



---

Creating A Single Global Electronic Market

## **Message Service Specification**

### **ebXML Transport, Routing & Packaging**

**Version 0.98**

6 March 2001

# 1 Status of this Document

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

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

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

*This version*

[http://www.ebxml.org/project\\_teams/transport/ebxml\\_message\\_service\\_specification\\_v-0.98.pdf](http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.98.pdf) *Latest version*

[http://www.ebxml.org/project\\_teams/transport/ebxml\\_message\\_service\\_specification\\_v-0.98.pdf](http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.98.pdf) *Previous version*

[http://www.ebxml.org/project\\_teams/transport/ebxml\\_message\\_service\\_specification\\_v-0.8\\_001110.pdf](http://www.ebxml.org/project_teams/transport/ebxml_message_service_specification_v-0.8_001110.pdf)

## 2 ebXML Participants

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

Ralph Berwanger – bTrade.com  
Jonathan Borden – Author of XMTP  
Jon Bosak – Sun Microsystems  
Marc Breissinger – webMethods  
Dick Brooks – Group 8760  
Doug Bunting – Ariba  
David Burdett – Commerce One  
Len Callaway – Drummond Group, Inc.  
David Craft – VerticalNet  
Philippe De Smedt – Viquity  
Lawrence Ding – WorldSpan  
Rik Drummond – Drummond Group, Inc.  
Andrew Eisenberg – Progress Software  
Colleen Evans—Progress / Sonic Software  
David Fischer, Drummond Group, Inc.  
Christopher Ferris – Sun Microsystems  
Robert Fox - Softshare  
Maryann Hondo – IBM  
Jim Hughes – Fujitsu  
John Ibbotson – IBM  
Ian Jones – British Telecommunications  
Ravi Kacker – Kraft Foods  
Henry Lowe – OMG  
Jim McCarthy – webXI  
Bob Miller – GSX  
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 – Columbine JDS Systems  
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
- 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 ebXML SOAP Extensions ..... 19
    - 8.2.3 #wildcard element content..... 19
  - 8.3 SOAP Header element..... 19
  - 8.4 MessageHeader element ..... 20
    - 8.4.1 version attribute..... 20
    - 8.4.2 SOAP mustUnderstand attribute ..... 20
  - 8.5 MessageHeader element description ..... 20
    - 8.5.1 From and To elements ..... 20
    - 8.5.2 CPAlid element ..... 21
    - 8.5.3 ConversationId element ..... 21
    - 8.5.4 Service element ..... 21
    - 8.5.5 Action element ..... 22
    - 8.5.6 MessageData element ..... 22

8.5.7	QualityOfServiceInfo element.....	23
8.5.8	eb:deliveryReceiptRequested="UnSigned"/> SequenceNumber element .....	24
8.5.9	Description element .....	25
8.5.10	MessageHeader sample .....	25
8.6	TraceHeaderList element .....	25
8.6.1	SOAP mustUnderstand attribute .....	26
8.6.2	version attribute.....	26
8.6.3	TraceHeader Element .....	26
8.6.4	Single Hop TraceHeader Sample.....	27
8.6.5	Multi-hop TraceHeader Sample .....	28
8.7	Via element .....	29
8.7.1	SOAP mustUnderstand attribute .....	29
8.7.2	SOAP actor attribute .....	29
8.7.3	version attribute.....	29
8.7.4	syncReply attribute.....	30
8.7.5	reliableMessagingMethod attribute .....	30
8.7.6	ackRequested attribute .....	30
8.7.7	CPAId element .....	30
8.7.8	Service and Action elements.....	30
8.7.9	Sample Via element .....	31
8.8	ErrorList element .....	31
8.8.1	id attribute .....	31
8.8.2	SOAP mustUnderstand attribute .....	31
8.8.3	version attribute.....	31
8.8.4	highestSeverity attribute.....	31
8.8.5	Error element .....	31
8.8.6	Examples .....	32
8.8.7	errorCode values.....	33
8.8.8	Reporting Errors in the ebXML Elements.....	33
8.8.9	Non-XML Document Errors.....	33
8.9	Signature element .....	34
8.10	SOAP Body Extensions .....	34
8.11	Manifest element .....	34
8.11.1	id attribute .....	35
8.11.2	SOAP mustUnderstand attribute .....	35
8.11.3	version attribute.....	35
8.11.4	Reference element.....	35
8.11.5	What References are Included in a Manifest.....	36
8.11.6	Manifest Validation.....	36
8.11.7	Manifest sample .....	36
8.12	StatusData Element .....	36
8.12.1	RefToMessageId element .....	37
8.12.2	Timestamp element.....	37
8.12.3	SOAP mustUnderstand attribute .....	37
8.12.4	version attribute.....	37
8.12.5	messageStatus attribute .....	37
8.13	Acknowledgment Element.....	37
8.13.1	Timestamp element.....	38
8.13.2	From element .....	38
8.13.3	SOAP mustUnderstand attribute .....	38
8.13.4	version attribute.....	38
8.13.5	type attribute .....	38
8.13.6	signed attribute.....	38
8.14	Combining ebXML SOAP Extension Elements.....	39

- 8.14.1 Manifest element..... 39
- 8.14.2 MessageHeader element ..... 39
- 8.14.3 TraceHeaderList element..... 39
- 8.14.4 StatusData element..... 39
- 8.14.5 ErrorList element..... 39
- 8.14.6 Acknowledgment element ..... 39
- 8.14.7 Signature element ..... 39
- 9 Message Service Handler Services ..... 40**
  - 9.1 Message Status Request Service ..... 40
    - 9.1.1 Message Status Request Message..... 40
    - 9.1.2 Message Status Response Message..... 40
    - 9.1.3 Security Considerations ..... 41
  - 9.2 Message Service Handler Ping Service..... 41
    - 9.2.1 Message Service Handler Ping Message ..... 41
    - 9.2.2 Message Service Handler Pong Message ..... 42
    - 9.2.3 Security Considerations ..... 42
- 10 Reliable Messaging ..... 43**
  - 10.1.1 Persistent Storage and System Failure..... 43
  - 10.1.2 Methods of Implementing Reliable Messaging..... 43
  - 10.2 Reliable Messaging Parameters ..... 43
    - 10.2.1 Delivery Semantics ..... 43
    - 10.2.2 Sync Reply ..... 44
    - 10.2.3 Time To Live ..... 44
    - 10.2.4 reliableMessagingMethod ..... 44
    - 10.2.5 AckRequested..... 45
    - 10.2.6 Timeout Parameter ..... 45
    - 10.2.7 Retries..... 45
    - 10.2.8 RetryInterval..... 45
    - 10.2.9 PersistDuration..... 45
  - 10.3 ebXML Reliable Messaging Protocol ..... 45
  - 10.4 Failed Message Delivery ..... 50
  - 10.5 MSH Time Accuracy..... 50
- 11 Error Reporting and Handling ..... 51**
  - 11.1 Definitions..... 51
  - 11.2 Types of Errors..... 51
  - 11.3 When to generate Error Messages ..... 51
    - 11.3.1 Security Considerations ..... 52
  - 11.4 Identifying the Error Reporting Location ..... 52
  - 11.5 Service and Action Element Values ..... 52
- 12 Security ..... 53**
  - 12.1 Security and Management ..... 53
  - 12.2 Collaboration Protocol Agreement ..... 53
  - 12.3 Countermeasure Technologies ..... 53
    - 12.3.1 Persistent Digital Signature ..... 53
    - 12.3.2 Persistent Signed Receipt..... 55
    - 12.3.3 Non-persistent Authentication ..... 55
    - 12.3.4 Non-persistent Integrity ..... 56
    - 12.3.5 Persistent Confidentiality..... 56
    - 12.3.6 Non-persistent Confidentiality ..... 56
    - 12.3.7 Persistent Authorization ..... 56

12.3.8	Non-persistent Authorization.....	56
12.3.9	Trusted Timestamp.....	56
<b>13</b>	<b>References .....</b>	<b>59</b>
13.1	Normative References .....	59
13.2	Non-Normative References.....	59
<b>14</b>	<b>Disclaimer .....</b>	<b>61</b>
<b>15</b>	<b>Contact Information .....</b>	<b>62</b>
<b>Appendix A ebXMLHeader Schema.....</b>		<b>64</b>
A.1	64	
<b>Appendix B Communication Protocol Bindings .....</b>		<b>69</b>
B.1	Introduction.....	69
B.2	HTTP .....	69
B.2.1	Minimum level of HTTP protocol.....	69
B.2.2	Sending ebXML Service messages over HTTP.....	69
B.2.3	HTTP Response Codes .....	71
B.2.4	SOAP Error conditions and Synchronous Exchanges.....	71
B.2.5	Synchronous vs. Asynchronous.....	71
B.2.6	Access Control .....	71
B.2.7	Confidentiality and Communication Protocol Level Security .....	72
B.3	SMTP .....	72
B.3.1	Minimum level of supported protocols.....	73
B.3.2	Sending ebXML Messages over SMTP.....	73
B.3.3	Response Messages.....	74
B.3.4	Access Control .....	75
B.3.5	Confidentiality and Communication Protocol Level Security .....	75
B.3.6	SMTP Model.....	75
B.4	Communication Errors during Reliable Messaging.....	76
<b>Copyright Statement .....</b>		<b>77</b>

## 1 4 Introduction

2 This is a draft standard for trial implementation. This specification is the one of a series of  
3 specifications. The main specification that is yet to be developed is the ebXML Service Interface  
4 specification that describes, in a language independent way, how an application or other process  
5 can interact with software that complies with this ebXML Message Service specification. The  
6 ebXML Service Interface specification is being developed as a separate document. It SHALL  
7 either be incorporated into a future version of this specification or referenced as an external  
8 specification as deemed most suitable by the ebXML Transport, Routing and Packaging Team.

### 9 4.1 Summary of Contents of Document

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

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

16 This specification is independent of both the payload and the communication protocol used,  
17 although Appendices to this specification describe how to use this specification with [RFC2068]  
18 and [RFC821].

19 This specification is organized around the following topics:

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

37 Appendices to this specification cover the following:

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



## 43 4.2 Document Conventions

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

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

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

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

## 73 4.3 Audience

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

## 76 4.4 Caveats and Assumptions

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

## 79 4.5 Related Documents

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

- 81 • **ebXML Message Services Requirements Specification** [EBXMLMSREQ] – defines the  
82 requirements of these Message Services
- 83 • **ebXML Technical Architecture** [EBXMLTA] – defines the overall technical architecture  
84 for ebXML
- 85 • **ebXML Technical Architecture Security Specification** [EBXMLTASEC] – defines the  
86 security mechanisms necessary to negate anticipated, selected threats

- 87 • **ebXML Collaboration Protocol Profile and Agreement Specification** [EBXMLTP]  
88 (under development) - defines how one party can discover and/or agree upon the  
89 information that party needs to know about another party prior to sending them a  
90 message that complies with this specification
- 91 • **ebXML Message Service Interface Specification** (to be developed) - defines an  
92 interface that may be used by software to interact with an ebXML Message Service
- 93 • **ebXML Registry/Repository Services Specification** [EBXMLRSS] – defines a registry  
94 service for the ebXML environment

## 95 **5 Design Objectives**

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

104 Every item in this specification will be prototyped by the ebXML Proof of Concept Team in order  
105 to ensure the clarity, accuracy and efficiency of this specification.

## 106 6 System Overview

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

### 112 6.1 What the Message Service does

113 The ebXML Message Service defines robust, yet basic, functionality to transfer messages using  
114 various existing communication protocols. The ebXML Message Service is structured to allow for  
115 messaging reliability, persistence, security and extensibility.

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

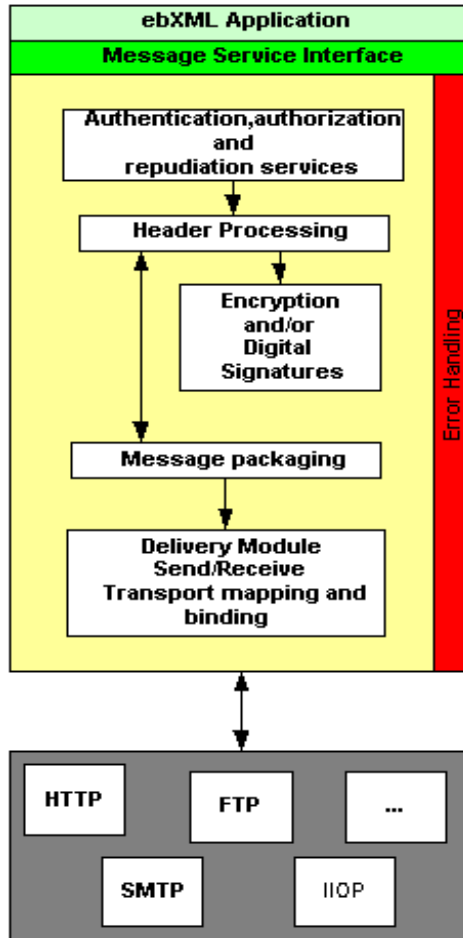
### 120 6.2 Message Service Overview

121 The *ebXML Messaging Service* may be conceptually broken down into following three parts: (1)  
122 an abstract *Service Interface*, (2) functions provided by the *Messaging Service Layer*, and (3) the  
123 mapping to underlying transport service(s).

