attribute **MUST** be namespace qualified for
1294 the ebXML SOAP **Envelope** extensions namespace defined above.

1295 8.13.4 messageStatus attribute

1296 The **messageStatus** attribute identifies the status of the message that is identified by the
1297 **RefToMessageId** element. It **SHALL** be set to one of the following values:

- 1298 • **Unauthorized** – the Message Status Request is not authorized or accepted
- 1299 • **NotRecognized** – the message identified by the **RefToMessageId** element in the
1300 **StatusResponse** element is not recognized
- 1301 • **Received** – the message identified by the **RefToMessageId** element in the
1302 **StatusResponse** element has been received by the MSH

1303 Note: if a Message Status Request is sent after the elapsed time indicated by *persistDuration* has passed since
 1304 the message being queried was sent, then the Message Status Response may indicate that the *MessageId* was
 1305 *NotRecognized* as the *MessageId* is no longer in persistent storage.

1306 8.13.5 StatusResponse sample

1307 An example of the *StatusResponse* element is given below:

```
1308 <eb:StatusResponse
1309   eb:version="1.0" eb:messageStatus="Received">
1310   <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
1311   <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp></eb:StatusResponse>
```

1312 8.14 DeliveryReceipt element

1313 The *DeliveryReceipt* element is an optional element that is used by the *To Party* that received a
 1314 message, to let the *From Party* that sent the original message, know that the message was received.
 1315 The *RefToMessageId* in a message containing a *DeliveryReceipt* element is used to identify the
 1316 message being for which the receipt is being generated by its *MessageId*.

1317 The *DeliveryReceipt* element consists of the following elements and attributes:

- 1318 • a *Timestamp* element
- 1319 • a *DigestValue* element
- 1320 • an *id* attribute
- 1321 • a REQUIRED *version* attribute

1322 8.14.1 Timestamp element

1323 The *Timestamp* element is a value representing the time that the message for which a *Delivery*
 1324 *Receipt* is being generated was received by the *To Party*. It must conform to an [XMLSchema]
 1325 *timeInstant*.

1326 8.14.2 DigestValue element

1327 The *ds:DigestValue* element contains a hash of the message for which a *Delivery Receipt* is being
 1328 generated. It must conform to the *DigestValue* element as defined in [XMLSchema]. How the digest
 1329 is calculated is specified by the designer of the service or business process that is using the *ebXML*
 1330 *Message Service*. The *ds:DigestValue* element is OPTIONAL.

1331 8.14.3 version attribute

1332 The REQUIRED *version* attribute indicates the version of the *ebXML Message Service Specification*
 1333 to which the *ebXML SOAP Header* extensions conform. Its purpose is to provide future versioning
 1334 capabilities. The value of the *version* attribute MUST be "1.0". Future versions of this specification
 1335 SHALL require other values of this attribute. The version attribute MUST be namespace qualified for
 1336 the *ebXML SOAP Envelope* extensions namespace defined above.

1337 8.14.4 DeliveryReceipt sample

1338 An example of the *DeliveryReceipt* element is given below:

```
1339 <eb:DeliveryReceipt eb:version="1.0">
1340   <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
1341   <eb:DigestValue>19284AH447S4729DE9FG</eb:DigestValue>
1342 </eb:DeliveryReceipt>
```

1344 8.15 Combining ebXML SOAP Extension Elements

1345 This section describes how the various ebXML SOAP extension elements may be used in
1346 combination.

1347 8.15.1 Manifest element

1348 The **Manifest** element MUST be present if there is any data associated with the message that is not
1349 present in the *Header Container*. This applies specifically to data in the *Payload Container* or
1350 elsewhere, e.g. on the web.

1351 8.15.2 MessageHeader element

1352 The **MessageHeader** element MUST be present in every message.

1353 8.15.3 TraceHeaderList element

1354 The **TraceHeaderList** element MAY be present in any message. It MUST be present if the message
1355 is being sent reliably (see section 10) or over multiple hops (see section 8.5.6).

1356 8.15.4 StatusRequest element

1357 A **StatusRequest** element MUST NOT be present with the following elements:

- 1358 • a **Manifest** element
- 1359 • an **ErrorList** element

1360 8.15.5 StatusResponse element

1361 This element MUST NOT be present with the following elements:

- 1362 • a **Manifest** element
- 1363 • a **StatusRequest** element
- 1364 • an **ErrorList** element with a **highestSeverity** attribute set to **Error**

1365 8.15.6 ErrorList element

1366 If the **highestSeverity** attribute on the **ErrorList** is set to **Warning**, then this element MAY be
1367 present with any other element.

1368 If the **highestSeverity** attribute on the **ErrorList** is set to **Error**, then this element MUST NOT be
1369 present with the following:

- 1370 • a **Manifest** element
- 1371 • a **StatusResponse** element

1372 8.15.7 Acknowledgment element

1373 An **Acknowledgment** element MAY be present on any message.

1374 8.15.8 Signature element

1375 One or more **Signature** elements MAY be present on any message.

1376 8.15.9 Via element

1377 One-and-only-one **Via** element MAY be present in any message.

1378 9 Message Service Handler Services

1379 The Message Service Handler MAY support two services that are designed to help provide
1380 smooth operation of a Message Handling Service implementation:

- 1381 • Message Status Request
- 1382 • Message Service Handler Ping

1383 If a receiving MSH does not support the service requested, it SHOULD return a SOAP Fault with
1384 a **faultCode** of **MustUnderstand**. Each service is described below:

1385 9.1 Message Status Request Service

1386 The Message Status Request Service consists of the following:

- 1387 • A Message Status Request message containing details regarding a message previously
1388 sent is sent to a Message Service Handler (MSH)
- 1389 • The Message Service Handler receiving the request responds with a Message Status
1390 Response message.

1391 A Message Service Handler SHOULD respond to Message Status Requests for messages that
1392 have been sent reliably (see section 10) and the **MessageId** in the **RefToMessageId** is present
1393 in *persistent storage* (see section 10.1.1).

1394 A Message Service Handler MAY respond to Message Status Requests for messages that have
1395 not been sent reliably.

1396 A Message Service SHOULD NOT use the Message Status Request Service to implement
1397 Reliable Messaging.

1398 9.1.1 Message Status Request Message

1399 A Message Status Request message consists of an *ebXML Message* containing no *ebXML*
1400 *Payload* and the following elements in the SOAP **Header**:

- 1401 • A **MessageHeader** element
- 1402 • A **TraceHeaderList** element
- 1403 • A **StatusRequest** element
- 1404 • A **Signature** element

1405 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1406 8.15.8).

1407 The **MessageHeader** element MUST contain the following:

- 1408 • a **From** element that identifies the *Party* that created the message status request
1409 message
- 1410 • a **To** element that identifies a *Party* who should receive the message. If a **TraceHeader**
1411 was present on the message whose status is being checked, this MUST be set using the
1412 **Receiver** from that message. All **PartyId** elements present in the **Receiver** element
1413 SHOULD be included in this **To** element.
- 1414 • a **Service** element that contains: **uri:www.ebxml.org/messageService**
- 1415 • an **Action** element that contains **StatusRequest**

1416 The message is then sent to the *To Party*.

1417 The **RefToMessageId** element in **StatusRequest** element in the SOAP **Body** contains the
1418 **MessageId** of the message whose status is being queried.

1419 9.1.2 Message Status Response Message

1420 Once the *To Party* receives the Message Status Request message, they SHOULD generate a
1421 Message Status Response message consisting of no ebXML Payload and the following elements
1422 in the SOAP **Header** and **Body**.

- 1423 • a **MessageHeader** element
- 1424 • a **TraceHeaderList** element
- 1425 • an **Acknowledgment** element
- 1426 • a **StatusResponse** element (see section 8.13)
- 1427 • a **Signature** element

1428 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1429 sections 8.5, 8.15.7 and 8.15.8).

1430 The **MessageHeader** element MUST contain the following:

- 1431 • a **From** element that identifies the sender of the Message Status Response message
- 1432 • a **To** element that is set to the value of the **From** element in the Message Status Request
1433 message
- 1434 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1435 • an **Action** element that contains **StatusResponse**
- 1436 • a **RefToMessaged** that identifies the Message Status Request message.

1437 The message is then sent to the *To Party*.

1438 9.1.3 Security Considerations

1439 Parties who receive a Message Status Request message SHOULD always respond to the
1440 message. However, they MAY ignore the message instead of responding with **messageStatus**
1441 set to **Unauthorized** if they consider that the sender of the message is unauthorized. The
1442 decision process that results in this course of action is implementation dependent.

1443 9.2 Message Service Handler Ping Service

1444 The Message Service Handler Ping Service enables one MSH to determine if another MSH is
1445 operating. It consists of:

- 1446 • sending a Message Service Handler Ping message to a MSH, and
- 1447 • the MSH that receives the Ping responding with a Message Service Handler Pong
1448 message.

1449 9.2.1 Message Service Handler Ping Message

1450 A Message Service Handler Ping (MSH Ping) message consists of an *ebXML Message*
1451 containing no ebXML Payload and the following elements in the SOAP **Header**.

- 1452 • A **MessageHeader** element
- 1453 • A **TraceHeaderList** element
- 1454 • A **Signature** element

1455 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.5 and
1456 8.15.8).

1457 The **MessageHeader** element MUST contain the following:

- 1458 • a **From** element that identifies the *Party* creating the MSH Ping message
- 1459 • a **To** element that identifies the *Party* that is being sent the MSH Ping message
- 1460 • a **Service** element that contains: **uri:www.ebxml.org/messageService/**
- 1461 • an **Action** element that contains **Ping**

1462 The message is then sent to the *To Party*.

1463 **9.2.2 Message Service Handler Pong Message**

1464 Once the *To Party* receives the MSH Ping message, they MAY generate a Message Service
1465 Handler Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML
1466 Payload and the following elements in the SOAP Header:

- 1467 • a **MessageHeader** element
- 1468 • a **TraceHeaderList** element
- 1469 • an **Acknowledgment** element
- 1470 • an OPTIONAL **Signature** element

1471 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see
1472 sections 8.5, 8.15.7 and 8.15.8).

1473 The **MessageHeader** element MUST contain the following:

- 1474 • a **From** element that identifies the creator of the MSH Pong message
- 1475 • a **To** element that identifies a *Party* that generated the MSH Ping message
- 1476 • a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 1477 • an **Action** element that contains the value **Pong**
- 1478 • a **RefToMessageId** that identifies the MSH Ping message.

1479 The message is then sent to the *To Party*.

1480 **9.2.3 Security Considerations**

1481 Parties who receive a MSH Ping message SHOULD always respond to the message. However,
1482 there is a risk that some parties might use the MSH Ping message to determine the existence of
1483 a Message Service Handler as part of a security attack on that MSH. Therefore, recipients of a
1484 MSH Ping MAY ignore the message if they consider that the sender of the message received is
1485 unauthorized or part of some attack. The decision process that results in this course of action is
1486 implementation dependent.

1487 10 Reliable Messaging

1488 Reliable Messaging defines an interoperable protocol such that the two Message Service
1489 Handlers (MSH) can “reliably” exchange messages that are sent using “reliable messaging”
1490 semantics, resulting in the *To Party* receiving the message once and only once.

1491 Reliability is achieved by a receiving MSH responding to a message with an *Acknowledgment*
1492 *Message*.

1493 10.1.1 Persistent Storage and System Failure

1494 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably
1495 in *persistent storage*. In this context *persistent storage* is a method of storing data that does not
1496 lose information after a system failure or interruption.

1497 This specification recognizes that different degrees of resilience may be realized depending on
1498 the technology that is used to persist the data. However, as a minimum, persistent storage that
1499 has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly
1500 RECOMMENDED though that implementers of this specification use technology that is resilient to
1501 the failure of any single hardware or software component.

1502 After a system interruption or failure, a MSH MUST ensure that messages in persistent storage
1503 are processed in the same way as if the system failure or interruption had not occurred. How this
1504 is done is an implementation decision.

1505 In order to support the filtering of duplicate messages, a receiving MSH SHOULD save the
1506 **MessageId** in *persistent storage*. It is also RECOMMENDED that the following be kept in
1507 *Persistent Storage*:

- 1508 • the complete message, at least until the information in the message has been passed to
1509 the application or other process that needs to process it
- 1510 • the time the message was received, so that the information can be used to generate the
1511 response to a Message Status Request (see section 9.1)
- 1512 • complete response message

1513 10.1.2 Methods of Implementing Reliable Messaging

1514 Support for Reliable Messaging MAY be implemented in one of the following two ways:

- 1515 • using the ebXML Reliable Messaging protocol, or
- 1516 • using ebXML SOAP structures together with commercial software products that are
1517 designed to provide reliable delivery of messages using alternative protocols.

1518 10.2 Reliable Messaging Parameters

1519 This section describes the parameters required to control reliable messaging. This parameter
1520 information can be specified in the *CPA* or in the **MessageHeader** (section 8.4.2).

1521 10.2.1 Delivery Semantics

1522 The **deliverySemantics** value MUST be used by the *From Party* MSH to indicate whether the
1523 Message MUST be sent reliably. Valid values are:

- 1524 • **OnceAndOnlyOnce**. The message must be sent using a **reliableMessagingMethod**
1525 that will result in the application or other process at the *To Party* receiving the message
1526 once and only once
- 1527 • **BestEffort**. The reliable delivery semantics are not used. In this case, the value of
1528 **reliableMessagingMethod** is ignored.

1529 The value for **deliverySemantics** is specified in the *CPA* or in **MessageHeader** (section 8.4.2).
1530 The default value for **deliverySemantics** is **BestEffort**.

1531 If **deliverySemantics** is set to **OnceAndOnlyOnce**, the *From Party* MSH and the *To Party* MSH
1532 must adopt a reliable messaging behavior that describes how messages are resent in the case of
1533 failure. The **deliverySemantic** value of **OnceAndOnlyOnce** will cause duplicate messages to
1534 be ignored.

1535 If **deliverySemantics** is set to **BestEffort**, a MSH that received a message that it is unable to
1536 deliver MUST NOT take any action to recover or otherwise notify anyone of the problem. The
1537 MSH that sent the message MUST NOT attempt to recover from any failure. This means that
1538 duplicate messages might be delivered to an application and persistent storage of messages is
1539 not required.

1540 If the *To Party* is unable to support the type of delivery semantics requested, the *To Party*
1541 SHOULD report the error to the *From Party* using an **ErrorCode** of **NotSupported** and a
1542 **Severity** of **Error**.

1543 10.2.2 mshTimeAccuracy

1544 The **mshTimeAccuracy** parameter indicates the minimum accuracy a Receiving MSH keeps the
1545 clocks it uses when checking, for example, **TimeToLive**. Its value is in the format "mm:ss" which
1546 indicates the accuracy in minutes and seconds.

1547 10.2.3 Time To Live

1548 The **TimeToLive** value indicates the time by which a message should be delivered to and
1549 processed by the *To Party*. It must conform to an XML Schema `timeInstant`.

1550 In this context, the **TimeToLive** has expired if the time of the internal clock of the receiving MSH
1551 is greater than the value of **TimeToLive** for the message.

1552 When setting a value for **TimeToLive** it is RECOMMENDED that the *From Party*'s MSH takes
1553 into account the accuracy of its own internal clocks as well as the **mshTimeAccuracy** parameter
1554 for the receiving MSH indicating the accuracy to which a MSH will keep its internal clocks. How a
1555 MSH ensures that its internal clocks are kept sufficiently accurate is an implementation decision.

1556 If the *To Party*'s MSH receives a message where **TimeToLive** has expired, it SHALL send a
1557 message to the *From Party* MSH, reporting that the **TimeToLive** of the message has expired.
1558 This message SHALL be comprised of an **ErrorList** containing an error that has the **errorCode**
1559 attribute set to **TimeToLiveExpired**, and the **severity** attribute set to **Error**.

1560 10.2.4 reliableMessagingMethod

1561 The **reliableMessagingMethod** attribute SHALL have one of the following values:

- 1562 • **ebXML**
- 1563 • **Transport**

1564 The default implied value for this attribute is **ebXML** and is case sensitive. Refer to section 8.7.5
1565 for discussion of the use of this attribute.

1566 10.2.5 ackRequested

1567 The **ackRequested** value is used by the sending MSH to request that the receiving MSH returns
1568 an *acknowledgment message* with an **Acknowledgment** element.

1569 Valid values for **ackRequested** are:

- 1570 • **Unsigned** - requests that an unsigned Acknowledgement is requested
- 1571 • **Signed** - requests that a signed Acknowledgement is requested, or
- 1572 • **None** - indicates that no Acknowledgement is requested.

1573 The default value is **None**.

1574

1575 **10.2.6 retries**

1576 The *retries* value is an integer value that specifies the maximum number of times a Sending
 1577 MSH SHOULD attempt to redeliver an unacknowledged *message* using the same
 1578 Communications Protocol.

1579 **10.2.7 rretryInterval**

1580 The *retryInterval* value is a time value, expressed as a duration in accordance with the
 1581 [XMLSchema] duration data type. This value specifies the minimum time the Sending MSH
 1582 MUST wait between retries, if an *Acknowledgment Message* is not received.

1583 **10.2.8 persistDuration**

1584 The *persistDuration* value is the minimum length of time, expressed as a [XMLSchema]
 1585 duration, that data from a reliably sent *Message*, is kept in *Persistent Storage* by a receiving
 1586 MSH.

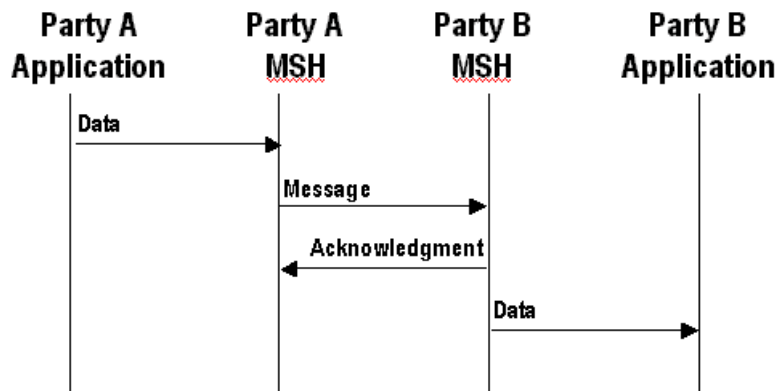
1587 If the *persistDuration* has passed since the message was first sent, a Sending MSH SHOULD
 1588 NOT resend a message with the same *MessageId* .

1589 If a message cannot be sent successfully before *persistDuration* has passed, then the Sending
 1590 MSH should report a delivery failure (see section 10.4).

1591 **10.3 ebXML Reliable Messaging Protocol**

1592 The ebXML Reliable Messaging Protocol described in this section MUST be followed if the
 1593 *deliverySemantics* parameter/element is set to *OnceAndOnlyOnce* and the
 1594 *reliableMessagingMethod* parameter/element is set to *ebXML* (the default).

1595 The ebXML Reliable Messaging Protocol is illustrated by the figure below.



1596
 1597 **Figure 10-1 Indicating that a message has been received**

1598 The receipt of the *Acknowledgment Message* indicates that the message being acknowledged
 1599 has been successfully received and either processed or persisted by the receiving MSH.

1600 An *Acknowledgment Message* MUST contain a *MessageData* element with a *RefToMessageId*
 1601 that contains the same value as the *MessageId* element in the *message being acknowledged*.

1602 **10.3.1.1 Sending Message Behavior**

1603 If a MSH is given data by an application that needs to be sent reliably (i.e. the
 1604 *deliverySemantics* is set to *OnceAndOnlyOnce*), then the MSH MUST do the following:

- 1605 1. Create a message from components received from the application that includes a
1606 **TraceHeader** element that identifies the sender and the receiver as described in Section
1607 8.5.4 **TraceHeader** element.
- 1608 2. Save the message in *persistent storage* (see section 10.1.1)
- 1609 3. Send the message to the *Receiver* MSH
- 1610 4. Wait for the *Receiver* MSH to return an *Acknowledgment Message* and, if it does not or a
1611 transient error is returned, then take the appropriate action as described in section 10.3.1.4

1612 10.3.1.2 Receiving Message Behavior

1613 If the **deliverySemantics** for the received message is set to **OnceAndOnlyOnce** then do the
1614 following:

- 1615 1. If the message is just an acknowledgement (i.e. the **Service** element is set to
1616 <http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment> and **Action**
1617 is set to **Acknowledgment**), then:
 - 1618 a) Look for a message in *persistent storage* that has a **MessageId** that is the same as the
1619 value of **RefToMessageId** on the received Message
 - 1620 b) If a message is found in *persistent storage* then mark the persisted message as delivered
- 1621 2. Otherwise, if the message is not just an acknowledgement, then check to see if the
1622 message is a duplicate (e.g. there is a **MessageId** held in *persistent storage* that was
1623 received earlier that contains the same value for the **MessageId**)
 - 1624 c) If the message is not a duplicate then do the following:
 - 1625 i) Save the **MessageId** of the received message in *persistent storage*. As an
1626 implementation decision, the whole message MAY be stored if there are other
1627 reasons for doing so.
 - 1628 ii) If the received message contains a **RefToMessageId** element then do the following:
 - 1629 (1) Look for a message in *persistent storage* that has a **MessageId** that is the same
1630 as the value of **RefToMessageId** on the received Message
 - 1631 (2) If a message is found in *persistent storage* then mark the persisted message as
1632 delivered
 - 1633 iii) Generate an *Acknowledgment Message* in response (see section 10.3.1.3).
 - 1634 d) If the message is a duplicate, then do the following:
 - 1635 i) Look in persistent storage for the first response to the received message and resend
1636 it (i.e. it contains a **RefToMessageId** that matches the **MessageId** of the received
1637 message)
 - 1638 ii) If a message was found in *persistent storage* then resend the persisted message
1639 back to the MSH that sent the received message,
 - 1640 iii) If no message was found in *persistent storage*, then:
 - 1641 (1) if **syncReply** is set to **True** and if the CPA indicates an application response is
1642 included, ignore the received message (i.e. no message was generated in
1643 response to the message, or the processing of the earlier message is not yet
1644 complete)
 - 1645 (2) if **syncReply** is set to **False** then generate an *Acknowledgment Message* (see
1646 section 10.3.1.3).

1647 **10.3.1.3 Generating an Acknowledgement Message**

1648 An *Acknowledgement Message* MUST be generated whenever a message is received with:

- 1649 • **deliverySemantics** set to **OnceAndOnlyOnce** and
- 1650 • **reliableMessagingMethod** set to **ebXML** (the default).

1651 As a minimum, it MUST contain a **MessageData** element with a **RefToMessageld** that contains
 1652 the same value as the **MessageId** element in the *message being acknowledged*.

1653 If **ackRequested** in the **Via** of the received message is set to **Signed** or **Unsigned** then the
 1654 acknowledgement message MUST also contain an **Acknowledgement** element.