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

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

147



148

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

## 150 7 Packaging Specification

### 151 7.1 Introduction

152 An ebXML Message has the following structure:

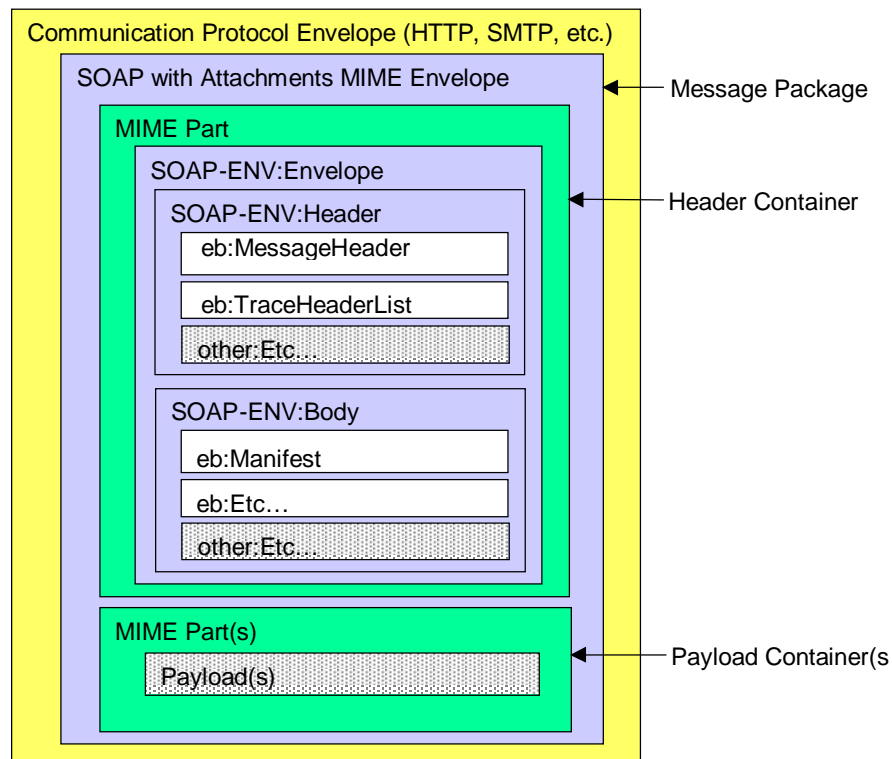
- 153 • a communication, “protocol independent” Multipart/MIME message envelope, structured  
154 in compliance with the Simple Object Access Protocol (SOAP) Messages with  
155 Attachments [SOAPATTACH] specification, referred to as the *Message Package*.
- 156 • there are two logical MIME parts within the *Message Package*:
  - 157 • a MIME part, referred to as the *Header Container*, containing one SOAP 1.1  
158 compliant message. This XML document is referred to as a *SOAP Message* for  
159 the remainder of this specification
  - 160 • zero or more MIME parts, referred to as *Payload Containers*, containing  
161 application level payloads

162 The *SOAP Message* is an XML document that consists of the SOAP Envelope element. This is  
163 the root element of the XML document representing the *SOAP Message*. The SOAP Envelope  
164 element consists of the following:

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

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

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



171

### 172 7.1.1 SOAP Structural Conformance

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

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

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

## 179 7.2 Message Package

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

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

```
190 Content-Type: multipart/related; type="text/xml"; boundary="-----boundaryValue";  
191 start="<cid-of-SOAP-message-body-part>"  
192
```

## 193 7.3 Header Container

194 The root body part of the *Message Package* is referred to in this specification as the *Header*  
195 *Container*. The *Header Container* is a MIME body part that MUST consist of one SOAP Message  
196 as defined in the SOAP Messages with Attachments [SOAPATTACH] specification.  
197

### 198 7.3.1 Content-Type

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

```
202 Content-Type: text/xml; charset="UTF-8"  
203
```

#### 204 7.3.1.1 charset Attribute

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

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

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

```
215 charset="UTF-8";  
216
```

217 **7.3.2 Header Container Example**

218 The following represents an example of a *Header Container* :

```

219
220 Content-ID: messagepackage-123@example.com      -- | Header Container
221 Content-Type: text/xml;                        |
222         charset="UTF-8"                       |
223
224 <SOAP-ENV:Envelope                             |--SOAP Message
225     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
226   <SOAP-ENV:Header>
227     ...
228   </SOAP-ENV:Header>
229   <SOAP-ENV:Body>
230     ...
231   </SOAP-ENV:Body>
232 </SOAP-ENV:Envelope>
    
```

233 A complete example of a *Header Container* is presented in Appendix B. That example includes  
 234 the `charset` attribute and portions of an *XML Prolog* (see sect 8.1), neither of which is required  
 235 to appear in a *Header Container*.

236 **7.4 Payload Container**

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

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

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

243 The contents of each *Payload Container* MUST be identified by the ebXML Message **Manifest**  
 244 element within the *SOAP Body* (see section 8.1.1).

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

250 **7.4.1 Example of a Payload Container**

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

```

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

260

261 **7.5 Additional MIME Parameters**

262 Any MIME part described by this specification MAY contain additional headers or MIME  
 263 parameters in conformance with the [RFC2045] specification. Implementations MAY ignore any  
 264 MIME parameter not defined in this specification. Implementations MUST ignore any MIME  
 265 parameter that they do not recognize.

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



## 268 **7.6 Reporting MIME Errors**

269 If a MIME error is detected in the *Message Package* then it MUST be reported by sending an  
270 ebXML Message containing an **ErrorList** element (see section 8.8), where **errorCode** is set to  
271 **MimeProblem** and a **severity** set to **Error**. See section 11 for more details on how to report an  
272 error.

## 273 8 ebXML SOAP Extensions

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

- 277 • different software components are likely to be used to generate that header-element,
- 278 • the element is not always present,
- 279 • the structure of the header element might vary independently of the other header-  
280 elements, or
- 281 • the data contained in the header-element MAY need to be digitally signed separately  
282 from the other header-elements.

### 283 8.1 XML Prolog

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

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

#### 290 8.1.1 XML Declaration

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

#### 295 8.1.2 Encoding Declaration

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

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

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

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

307 For example:

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

## 310 8.2 ebXML SOAP Envelope Extensions

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

### 318 8.2.1 Namespace pseudo attribute

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

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

### 326 8.2.2 ebXML SOAP Extensions

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

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

### 347 8.2.3 #wildcard element content

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

352 Any foreign namespace-qualified element added SHOULD include the SOAP **mustUnderstand**  
353 attribute. If the SOAP **mustUnderstand** attribute is NOT present, the default value implied is '0'  
354 (false). If an implementation of the MSH does not recognize the namespace of the element and  
355 the value of the SOAP **mustUnderstand** attribute is '1' (true) then the MSH SHALL respond with  
356 a message that includes an **errorCode** of **NotSupported** in an **Error** element as defined in  
357 section 8.8. If the value of the **mustUnderstand** attribute is '0' or if the **mustUnderstand**  
358 attribute is not present then an implementation of the MSH MAY ignore the namespace-qualified  
359 element and its content.

## 360 8.3 SOAP Header element

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

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

367 </SOAP-ENV:Envelope>

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

## 370 8.4 MessageHeader element

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

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

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

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

- 386 • SOAP **mustUnderstand**
- 387 • **version**

### 388 8.4.1 version attribute

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

### 394 8.4.2 SOAP mustUnderstand attribute

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

## 399 8.5 MessageHeader element description

400 The following sections describe the contents of all the elements in the **MessageHeader** element  
401 defined in the previous section.

### 402 8.5.1 From and To elements

403 The REQUIRED **From** element identifies the *Party* that originated the message. The REQUIRED  
404 **To** element identifies the *Party* that is the intended recipient of the message. Both **To** and **From**  
405 can be logical identifiers such as a DUNS number, or identifiers that also imply a physical location  
406 such as an email address.

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

408 The **PartyId** element has a single attribute, **type** and content that is a string value. The **type**  
409 attribute indicates the domain of names to which the string in the content of the **PartyId** element

410 belongs. The value of the **type** attribute MUST be mutually agreed and understood by each of the  
411 *Parties*. It is RECOMMENDED that the value of the **type** attribute be a URI.

412 If the **PartyId type** attribute is not present, the content of the **PartyId** element MUST be a URI  
413 [RFC2396], otherwise report an error (see section 11) with **errorCode** set to **Inconsistent** and  
414 **severity** set to **Error**. It is strongly RECOMMENDED that the content of the **PartyID** element be  
415 a URN.

416 The following fragment demonstrates usage of the **From** and **To** elements. The first illustrates a  
417 user-defined numbering scheme, and the second a URN.

418  
419  
420  
421  
422  
423  
424

```
<eb:From>
  <eb:PartyId type="urn:duns.com">1234567890123</eb:PartyId>
</eb:From>
<eb:To>
  <eb:PartyId>smtp:joe@example.com</eb:PartyId>
</eb:To>
```

### 425 8.5.2 CPAId element

426 The REQUIRED **CPAId** element is a string that identifies the parameters that govern the  
427 exchange of messages between the parties. The recipient of a message MUST be able to  
428 resolve the **CPAId** to an individual set of parameters, taking into account the sender of the  
429 message.

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

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

437  
438

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

### 439 8.5.3 ConversationId element

440 The REQUIRED **ConversationId** element is a string that identifies the set of related messages  
441 that make up a conversation between two **Parties**. The **Party** that initiates a conversation  
442 determines the value of the **ConversationId** element that SHALL be reflected in all messages  
443 pertaining to that conversation.

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

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

449

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

450 *Note that implementations are free to choose how they will identify and store conversational state*  
451 *related to a specific Conversation. Implementations SHOULD provide a facility for mapping*  
452 *between their identification schema and a **ConversationId** generated by another implementation.*

### 453 8.5.4 Service element

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

- 456
- a standards organization, or
  - an individual or enterprise
- 457

458 Note that in the context of an ebXML Business Process model, a **Service** element identifies a  
 459 Business Transaction. An example of the **Service** element follows:

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

461 Note: URIs in the **Service** element that start with the namespace:  
 462 <http://www.ebxml.org/namespaces/messageService> are reserved for use by this specification.

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

#### 464 8.5.4.1 type attribute

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

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

#### 471 8.5.5 Action element

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

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

476

#### 477 8.5.6 MessageData element

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

- 480 • **MessageId**
- 481 • **Timestamp**
- 482 • **RefToMessageId**
- 483 • **TimeToLive**

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