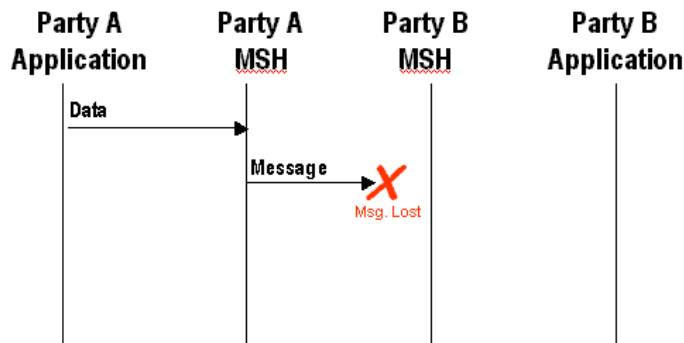
1655 Depending on the value of the **syncReply** parameter, the *Acknowledgement Message* can also
 1656 be sent at the same time as the response to the received message. In this case, the values for
 1657 the **MessageHeader** elements of the *Acknowledgement Message* are set by the designer of the
 1658 Service.

1659 If an **Acknowledgment** element is being sent on its own, then the value of the **MessageHeader**
 1660 elements MUST be set as follows:

- 1661 1) The **Service** element MUST be set to: **uri:www.ebxml.org/messageService/**
- 1662 2) The **Action** element MUST be set to **Acknowledgment**.
- 1663 3) The **From** element MAY be populated with the **To** element extracted from the message
 1664 received, or it MAY be set using the **Receiver** from the last **TraceHeader** in the *message* that
 1665 has just been received. In either case, all **PartyId** elements from the message received
 1666 SHOULD be included in this **From** element.
- 1667 4) The **To** element MAY be populated with the **From** element extracted from the message
 1668 received, or it MAY be set using the **Sender** from the last **TraceHeader** in the *message* that
 1669 has just been received. In either case, all **PartyId** elements from the message received
 1670 SHOULD be included in this **To** element.
- 1671 5) The **RefToMessageld** element MUST be set to the **MessageId** of the *message* that has just
 1672 been received

1673 **10.3.1.4 Resending Lost Messages and Duplicate Filtering**

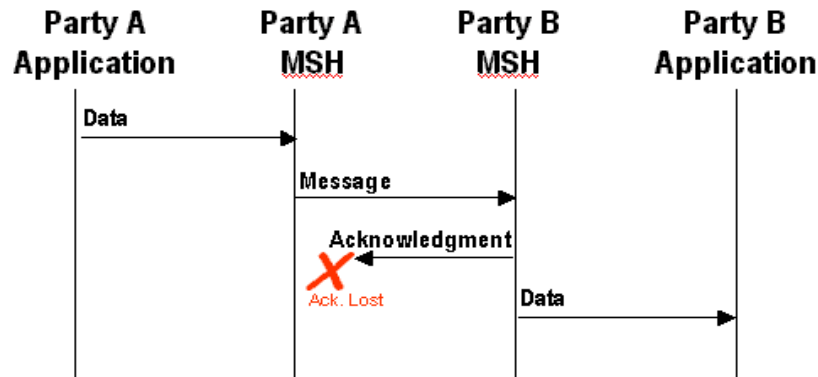
1674 This section describes the behavior that is required by the sender and receiver of a message in
 1675 order to handle when messages are lost. A message is "lost" when a sending MSH does not
 1676 receive a response to a message. For example, it is possible that a *message* was lost, for
 1677 example:



1678

1679 **Figure 10-2 Undelivered Message**

1680 It is also possible that the *Acknowledgment Message* was lost, for example:



1681

1682 **Figure 10-3 Lost Acknowledgment Message**

1683 The rules that apply are as follows:

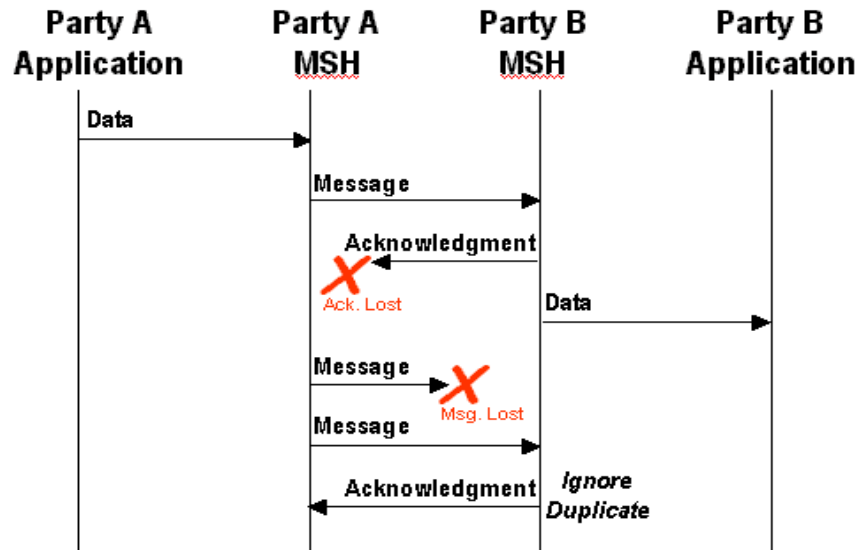
- 1684 1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has
 1685 not been received from the Receiving MSH and the following are both true:
- 1686 a) At least the time specified in the *retryInterval* has passed since the message was last
 1687 sent, and
 - 1688 b) The message has been resent less than the number of times specified in the *retries*
 1689 Parameter
- 1690 2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum
 1691 number of retries, the Sending MSH SHOULD notify the application and/or system
 1692 administrator function of the failure to receive an acknowledgement.
- 1693 3) If the Sending MSH detects an unrecoverable communications protocol error at the transport
 1694 protocol level, the Sending MSH SHOULD resend the message.

1695 **10.3.1.5 Duplicate Message Handling**

1696 In the context of this specification, a duplicate message is:

- 1697 • an *identical message* is a *message* that contains, apart from perhaps an additional
 1698 **TraceHeader** element, the same *ebXML SOAP Header, Body and ebXML Payload* as
 1699 the earlier *message* that was sent.
- 1700 • a *duplicate message* is a *message* that contains the same **MessageId** as an earlier
 1701 message that was received.
- 1702 • the first *message* is the message with the earliest **Timestamp** in the **MessageData**
 1703 element that has the same **RefToMessageId** as the duplicate message.

1704



1705

1706 **Figure 10-4 Resending Unacknowledged Messages**

1707 The diagram above shows the behavior that MUST be followed by the sending and receiving
 1708 MSH that are sent with **deliverySemantics** of **OnceAndOnlyOnce**. Specifically:

- 1709 1) The sender of the *message* (e.g. Party A) MUST resend the *identical message* if no
 1710 *Acknowledgment Message* is received
- 1711 2) When the recipient (Party B) of the *message* receives a *duplicate message*, it MUST resend
 1712 to the sender (Party A) a message identical to the *first message* that was sent to the sender
 1713 Party A)
- 1714 3) The recipient of the *message* (Party B) MUST NOT forward the message a second time to
 1715 the application/process.

1716 **10.4 Failed Message Delivery**

1717 If a message sent with **deliverySemantics** set to **OnceAndOnlyOnce** cannot be delivered, the
 1718 MSH or process SHOULD send a delivery failure notification to the *From Party*. The delivery
 1719 failure notification message contains:

- 1720 • a **From** element that identifies the *Party* who detected the problem
- 1721 • a **To** element that identifies the *From Party* that created the message that could not be
 1722 delivered
- 1723 • a **Service** element and **Action** element set as described in 11.5
- 1724 • an **Error** element with a severity of:
- 1725 - **Error** if the party who detected the problem could not transmit the message (e.g. the
 1726 communications transport was not available)
- 1727 - **Warning** if the message was transmitted, but an *acknowledgment message* was not
 1728 received. This means that the message probably was not delivered although there is
 1729 a small probability that it was.
- 1730 • an **ErrorCode** of **DeliveryFailure**

1731 It is possible that an error message with an **Error** element with an **ErrorCode** set to
 1732 **DeliveryFailure** cannot be delivered successfully for some reason. If this occurs, then the *From*
 1733 *Party* that is the ultimate destination for the error message SHOULD be informed of the problem
 1734 by other means. How this is done is outside the scope of this specification.

1735 11 Error Reporting and Handling

1736 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects in
1737 an ebXML Message to another MSH. The *ebXML Message Service* error reporting and handling is to
1738 be considered as being a layer of processing above the SOAP Processor layer. This means that the
1739 ebXML MSH is essentially an application-level handler of a *SOAP Message* from the perspective of
1740 the SOAP Processor. The SOAP Processor MAY generate SOAP Fault messages if it is unable to
1741 process the message. A Sending MSH MUST be prepared to accept and process these SOAP
1742 Faults.

1743 It is possible for the ebXML MSH software to cause a SOAP Fault to be generated and returned to
1744 the sender of a *SOAP Message*. In this event, the returned message MUST conform to the [SOAP]
1745 specification processing guidelines for SOAP Faults.

1746 An ebXML *SOAP Message* that reports an error that has a **highestSeverity** of **Warning** SHALL NOT
1747 be reported or returned as a SOAP Fault.

1748 11.1 Definitions

1749 For clarity two phrases are defined that are used in this section:

- 1750 • *message in error*. A *message* that contains or causes an error of some kind
- 1751 • *message reporting the error*. A *message* that contains an ebXML **ErrorList** element that
1752 describes the error(s) found in a *message in error*.

1753 11.2 Types of Errors

1754 One MSH needs to report to another MSH errors in a *message in error*. For example, errors
1755 associated with:

- 1756 • ebXML namespace qualified content of the *SOAP Message* document (see section 0),
- 1757 • reliable messaging failures (see section 10), or
- 1758 • security (see section 12).

1759 Unless specified to the contrary, all references to "an error" in the remainder of this specification imply
1760 any or all of the types of errors listed above.

1761 Errors associated with Data Communication protocols are detected and reported using the standard
1762 mechanisms supported by that data communication protocol and are do not use the error reporting
1763 mechanism described here.

1764 11.3 When to generate Error Messages

1765 When a MSH detects an error in a message it is strongly RECOMMENDED that the error is reported
1766 to the MSH that sent the message that had an error if:

- 1767 • the Error Reporting Location (see section 11.4) to which the *message reporting the error*
1768 should be sent can be determined, and
- 1769 • the *message in error* does not have an **ErrorList** element with **highestSeverity** set to **Error**.

1770 If the Error Reporting Location cannot be found or the *message in error* has an **ErrorList** element
1771 with **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1772 • the error is logged,
- 1773 • the problem is resolved by other means, and
- 1774 • no further action is taken.

1775 11.3.1 Security Considerations

1776 Parties that receive a Message containing an error in the header SHOULD always respond to the
1777 message. However, they MAY ignore the message and not respond if they consider that the message
1778 received is unauthorized or is part of some security attack. The decision process that results in this
1779 course of action is implementation dependent.

1780 11.4 Identifying the Error Reporting Location

1781 The Error Reporting Location is a URI that is specified by the sender of the *message in error* that
1782 indicates where to send a *message reporting the error*.

1783 The **ErrorURI** implied by the CPA, identified by the **CPAId** on the message, SHOULD be used. If no
1784 **ErrorURI** is implied by the CPA and a **TraceHeaderList** is present in the *message in error*, the value
1785 of the **Location** element in the **Sender** of the topmost **TraceHeader** MUST be used. Otherwise, the
1786 recipient MAY resolve an **ErrorURI** using the **From** element of the *message in error*. If this is not
1787 possible, no error will be reported to the sending *Party*.

1788 Even if the *message in error* cannot be successfully analyzed or parsed, MSH implementers
1789 SHOULD try to determine the Error Reporting Location by other means. How this is done is an
1790 implementation decision.

1791 11.5 Service and Action Element Values

1792 An **ErrorList** element can be included in a **SOAP Header** that is part of a *message* that is being sent
1793 as a result of processing of an earlier message. In this case, the values for the **Service** and **Action**
1794 elements are set by the designer of the Service.

1795 An **ErrorList** element can also be included in an **SOAP Header** that is not being sent as a result of
1796 the processing of an earlier message. In this case, if the **highestSeverity** is set to **Error**, the values
1797 of the **Service** and **Action** elements MUST be set as follows:

- 1798 • The **Service** element MUST be set to: **uri:www.ebxml.org/messageService/**
- 1799 • The **Action** element MUST be set to **MessageError**.

1800 If the **highestSeverity** is set to **Warning**, the **Service** and **Action** elements MUST NOT be used.

1801 **12 Security**

1802 The *ebXML Message Service*, by its very nature, presents certain security risks. A Message Service
1803 may be at risk by means of:

- 1804 • Unauthorized access
- 1805 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1806 • Denial-of-Service and spoofing

1807 Each security risk is described in detail in the ebXML Technical Architecture Security Specification
1808 [ebTASEC].

1809 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a
1810 combination, of the countermeasures described in this section. This specification describes a set of
1811 profiles, or combinations of selected countermeasures, that have been selected to address key risks
1812 based upon commonly available technologies. Each of the specified profiles includes a description of
1813 the risks that are not addressed.

1814 Application of countermeasures SHOULD be balanced against an assessment of the inherent risks
1815 and the value of the asset(s) that might be placed at risk.

1816 **12.1 Security and Management**

1817 No technology, regardless of how advanced it might be, is an adequate substitute to the effective
1818 application of security management policies and practices.

1819 It is strongly RECOMMENDED that the site manager of an *ebXML Message Service* apply due
1820 diligence to the support and maintenance of its; security mechanism, site (or physical) security
1821 procedures, cryptographic protocols, update implementations and apply fixes as appropriate. (See
1822 <http://www.cert.org/> and <http://ciac.llnl.gov/>)

1823 **12.2 Collaboration Protocol Agreement**

1824 The configuration of Security for MSHs may be specified in the CPA. Three areas of the CPA have
1825 security definitions as follows:

- 1826 • The Document Exchange section addresses security to be applied to the payload of the
1827 message. The MSH is not responsible for any security specified at this level but may offer
1828 these services to the message sender.
- 1829 • The Message section addresses security applied to the entire ebXML Document, which
1830 includes the header and the payload.

1831 **12.3 Countermeasure Technologies**

1832 **12.3.1 Persistent Digital Signature**

1833 If signatures are being used to digitally sign an ebXML Message then XML Signature [DSIG] MUST
1834 be used to bind the ebXML SOAP **Header** and **Body** to the ebXML Payload or data elsewhere on the
1835 web that relates to the message. It is also strongly RECOMMENDED that XML Signature be used to
1836 digitally sign the Payload on its own.

1837 The only available technology that can be applied to the purpose of digitally signing an ebXML
1838 Message (the ebXML SOAP **Header** and **Body** and its associated payload objects) is provided by
1839 technology that conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML
1840 Signature conforming to this specification can selectively sign portions of an XML document(s),
1841 permitting the documents to be augmented (new element content added) while preserving the validity
1842 of the signature(s).

1843 An ebXML Message that requires a digital signature SHALL be signed following the process defined
1844 in this section of the specification and SHALL be in full compliance with [XMLDSIG].

1845 **12.3.1.1 Signature Generation**

1846 1) Create a **SignedInfo** element with **SignatureMethod**, **CanonicalizationMethod**, and
1847 **Reference(s)** elements for the SOAP Header and any required payload objects, as prescribed by
1848 [XMLDSIG].

1849 2) Canonicalize and then calculate the **SignatureValue** over **SignedInfo** based on algorithms
1850 specified in **SignedInfo** as specified in [XMLDSIG].

1851 3) Construct the **Signature** element that includes the **SignedInfo**, **KeyInfo** (RECOMMENDED), and
1852 **SignatureValue** elements as specified in [XMLDSIG].

1853 4) Include the namespace qualified **Signature** element in the SOAP **Header** just signed, following
1854 the **TraceHeaderList** element.

1855 The **ds:SignedInfo** element SHALL be composed of zero or one **ds:CanonicalizationMethod**
1856 element, the **ds:SignatureMethod** and one or more **ds:Reference** elements.

1857 The **ds:CanonicalizationMethod** element is defined as OPTIONAL in [XMLDSIG], meaning that the
1858 element need not appear in an instance of a **ds:SignedInfo** element. The default canonicalization
1859 method that is applied to the data to be signed is [XMLC14N] in the absence of a
1860 **ds:Canonicalization** element that specifies otherwise. This default SHALL also serve as the default
1861 canonicalization method for the *ebXML Message Service*.

1862 The **ds:SignatureMethod** element SHALL be present and SHALL have an Algorithm attribute. The
1863 RECOMMENDED value for the Algorithm attribute is:

1864 <http://www.w3.org/2000/09/xmlsig#dsa-sha1>

1865 This RECOMMENDED value SHALL be supported by all compliant *ebXML Message Service*
1866 software implementations.

1867 The **ds:Reference** element for the SOAP **Header** document SHALL have a URI attribute value of ""
1868 to provide for the signature to be applied to the document that contains the **ds:Signature** element
1869 (the SOAP **Header**).

1870 The **ds:Reference** element for the SOAP **Header** MAY include a **Type** attribute that has a value
1871 "http://www.w3.org/2000/09/xmlsig#Object" in accordance with [XMLDSIG]. This attribute is purely
1872 informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared to handle
1873 either case. The **ds:Reference** element MAY include the optional **id** attribute.

1874 The **ds:Reference** element for the SOAP **Header** SHALL include a child **ds:Transform** element that
1875 excludes the containing **ds:Signature** element and all its descendants as well as the
1876 **TraceHeaderList** element and all its descendants as these elements are subject to change. The
1877 **ds:Transform** element SHALL include a child **ds:XPath** element that has a value of:
1878 /descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1']) and not (ancestor-or-
1879 self::VIA)]

1880 Each payload object that requires signing SHALL be represented by a **ds:Reference** element that
1881 SHALL have a **URI** attribute that resolves to that payload object. This MAY be either the `Content-`
1882 `Id` URI of the MIME body part of the payload object, or a URI that matches the `Content-Location` of
1883 the MIME body part of the payload object, or a URI that resolves to an external payload object
1884 external to the Message Package. It is strongly RECOMMENDED that the URI attribute value match
1885 the `xlink:href` URI value of the corresponding **Manifest/Reference** element for that payload object.
1886 However, this is NOT REQUIRED.