```
485 eb:MessageData>  
486     <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>  
487  
488 <eb:RefToMessageId>example.com.20001209-133003-28571</eb:RefToMessageId>  
489     <eb:Timestamp>20010215111212Z</Timestamp>  
490 </eb:MessageData>
```

##### 491 8.5.6.1 MessageId element

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

##### 495 8.5.6.2 Timestamp element

496 The **Timestamp** is a value representing the time that the message header was created  
 497 conforming to an [XMLSchema] `timeInstant`. The format of `CCYYMMDDTHHMMSS.SSSZ` is  
 498 REQUIRED to be used. This time format is Coordinated Universal Time (UTC).

### 499 8.5.6.3 RefToMessageId element

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

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

505 For Acknowledgment Messages, the **RefToMessageId** element is REQUIRED, and its value  
506 MUST be the **MessageId** value of the ebXML Message being acknowledged. See also sections  
507 8.13 and 10.

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

### 510 8.5.6.4 TimeToLive element

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

### 514 8.5.7 QualityOfServiceInfo element

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

- 517 • **deliverySemantics**
- 518 • **messageOrderSemantics**
- 519 • **deliveryReceiptRequested**

520 The **QualityOfServiceInfo** element SHOULD be present if any of the attributes within the  
521 element need to be set to their non-default value. The **deliverySemantics** attribute supports  
522 Reliable Messaging and is discussed in detail in section.

#### 523 8.5.7.1 messageOrderSemantics attribute

524 The **messageOrderSemantics** parameter/attribute MUST be used by the *From Party* MSH to  
525 indicate whether the Message is passed to the receiving application in the order that the sending  
526 application specified. Valid Values are:

- 527 • **Guaranteed**. The messages are passed to the receiving application in the order that the  
528 sending application specified.
- 529 • **NotGuaranteed** The messages may be passed to the receiving application in different  
530 order from the order which sending application specified.

531 The default value for **messageOrderSemantics** is specified in the CPA. If no value is specified in  
532 the CPA then the default value is **NotGuaranteed**.

533 If **messageOrderSemantics** is set to **Guaranteed** then the *To Party* MSH MUST correct invalid  
534 order of messages using the value of **SequenceNumber** in the conversation specified by the  
535 **ConversationId**. The **Guaranteed** semantics can be set only when **deliverySemantics** is  
536 **OnceAndOnlyOnce**. If **deliverySemantics** is not **OnceAndOnlyOnce** then report the error to  
537 the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see section 10).

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

540 If the *To Party* is unable to support the type of **messageOrderSemantics** requested, then the *To*  
541 *Party* MUST report the error to the *From Party* using an **errorCode** of **NotSupported** and a  
542 **severity** of **Error**. A sample of **messageOrderSemantics** follows.  
543

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

546 **8.5.7.2 deliveryReceiptRequested attribute**

547 The **deliveryReceiptRequested** attribute MUST be used by a *From Party* to indicate whether a  
 548 message received by the *To Party* should result in the *To Party* returning an acknowledgment  
 549 message containing an **Acknowledgment** element with a **type** of **deliveryReceipt**.

550 The **deliveryReceiptRequested** element is frequently used to provide a business-level  
 551 acknowledgment that the message has been received and is being processed. This is separate  
 552 from a Reliable Messaging acknowledgment message which only indicates that a receiving MSH  
 553 has successfully received a message.

554 Before setting the value of **deliveryReceiptRequested**, the *From Party* SHOULD check the  
 555 **deliveryReceiptSupported** parameter for the *To Party* in the CPA to make sure that its value is  
 556 compatible.

557 Valid values for **deliveryReceiptRequested** are:

- 558 • **Unsigned** - requests that an unsigned Delivery Receipt is requested
- 559 • **Signed** - requests that a signed Delivery Receipt is requested, or
- 560 • **None** - indicates that no Delivery Receipt is requested.

561 The default value for **deliveryReceiptRequested** is **None**. When a *To Party* receives a message  
 562 with **deliveryReceiptRequested** attribute set to **Signed** or **Unsigned** then it should verify that it  
 563 is able to support the type of Delivery Receipt requested.

564 If the *To Party* can produce the Delivery Receipt of the type requested, then it MUST return to the  
 565 *From Party* on the message just received, a message containing an **Acknowledgment** element  
 566 with the value of the **type** attribute set to **DeliveryReceipt**.

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

569 An example of **deliveryReceiptRequested** follows:

570 <eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"  
 571 eb:messageOrderSemantics="Guaranteed"

572 **8.5.8 eb:deliveryReceiptRequested="Unsigned"/> SequenceNumber**  
 573 **element**

574 The **SequenceNumber** is an element that indicates the sequence in which messages MUST be  
 575 processed by a *To Party* receiving MSH. The **SequenceNumber** is unique within the  
 576 **ConversationId** and *From Party* MSH. It is set to zero on the first message from that MSH for a  
 577 Conversation and then incremented by one for each subsequent message sent. The  
 578 **SequenceNumber** element MUST appear only when **deliverySemantics** is **OnceAndOnlyOnce**  
 579 and **messageOrderSemantics** is **Guaranteed**. If it does not meet these criteria, then there is an  
 580 error that must be reported to the *From Party* MSH with an **errorCode** of **Inconsistent** and a  
 581 **severity** of **Error**.

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

586 If the implementation defined limit for saved out-of-sequence messages is reached, then the *To*  
 587 *Party* MSH MUST indicate a delivery failure to the *From Party* MSH with **errorCode** set to  
 588 **DeliveryFailure** and **severity** set to **Error** (see section 11).



589 The **SequenceNumber** element is an integer value that is incremented (e.g. 0, 1, 2, 3, 4...) for  
 590 each From Party application-prepared message sent to the *To Party* application in the  
 591 **ConversationId**. The next value of 99999999 in the increment is "0". The Sequence Number  
 592 consists of ASCII numerals in the range 0-99999999. In following cases, the Sequence Number  
 593 takes the value "0":

- 594 1) First message from the within the Conversation
- 595 2) First message after resetting Sequence Number information by the From Party MSH
- 596 3) First message after wraparound (next value after 99999999)

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

- 599 • **Reset** – the Sequence Number is reset as shown in 1 or 2 above
- 600 • **Continue** – the Sequence Number continues sequentially (including 3 above)

601 When the Sequence Number is set to "0" because of 1 or 2 above, the **status** attribute of the  
 602 messages MUST be set to **Reset**. In all other cases, including 3 above, the **status** attribute  
 603 MUST be set to **Continue**. Before the *From Party* MSH resets the **SequenceNumber** of a  
 604 Conversation, the *From Party* MUST wait for receiving of all the *Acknowledgement Messages* for  
 605 Messages previously sent for the Conversation. Only when all the sent Messages are  
 606 acknowledged, can the *From Party* reset the **SequenceNumber**. An example of  
 607 **SequenceNumber** follows.

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

### 610 8.5.9 Description element

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

### 616 8.5.10 MessageHeader sample

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

```
619 <eb:MessageHeader id="..." eb:version="1.0">
620   <eb:From>uri:example.com</eb:From>
621   <eb:To type="someType">QRS543</eb:To>
622   <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
623   <eb:ConversationId>987654321</eb:ConversationId>
624   <eb:Service type="myservicetypes">QuoteToCollect</eb:Service>
625   <eb:Action>NewPurchaseOrder</eb:Action>
626   <eb:MessageData>
627     <eb:MessageId>mid:UUID-2</eb:MessageId>
628     <eb:Timestamp>20000725T121905.000Z</eb:Timestamp>
629     <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
630   </eb:MessageData>
631   <eb:QualityOfServiceInfo
632     deliverySemantics="OnceAndOnlyOnce"
633     deliveryReceiptRequested="Signed" />
634 </eb:MessageHeader>
```

### 636 8.6 TraceHeaderList element

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

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

Message Service Specification 0.98

Page 25 of 77

- 641 • the message is being sent over a single hop (see sections 8.6.3.5 and 8.6.5), and
- 642 • the message is not being sent reliably (see section 10)

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

- 644 • SOAP **mustUnderstand**
- 645 • **version**

### 646 8.6.1 SOAP mustUnderstand attribute

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

### 651 8.6.2 version attribute

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

### 657 8.6.3 TraceHeader Element

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

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

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

669 The **TraceHeader** element MAY contain either or both of the following attributes:

- 670 • **reliableMessagingMethod**
- 671 • **ackRequested**

672

#### 673 8.6.3.1 SenderURI element

674 This element contains the URI of the Sender's Message Service Handler. Unless there is another  
675 URI identified within the CPA, the recipient of the message uses the URI to send a message,  
676 when required that:

- 677 • responds to an earlier message
- 678 • acknowledges an earlier message
- 679 • reports an error in an earlier message.

#### 680 8.6.3.2 ReceiverURI element

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

683 **8.6.3.3 Timestamp element**

684 The **Timestamp** element is the time the individual **TraceHeader** was created. It is in the same  
 685 format as in the **Timestamp** element in the **MessageData** element.

686 **8.6.3.4 #wildcard element**

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

688 **8.6.3.5 reliableMessagingMethod attribute**

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

- 691 • **ebXML**
- 692 • **Transport**

693 The default implied value for this attribute is **ebXML**. Refer to section 10.1.2 for discussion of the  
 694 use of this attribute.

695 **8.6.3.6 ackRequested attribute**

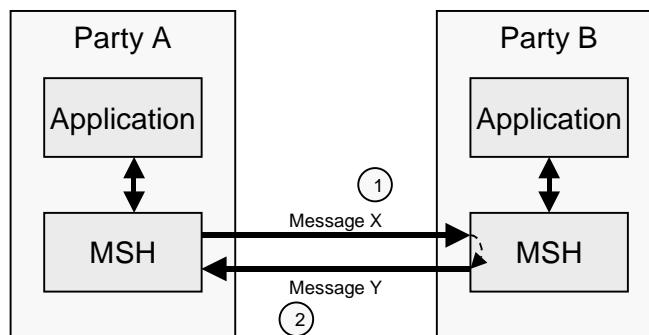
696 The **ackRequested** attribute is an enumeration that SHALL have one of the following values:

- 697 • **Signed**
- 698 • **Unsigned**
- 699 • **None**

700 The default implied value for this attribute is **None**. Refer to section 8.7.6 for discussion of the  
 701 use of this attribute.

702 **8.6.4 Single Hop TraceHeader Sample**

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



704

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

706 The content of the corresponding messages could include:

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

708

```

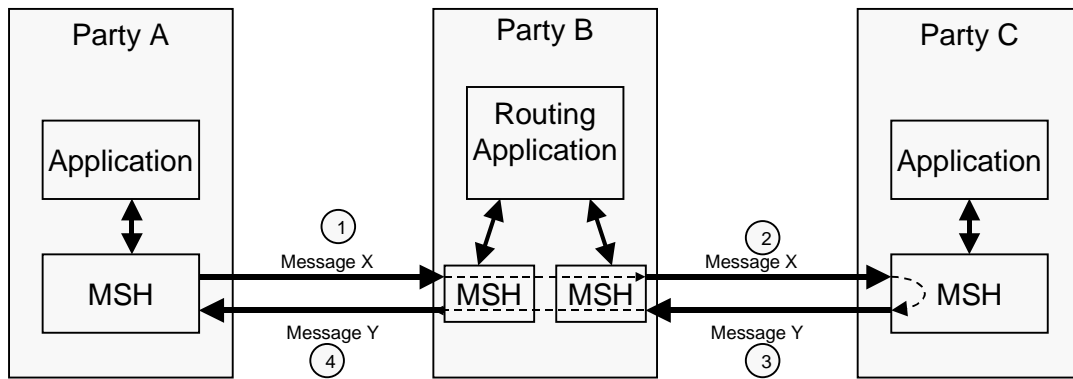
709 <eb:MessageHeader id="..." eb:version="1.0">
710 <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
711 <eb:To>urn:myscheme.com:id:PartyB-id</eb:To>
712 <eb:ConversationId>219cdj89dj2398djfn</eb:ConversationId>
713 ...
714 <eb:MessageData>
715 <eb:MessageId>29dmridj103kvna</eb:MessageId>
716 ...
    
```

```

717 </eb:MessageData>
718 ...
719 </eb:MessageHeader>
720
721 <eb:TraceHeaderList id="..." eb:version="1.0">
722 <eb:TraceHeader>
723 <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
724 <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
725 <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
726 </eb:TraceHeader>
727 </eb:TraceHeaderList>
    
```

728 **8.6.5 Multi-hop TraceHeader Sample**

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



731

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

733 The content of the corresponding messages could include:

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

```

734 <eb:MessageHeader id="..." eb:version="1.0">
735 <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
736 <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
737 <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
738 ...
739 <eb:MessageData>
740 <eb:MessageId>29dmridj103kvna</eb:MessageId>
741 ...
742 </eb:MessageData>
743 ...
744 </eb:MessageHeader>
745
746 <eb:TraceHeaderList id="..." eb:version="1.0">
747 <eb:TraceHeader>
748 <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
749 <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
750 <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
751 </eb:TraceHeader>
752 </eb:TraceHeaderList>
    
```

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

```

754 <eb:MessageHeader id="..." eb:version="1.0">
755 <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
756 <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
757 <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
758 ...
759 <eb:MessageData>
760 <eb:MessageId>29dmridj103kvna</eb:MessageId>
761 ...
762 </eb:MessageData>
763 ...
764 </eb:MessageHeader>
    
```

765

```

766 <eb:TraceHeaderList id="..." eb:version="1.0">
767   <eb:TraceHeader>
768     <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
769     <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
770     <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
771   </eb:TraceHeader>
772   <eb:TraceHeader>
773     <eb:SenderURI>url:PartyB.com/PartyAMsh</eb:SenderURI>
774     <eb:ReceiverURI>url:PartyC.com/PartyBMsh</eb:ReceiverURI>
775     <eb:Timestamp>20001216T21:19:45.483Z-6</eb:Timestamp>
776   </eb:TraceHeader>
777 </eb:TraceHeaderList>
778

```

## 779 8.7 Via element

780 The **Via** element is a *SOAP Header* that is used to convey information to the next ebXML  
781 Message Service Handler (MSH) that receives the message. Note that this MSH can be a MSH  
782 operated by an intermediary or by the *To Party*. In particular, the **Via** element is used to hold data  
783 that can vary from one hop to another.

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

- 785 • SOAP **mustUnderstand** attribute with a value of '1'
- 786 • SOAP **actor** attribute with the value "<http://schemas.xmlsoap.org/soap/actor/next>"
- 787 • a **version** attribute with a value of '1.0'

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

- 789 • **syncReply** attribute
- 790 • **reliableMessagingMethod** attribute
- 791 • **ackRequested** attribute
- 792 • **CPAId** element

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

- 794 • **Service** element
- 795 • **Action** element

### 796 8.7.1 SOAP mustUnderstand attribute

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

### 804 8.7.2 SOAP actor attribute

805 The **Via** element MUST contain a SOAP **actor** attribute with the value  
806 <http://schemas.xmlsoap.org/soap/actor/next> and be interpreted and processed as defined in the  
807 [SOAP] specification. This means that the **Via** element MUST be processed by the SOAP  
808 processor that receives the message and SHOULD NOT be forwarded to the next SOAP  
809 processor. The **Via** element is specifically intended for the recipient of a message that contains  
810 the element.

### 811 8.7.3 version attribute

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

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

#### 817 **8.7.4 syncReply attribute**

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

#### 825 **8.7.5 reliableMessagingMethod attribute**

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

- 828 • **ebXML**
- 829 • **Transport**

830 The default implied value for this attribute is **ebXML**. Refer to section 10.2.4 for discussion of the  
831 use of this attribute.

#### 832 **8.7.6 ackRequested attribute**

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

- 834 • Signed
- 835 • UnSigned
- 836 • None

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

#### 841 **8.7.7 CPAId element**

842 The **CPAId** element is a string that identifies the parameters that govern the exchange of  
843 messages between two MSH instances. It has the same meaning as the **CPAId** in the  
844 **MessageHeader** except that the parameters identified by the **CPAId** apply just to the exchange  
845 of messages between the two MSH instances rather than between the **Parties** identified in the **To**  
846 and **From** elements of the **MessageHeader**. This allows different parameters, transport  
847 protocols, etc, to be used on different hops when a message is passed through intermediaries.

848 If the **CPAId** element is present, the parameter values that are identified SHOULD be used  
849 instead of the values identified by the **CPAId** in the **MessageHeader** element.

#### 850 **8.7.8 Service and Action elements**

851 The **Service** and **Action** elements have the same meaning as the **Service** and **Action** elements  
852 in the **MessageHeader** element (see sections 8.5.4 and 8.5.5) except that they are interpreted  
853 and acted on by the next MSH whether or not the MSH is operated by the **To Party**.

854 The designer of the service or business process that is using the ebXML messaging service  
855 defines the values used for Service and Action.

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

## 858 8.7.9 Sample Via element

859 The following is a sample *Via* element.