1887 Example of digitally signed ebXML SOAP Message:

1888
1889
1890

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
```



```

1891  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
1892  xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
1893  xmlns:xlink="http://www.w3.org/1999/xlink">
1894  <SOAP-ENV:Header>
1895    <eb:MessageHeader eb:id="..." eb:version="1.0">
1896      ...
1897    </eb:MessageHeader>
1898    <eb:TraceHeaderList eb:id="..." eb:version="1.0">
1899      <eb:TraceHeader>
1900        ...
1901      </eb:TraceHeader>
1902    </eb:TraceHeaderList>
1903    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
1904      <ds:SignedInfo>
1905        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
1906        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
1907        <ds:Reference URI="">
1908          <ds:Transforms>
1909            <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
1910              <XPath>/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1'])
1911                or(ancestor-or-self::eb:TraceHeaderList)]</XPath>
1912            </ds:Transform>
1913          </ds:Transforms>
1914          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
1915          <ds:DigestValue>...</ds:DigestValue>
1916        </ds:Reference>
1917        <ds:Reference URI="cid://blahblahblah/">
1918          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
1919          <ds:DigestValue>...</ds:DigestValue>
1920        </ds:Reference>
1921      </ds:SignedInfo>
1922    <ds:SignatureValue>...</ds:SignatureValue>
1923    <ds:KeyInfo>...</ds:KeyInfo>
1924  </ds:Signature>
1925 </SOAP-ENV:Header>
1926 <SOAP-ENV:Body>
1927   <eb:Manifest eb:id="Mani01" eb:version="1.0">
1928     <eb:Reference xlink:href="cid://blahblahblah"
1929       xlink:role="http://ebxml.org/gci/invoice">
1930       <eb:Schema eb:version="1.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
1931     </eb:Reference>
1932   </eb:Manifest>
1933 </SOAP-ENV:Body>
1934 </SOAP-ENV:Envelope>

```

1935

1936 12.3.2 Persistent Signed Receipt

1937 An *ebXML Message* that has been digitally signed MAY be acknowledged with a **DeliveryReceipt**
 1938 acknowledgment message that itself is digitally signed in the manner described in the previous
 1939 section. The acknowledgment message MUST contain the set of **ds:DigestValue** elements
 1940 contained in the **ds:Signature** element of the original message within the **Acknowledgment**
 1941 element.

1942 12.3.3 Non-persistent Authentication

1943 Non-persistent authentication is provided by the communications channel used to transport the
 1944 *ebXML Message*. This authentication MAY be either in one direction—from the session initiator to the
 1945 receiver—or bi-directional. The specific method will be determined by the communications protocol
 1946 used. For instance, the use of a secure network protocol, such as [RFC2246] or [IPSEC] provides
 1947 the sender of an *ebXML Message* with a way to authenticate the destination for the TCP/IP
 1948 environment.

1949 12.3.4 Non-persistent Integrity

1950 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured so as to provide
 1951 for integrity check CRCs of the packets transmitted via the network connection.

1952 12.3.5 Persistent Confidentiality

1953 XML Encryption is a W3C/IETF joint activity that is actively engaged in the drafting of a specification
1954 for the selective encryption of an XML document(s). It is anticipated that this specification will be
1955 completed within the next year. The ebXML Transport, Routing and Packaging team has identified
1956 this technology as the only viable means of providing persistent, selective confidentiality of elements
1957 within an *ebXML Message* including the SOAP **Header**.

1958 Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH. However,
1959 this specification states that it is not the responsibility of the MSH to provide security for the ebXML
1960 Payloads. Payload confidentiality MAY be provided by using XML Encryption (when available) or
1961 some other cryptographic process, such as [S/MIME], [S/MIMEV3], or [PGP/MIME], that is bilaterally
1962 agreed upon by the parties involved. Since XML Encryption is not currently available, it is
1963 RECOMMENDED that [S/MIME] encryption methods be used for ebXML Payloads. The XML
1964 Encryption standard SHALL be the default encryption method when XML Encryption has achieved
1965 W3C Recommendation status.

1966 12.3.6 Non-persistent Confidentiality

1967 Use of a secure network protocol such as [RFC2246] or [IPSEC] provides transient confidentiality of a
1968 message as it is transferred between two ebXML MSH nodes.

1969 12.3.7 Persistent Authorization

1970 The OASIS Security Services TC is actively engaged in the definition of a specification that provides
1971 for the exchange of security credentials, including NameAssertion and Entitlements that is based on
1972 [S2ML]. Use of technology that is based on this anticipated specification MAY be used to provide
1973 persistent authorization for an *ebXML Message* once it becomes available. ebXML has a formal
1974 liaison to this TC. There are also many ebXML member organizations and contributors that are active
1975 members of the OASIS Security Services TC such as Sun, IBM, CommerceOne, Cisco and others
1976 that are endeavoring to ensure that the specification meets the requirements of providing persistent
1977 authorization capabilities for the *ebXML Message Service*.

1978 12.3.8 Non-persistent Authorization

1979 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured to provide for
1980 bilateral authentication of certificates prior to establishing a session. This provides for the ability for an
1981 ebXML MSH to authenticate the source of a connection that can be used to recognize the source as
1982 an authorized source of *ebXML Messages*.

1983 12.3.9 Trusted Timestamp

1984 At the time of this specification, services that offer trusted timestamp capabilities are becoming
1985 available. Once these become more widely available, and a standard has been defined for their use
1986 and expression, these standards, technologies and services will be evaluated and considered for use
1987 in providing this capability.

1988 12.3.10 Supported Security Services

1989 The general architecture of the ebXML Message Service Specification is intended to support all the
1990 security services required for electronic business. The following table combines the security services
1991 of the Message Service Handler into a set of security profiles. These profiles, or combinations of
1992 these profiles, support the specific security policy of the ebXML user community. Due to the
1993 immature state of XML security specifications, this version of the specification requires support for
1994 profiles 0 and 1 only. This does not preclude users from employing additional security features to
1995 protect ebXML exchanges; however, interoperability between parties using any profiles other than 0
1996 and 1 cannot be guaranteed.

1997

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
✓	Profile 0										no security services are applied to data
✓	Profile 1	✓									sending MSH applies XML/DSIG structures to message
	Profile 2		✓						✓		sending MSH authenticates and receiving MSH authorizes sender based on communication channel credentials.
	Profile 3		✓			✓					sending MSH authenticates and both MSHs negotiate a secure channel to transmit data
	Profile 4		✓		✓						sending MSH authenticates, the receiving MSH performs integrity checks using communications protocol
	Profile 5		✓								sending MSH authenticates the communication channel only (e.g., SSL 3.0 over TCP/IP)
	Profile 6	✓					✓				sending MSH applies XML/DSIG structures to message and passes in secure communications channel
	Profile 7	✓		✓							sending MSH applies XML/DSIG structures to message and receiving MSH returns a signed receipt
	Profile 8	✓		✓			✓				combination of profile 6 and 7
	Profile 9	✓							✓		Profile 5 with a trusted timestamp applied
	Profile 10	✓		✓					✓		Profile 9 with receiving MSH returning a signed receipt
	Profile 11	✓					✓		✓		Profile 6 with the receiving MSH applying a trusted timestamp
	Profile 12	✓		✓			✓		✓		Profile 8 with the receiving MSH applying a trusted timestamp
	Profile 13	✓				✓					sending MSH applies XML/DSIG structures to message and applies confidentiality structures (XML-Encryption)
	Profile 14	✓		✓		✓					Profile 13 with a signed receipt
	Profile 15	✓		✓					✓		sending MSH applies XML/DSIG structures to message, a trusted timestamp is added to message, receiving MSH returns a signed receipt

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
	Profile 16	✓				✓				✓	Profile 13 with a trusted timestamp applied
	Profile 17	✓		✓		✓				✓	Profile 14 with a trusted timestamp applied
	Profile 18	✓						✓			sending MSH applies XML/DSIG structures to message and forwards authorization credentials (S2ML)
	Profile 19	✓		✓				✓			Profile 18 with receiving MSH returning a signed receipt
	Profile 20	✓		✓				✓		✓	Profile 19 with the a trusted timestamp being applied to the sending MSH message
	Profile 21	✓		✓		✓		✓		✓	Profile 19 with the sending MSH applying confidentiality structures (XML-Encryption)
	Profile 22					✓					sending MSH encapsulates the message within confidentiality structures (XML-Encryption)

1998

1999 **13 References**2000 **13.1 Normative References**

- 2001 [HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, January 1997
- 2002 [RFC822] Standard for the format of ARPA Internet text messages. D. Crocker. Aug-13-1982.
- 2003 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein, Published November 1996
- 2004 [RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N. Borenstein. November 1996.
- 2005 [RFC2246] RFC 2246 - Dierks, T. and C. Allen, "The TLS Protocol", January 1999.
- 2006 [RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.
- 2007 [RFC2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E. Levinson, Published August 1998
- 2008 [RFC2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, Published August 1998
- 2009 [RFC2487] SMTP Service Extension for Secure SMTP over TLS. P. Hoffman. January 1999.
- 2010 [RFC2554] SMTP Service Extension for Authentication. J. Myers. March 1999.
- 2011 [RFC2616] RFC 2616 - Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", , June 1999.
- 2012 [RFC2617] 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.
- 2013 [RFC2817] RFC 2817 - Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", May 2000.
- 2014 [RFC2818] RFC 2818 - Rescorla, E., "HTTP Over TLS", , May 2000.[SOAP] Simple Object Access Protocol
- 2015 [SMTP] IETF RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982
- 2016 [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, <http://www.w3.org/TR/SOAP>
- 2017 [SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000 <http://www.w3.org/TR/SOAP-attachments>
- 2018 [SSL3] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996.
- 2019 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage conventions.
- 2020 [XLINK] W3C XML Linking Candidate Recommendation, <http://www.w3.org/TR/xlink/>

- 2041 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second
2042 Edition), October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 2043 [XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web Consortium,
2044 14 January 1999, <http://www.w3.org/TR/REC-xml-names>
- 2045 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,
2046 <http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/>
- 2047 [XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent, January 2001
- 2048 **13.2 Non-Normative References**
- 2049 [ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 0.92,
2050 published 3 March, 2001
- 2051 [ebTA] ebXML Technical Architecture, version 1.04 published 16 February, 2001
- 2052 [ebTASEC] ebXML Technical Architecture Security Specification, version 0.3 published 19
2053 February, 2001
- 2054 [ebRS] ebXML Registry Services Specification, version 0.84
- 2055 [ebMSREQ] ebXML Transport, Routing and Packaging: Overview and Requirements, Version
2056 0.96, Published 25 May 2000
- 2057 [Glossary] ebXML Glossary, see ebXML Project Team Home Page
2058 http://www.ebxml.org/project_teams/technical_coord/ebxml_ta_glossary95.xls
- 2059 [IPSEC] IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November 1998.
2060 RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson.
2061 November 1998.
- 2062 [PGP/MIME] IETF RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.
2063 October 1996.
- 2064 [S/MIME] IETF RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P.
2065 Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.
- 2066 [S/MIMECH] IETF RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman,
2067 B. Ramsdell, J. Weinstein. March 1998.
- 2068 [S/MIMEV3] IETF RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed..
2069 June 1999.
- 2070 [TLS] RFC2246, T. Dierks, C. Allen. January 1999.
- 2071 [XMLSchema] W3C XML Schema Candidate Recommendation,
2072 <http://www.w3.org/TR/xmlschema-0/>
2073 <http://www.w3.org/TR/xmlschema-1/>
2074 <http://www.w3.org/TR/xmlschema-2/>
- 2075 [XMTP] XMTP - Extensible Mail Transport Protocol
2076 <http://www.openhealth.org/documents/xmtp.htm>

2077 **14 Disclaimer**

2078 The views and specification expressed in this document are those of the authors and are not
2079 necessarily those of their employers. The authors and their employers specifically disclaim
2080 responsibility for any problems arising from correct or incorrect implementation or use of this design.

2081 **15 Contact Information**

2082 **Team Leader**

2083 Name Rik Drummond
 2084 Company Drummond Group, Inc.
 2085 Street 5008 Bentwood Ct.
 2086 City, State, Postal Code Fort Worth, Texas 76132
 2087 Country USA
 2088 Phone +1 (817) 294-7339
 2089 EMail: rik@drummondgroup.com

2091 **Vice Team Leader**

2092 Name Christopher Ferris
 2093 Company Sun Microsystems
 2094 Street One Network Drive
 2095 City, State, Postal Code Burlington, MA 01803-0903
 2096 Country USA
 2097 Phone: +1 (781) 442-3063
 2098 EMail: chris.ferris@sun.com

2100 **Team Editor**

2101 Name David Burdett
 2102 Company Commerce One
 2103 Street 4400 Rosewood Drive
 2104 City, State, Postal Code Pleasanton, CA 94588
 2105 Country USA
 2106 Phone: +1 (925) 520-4422
 2107 EMail: david.burdett@commerceone.com

2109 **Authors**

2110 Name Dick Brooks
 2111 Company Group 8760
 2112 Street 110 12th Street North, Suite F103
 2113 City, State, Postal Code Birmingham, Alabama 35203
 2114 Phone: +1 (205) 250-8053
 2115 Email: dick@8760.com

2117 Name David Burdett
 2118 Company Commerce One
 2119 Street 4400 Rosewood Drive
 2120 City, State, Postal Code Pleasanton, CA 94588
 2121 Country USA
 2122 Phone: +1 (925) 520-4422
 2123 EMail: david.burdett@commerceone.com

2125 Name Christopher Ferris
 2126 Company Sun Microsystems
 2127 Street One Network Drive
 2128 City, State, Postal Code Burlington, MA 01803-0903
 2129 Country USA
 2130 Phone: +1 (781) 442-3063
 2131 EMail: chris.ferris@east.sun.com

2132
 2133 Name John Ibbotson
 2134 Company IBM UK Ltd

2135 Street Hursley Park
 2136 City, State, Postal Code Winchester SO21 2JN
 2137 Country United Kingdom
 2138 Phone: +44 (1962) 815188
 2139 Email: john_ibbotson@uk.ibm.com
 2140
 2141 Name Masayoshi Shimamura
 2142 Company Fujitsu Limited
 2143 Street Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
 2144 City, State, Postal Code Kohoku-ku, Yokohama 222-0033, Japan
 2145 Phone: +81-45-476-4590
 2146 EMail: shima@rp.open.cs.fujitsu.co.jp
 2147
2148 Document Editing Team
 2149 Name Ralph Berwanger
 2150 Company bTrade.com
 2151 Street 2324 Gateway Drive
 2152 City, State, Postal Code Irving, TX 75063
 2153 Country USA
 2154 Phone: +1 (972) 580-3970
 2155 EMail: rberwanger@btrade.com
 2156
 2157 Name Colleen Evans
 2158 Company Progress/Sonic Software
 2159 Street 14 Oak Park
 2160 City,State,Postal Code Bedford, MA 01730
 2161 Country USA
 2162 Phone +1 (720) 480-3919
 2163 Email cevans@progress.com
 2164
 2165 Name Ian Jones
 2166 Company British Telecommunications
 2167 Street Enterprise House, 84-85 Adam Street
 2168 City, State, Postal Code Cardiff, CF24 2XF
 2169 Country United Kingdom
 2170 Phone: +44 29 2072 4063
 2171 EMail: ian.c.jones@bt.com
 2172
 2173 Name Martha Warfelt
 2174 Company DaimlerChrysler Corporation
 2175 Street 800 Chrysler Drive
 2176 City, State, Postal Code Auburn Hills, MI
 2177 Country USA
 2178 Phone: +1 (248) 944-5481
 2179 EMail: maw2@daimlerchrysler.com
 2180
 2181 Name David Fischer
 2182 Company Drummond Group, Inc
 2183 Street 5008 Bentwood Ct
 2184 City, State, Postal Code Fort Worth, TX 76132
 2185 Phone +1 (817-294-7339
 2186 EMail david@drummondgroup.com

2187 Appendix AebXML SOAP Extension Elements Schema

2188 The ebXML SOAP extension elements schema has been specified using the Candidate
 2189 Recommendation draft of the XMLSchema specification[XMLSchema]. Because ebXML has
 2190 adopted SOAP 1.1 for the message format, and because the SOAP 1.1 schema resolved by the
 2191 SOAP 1.1 namespace URI was written to an earlier draft of the XMLSchema specification, the
 2192 ebXML TRP team has created a version of the SOAP 1.1 envelope schema that is specified
 2193 using the schema vocabulary that conforms to the W3C XMLSchema Candidate
 2194 Recommendation specification[XMLSchema].

2195 In addition, it was necessary to craft a schema for the [XLINK] attribute vocabulary and for the
 2196 XML xml:lang attribute.

2197 Finally, because certain authoring tools do not correctly resolve local entities when importing
 2198 schema, a version of the W3C XMLSignature Core schema has also been provided and
 2199 referenced by the ebXML SOAP extension elements schema defined in this Appendix.

2200 These alternative schema SHALL be available from the following URL's:

2201 XMLSignature Core – http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd

2202 Xlink - http://ebxml.org/project_teams/transport/xlink.xsd

2203 xml:lang - http://ebxml.org/project_teams/transport/xml_lang.xsd

2204 SOAP1.1 - http://ebxml.org/project_teams/transport/envelope.xsd

2205 Note: if inconsistencies exist between the specification and this schema, the specification supersedes this example
 2206 schema.