```
860
861 <eb:Via SOAP-ENV:mustUnderstand="1"
862   SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next"
863   eb:version="1.0"
864   eb:syncReply="false">
865   <eb:CPAId>yaddaydda</eb:CPAId>
866   <eb:Service>Proxy</eb:Service>
867   <eb:Action>LogActivity</eb:Action>
868 </eb:Via>
```

## 869 8.8 ErrorList element

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

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

- 873 • *id* attribute
- 874 • SOAP *mustUnderstand* attribute
- 875 • *version attribute*
- 876 • *highestSeverity* attribute

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

### 878 8.8.1 id attribute

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

### 880 8.8.2 SOAP mustUnderstand attribute

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

### 885 8.8.3 version attribute

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

### 891 8.8.4 highestSeverity attribute

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

### 895 8.8.5 Error element

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

- 897 • *codeContext*
- 898 • *errorCode*
- 899 • *severity*
- 900 • *location*
- 901 • *xml:lang*

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

### 903 8.8.5.1 codeContext attribute

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

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

### 911 8.8.5.2 errorCode attribute

912 The REQUIRED **errorCode** attribute indicates the nature of the error in the *message in error*.  
 913 Valid values for the **errorCode** and a description of the code's meaning are given in sections  
 914 8.8.8 and 8.8.9.

### 915 8.8.5.3 severity attribute

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

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

### 921 8.8.5.4 location attribute

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

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

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

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

### 930 8.8.5.5 Error element Content

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

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

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

### 937 8.8.6 Examples

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

```
939
940 <eb:ErrorList id='3490sdo9', highestSeverity="error" eb:version="1.0">
941   <eb:Error errorCode='SecurityFailure' severity="Error"
942     location='URI_of_ds:Signature_goes_here' xml:lang="us-en"
943     errorMessage='Validation of signature failed' />
944   <eb:Error ... />
945 </eb:ErrorList>
```



946 **8.8.7 errorCode values**

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

- 949 • the first column contains the value to be used as an **errorCode**, e.g. **SecurityFailure**
- 950 • the second column contains a "Short Description" of the **errorCode**. Note that this  
 951 narrative MUST NOT be used in the **errorMessage** attribute.
- 952 • the third columns contains a "Long Description" that provides an explanation of the  
 953 meaning of the error and provides guidance on when the particular **errorCode** should be  
 954 used.

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

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

960 **8.8.8 Reporting Errors in the ebXML Elements**

961 The following list contains error codes that can be associated with ebXML elements:  
 962

Error Code	Short Description	Long Description
<b>ValueNotRecognized</b>	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 ebXML Message Service.
<b>NotSupported</b>	Element or attribute not supported	Although the document is well formed and valid, an element or attribute is present that: <ul style="list-style-type: none"> <li>• is consistent with the rules and constraints contained in this specification, but</li> <li>• is not supported by the ebXML Message Service processing the message.</li> </ul>
<b>Inconsistent</b>	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.
<b>OtherXml</b>	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 <b>errorMessage</b> attribute should be used to indicate the nature of the problem.

963 **8.8.9 Non-XML Document Errors**

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

Error Code	Short Description	Long Description
<b>DeliveryFailure</b>	Message Delivery Failure	A message has been received that either probably or definitely could not be sent to its next destination. Note that if <b>severity</b> is set to <b>Warning</b> then there is a small probability that the message was delivered.

<b>TimeToLiveExpired</b>	Message Time To Live Expired	A message has been received that arrived after the time specified in the <b>TimeToLive</b> element of the <b>Header</b> element
<b>SecurityFailure</b>	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
<b>Unknown</b>	Unknown Error	Indicates that an error has occurred that is not covered explicitly by any of the other errors. The <b>errorMessage</b> attribute should be used to indicate the nature of the problem.

965 **8.9 Signature element**

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

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

976 **8.10 SOAP Body Extensions**

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

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

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

- 987 • **Manifest** element
- 988 • **StatusData** element
- 989 • **Acknowledgement** element

990 Each is defined in the following sections.

991 **8.11 Manifest element**

992 The **Manifest** element is a composite element consisting of one or more **Reference** elements.  
 993 Each **Reference** element identifies data associated with the message, whether included as part  
 994 of the message as payload document(s) contained in a **Payload Container**, or remote resources  
 995 accessible via a URL. The purpose of the **Manifest** is as follows:

- 996 • to make it easier to directly extract a particular payload associated with this ebXML  
 997 Message,
- 998 • to enable a MSH to check the integrity of an ebXML Message
- 999 • to allow an application to determine whether it can process the payload without having to  
 1000 parse it.

1001 The **Manifest** element is comprised of the following attributes and elements, each of which is  
1002 described below:

- 1003 • a REQUIRED **id** attribute
- 1004 • a REQUIRED SOAP **mustUnderstand** attribute
- 1005 • a REQUIRED **version** attribute
- 1006 • one or more **Reference** elements
- 1007 • **#wildcard**

#### 1008 **8.11.1 id attribute**

1009 The **Manifest** element MUST have an **id** attribute that is an XML ID.

#### 1010 **8.11.2 SOAP mustUnderstand attribute**

1011 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP  
1012 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the  
1013 **Manifest** element MUST be understood by a receiving process or else the message MUST be  
1014 rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

#### 1015 **8.11.3 version attribute**

1016 The REQUIRED **version** attribute indicates the version of the ebXML Message Service Header  
1017 Specification to which the ebXML SOAP **Header** extensions conform. Its purpose is to provide  
1018 future versioning capabilities. The value of the **version** attribute MUST be "0.98". Future versions  
1019 of this specification SHALL require other values of this attribute. The version attribute MUST be  
1020 namespace qualified for the ebXML Message Header namespace defined above.

#### 1021 **8.11.4 Reference element**

1022 The **Reference** element is a composite element consisting of the following subordinate elements:

- 1023 • **Schema** - information about the schema(s) that define the instance document identified  
1024 in the parent **Reference** element
- 1025 • **Description** - a textual description of the payload object referenced by the parent  
1026 **Reference** element
- 1027 • **#wildcard** - any namespace-qualified element content belonging to a foreign namespace

1028 The **Reference** element itself is an [XLINK] simple link. XLINK is presently a Candidate  
1029 Recommendation (CR) of the W3C. It should be noted that the use of XLINK in this context is  
1030 chosen solely for the purpose of providing a concise vocabulary for describing an association.  
1031 Use of an XLINK processor or engine is NOT REQUIRED, but MAY prove useful in certain  
1032 implementations.

1033 The **Reference** element has the following attribute content in addition to the element content  
1034 described above:

- 1035 • **id** - a REQUIRED XML ID for the **Reference** element,
- 1036 • **xlink:type** - this attribute defines the element as being an XLINK simple link. It has a  
1037 fixed value of 'simple',
- 1038 • **xlink:href** - this REQUIRED attribute has a value that is the URI of the payload object  
1039 referenced. It SHALL conform to the [XLINK] specification criteria for a simple link.
- 1040 • **xlink:role** - this attribute identifies some resource that describes the payload object or its  
1041 purpose. If present, then it SHALL have a value that is a valid URI in accordance with the  
1042 [XLINK] specification,
- 1043 • Any other namespace-qualified attribute MAY be present. A receiving MSH MAY choose  
1044 to ignore any foreign namespace attributes other than those defined above.

#### 1045 8.11.4.1 Schema element

1046 If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema,  
1047 DTD, or a database schema), then the **Schema** element SHOULD be present as a child of the  
1048 **Reference** element. It provides a means of identifying the schema and its version defining the  
1049 payload object identified by the parent **Reference** element. The **Schema** element contains the  
1050 following attributes:

- 1051 • **location** - the REQUIRED URI of the schema
- 1052 • **version** – a version identifier of the schema

#### 1053 8.11.4.2 Description element

1054 The **Reference** element MAY contain zero or more **Description** elements. The **Description** is a  
1055 textual description of the payload object referenced by the parent **Reference** element. The  
1056 language of the description is defined by a REQUIRED **xml:lang** attribute. The **xml:lang** attribute  
1057 MUST comply with the rules for identifying languages specified in [XML]. This element is provided  
1058 to allow a human readable description of the payload object identified by the parent **Reference**  
1059 element. If multiple **Description** elements are present, each SHOULD have a unique **xml:lang**  
1060 attribute value. An example of a **Description** element follows.

```
1061 <eb:Description xml:lang="en-gb">Purchase Order for 100,000 widgets</eb:Description>
```

#### 1062 8.11.4.3 #wildcard element

1063 Refer to section 8.2.3 for discussion of #wildcard element handling.

#### 1064 8.11.5 What References are Included in a Manifest

1065 The designer of the business process or information exchange that is using ebXML Messaging  
1066 decides what payload data is referenced by the Manifest and the values to be used for **xlink:role**.

#### 1067 8.11.6 Manifest Validation

1068 If an **xlink:href** attribute contains a URI that is a content id (URI scheme "cid") then a MIME  
1069 part with that `content-id` MUST be present in the *Payload Container* of the message. If it is  
1070 not, then the error SHALL be reported to the *From Party* with an **errorCode** of **MimeProblem**  
1071 and a **severity** of **Error**.

1072 If an **xlink:href** attribute contains a URI that is not a content id (URI scheme "cid") and that URI  
1073 cannot be resolved, then it is an implementation decision on whether to report the error. If the  
1074 error is to be reported, then it SHALL be reported to the *From Party* with an **errorCode** of  
1075 **MimeProblem** and a **severity** of **Error**.

#### 1076 8.11.7 Manifest sample

1077 The following fragment demonstrates a typical **Manifest** for a message with a single payload  
1078 MIME body part:

```
1079
1080 <eb:Manifest id="Manifest" eb:version="1.0">
1081   <eb:Reference id="pay01"
1082     xlink:href="cid:payload-1"
1083     xlink:role="http://regrep.org/gci/purchaseOrder">
1084     <eb:Description>Purchase Order for 100,000 widgets</eb:Description>
1085     <eb:Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
1086       version="1.0"/>
1087   </eb:Reference>
1088 </eb:Manifest>
```

#### 1089 8.12 StatusData Element

1090 The **StatusData** element is used by one MSH to respond to a request on the status of the  
1091 processing of a message that was previously sent (see also section 9.1).

1092 The **StatusData** element consists of the following elements and attributes:

- 1093 • a REQUIRED **RefToMessageld** element
- 1094 • a **Timestamp** element
- 1095 • a REQUIRED SOAP **mustUnderstand** attribute
- 1096 • a REQUIRED **version** attribute
- 1097 • a **messageStatus** attribute

#### 1098 8.12.1 RefToMessageld element

1099 A REQUIRED **RefToMessageld** element that contains the **MessageId** of the message whose  
1100 status is being reported.

#### 1101 8.12.2 Timestamp element

1102 The **Timestamp** element contains the time that the message, whose status is being reported,  
1103 was received. This MUST be omitted if the message whose status is being reported is  
1104 **NotRecognized** or the request was **Unauthorized**.

#### 1105 8.12.3 SOAP mustUnderstand attribute

1106 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP  
1107 namespace (<http://schemas.xmlsoap.org/soap/envelope/>), indicates that the contents of the  
1108 **StatusData** element MUST be understood by a receiving process or else the message MUST be  
1109 rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

#### 1110 8.12.4 version attribute

1111 The REQUIRED **version** attribute indicates the version of the ebXML Message Service Header  
1112 Specification to which the ebXML SOAP Header extensions conform. Its purpose is to provide  
1113 future versioning capabilities. The value of the **version** attribute MUST be "0.98". Future versions  
1114 of this specification SHALL require other values of this attribute. The **version** attribute MUST be  
1115 namespace qualified for the ebXML Message Header namespace defined above.

#### 1116 8.12.5 messageStatus attribute

1117 The **messageStatus** attribute identifies the status of the message that is identified by the  
1118 **RefToMessageld** element. It SHALL be set to one of the following values:

- 1119 - **Unauthorized** – the Message Status Request is not authorized or accepted
- 1120 - **NotRecognized** – the message identified by the **RefToMessageld** element in the  
1121 **StatusData** element is not recognized
- 1122 - **Received** – the message identified by the **RefToMessageld** element in the  
1123 **StatusData** element has been received by the MSH, but has not been processed by  
1124 an application or forwarded to another MSH

1125 Note that if a Message Status Request is sent after the elapsed time indicated by  
1126 **persistDuration** has passed since the message being queried was sent, then the Message  
1127 Status Response may indicate that the **MessageId** was **NotRecognized** as the **MessageId** is no  
1128 longer in persistent storage.

### 1129 8.13 Acknowledgment Element

1130 The **Acknowledgment** element is an optional element that is used by one Message Service  
1131 Handler to indicate that another Message Service Handler has received a message.

- 1132 •

1133 The **RefToMessageld** in a message containing an Acknowledgement element is used to identify  
1134 the message being acknowledged by its **MessageId**.

1135 The **Acknowledgment** element consists of the following elements and attributes:

- 1136 • a **Timestamp** element
- 1137 • a **From** element
- 1138 • a REQUIRED SOAP **mustUnderstand** attribute
- 1139 • a REQUIRED **version** attribute
- 1140 • a **type** attribute
- 1141 • a **signed** attribute

#### 1142 **8.13.1 Timestamp element**

1143 The **Timestamp** element is a value representing the time that the *message being acknowledged*  
1144 was received by the Party generating the *acknowledgment message*. It must conform to an  
1145 [XMLSchema] `timeInstant`.

#### 1146 **8.13.2 From element**

1147 This is the same element as the **From** element within **MessageHeader** element (see section  
1148 8.5.1). However, when used in the context of an **Acknowledgment** element, it contains the  
1149 identifier of the *Party* that is generating the *acknowledgment message*.

1150 If the **From** element is omitted then the *Party* that is sending the element is identified by the **From**  
1151 element in the **MessageHeader** element.

#### 1152 **8.13.3 SOAP mustUnderstand attribute**

1153 The REQUIRED SOAP **mustUnderstand** attribute, namespace qualified to the SOAP  
1154 namespace (`http://schemas.xmlsoap.org/soap/envelope`), indicates that the contents of the  
1155 **Acknowledgment** element MUST be understood by a receiving process or else the message  
1156 MUST be rejected in accordance with [SOAP]. This attribute MUST have a value of '1' (true).

#### 1157 **8.13.4 version attribute**

1158 The REQUIRED **version** attribute indicates the version of the ebXML Message Service Header  
1159 Specification to which the ebXML SOAP Header extensions conform. Its purpose is to provide  
1160 future versioning capabilities. The value of the **version** attribute MUST be "0.98". Future versions  
1161 of this specification SHALL require other values of this attribute. The version attribute MUST be  
1162 namespace qualified for the ebXML Message Header namespace defined above.

#### 1163 **8.13.5 type attribute**

1164 The **type** attribute indicates who sent the *acknowledgment message*. It MUST contain either:

- 1165 • **DeliveryReceipt** - indicates that the *acknowledgment message* was generated by the *To*  
1166 *Party* identified by the **To** element of the *message being acknowledged*, or
- 1167 • **Acknowledgement** - indicates that the *acknowledgment message* was generated by a  
1168 *Party* that is not the *To Party* identified by the **To** element of the *message being*  
1169 *acknowledged*. Typically this will be a *Party* that has received the message and is  
1170 forwarding it to either the *To Party* or another *Party* with the intention that the message is  
1171 sent to the *To Party*.

1172 The default value for **type** is **DeliveryReceipt**.

#### 1173 **8.13.6 signed attribute**

1174 The **signed** attribute indicates whether the *acknowledgment message* is digitally signed. It MUST  
1175 contain either:

- 1176 • **true** - indicates that the *acknowledgment message* is digitally signed, or
- 1177 • **false** - indicates that the *acknowledgment message* is not digitally signed

1178 The default value for **signed** is **false**.

1179 See section 12 for details on what should be signed and how a signature that signs an  
1180 *acknowledgment message* should be checked.

## 1181 8.14 Combining ebXML SOAP Extension Elements

1182 This section describes how the various ebXML SOAP extension elements may be used in  
1183 combination.

### 1184 8.14.1 Manifest element

1185 The *Manifest* element MUST be present if there is any data associated with the message that is  
1186 not present in the *Header Container*. This applies specifically to data in the *Payload Container* or  
1187 elsewhere, e.g. on the web.

### 1188 8.14.2 MessageHeader element

1189 The *MessageHeader* element MUST be present in every message.

### 1190 8.14.3 TraceHeaderList element

1191 The *TraceHeaderList* element MAY be present in any message. It MUST be present if the  
1192 message is being sent reliably (see section 10) or over multiple hops (see section 8.6.5).

### 1193 8.14.4 StatusData element

1194 This element MUST NOT be present with the following elements:

- 1195 • a *Manifest* element
- 1196 • an *ErrorList* element with a *highestSeverity* attribute set to *Error*

### 1197 8.14.5 ErrorList element

1198 If the *highestSeverity* attribute on the *ErrorList* is set to *Warning*, then this element MAY be  
1199 present with any other element.

1200 If the *highestSeverity* attribute on the *ErrorList* is set to *Error*, then this element MUST NOT be  
1201 present with the following:

- 1202 • a *Manifest* element
- 1203 • a *StatusData* element

### 1204 8.14.6 Acknowledgment element

1205 An *Acknowledgment* element MAY be present on any message.

### 1206 8.14.7 Signature element

1207 A *Signature* element MAY be present on any message.

## 1208 9 Message Service Handler Services

1209 The Message Service Handler MUST support two services that are designed to help provide  
1210 smooth operation of a Message Handling Service implementation:

- 1211 • Message Status Request
- 1212 • Message Service Handler Ping

1213 Each service is described below:

### 1214 9.1 Message Status Request Service

1215 The Message Status Request Service consists of the following:

- 1216 • A Message Status Request message containing details regarding a message previously  
1217 sent is sent to a Message Service Handler (MSH)
- 1218 • The Message Service Handler receiving the request responds with a Message Status  
1219 Response message.

1220 A Message Service Handler SHOULD respond to Message Status Requests that have been sent  
1221 reliably (see section 10) and the **MessageId** in the **RefToMessageId** is present in *persistent*  
1222 *storage* (see section 10.1.1).

1223 An implementation MAY also respond to Message Status Requests that have not been sent  
1224 reliably.

1225 A Message services also SHOULD NOT use the Message Status Request Service to implement  
1226 Reliable Messaging.

#### 1227 9.1.1 Message Status Request Message

1228 A Message Status Request message consists of an ebXML Message containing no *ebXML*  
1229 *Payload* and the following elements in the ebXML Header:

- 1230 • A **Header** element
- 1231 • A **TraceHeaderList** element
- 1232 • A **Signature** element

1233 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.6 and  
1234 8.14.7).

1235 The **Header** element MUST contain the following:

- 1236 • a **From** element that identifies the party that created the message status request  
1237 message
- 1238 • a **To** element that identifies a party who should receive the message. If a **TraceHeader**  
1239 was present on the message whose status is being checked, this MUST be the  
1240 **ReceiverURI** from that message.
- 1241 • a **Service** element that contains:  
1242 **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1243 • an **Action** element that contains **Request**

1244 The message is then sent to the **To** party.

#### 1245 9.1.2 Message Status Response Message

1246 Once the **To** party receives the Message Status Request message, they SHOULD generate a  
1247 Message Status Response message consisting of no ebXML Payload and the following elements  
1248 in the ebXML Header.

- 1249 • a **Header** element



- 1250 • a **TraceHeaderList** element
- 1251 • an **Acknowledgment** element
- 1252 • a **StatusData** element (see section 8.12)
- 1253 • a **Signature** element

1254 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see  
1255 sections 8.6, 8.14.6 and 8.14.7).

1256 The **Header** element MUST contain the following:

- 1257 • a **From** element that identifies the sender of the Message Status Response message
- 1258 • a **To** element that is set to the value of the **From** element in the Message Status Request  
1259 message
- 1260 • a **Service** element that contains the value:  
1261 **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1262 • an **Action** element that contains **Response**
- 1263 • a **RefToMessageId** that identifies the Message Status Request message.

1264 The message is then sent to the **To** party.

### 1265 9.1.3 Security Considerations

1266 Parties who receive a Message Status Request message SHOULD always respond to the  
1267 message. However, they MAY ignore the message instead of responding with **messageStatus**  
1268 set to **Unauthorized** if they consider that the sender of the message is unauthorized. The  
1269 decision process that results in this course of action is implementation dependent.

## 1270 9.2 Message Service Handler Ping Service

1271 The Message Service Handler Ping Service enables one MSH to determine if another MSH is  
1272 operating. It consists of:

- 1273 • sending a Message Service Handler Ping message to a MSH, and
- 1274 • the MSH that receives the Ping responding with a Message Service Handler Pong  
1275 message.

### 1276 9.2.1 Message Service Handler Ping Message

1277 A Message Service Handler Ping (MSH Ping) message consists of an ebXML Message  
1278 containing no ebXML Payload and the following elements in the ebXML Header:

- 1279 • A **Header** element
- 1280 • A **TraceHeaderList** element
- 1281 • A **Signature** element

1282 The **TraceHeaderList** and the **Signature** elements MAY be omitted (see sections 8.6 and  
1283 8.14.7).

1284 The **Header** element MUST contain the following:

- 1285 • a **From** element that identifies the party creating the MSH Ping message
- 1286 • a **To** element that identifies the party that is being sent the MSH Ping message
- 1287 • a **Service** element that contains:  
1288 **<http://www.ebxml.org/namespaces/messageService/MSHStatus>**
- 1289 • an **Action** element that contains **Ping**

1290 The message is then sent to the **To** party.

### 1291 **9.2.2 Message Service Handler Pong Message**

1292 Once the **To** party receives the MSH Ping message, they MAY generate a Message Service  
1293 Handler Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML  
1294 Payload and the following elements in the ebXML Header:

- 1295 • a **Header** element
- 1296 • a **TraceHeaderList** element
- 1297 • an **Acknowledgment** element
- 1298 • a **Signature** element

1299 The **TraceHeaderList**, **Acknowledgment** and **Signature** elements MAY be omitted (see  
1300 sections 8.6, 8.14.6 and 8.14.7).

1301 The **Header** element MUST contain the following:

- 1302 • a **From** element that identifies the creator of the MSH Pong message
- 1303 • a **To** element that identifies a Party that generated the MSH Ping message
- 1304 • a **Service** element that contains the value:  
1305 ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- 1306 • an **Action** element that contains the value ***Pong***
- 1307 • a **RefToMessageId** that identifies the MSH Ping message.

1308 The message is then sent to the **To** party.

### 1309 **9.2.3 Security Considerations**

1310 Parties who receive a MSH Ping message SHOULD always respond to the message. However,  
1311 there is a risk that some parties might use the MSH Ping message to determine the existence of  
1312 a Message Service Handler as part of a security attack on that MSH. Therefore, recipients of a  
1313 MSH Ping MAY ignore the message if they consider that the sender of the message received is  
1314 unauthorized or part of some attack. The decision process that results in this course of action is  
1315 implementation dependent.

## 1316 10 Reliable Messaging

1317 Reliable Messaging defines an interoperable protocol such that the two Message Service  
1318 Handlers (MSH) can “reliably” exchange messages that are sent using “reliable messaging”  
1319 semantics, resulting in the *To Party* receiving the message once and only once.

1320 Reliability is achieved by a receiving MSH responding to a message with an *Acknowledgment*  
1321 *Message*.