```

2207
2208 <?xml version="1.0" encoding="UTF-8"?>
2209 <schema targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2210 xmlns:xml="http://www.w3.org/XML/1998/namespace"
2211 xmlns:tns="http://www.ebxml.org/namespaces/messageHeader"
2212 xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xlink="http://www.w3.org/1999/xlink"
2213 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
2214 xmlns="http://www.w3.org/2000/10/XMLSchema" version="1.0">
2215   <import namespace="http://www.w3.org/2000/09/xmldsig#"
2216   schemaLocation="http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"/>
2217   <import namespace="http://www.w3.org/1999/xlink"
2218   schemaLocation="http://ebxml.org/project_teams/transport/xlink.xsd"/>
2219   <import namespace="http://schemas.xmlsoap.org/soap/envelope/"
2220   schemaLocation="http://ebxml.org/project_teams/transport/envelope.xsd"/>
2221   <import namespace="http://www.w3.org/XML/1998/namespace"
2222   schemaLocation="http://ebxml.org/project_teams/transport/xml_lang.xsd"/>
2223   <!-- MANIFEST -->
2224   <element name="Manifest">
2225     <complexType>
2226       <sequence>
2227         <element ref="tns:Reference" maxOccurs="unbounded"/>
2228         <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2229       </sequence>
2230       <attribute ref="tns:id"/>
2231       <attribute ref="tns:version"/>
2232       <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2233       processContents="lax"/>
2234     </complexType>
2235   </element>
2236   <element name="Reference">
2237     <complexType>
2238       <sequence>
2239         <element ref="tns:Schema" minOccurs="0" maxOccurs="unbounded"/>
2240         <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
2241         <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2242       </sequence>

```

```

2243     <attribute ref="tns:id"/>
2244     <attribute ref="xlink:type" use="fixed" value="simple"/>
2245     <attribute ref="xlink:href" use="required"/>
2246     <attribute ref="xlink:role"/>
2247   </complexType>
2248 </element>
2249 <element name="Schema">
2250   <complexType>
2251     <attribute name="location" type="uriReference" use="required"/>
2252     <attribute name="version" type="tns:non-empty-string"/>
2253   </complexType>
2254 </element>
2255 <!-- MESSAGEHEADER -->
2256 <element name="MessageHeader">
2257   <complexType>
2258     <sequence>
2259       <element ref="tns:From"/>
2260       <element ref="tns:To"/>
2261       <element ref="tns:CPAId"/>
2262       <element ref="tns:ConversationId"/>
2263       <element ref="tns:Service"/>
2264       <element ref="tns:Action"/>
2265       <element ref="tns:MessageData"/>
2266       <element ref="tns:QualityOfServiceInfo" minOccurs="0"/>
2267       <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
2268       <element ref="tns:SequenceNumber" minOccurs="0"/>
2269     </sequence>
2270     <attribute ref="tns:id"/>
2271     <attribute ref="tns:version"/>
2272     <attribute ref="soap:mustUnderstand"/>
2273     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2274 processContents="lax"/>
2275   </complexType>
2276 </element>
2277 <element name="CPAId" type="tns:non-empty-string"/>
2278 <element name="ConversationId" type="tns:non-empty-string"/>
2279 <element name="Service" type="tns:non-empty-string"/>
2280 <element name="Action" type="tns:non-empty-string"/>
2281 <element name="MessageData">
2282   <complexType>
2283     <sequence>
2284       <element ref="tns:MessageId"/>
2285       <element ref="tns:Timestamp"/>
2286       <element ref="tns:RefToMessageId" minOccurs="0"/>
2287       <element ref="tns:TimeToLive" minOccurs="0"/>
2288     </sequence>
2289   </complexType>
2290 </element>
2291 <element name="MessageId" type="tns:non-empty-string"/>
2292 <element name="TimeToLive" type="timeDuration"/>
2293 <!--
2294 <element name="TimeToLive" type="duration"/>
2295 -->
2296 <element name="QualityOfServiceInfo">
2297   <complexType>
2298     <attribute name="deliverySemantics" type="tns:deliverySemantics.type" use="default"
2299 value="BestEffort"/>
2300     <attribute name="messageOrderSemantics" type="tns:messageOrderSemantics.type"
2301 use="default" value="NotGuaranteed"/>
2302     <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type" use="default"
2303 value="None"/>
2304   </complexType>
2305 </element>
2306 <!-- TRACE HEADER LIST -->
2307 <element name="TraceHeaderList">
2308   <complexType>
2309     <sequence>
2310       <element ref="tns:TraceHeader" maxOccurs="unbounded"/>
2311     </sequence>
2312     <attribute ref="tns:id"/>
2313     <attribute ref="tns:version"/>

```

```

2314     <attribute ref="soap:mustUnderstand" use="required" />
2315     <attribute ref="soap:actor" use="required" />
2316     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2317 processContents="lax" />
2318   </complexType>
2319 </element>
2320 <element name="TraceHeader">
2321   <complexType>
2322     <sequence>
2323       <element ref="tns:Sender" />
2324       <element ref="tns:Receiver" />
2325       <element ref="tns:Timestamp" />
2326       <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
2327     </sequence>
2328     <attribute ref="tns:id" />
2329   </complexType>
2330 </element>
2331 <element name="Sender" type="tns:senderReceiver.type" />
2332 <element name="Receiver" type="tns:senderReceiver.type" />
2333 <element name="SequenceNumber" type="positiveInteger" />
2334 <!-- DELIVERY RECEIPT -->
2335 <element name="DeliveryReceipt">
2336   <complexType>
2337     <sequence>
2338       <element ref="tns:Timestamp" />
2339       <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded" />
2340     </sequence>
2341     <attribute ref="tns:id" />
2342     <attribute ref="tns:version" />
2343     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2344 processContents="lax" />
2345     <!-- <attribute name="signed" type="boolean" /> -->
2346   </complexType>
2347 </element>
2348 <!-- ACKNOWLEDGEMENT -->
2349 <element name="Acknowledgment">
2350   <complexType>
2351     <sequence>
2352       <element ref="tns:Timestamp" />
2353       <element ref="tns:From" minOccurs="0" />
2354       <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded" />
2355     </sequence>
2356     <attribute ref="tns:id" />
2357     <attribute ref="tns:version" />
2358     <attribute ref="soap:mustUnderstand" use="required" />
2359     <attribute ref="soap:actor" use="required" />
2360     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2361 processContents="lax" />
2362   </complexType>
2363 </element>
2364 <!-- ERROR LIST -->
2365 <element name="ErrorList">
2366   <complexType>
2367     <sequence>
2368       <element ref="tns:Error" maxOccurs="unbounded" />
2369     </sequence>
2370     <attribute ref="tns:id" />
2371     <attribute ref="tns:version" />
2372     <attribute ref="soap:mustUnderstand" use="required" />
2373     <attribute name="highestSeverity" type="tns:severity.type" use="default"
2374 value="Warning" />
2375     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2376 processContents="lax" />
2377   </complexType>
2378 </element>
2379 <element name="Error">
2380   <complexType>
2381     <attribute ref="tns:id" />
2382     <attribute name="codeContext" type="uriReference" use="required" />
2383     <attribute name="errorCode" type="tns:non-empty-string" use="required" />
2384     <attribute name="severity" type="tns:severity.type" use="default" value="Warning" />

```

```

2385     <attribute name="location" type="tns:non-empty-string"/>
2386     <attribute ref="xml:lang"/>
2387     <attribute name="errorMessage" type="tns:non-empty-string"/>
2388   </complexType>
2389 </element>
2390 <!-- STATUS RESPONSE -->
2391 <element name="StatusResponse">
2392   <complexType>
2393     <sequence>
2394       <element ref="tns:RefToMessageId"/>
2395       <element ref="tns:Timestamp" minOccurs="0"/>
2396     </sequence>
2397     <attribute ref="tns:id"/>
2398     <attribute ref="tns:version"/>
2399     <attribute name="messageStatus" type="tns:messageStatus.type"/>
2400     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2401 processContents="lax"/>
2402   </complexType>
2403 </element>
2404 <!-- STATUS REQUEST -->
2405 <element name="StatusRequest">
2406   <complexType>
2407     <sequence>
2408       <element ref="tns:RefToMessageId"/>
2409     </sequence>
2410     <attribute ref="tns:id"/>
2411     <attribute ref="tns:version"/>
2412     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2413 processContents="lax"/>
2414   </complexType>
2415 </element>
2416 <!-- VIA -->
2417 <element name="Via">
2418   <complexType>
2419     <sequence>
2420       <element ref="tns:CPAId" minOccurs="0"/>
2421       <element ref="tns:Service" minOccurs="0"/>
2422       <element ref="tns:Action" minOccurs="0"/>
2423     </sequence>
2424     <attribute ref="tns:id"/>
2425     <attribute ref="tns:version"/>
2426     <attribute ref="soap:mustUnderstand" use="required"/>
2427     <attribute ref="soap:actor" use="required"/>
2428     <attribute name="syncReply" type="boolean"/>
2429     <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type" use="default"
2430 value="None"/>
2431     <attribute name="reliableMessagingMethod" type="tns:rmm.type"/>
2432     <attribute name="ackRequested" type="boolean"/>
2433     <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2434 processContents="lax"/>
2435   </complexType>
2436 </element>
2437 <!-- COMMON TYPES -->
2438 <complexType name="senderReceiver.type">
2439   <sequence>
2440     <element ref="tns:PartyId" maxOccurs="unbounded"/>
2441     <element name="Location" type="uriReference"/>
2442   </sequence>
2443 </complexType>
2444 <simpleType name="messageStatus.type">
2445   <restriction base="NMTOKEN">
2446     <enumeration value="Unauthorized"/>
2447     <enumeration value="NotRecognized"/>
2448     <enumeration value="Received"/>
2449     <enumeration value="Processed"/>
2450     <enumeration value="Forwarded"/>
2451   </restriction>
2452 </simpleType>
2453 <simpleType name="type.type">
2454   <restriction base="NMTOKEN">
2455     <enumeration value="DeliveryReceipt"/>

```

```

2456     <enumeration value="IntermediateAck" />
2457   </restriction>
2458 </simpleType>
2459 <simpleType name="messageOrderSemantics.type">
2460   <restriction base="NMTOKEN">
2461     <enumeration value="Guaranteed" />
2462     <enumeration value="NotGuaranteed" />
2463   </restriction>
2464 </simpleType>
2465 <simpleType name="deliverySemantics.type">
2466   <restriction base="NMTOKEN">
2467     <enumeration value="OnceAndOnlyOnce" />
2468     <enumeration value="BestEffort" />
2469   </restriction>
2470 </simpleType>
2471 <simpleType name="non-empty-string">
2472   <restriction base="string">
2473     <minLength value="1" />
2474   </restriction>
2475 </simpleType>
2476 <simpleType name="rmm.type">
2477   <restriction base="NMTOKEN">
2478     <enumeration value="ebXML" />
2479     <enumeration value="Transport" />
2480   </restriction>
2481 </simpleType>
2482 <simpleType name="signedUnsigned.type">
2483   <restriction base="NMTOKEN">
2484     <enumeration value="Signed" />
2485     <enumeration value="Unsigned" />
2486     <enumeration value="None" />
2487   </restriction>
2488 </simpleType>
2489 <simpleType name="severity.type">
2490   <restriction base="NMTOKEN">
2491     <enumeration value="Warning" />
2492     <enumeration value="Error" />
2493   </restriction>
2494 </simpleType>
2495 <!-- COMMON ATTRIBUTES and ELEMENTS -->
2496 <attribute name="id" type="ID" form="unqualified" />
2497 <attribute name="version" type="tns:non-empty-string" use="fixed" value="1.0" />
2498 <element name="PartyId">
2499   <complexType>
2500     <simpleContent>
2501       <extension base="tns:non-empty-string">
2502         <attribute name="type" type="tns:non-empty-string" />
2503       </extension>
2504     </simpleContent>
2505   </complexType>
2506 </element>
2507 <element name="To">
2508   <complexType>
2509     <sequence>
2510       <element ref="tns:PartyId" maxOccurs="unbounded" />
2511     </sequence>
2512   </complexType>
2513 </element>
2514 <element name="From">
2515   <complexType>
2516     <sequence>
2517       <element ref="tns:PartyId" maxOccurs="unbounded" />
2518     </sequence>
2519   </complexType>
2520 </element>
2521 <element name="Description">
2522   <complexType>
2523     <simpleContent>
2524       <extension base="tns:non-empty-string">
2525         <attribute ref="xml:lang" />
2526       </extension>

```

```
2527     </simpleContent>
2528   </complexType>
2529 </element>
2530 <element name="RefToMessageId" type="tns:non-empty-string"/>
2531 <element name="Timestamp" type="timeInstant"/>
2532 <!--
2533 <element name="Timestamp" type="dateTime"/>
2534 -->
2535 </schema>
2536
```

2537 **Appendix B Communication Protocol Bindings**

2538 **B.1 Introduction**

2539 One of the goals of ebXML's Transport, Routing and Packaging team is to design a message
2540 handling service that is usable over a variety of network and application level communication
2541 protocols. These protocols serve as the "carrier" of ebXML Messages and provide the underlying
2542 services necessary to carry out a complete ebXML Message exchange between two parties.
2543 HTTP, FTP, Java Message Service (JMS) and SMTP are examples of application level
2544 communication protocols. TCP and SNA/LU6.2 are examples of network transport protocols.
2545 Communication protocols vary in their support for data content, processing behavior and error
2546 handling and reporting. For example, it is customary to send binary data in raw form over HTTP.
2547 However, in the case of SMTP it is customary to "encode" binary data into a 7-bit representation.
2548 HTTP is equally capable of carrying out synchronous or asynchronous message exchanges
2549 whereas it is likely that message exchanges occurring over SMTP will be asynchronous. This
2550 section describes the technical details needed to implement this abstract ebXML Message
2551 Handling Service over particular communication protocols.

2552 This section specifies communication protocol bindings and technical details for carrying *ebXML*
2553 *Message Service* messages for the following communication protocols:

- 2554 • Hypertext Transfer Protocol [HTTP], in both asynchronous and synchronous forms of
2555 transfer.
- 2556 • Simple Mail Transfer Protocol [SMTP], in asynchronous form of transfer only.

2557 **B.2 HTTP**

2558 **B.2.1 Minimum level of HTTP protocol**

2559 Hypertext Transfer Protocol Version 1.1 [HTTP] (<http://www.ietf.org/rfc2616.txt>) is the minimum
2560 level of protocol that MUST be used.

2561 **B.2.2 Sending ebXML Service messages over HTTP**

2562 Even though several HTTP request methods are available, this specification only defines the use
2563 of HTTP POST requests for sending *ebXML Message Service* messages over HTTP. The identity
2564 of the ebXML MSH (e.g. ebxmlhandler) may be part of the HTTP POST request:

2565
2566 `POST /ebxmlhandler HTTP/1.1`
2567

2568 Prior to sending over HTTP, an ebXML Message MUST be formatted according to ebXML
2569 Message Service Specification sections 7 and 0. Additionally, the messages MUST conform to
2570 the HTTP specific MIME canonical form constraints specified in section 19.4 of RFC 2616 [HTTP]
2571 specification (see: <http://www.ietf.org/rfc2616.txt>).

2572 HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is OPTIONAL
2573 for such parts in an ebXML Service Message prior to sending over HTTP. However, content-
2574 transfer-encoding of such parts (e.g. using base64 encoding scheme) is not precluded by this
2575 specification.

2576 The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- 2577 • The **Content-Type: Multipart/Related** MIME header with the associated
2578 parameters, from the ebXML Service Message Envelope **MUST** appear as an HTTP
2579 header.
 - 2580 • All other MIME headers that constitute the ebXML Message Envelope **MUST** also
2581 become part of the HTTP header.
 - 2582 • The mandatory SOAPAction HTTP header field must also be included in the HTTP
2583 header and **MAY** have a value of "ebXML".
- 2584 **SOAPAction: "ebXML"**
- 2585 • Other headers with semantics defined by MIME specifications, such as Content-Transfer-
2586 Encoding, **SHALL NOT** appear as HTTP headers. Specifically, the "MIME-Version: 1.0"
2587 header **MUST NOT** appear as an HTTP header. However, HTTP-specific MIME-like
2588 headers defined by HTTP 1.1 **MAY** be used with the semantic defined in the HTTP
2589 specification.
 - 2590 • All ebXML Service Message parts that follow the ebXML Message Envelope, including
2591 the MIME boundary string, constitute the HTTP entity body. This encompasses the SOAP
2592 envelope and the constituent ebXML parts and attachments including the trailing MIME
2593 boundary strings.

2594 The example below shows an example instance of an HTTP POST'ed ebXML Service Message:

```

2595 POST /servlet/ebXMLhandler HTTP/1.1
2596 Host: www.example2.com
2597 SOAPAction: "ebXML"
2598 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2599 start=" <ebxhheader111@example.com>"
2600
2601 --Boundary
2602 Content-ID: <ebxhheader111@example.com>
2603 Content-Type: text/xml
2604 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2605 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2606 <SOAP-ENV:Header>
2607 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2608 <eb:From>
2609 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2610 </eb:From>
2611 <eb:To>
2612 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2613 </eb:To>
2614 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2615 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2616 <eb:Service>OrderProcessing</eb:Service>
2617 <eb:Action>NewOrder</eb:Action>
2618 <eb:MessageData>
2619 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2620 <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
2621 </eb:MessageData>
2622 <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort"/>
2623 </eb:MessageHeader>
2624 </SOAP-ENV:Header>
2625 <SOAP-ENV:Body>
2626 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2627 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2628 xlink:role="XLinkRole"
2629 xlink:type="simple">
2630 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2631 </eb:Reference>
2632 </eb:Manifest>
2633 </SOAP-ENV:Body>
2634 </SOAP-ENV:Envelope>
2635
2636 --Boundary
2637 Content-ID: <ebxmlpayload111@example.com>
2638 Content-Type: text/xml
    