### 1322 10.1.1 Persistent Storage and System Failure

1323 A MSH that supports Reliable Messaging MUST keep messages that are sent or received reliably  
1324 in *persistent storage*. In this context *persistent storage* is a method of storing data that does not  
1325 lose information after a system failure or interruption.

1326 This specification recognizes that different degrees of resilience may be realized depending on  
1327 the technology that is used to persist the data. However, as a minimum, persistent storage that  
1328 has the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly  
1329 RECOMMENDED though that implementers of this specification use technology that is resilient to  
1330 the failure of any single hardware or software component.

1331 After a system interruption or failure, an MSH MUST ensure that messages in persistent storage  
1332 are processed in the same way as if the system failure or interruption had not occurred. How this  
1333 is done is an implementation decision.

1334 In order to support the filtering of duplicate messages, a receiving MSH SHOULD save the  
1335 **MessageId** in *persistent storage*. It is also RECOMMENDED that the following be kept in  
1336 *Persistent Storage*:

- 1337 • the complete message, at least until the information in the message has been passed to  
1338 the application or other process that needs to process it
- 1339 • the time the message was received, so that the information can be used to generate the  
1340 response to a Message Status Request (see section 9.1)

1341

### 1342 10.1.2 Methods of Implementing Reliable Messaging

1343 Support for Reliable Messaging MAY be implemented in one of the following two ways:

- 1344 • using the ebXML Reliable Messaging protocol, or
- 1345 • using ebXML Header and Message structures together with commercial software  
1346 products that are designed to provide reliable delivery of messages using alternative  
1347 protocols.

## 1348 10.2 Reliable Messaging Parameters

1349 This section describes the parameters required to control reliable messaging. This parameter  
1350 information is contained in the *CPA* that governs the processing of a message.

### 1351 10.2.1 Delivery Semantics

1352 The **deliverySemantics** value MUST be used by the **From** party MSH to determine whether the  
1353 Message must be sent reliably. Valid Values are:

- 1354 • **OnceAndOnlyOnce**. The message must be sent using a **reliableMessagingMethod**  
1355 that will result in the application or other process at the **To** party receiving the message  
1356 once and only once
- 1357 • **BestEffort** The reliable delivery semantics are not used. In this case, the value of  
1358 **reliableMessagingMethod** is ignored.

1359 The value for **deliverySemantics** is specified in the CPA. If no value is specified in the CPA, the  
1360 default value is **BestEffort**.

1361 If **deliverySemantics** is set to **OnceAndOnlyOnce**, the **From** party MSH and the **To** party MSH  
1362 must adopt the Reliable Messaging behavior (see section 10) that describes how messages are  
1363 resent in the case of failure and duplicates are ignored.

1364 If **deliverySemantics** is set to **BestEffort**, a MSH that received a message that it is unable to  
1365 deliver MUST NOT take any action to recover or otherwise notify anyone of the problem. The  
1366 MSH that sent the message must not attempt to recover from any failure. This means that  
1367 duplicate messages might be delivered to an application and persistent storage of messages is  
1368 not required.

1369 If the **To** party is unable to support the type of delivery semantics requested, the **To** party  
1370 SHOULD report the error to the **From** party using an **ErrorCode** of **NotSupported** and a  
1371 **Severity** of **Error**.

## 1372 10.2.2 Sync Reply

1373 The **syncReply** is an optional value that indicates whether a response to a message must be  
1374 returned at the same time as any acknowledgments. It has two values:

- 1375 • **true** indicates that the receiving MSH MUST NOT respond to the original message until it  
1376 has been processed by an application/process;
- 1377 • **false** indicates that the receiving MSH MAY send an acknowledgment before processing  
1378 of the message by an application/process.

1379 The default value is **false**.

## 1380 10.2.3 Time To Live

1381 The **TimeToLive** value indicates the time by which a message should be delivered to and  
1382 processed by the **To** party. It must conform to an XML Schema `timeInstant`.

1383 In this context, the **TimeToLive** has expired if the time of the internal clock of the receiving MSH  
1384 is greater than the value of **TimeToLive** for the message.

1385 When setting a value for **TimeToLive** it is RECOMMENDED that the *From Party's* MSH takes  
1386 into account the accuracy of its own internal clocks as well as the MSH **TimeAccuracy**  
1387 parameter for the receiving MSH, specified in the CPA, that indicates the accuracy to which a  
1388 MSH will keep its internal clocks. How a MSH ensures that its internal clocks are kept sufficiently  
1389 accurate is an implementation decision.

1390 If the *To Party's* MSH receives a message where **TimeToLive** has expired, it SHALL send a  
1391 message to the *From Party* MSH, reporting that the **TimeToLive** of the message has expired.  
1392 This message SHALL be comprised of an **ErrorList** containing an error that has the **errorCode**  
1393 attribute set to **TimeToLiveExpired**, and the **severity** attribute set to **Error**.

## 1394 10.2.4 reliableMessagingMethod

1395 The **reliableMessagingMethod** value SHALL have one of the following values:

- 1396 • **ebXML**
- 1397 • **Transport**

1398 The default implied value for this attribute is **ebXML** and is case sensitive. Refer to section  
1399 10.1.2 for discussion of the use of this attribute.

1400

### 1401 **10.2.5 AckRequested**

1402 The **AckRequested** value is used by the sending MSH to request that the receiving MSH returns  
1403 an *acknowledgment message* with an **Acknowledgment** element with a **type** of  
1404 **Acknowledgment**.

1405 Valid values for **AckRequested** are:

- 1406 • **Unsigned** - requests that an unsigned Acknowledgement is requested
- 1407 • **Signed** - requests that a signed Acknowledgement is requested, or
- 1408 • **None** - indicates that no Acknowledgement is requested.

1409 The default value is **None**.

### 1410 **10.2.6 Timeout Parameter**

1411 The **timeout** parameter is an integer value that specifies the time expressed as a [XMLSchema]  
1412 timeDuration, that the Sending MSH MUST wait for an Acknowledgement Message before first  
1413 resending a message to the Receiving MSH.

### 1414 **10.2.7 Retries**

1415 The **retries** value is an integer value that specifies the maximum number of times a Sending  
1416 MSH SHOULD attempt to redeliver an unacknowledged *message* using the same  
1417 Communications Protocol.

### 1418 **10.2.8 RetryInterval**

1419 The **retryInterval** value is a time value, expressed as a duration in accordance with the  
1420 [XMLSchema] timeDuration data type. This value specifies the minimum time the Sending MSH  
1421 MUST wait between retries, if an *Acknowledgment Message* is not received.

### 1422 **10.2.9 PersistDuration**

1423 The **persistDuration** value is the minimum length of time, expressed as a [XMLSchema]  
1424 timeDuration, that data from a reliably sent *Message*, is kept in *Persistent Storage* by a receiving  
1425 MSH.

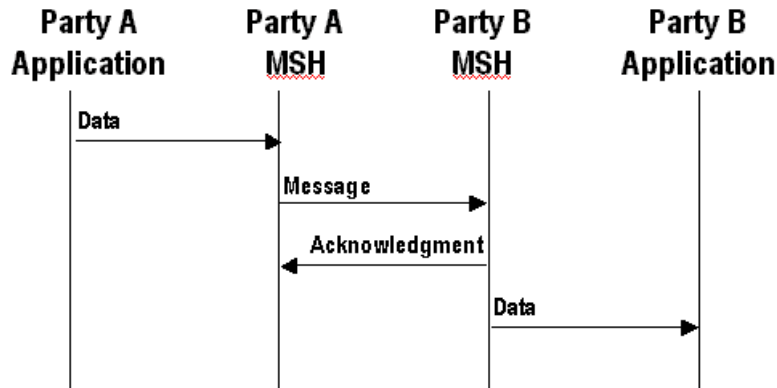
1426 If the **persistDuration** has passed since the message was first sent, a Sending MSH SHOULD  
1427 NOT resend a message with the same **MessageId**.

1428 If a message cannot be sent successfully before **persistDuration** has passed, then the Sending  
1429 MSH should report a delivery failure (see section 10.4).

## 1430 **10.3 ebXML Reliable Messaging Protocol**

1431 The ebXML Reliable Messaging Protocol described in this section MUST be followed if the  
1432 **deliverySemantics** parameter/element is set to **OnceAndOnlyOnce** and the  
1433 **reliableMessagingMethod** parameter/element is set to **ebXML** (the default).

1434 The ebXML Reliable Messaging Protocol is illustrated by the figure below.



1435

1436

1437 **Figure 10-1 Indicating that a message has been received**

1438 The receipt of the *acknowledgment message* indicates that the message being acknowledged  
 1439 has been successfully received and either processed or persisted by the receiving MSH.

1440 An *acknowledgment message* MUST contain a **MessageData** element with a **RefToMessageId**  
 1441 that contains the same value as the **MessageId** element in the *message being acknowledged*.

1442 **10.3.1.1 Sending Message Behavior**

1443 If a MSH is given data by an application that needs to be sent reliably (i.e. the  
 1444 **deliverySemantics** parameter in the CPA is set to **OnceAndOnlyOnce**), then the MSH MUST  
 1445 do the following:

1446 Create a message from components received from the application that includes:

- 1447 1) a **TraceHeader** element that identifies the sender and the receiver URIs
- 1448 2) Save the message in *persistent storage* (see section 10.1.1)
- 1449 3) Send the message to the *Receiver* MSH
- 1450 4) Wait for the *Receiver* MSH to return an *acknowledgment message* and, if it does not, then  
 1451 take the appropriate action as described in section 10.3.1.4

1452 **10.3.1.2 Receiving Message Behavior**

1453 If the CPA indicates that the **deliverySemantics** for the received message is set to  
 1454 **OnceAndOnlyOnce** then do the following:

- 1455 1) If the message is just an acknowledgement (i.e. the **Service** element is set to  
 1456 <http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment> and **Action**  
 1457 is set to **Acknowledgment**), then:
  - 1458 a) Look for a message in *persistent storage* that has a **MessageId** that is the same as the  
 1459 value of **RefToMessageId** on the received Message
  - 1460 b) If a message is found in *persistent storage* then mark the persisted message as delivered
- 1461 2) Otherwise, if the message is not just an acknowledgement, then check to see if the  
 1462 message is a duplicate (e.g. there is a **MessageId** held in *persistent storage* that was  
 1463 received earlier that contains the same value for the **MessageId**)
- 1464 c) If the message is not a duplicate then do the following:

- 1465 i) Save the **MessageId** of the received message in *persistent storage*. As an  
 1466 implementation decision, the whole message MAY be stored if there are other  
 1467 reasons for doing so.
- 1468 ii) If the received message contains a **RefToMessageId** element then do the following:
- 1469 (1) Look for a message in *persistent storage* that has a **MessageId** that is the same  
 1470 as the value of **RefToMessageId** on the received Message
- 1471 (2) If a message is found in *persistent storage* then mark the persisted message as  
 1472 delivered
- 1473 iii) Generate an *Acknowledgement Message* in response (see section 10.3.1.3).
- 1474 d) If the message is a duplicate, then do the following:
- 1475 i) Look in persistent storage for the first response to the received message and resend  
 1476 it (i.e. it contains a **RefToMessageId** that matches the **MessageId** of the received  
 1477 message)
- 1478 ii) If a message was found in *persistent storage* then resend the persisted message  
 1479 back to the MSH that sent the received message,
- 1480 iii) If no message was found in *persistent storage*, then:
- 1481 (1) if **syncReply** is set to **True** and the CPA indicates that an application response is  
 1482 included, ignore the received message (i.e. no message was generated in  
 1483 response to the message, or the processing of the earlier message is not yet  
 1484 complete)
- 1485 (2) if **syncReply** is set to **False** then generate an *Acknowledgement Message* (see  
 1486 section 10.3.1.3).

### 1487 10.3.1.3 Generating an Acknowledgement Message

1488 An *Acknowledgement Message* MUST be generated whenever a message is received with:

- 1489 • **deliverySemantics** set to **OnceAndOnlyOnce** and
- 1490 • **reliableMessagingMethod** set to **ebXML** (the default).

1491 As a minimum, it MUST contain a **MessageData** element with a **RefToMessageId** that contains  
 1492 the same value as the **MessageId** element in the *message being acknowledged*.

1493 If **ackRequested** in the **TraceHeader** of the received message is set to **Signed** or **Unsigned**  
 1494 then the acknowledgement message MUST also contain an **Acknowledgement** element.

1495 Depending on the value of the **syncReplyMode** parameter, the *Acknowledgement Message* can  
 1496 also be sent at the same time as the response to the received message. In this case, the values  
 1497 for the **Header** elements of the *Acknowledgement Message* are set by the designer of the  
 1498 Service.

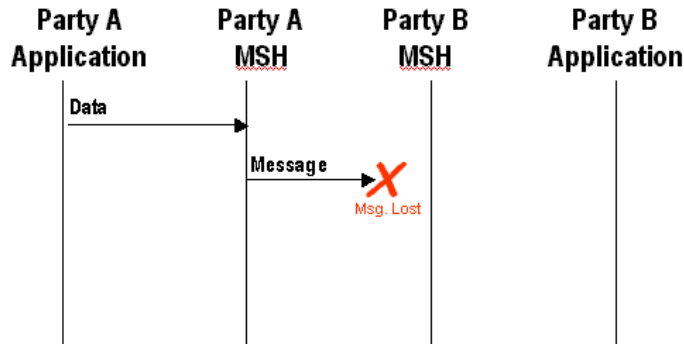
1499 If an **Acknowledgment** element is being sent on its own, then the value of the **Header** elements  
 1500 MUST be set as follows:

- 1501 1) The **Service** element MUST be set to:  
 1502 <http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment>
- 1503 2) The **Action** element MUST be set to **Acknowledgment**.
- 1504 3) The **From** element MUST be set to the **ReceiverURI** from the last **TraceHeader** in the  
 1505 *message* that has just been received
- 1506 4) The **To** element MUST be set to the **SenderURI** from the last **TraceHeader** in the *message*  
 1507 that has just been received

1508 5) The **RefToMessageId** element MUST be set to the **MessageId** of the *message* that has just  
 1509 been received  
 1510

1511 **10.3.1.4 Resending Lost Messages and Duplicate Filtering**

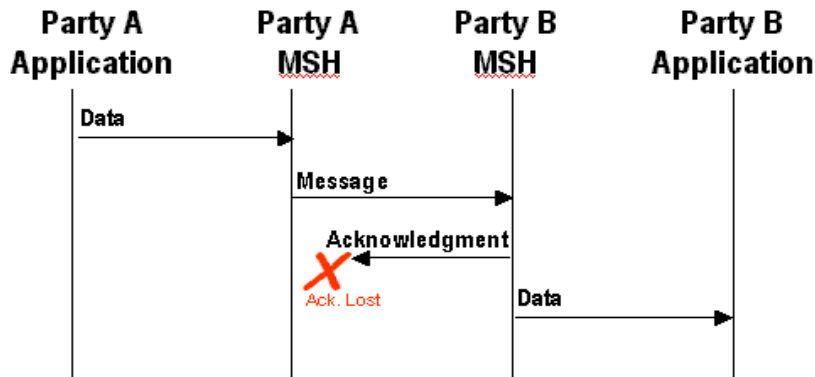
1512 This section describes the behavior that is required by the sender and receiver of a message in  
 1513 order to handle when messages are lost. A message is "lost" when a sending MSH does not  
 1514 receive a response to a message. For example, it is possible that a *message* was lost, for  
 1515 example:



1516

1517 **Figure 10-2 Undelivered Message**

1518 It is also possible that the *Acknowledgment Message* was lost, for example:



1519

1520 **Figure 10-3 Lost Acknowledgment Message**

1521 The rules that apply are as follows:

- 1522 1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has  
 1523 not been received from the Receiving MSH and either of the following are true:
  - 1524 a) The message has not yet been resent and at least the time specified in the **timeout**  
 1525 parameter has passed since the first message was sent, or
  - 1526 b) The message has been resent, and the following are both true:
    - 1527 i) At least the time specified in the **retryInterval** has passed since the last time the  
 1528 message was resent, and



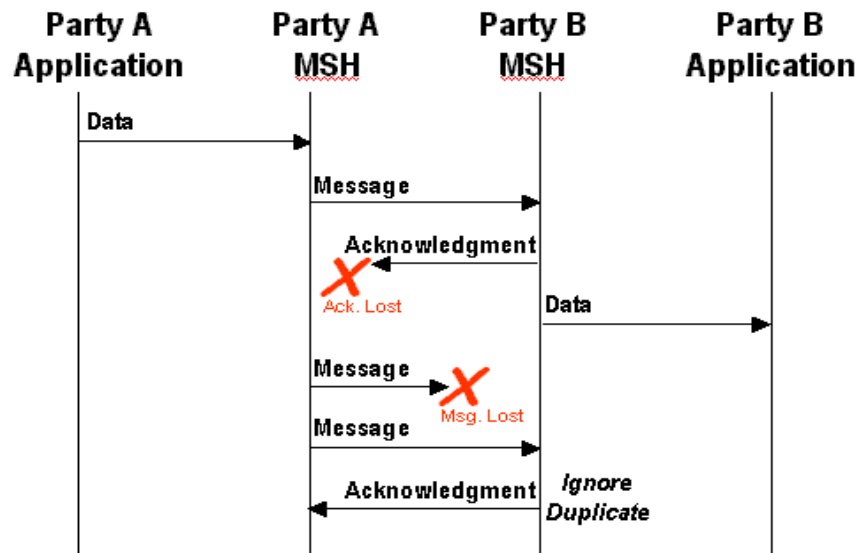
- 1529           ii) The message has been resent less than the number of times specified in the **retries**  
 1530           Parameter
- 1531   2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum  
 1532   number of retries, the Sending MSH SHOULD notify the application and/or system  
 1533   administrator function of the failure to receive an acknowledgement.
- 1534   3) If the Sending MSH detects a communications protocol error that is unrecoverable at the  
 1535   transport protocol level, the Sending MSH SHOULD resend the message using the rules as  
 1536   defined in the CPA.