```

```
2640 <?xml version="1.0" encoding="UTF-8"?>
2641 <purchase_order>
2642   <po_number>1</po_number>
2643   <part_number>123</part_number>
2644   <price currency="USD">500.00</price>
2645 </purchase_order>
2646
2647 --Boundary--
```

2648 B.2.3 HTTP Response Codes

2649 In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be
2650 followed, for returning the HTTP level response codes. A 2xx code MUST be returned when the
2651 HTTP Posted message is successfully received by the receiving HTTP entity. However, see
2652 exception for SOAP error conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx
2653 range MAY be returned for conditions corresponding to them. However, error conditions
2654 encountered while processing an ebXML Service Message MUST be reported using the error
2655 mechanism defined by the ebXML Message Service Specification (see section 11).

2656 B.2.4 SOAP Error conditions and Synchronous Exchanges

2657 The SOAP 1.1 specification states:

2658 *"In case of a SOAP error while processing the request, the SOAP HTTP server MUST issue an*
2659 *HTTP 500 "Internal Server Error" response and include a SOAP message in the response*
2660 *containing a SOAP Fault element indicating the SOAP processing error."*

2661 However, the scope of the SOAP 1.1 specification is limited to synchronous mode of message
2662 exchange over HTTP, whereas the ebXML Message Service Specification specifies both
2663 synchronous and asynchronous modes of message exchange over HTTP. Hence, the SOAP 1.1
2664 specification MUST be followed for synchronous mode of message exchange, where the SOAP
2665 Message containing a SOAP Fault element indicating the SOAP processing error MUST be
2666 returned in the HTTP response with a response code of "HTTP 500 Internal Server Error". When
2667 asynchronous mode of message exchange is being used, a HTTP response code in the range
2668 2xx MUST be returned when the message is received successfully and any error conditions
2669 (including SOAP errors) must be returned via a separate HTTP Post.

2670 B.2.5 Synchronous vs. Asynchronous

2671 When the **syncReply** parameter in the **Via** element is set to "true", the response message(s)
2672 MUST be returned on the same HTTP connection as the inbound request, with an appropriate
2673 HTTP response code, as described above. When the **syncReply** parameter is set to "false", the
2674 response messages are not returned on the same HTTP connection as the inbound request, but
2675 using an independent HTTP Post request. An HTTP response with a response code as defined
2676 in section B.2.3 above and with an empty HTTP body MUST be returned in response to the HTTP
2677 Post.

2678 B.2.6 Access Control

2679 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access
2680 through the use of an access control mechanism. The HTTP access authentication process
2681 described in "HTTP Authentication: Basic and Digest Access Authentication" [RFC2617] defines
2682 the access control mechanisms allowed to protect an ebXML Message Service Handler from
2683 unauthorized access.

2684 Implementers MAY support all of the access control schemes defined in [RFC2617] however they
2685 MUST support the Basic Authentication mechanism, as described in section 2, when Access
2686 Control is used.

2687 Implementers that use basic authentication for access control SHOULD also use communication
2688 protocol level security, as specified in the section titled "Confidentiality and Communication
2689 Protocol Level Security" in this document.

2690 **B.2.7 Confidentiality and Communication Protocol Level Security**

2691 An ebXML Message Service Handler MAY use transport layer encryption to protect the
2692 confidentiality of ebXML Messages and HTTP transport headers. The IETF Transport Layer
2693 Security specification [RFC2246] provides the specific technical details and list of allowable
2694 options, which may be used by ebXML Message Service Handlers. ebXML Message Service
2695 Handlers MUST be capable of operating in backwards compatibility mode with SSL [SSL3], as
2696 defined in Appendix E of [RFC2246].

2697 ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key
2698 sizes specified within [RFC2246]. At a minimum ebXML Message Service Handlers MUST
2699 support the key sizes and algorithms necessary for backward compatibility with [SSL3].

2700 The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that
2701 stronger encryption keys/algorithms SHOULD be used.

2702 Both [RFC2246] and [SSL3] require the use of server side digital certificates. In addition client
2703 side certificate based authentication is also permitted. ebXML Message Service handlers MUST
2704 support 3rd party signed certificates as well as "self signed" certificates.

2705 **B.3 SMTP**

2706 The Simple Mail Transfer Protocol [SMTP] and its companion documents [RFC822] and [ESMTP]
2707 makeup the suite of specifications commonly referred to as Internet Electronic Mail. These
2708 specifications have been augmented over the years by other specifications, which define
2709 additional functionality "layered on top" of these baseline specifications. These include:

- 2710 • Multipurpose Internet Mail Extensions (MIME) [RFC2045], [RFC2046], [RFC2387]
- 2711 • SMTP Service Extension for Authentication [RFC2554]
- 2712 • SMTP Service Extension for Secure SMTP over TLS [RFC2487]

2713 Typically, Internet Electronic Mail Implementations consist of two "agent" types:

- 2714 • Message Transfer Agent (MTA): Programs that send and receive mail messages with
2715 other MTA's on behalf of MUA's. Microsoft Exchange Server is an example of a MTA
- 2716 • Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail
2717 messages and communicate with an MTA to send/retrieve mail messages. Microsoft
2718 Outlook is an example of a MUA.

2719 MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

2720 MUA's are responsible for constructing electronic mail messages in accordance with the Internet
2721 Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML
2722 compliant message for transport via eMail from the perspective of a MUA. No attempt is made to
2723 define the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.

2724 **B.3.1 Minimum level of supported protocols**

- 2725 • Simple Mail Transfer Protocol [RFC821] and [RFC822]
- 2726 • MIME [RFC2045] and [RFC2046]
- 2727 • Multipart/Related MIME [RFC2387]

2728 **B.3.2 Sending ebXML Messages over SMTP**

2729

2730 Prior to sending messages over SMTP an ebXML Message MUST be formatted according to
 2731 ebXML Message Service Specification sections 7 and 0. Additionally the messages must also
 2732 conform to the syntax, format and encoding rules specified by MIME [RFC2045], [RFC2046] and
 2733 [RFC2387].

2734 Many types of data that a party might desire to transport via email are represented as 8bit
 2735 characters or binary data. Such data cannot be transmitted over SMTP [SMTP], which restricts
 2736 mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including any
 2737 trailing CRLF line separator. If a sending Message Service Handler knows that a receiving MTA,
 2738 or ANY intermediary MTA's, are restricted to handling 7-bit data then any document part that
 2739 uses 8 bit (or binary) representation must be "transformed" according to the encoding rules
 2740 specified in section 6 of [RFC2045]. In cases where a Message Service Handler knows that a
 2741 receiving MTA and ALL intermediary MTA's are capable of handling 8-bit data then no
 2742 transformation is needed on any part of the ebXML Message.

2743 The rules for forming an ebXML Message for transport via SMTP are as follows:

- 2744 • If using [RFC821] restricted transport paths, apply transfer encoding to all 8-bit data that
 2745 will be transported in an ebXML message, according to the encoding rules defined in
 2746 section 6 of [RFC2045]. The Content-Transfer-Encoding MIME header MUST be
 2747 included in the MIME envelope portion of any body part that has been transformed
 2748 (encoded).
- 2749 • The Content-Type: Multipart/Related MIME header with the associated
 2750 parameters, from the ebXML Message Envelope MUST appear as an eMail MIME
 2751 header.
- 2752 • All other MIME headers that constitute the ebXML Message Envelope MUST also
 2753 become part of the eMail MIME header.
- 2754 • The SOAPAction MIME header field must also be included in the eMail MIME header and
 2755 MAY have the value of ebXML:

2756 SOAPAction: "ebXML"

2757 Where Service and Action are values of the corresponding elements from the ebXML
 2758 MessageHeader.

- 2759 • The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- 2760 • The eMail header "To:" MUST contain the [RFC822] compliant eMail address of the
 2761 ebXML Message Service Handler.
- 2762 • The eMail header "From:" MUST contain the [RFC822] compliant eMail address of the
 2763 senders ebXML Message Service handler.
- 2764 • Construct a "Date:" eMail header in accordance with [RFC822]
- 2765 • Other headers MAY occur within the eMail message header in accordance with [RFC822]
 2766 and [RFC2045], however ebXML Message Service Handlers MAY choose to ignore
 2767 them.

2768 The example below shows a minimal example of an eMail message containing an ebXML
 2769 Message:

```

2770
2771 From: ebXMLhandler@example.com
2772 To: ebXMLhandler@example2.com
2773 Date: Thu, 08 Feb 2001 19:32:11 CST
2774 MIME-Version: 1.0
2775 SOAPAction: "ebXML"
2776 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2777 start=" <ebxhmheader111@example.com>"
2778
2779 --Boundary
2780 Content-ID: <ebxhmheader111@example.com>
2781 Content-Type: text/xml
2782
2783 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2784 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2785 <SOAP-ENV:Header>
2786 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2787 <eb:From>
2788 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2789 </eb:From>
2790 <eb:To>
2791 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2792 </eb:To>
2793 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2794 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2795 <eb:Service>OrderProcessing</eb:Service>
2796 <eb:Action>NewOrder</eb:Action>
2797 <eb:MessageData>
2798 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2799 <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
2800 </eb:MessageData>
2801 <eb:QualityOfServiceInfo eb:deliverySemantics="BestEffort"/>
2802 </eb:MessageHeader>
2803 </SOAP-ENV:Header>
2804 <SOAP-ENV:Body>
2805 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2806 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2807 xlink:role="XLinkRole"
2808 xlink:type="simple">
2809 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2810 </eb:Reference>
2811 </eb:Manifest>
2812 </SOAP-ENV:Body>
2813 </SOAP-ENV:Envelope>
2814
2815 --Boundary
2816 Content-ID: <ebxhmheader111@example.com>
2817 Content-Type: text/xml
2818 <?xml version="1.0" encoding="UTF-8"?>
2819 <purchase_order>
2820 <po_number>1</po_number>
2821 <part_number>123</part_number>
2822 <price currency="USD">500.00</price>
2823 </purchase_order>
2824
2825 --Boundary--

```

2826 B.3.3 Response Messages

2827 All ebXML response messages, including errors and acknowledgements, are delivered
 2828 asynchronously between ebXML Message Service Handlers. Each response message MUST be
 2829 constructed in accordance with the rules specified in the section titled "Sending ebXML
 2830 messages over SMTP" elsewhere in this document.

2831 ebXML Message Service Handlers MUST be capable of receiving a delivery failure notification
 2832 message sent by an MTA. A MSH that receives a delivery failure notification message SHOULD
 2833 examine the message to determine which ebXML message, sent by the MSH, resulted in a
 2834 message delivery failure. The MSH SHOULD attempt to identify the application responsible for
 2835 sending the offending message that caused the failure. The MSH SHOULD attempt to notify the
 2836 application that a message delivery failure has occurred. If the MSH is unable to determine the
 2837 source of the offending message the MSH administrator should be notified.

2838 MSH's which cannot identify a received message as a valid ebXML message or a message
 2839 delivery failure SHOULD retain the unidentified message in a "dead letter" folder.

2840 A MSH SHOULD place an entry in an audit log indicating the disposition of each received
 2841 message.

2842 **B.3.4 Access Control**

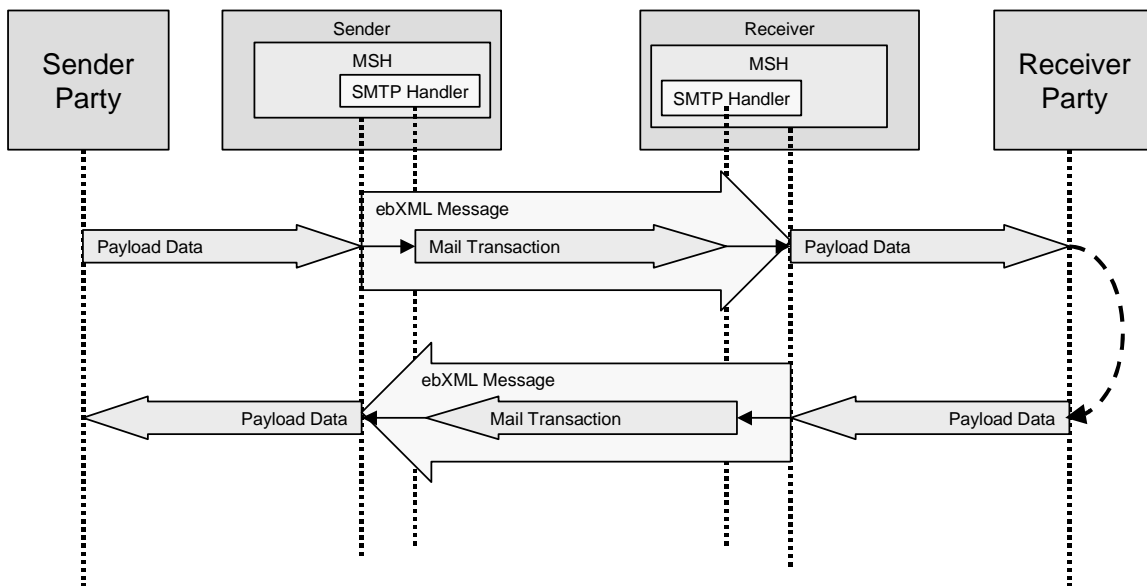
2843 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access
 2844 through the use of an access control mechanism. The SMTP access authentication process
 2845 described in "SMTP Service Extension for Authentication" [RFC2554] defines the ebXML
 2846 recommended access control mechanism to protect a SMTP based ebXML Message Service
 2847 Handler from unauthorized access.

2848 **B.3.5 Confidentiality and Communication Protocol Level Security**

2849 An ebXML Message Service Handler MAY use transport layer encryption to protect the
 2850 confidentiality of ebXML messages. The IETF "SMTP Service Extension for Secure SMTP over
 2851 TLS" specification [RFC2487] provides the specific technical details and list of allowable options,
 2852 which may be used.

2853 **B.3.6 SMTP Model**

2854 All *ebXML Message Service* messages carried as mail in an [SMTP] Mail Transaction as shown
 2855 in the figure below.



2856
 2857

2858 B.4 Communication Errors during Reliable Messaging

2859 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP,
2860 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport
2861 recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does
2862 Reliable Messaging recovery take place (see section 10).

2863 **Copyright Statement**

2864 Copyright © ebXML 2001. All Rights Reserved.

2865 This document and translations of it MAY be copied and furnished to others, and derivative works
2866 that comment on or otherwise explain it or assist in its implementation MAY be prepared, copied,
2867 published and distributed, in whole or in part, without restriction of any kind, provided that the
2868 above copyright notice and this paragraph are included on all such copies and derivative works.
2869 However, this document itself MAY not be modified in any way, such as by removing the
2870 copyright notice or references to the ebXML, UN/CEFACT, or OASIS, except as required to
2871 translate it into languages other than English.

2872 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
2873 successors or assigns.

2874 This document and the information contained herein is provided on an "AS IS" basis and ebXML
2875 disclaims all warranties, express or implied, including but not limited to any warranty that the use
2876 of the information herein will not infringe any rights or any implied warranties of merchantability or
2877 fitness for a particular purpose.