- 1537

1538   **10.3.1.5 Duplicate Message Handling**

1539   In this context:

- 1540   • an *identical message* is a message that contains, apart from perhaps an additional  
 1541   **TraceHeader** element, the same *ebXML Header* and *ebXML Payload* as the earlier  
 1542   message that was sent.
- 1543   • a *duplicate message* is a message that contains the same **MessageId** as an earlier  
 1544   message that was received.
- 1545   • the first *message* is the message with the earliest **Timestamp** in the **MessageData**  
 1546   element that has the same **RefToMessageId** as the duplicate message.

1547   Note that the Communication Protocol Envelope MAY be different. This means that the same  
 1548   message MAY be sent using different communication protocols and the reliable messaging  
 1549   behavior described in this section will still apply. The ability to use alternative communication  
 1550   protocols is specified in the CPA and is an OPTIONAL implementation specific feature.



1551

1552   **Figure 10-4 Resending Unacknowledged Messages**

1553   The diagram above shows the behavior that MUST be followed by the sending and receiving  
 1554   MSH that are sent with **deliverySemantics** of **OnceAndOnlyOnce**. Specifically:

- 1555   1) The sender of the *message* (e.g. Party A) MUST resend the *identical message* if no  
 1556   *Acknowledgment Message* is received

- 1557 2) When the recipient (Party B) of the *message* receives a *duplicate message*, it MUST resend  
1558 to the sender (Party A) a message identical to the *first message* that was sent to the sender  
1559 Party A)
- 1560 3) The recipient of the *message* (Party B) MUST NOT forward the message a second time to  
1561 the application/process.

#### 1562 **10.4 Failed Message Delivery**

1563 If a message sent with ***deliverySemantics*** set to ***OnceAndOnlyOnce*** cannot be delivered, the  
1564 MSH or process SHOULD send a delivery failure notification to the ***From Party***. The delivery  
1565 failure notification message contains:

- 1566 • a ***From Party*** that identifies the party who detected the problem
- 1567 • a ***To Party*** that identifies the ***From Party*** that created the message that could not be  
1568 delivered
- 1569 • a ***Service*** element and ***Action*** element set as described in 11.5
- 1570 • an ***Error*** element with a severity of:
  - 1571 - ***Error*** if the party who detected the problem could not transmit the message (e.g. the  
1572 communications transport was not available)
  - 1573 - ***Warning*** if the message was transmitted, but an *acknowledgment message* was not  
1574 received. This means that the message probably was not delivered although there is  
1575 a small probability that it was.
- 1576 • an ***ErrorCode*** of ***DeliveryFailure***

1577 It is possible that an error message with an ***Error*** element with an ***ErrorCode*** set to  
1578 ***DeliveryFailure*** cannot be delivered successfully for some reason. If this occurs, then the From  
1579 Party that is the ultimate destination for the error message SHOULD be informed of the problem  
1580 by other means. How this is done is outside the scope of this specification.

#### 1581 **10.5 MSH Time Accuracy**

1582 The ***mshTimeAccuracy*** parameter in the CPA indicates the minimum accuracy that a Receiving  
1583 MSH keeps the clocks it uses when checking, for example, ***TimeToLive***. Its value is in the format  
1584 “mm:ss” which indicates the accuracy in minutes and seconds.

## 1585 11 Error Reporting and Handling

1586 This section describes how one ebXML Message Service Handler (MSH) reports errors it detects  
1587 in an ebXML Message to another MSH. The ebXML Message Service error reporting and  
1588 handling is to be considered as being a layer of processing above the SOAP Processor layer.  
1589 This means that the ebXML MSH is essentially an application-level handler of a SOAP message  
1590 from the perspective of the SOAP Processor. The SOAP Processor MAY generate SOAP Fault  
1591 messages if it is unable to process the message. A Sending MSH MUST be prepared to accept  
1592 and process these SOAP Faults.

1593 It is possible for the ebXML MSH software to cause a SOAP Fault to be generated and returned  
1594 to the sender of a SOAP message. In this event, the returned message MUST conform to the  
1595 [SOAP] specification processing guidelines for SOAP Faults.

1596 An ebXML SOAP message that reports an error that has a **highestSeverity** of **Warning** SHALL  
1597 NOT be reported or returned as a SOAP Fault.

### 1598 11.1 Definitions

1599 For clarity two phrases are defined that are used in this section:

- 1600 • *message in error*. A message that contains or causes an error of some kind
- 1601 • *message reporting the error*. A message that contains an ebXML **ErrorList element** that  
1602 describes the error(s) found in a *message in error*.

### 1603 11.2 Types of Errors

1604 One MSH needs to report to another MSH errors in a *message in error*. For example, errors  
1605 associated with:

- 1606 • the structure or content of the *Message Package* (e.g. MIME) (see section 7),
- 1607 • the ebXML namespace qualified content of the SOAP message document (see section  
1608 8),
- 1609 • reliable messaging failures (see section 10), or
- 1610 • security (see section 12).

1611 Unless specified to the contrary, all references to "an error" in the remainder of this specification  
1612 imply any or all of the types of errors listed above.

1613 Errors associated with Data Communication protocols are detected and reported using the  
1614 standard mechanisms supported by that data communication protocol and are do not use the  
1615 error reporting mechanism described here.

### 1616 11.3 When to generate Error Messages

1617 When an MSH detects an error in a message it is strongly RECOMMENDED that the error is  
1618 reported to the MSH that sent the message that had an error if:

- 1619 • the Error Reporting Location (see section 11.4) to which the *message reporting the error*  
1620 should be sent can be determined, and
- 1621 • the *message in error* does not have an **ErrorList** element with **highestSeverity** set to  
1622 **Error**.

1623 If the Error Reporting Location cannot be found or the *message in error* has an **ErrorList** element  
1624 with **highestSeverity** set to **Error**, it is RECOMMENDED that:

- 1625 • the error is logged,
- 1626 • the problem is resolved by other means, and

- 1627       • no further action is taken.

### 1628   **11.3.1 Security Considerations**

1629   Parties that receive a Message containing an error in the header SHOULD always respond to the  
1630   message. However they MAY ignore the message and not respond if they consider that the  
1631   message received is unauthorized or is part of some security attack. The decision process that  
1632   results in this course of action is implementation dependent.

### 1633   **11.4 Identifying the Error Reporting Location**

1634   The Error Reporting Location is a URI that is specified by the sender of the *message in error* that  
1635   indicates where to send a *message reporting the error*.

1636   The **ErrorURI** implied by the CPA identified by the **CpaID** on the message SHOULD be used. If  
1637   no **ErrorURI** is implied by the CPA, then the **SenderURI** MUST be used.

1638   Even if the *message in error* cannot be successfully analyzed or parsed, MSH implementers  
1639   SHOULD try to determine the Error Reporting Location by other means. How this is done is an  
1640   implementation decision.

### 1641   **11.5 Service and Action Element Values**

1642   An **ErrorList** element can be included in an **ebXMLHeader** that is part of a *message* that is being  
1643   sent as a result of processing of an earlier message. In this case, the values for the **Service** and  
1644   **Action** elements are set by the designer of the Service.

1645   An **ErrorList** element can also be included in an **ebXMLHeader** that is not being sent as a result  
1646   of the processing of an earlier message. In this case, the values of the **Service** and **Action**  
1647   elements MUST be set as follows:

- 1648       • The **Service** element MUST be set to:  
1649         **<http://www.ebxml.org/namespaces/messageService/MessageStatus>**
- 1650       • The **Action** element MUST be set to **MessageError**.

## 1651 **12 Security**

1652 The ebXML Message Service, by its very nature, presents certain security risks. A Message  
1653 Service may be at risk by means of:

- 1654 • Unauthorized access
- 1655 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 1656 • Denial-of-Service, spoofing, bombing attacks

1657 Each security risk is described in detail in the ebXML Technical Architecture Security  
1658 Specification [EBXMLSEC].

1659 Each of these security risks MAY be addressed in whole, or in part, by the application of one, or a  
1660 combination, of the countermeasures described in this section. This specification describes a set  
1661 of profiles, or combinations of selected countermeasures, that have been selected to address key  
1662 risks based upon commonly available technologies. Each of the specified profiles includes a  
1663 description of the risks that are not addressed.

1664 Application of countermeasures SHOULD be balanced against an assessment of the inherent  
1665 risks and the value of the asset(s) that might be placed at risk.

### 1666 **12.1 Security and Management**

1667 No technology, regardless of how advanced it might be, is an adequate substitute to the effective  
1668 application of security management policies and practices.

1669 It is **STRONGLY RECOMMENDED** that the site manager of an ebXML Message Service apply  
1670 due diligence to the support and maintenance of its; security mechanism, site (or physical)  
1671 security procedures, cryptographic protocols, update implementations and apply fixes as  
1672 appropriate. (See <http://www.cert.org/> and <http://ciac.llnl.gov/>)

### 1673 **12.2 Collaboration Protocol Agreement**

1674 The configuration of Security for MSHs is specified in the CPA. Three areas of the CPA have  
1675 security definitions as follows:

- 1676 • The Document Exchange section addresses security to be applied to the payload of the  
1677 message. The MSH is not responsible for any security specified at this level but may  
1678 offer these services to the message sender.
- 1679 • The Message section addresses security applied to the entire ebXML Document, which  
1680 includes the header and the payload.
- 1681 • The Transport section addresses the Transport level. The MSH is not responsible for  
1682 any security specified at this level.

### 1683 **12.3 Countermeasure Technologies**

#### 1684 **12.3.1 Persistent Digital Signature**

1685 If signatures are being used to digitally sign an ebXML Message then XML Signature [DSIG]  
1686 MUST be used to bind the ebXML Header Document to the ebXML Payload or data elsewhere on  
1687 the web that relates to the message. It is also strongly **RECOMMENDED** that XML Signature is  
1688 used to digitally sign the Payload on its own.

1689 The only available technology that can be applied to the purpose of digitally signing an ebXML  
1690 Message (both the ebXML Header and its associated payload objects) is provided by technology  
1691 that conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature  
1692 conforming to this specification can selectively sign portions of an XML document(s), permitting

1693 the documents to be augmented (new element content added) while preserving the validity of the  
1694 signature(s).

1695 An ebXML Message that requires a digital signature SHALL be signed following the process  
1696 defined in this section of the specification and SHALL be in full compliance with [XMLDSIG].

### 1697 12.3.1.1 Signature Generation

1698 1) Create a **SignedInfo** element with SignatureMethod, CanonicalizationMethod, and  
1699 Reference(s) elements for the ebXML Header document and any required payload objects,  
1700 as prescribed by [XMLDSIG].

1701 2) Canonicalize and then calculate the SignatureValue over **SignedInfo** based on algorithms  
1702 specified in SignedInfo as specified in [XMLDSIG].

1703 3) Construct the Signature element that includes the **SignedInfo**, **KeyInfo** (RECOMMENDED),  
1704 and **SignatureValue** elements as specified in [XMLDSIG].

1705 4) Include the namespace qualified **Signature** element in the ebXML Header document just  
1706 signed, following the **TraceHeaderList** element.

1707 The **ds:SignedInfo** element SHALL be composed of zero or one **ds:CanonicalizationMethod**  
1708 element, the **ds:SignatureMethod** and one or more **ds:Reference** elements.

1709 The **ds:CanonicalizationMethod** element is defined as OPTIONAL in [XMLDSIG], meaning that  
1710 the element need not appear in an instance of a **ds:SignedInfo** element. The default  
1711 canonicalization method that is applied to the data to be signed is [XMLC14N] in the absence of a  
1712 **ds:Canonicalization** element that specifies otherwise. This default SHALL also serve as the  
1713 default canonicalization method for the ebXML Message Service.

1714 The **ds:SignatureMethod** element SHALL be present and SHALL have an Algorithm attribute.  
1715 The RECOMMENDED value for the Algorithm attribute is:

1716 <http://www.w3.org/2000/02/xmlsig#sha1>

1717 This RECOMMENDED value SHALL be supported by all compliant ebXML Message Service  
1718 software implementations.

1719 The **ds:Reference** element for the ebXML Header document SHALL have a URI attribute value  
1720 of "" to provide for the signature to be applied to the document that contains the **ds:Signature**  
1721 element (the ebXML Header document). The **ds:Reference** element for the ebXML Header  
1722 document MAY include a **Type** attribute that has a value  
1723 "http://www.w3.org/2000/02/xmlsig#Object" in accordance with [XMLDSIG]. This attribute is  
1724 purely informative. It MAY be omitted. Implementations of the ebXML MSH SHALL be prepared  
1725 to handle either case. The **ds:Reference** element MAY include the optional **id** attribute.

1726 The **ds:Reference** element for the ebXML Header document SHALL include a child  
1727 **ds:Transform** element that excludes the containing **ds:Signature** element and all its  
1728 descendants as well as the **TraceHeaderList** element and all its descendants as these elements  
1729 are subject to change. The **ds:Transform** element SHALL include a child **ds:XPath** element that  
1730 has a value of:  
1731 [/descendant-or-self::node\(\)\[not\(ancestor-or-self::ds:Signature\[@id='S1'\]\)\]](#) and not (ancestor-or-  
1732 self::TraceHeaderList])

1733 Each payload object that requires signing SHALL be represented by a **ds:Reference** element  
1734 that SHALL have a **URI** attribute that resolves to that payload object. This MAY be either the  
1735 Content-Id URI of the payload object enveloped in the MIME ebXML Payload Container, or a URI  
1736 that matches the Content-Location header of the payload object enveloped in the ebXML Payload  
1737 Container, or a URI that resolves to an external payload object that is external to the ebXML  
1738 Payload Container. It is STRONGLY RECOMMENDED that the URI attribute value match the  
1739 xlink:href URI value of the corresponding **Manifest/Reference** element for that payload object.  
1740 However, this is NOT REQUIRED.

1741 Example of digitally signed ebXML SOAP message:

```

1742
1743 <?xml version="1.0" encoding="utf-8"?>
1744 <SOAP-ENV:Envelope
1745   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
1746   xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
1747   xmlns:xlink="http://www.w3.org/1999/xlink">
1748   <SOAP-ENV:Header>
1749     <eb:MessageHeader id="..." eb:version="1.0">
1750       ...
1751     </eb:MessageHeader>
1752     <eb:TraceHeaderList id="..." eb:version="1.0">
1753       <eb:TraceHeader>
1754         ...
1755       </eb:TraceHeader>
1756     </eb:TraceHeaderList>
1757     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlds#">
1758       <ds:SignedInfo>
1759         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20001011"/>
1760         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#dsa-sha1"/>
1761         <ds:Reference URI="">
1762           <ds:Transforms>
1763             <ds:Transform>
1764               <XPath>/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1'])]
1765               and not(ancestor-or-self::TraceHeaderList)]</XPath>
1766             </ds:Transform>
1767           </ds:Transforms>
1768           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
1769           <ds:DigestValue>...</ds:DigestValue>
1770         </ds:Reference>
1771         <ds:Reference URI="cid://blahblahblah/">
1772           <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
1773           <ds:DigestValue>...</ds:DigestValue>
1774         </ds:Reference>
1775       </ds:SignedInfo>
1776       <ds:SignatureValue>...</ds:SignatureValue>
1777       <ds:KeyInfo>...</ds:KeyInfo>
1778     </ds:Signature>
1779   </SOAP-ENV:Header>
1780   <SOAP-ENV:Body>
1781     <eb:Manifest id="Mani01" eb:version="1.0">
1782       <eb:Reference xlink:href="cid://blahblahblah"
1783         xlink:role="http://ebxml.org/gci/invoice">
1784         <eb:Schema eb:version="1.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
1785       </eb:Reference>
1786     </eb:Manifest>
1787   </SOAP-ENV:Body>
1788 </SOAP-ENV:Envelope>

```

1789

### 1790 12.3.2 Persistent Signed Receipt

1791 An ebXML Message that has been digitally signed MAY be acknowledged with a DeliveryReceipt  
 1792 acknowledgment message that itself is digitally signed in the manner described in the previous  
 1793 section. The acknowledgment message MUST contain the set of **ds:DigestValue** elements  
 1794 contained in the **ds:Signature** element of the original message within the **Acknowledgment**  
 1795 element.

### 1796 12.3.3 Non-persistent Authentication

1797 Non-persistent authentication is provided by the communications channel used to transport the  
 1798 ebXML Message. This authentication MAY be either in one direction—from the session initiator to  
 1799 the receiver—or bi-directional. The specific method will be determined by the communications  
 1800 protocol used. For instance, the use of a secure network protocol, such as [RFC2246] or [IPSEC]  
 1801 provides the sender of an ebXML Message to authenticate the destination for the TCP/IP  
 1802 environment.

**1803 12.3.4 Non-persistent Integrity**

1804 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured so as to  
1805 provide for integrity check CRCs of the packets transmitted via the network connection.

**1806 12.3.5 Persistent Confidentiality**

1807 XML Encryption is a W3C/IETF joint activity that is actively engaged in the drafting of a  
1808 specification for the selective encryption of an XML document(s). It is anticipated that this  
1809 specification will be completed within the next year. The ebXML Transport, Routing and  
1810 Packaging team has identified this technology as the only viable means of providing persistent,  
1811 selective confidentiality of elements within an ebXML Message including the ebXML Header  
1812 document.

1813 Confidentiality for ebXML Payloads MAY be provided by functionality possessed by a MSH.  
1814 However, this specification states that it is not the responsibility of the MSH to provide security for  
1815 the ebXML Payloads. Payload confidentiality MAY be provided by using XML Encryption (when  
1816 available) or some other cryptographic process, such as [S/MIME], [S/MIMEV3], or [PGP/MIME],  
1817 that is bilaterally agreed upon by the parties involved. Since XML Encryption is not currently  
1818 available, it is RECOMMENDED that [S/MIME] encryption methods be used for ebXML Payloads.  
1819 The XML Encryption standard SHALL be the default encryption method when XML Encryption  
1820 has achieved W3C Recommendation status.

**1821 12.3.6 Non-persistent Confidentiality**

1822 Use of a secure network protocol such as [RFC2246] or [IPSEC] provides transient confidentiality  
1823 of a message as it is transferred between two ebXML MSH nodes.

**1824 12.3.7 Persistent Authorization**

1825 The OASIS Security Services TC is actively engaged in the definition of a specification that  
1826 provides for the exchange of security credentials, including NameAssertion and Entitlements that  
1827 is based on [S2ML]. Use of technology that is based on this anticipated specification MAY be  
1828 used to provide persistent authorization for an ebXML Message once it becomes available.  
1829 ebXML has a formal liaison to this TC. There are also many ebXML member organizations and  
1830 contributors that are active members of the OASIS Security Services TC such as Sun, IBM,  
1831 CommerceOne, Cisco and others that are endeavoring to ensure that the specification meets the  
1832 requirements of providing persistent authorization capabilities for the ebXML Message Service.

**1833 12.3.8 Non-persistent Authorization**

1834 Use of a secure network protocol such as [RFC2246] or [IPSEC] MAY be configured to provide  
1835 for bilateral authentication of certificates prior to establishing a session. This provides for the  
1836 ability for an ebXML MSH to authenticate the source of a connection that can be used to  
1837 recognize the source as an authorized source of ebXML Messages.

**1838 12.3.9 Trusted Timestamp**

1839 At the time of this specification, services that offer trusted timestamp capabilities are becoming  
1840 available. Once these become more widely available, and a standard has been defined for their  
1841 use and expression, these standards, technologies and services will be evaluated and considered  
1842 for use in providing this capability.



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

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
	Profile 15	✓		✓						✓	sending MSH applies XML/DSIG structures to message, a trusted timestamp is added to message, receiving MSH returns a signed receipt
	Profile 16	✓				✓				✓	Profile 13 with a trusted timestamp applied
	Profile 17	✓		✓		✓				✓	Profile 14 with a trusted timestamp applied
	Profile 18	✓						✓			sending MSH applies XML/DSIG structures to message and forwards authorization credentials (S2ML)
	Profile 19	✓		✓				✓			Profile 18 with receiving MSH returning a signed receipt
	Profile 20	✓		✓				✓		✓	Profile 19 with the a trusted timestamp being applied to the sending MSH message
	Profile 21	✓		✓		✓		✓		✓	Profile 19 with the sending MSH applying confidentiality structures (XML-Encryption)
	Profile 22					✓					sending MSH encapsulates the message within confidentiality structures (XML-Encryption)

1843

1844 **13 References**1845 **13.1 Normative References**

- 1846 [HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J.  
1847 Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, January 1997
- 1848 [RFC 2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E.  
1849 Levinson, Published August 1998
- 1850 [RFC 2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T  
1851 Berners-Lee, Published August 1998
- 1852 [RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One:  
1853 Format of Internet Message Bodies, N Freed & N Borenstein, Published  
1854 November 1996
- 1855 [SMTP] IETF RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982
- 1856 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box,  
1857 DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman,  
1858 Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus  
1859 Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08  
1860 May 2000, <http://www.w3.org/TR/SOAP>
- 1861 [SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs;  
1862 Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000  
1863 <http://www.w3.org/TR/SOAP-attachments>
- 1864 [UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage  
1865 conventions.
- 1866 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second  
1867 Edition), October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 1868 [XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web  
1869 Consortium, 14 January 1999, <http://www.w3.org/TR/REC-xml-names>
- 1870 [XLINK] W3C XML Linking Candidate Recommendation, <http://www.w3.org/TR/xlink/>
- 1871 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,  
1872 <http://www.w3.org/TR/2000/CR-xmlsig-core-20001031/>
- 1873 [XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent, January 2001  
1874

1875 **13.2 Non-Normative References**

- 1876 [Glossary] ebXML Glossary, see ebXML Project Team Home Page
- 1877 [PGP/MIME] IETF RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.  
1878 October 1996.
- 1879 [S/MIME] IETF RFC2311, "S/MIME Version 2 Message Specification", S. Dusse, P.  
1880 Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.
- 1881 [S/MIMECH] IETF RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P.  
1882 Hoffman, B. Ramsdell, J. Weinstein. March 1998.
- 1883 [S/MIMEV3] IETF RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed..  
1884 June 1999.

1885	[TRPREQ]	ebXML Transport, Routing and Packaging: Overview and Requirements, Version 0.96, Published 25 May 2000
1886		
1887	[EBXMLTP]	ebXML Collaboration Protocol Profile and Agreement specification, Version 0.92, published 3 March, 2001
1888		
1889	[EBXMLTA]	ebXML Technical Architecture, version 1.04 published 16 February, 2001
1890	[EBXMLTASEC]	ebXML Technical Architecture Security Specification, version 0.3 published 19 February, 2001
1891		
1892	[EBXMLRSS]	ebXML Registry Services Specification, version 0.84
1893	[TLS]	RFC2246, T. Dierks, C. Allen. January 1999.
1894	[IPSEC]	IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November 1998. RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November 1998.
1895		
1896		
1897	[XMLSchema]	W3C XML Schema Candidate Recommendation,
1898		<a href="http://www.w3.org/TR/xmlschema-0/">http://www.w3.org/TR/xmlschema-0/</a>
1899		<a href="http://www.w3.org/TR/xmlschema-1/">http://www.w3.org/TR/xmlschema-1/</a>
1900		<a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a>
1901	[XMTP]	XMTP - Extensible Mail Transport Protocol
1902		<a href="http://www.openhealth.org/documents/xmtp.htm">http://www.openhealth.org/documents/xmtp.htm</a>

1903 **14 Disclaimer**

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

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

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

1917

1918 **Vice Team Leader**

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

1926

1927 **Team Editor**

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

1935

1936 **Authors**

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

1943

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

1951

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

1959

1960 Name John Ibbotson  
 1961 Company IBM UK Ltd

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

## 2014 Appendix A ebXMLHeader Schema

### 2015 A.1

2016 The following is the definition of the ebXML SOAP Header extension elements as a schema that  
2017 conforms to [XMLSchema].

```

2018 <?xml version="1.0" encoding="UTF-8"?>
2019 <xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
2020 targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
2021 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" version="0.98"
2022 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
2023   <xsd:import namespace="http://www.w3.org/1999/xlink"
2024     schemaLocation="http://www.w3.org/1999/xlink"/>
2025   <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
2026     schemaLocation="http://schemas.xmlsoap.org/soap/envelope"/>
2027   <!-- MANIFEST -->
2028   <xsd:element name="Manifest">
2029     <xsd:complexType>
2030       <xsd:sequence>
2031         <xsd:element ref="Reference" maxOccurs="unbounded"/>
2032         <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2033       </xsd:sequence>
2034       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2035       <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2036       <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2037     </xsd:complexType>
2038   </xsd:element>
2039   <xsd:element name="Reference">
2040     <xsd:complexType>
2041       <xsd:sequence>
2042         <xsd:element ref="Schema" minOccurs="0" maxOccurs="unbounded"/>
2043         <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2044         <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2045       </xsd:sequence>
2046       <xsd:attribute name="id" use="required" type="xsd:ID"/>
2047       <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
2048       <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
2049       <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
2050     </xsd:complexType>
2051   </xsd:element>
2052   <xsd:element name="Schema">
2053     <xsd:complexType>
2054       <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
2055       <xsd:attribute name="version" type="xsd:string"/>
2056     </xsd:complexType>
2057   </xsd:element>
2058   <!-- HEADER -->
2059   <xsd:element name="MessageHeader">
2060     <xsd:complexType>
2061       <xsd:sequence>
2062         <xsd:element ref="From"/>
2063         <xsd:element ref="To"/>
2064         <xsd:element ref="CPAId"/>
2065         <xsd:element ref="ConversationId"/>
2066         <xsd:element ref="Service"/>
2067         <xsd:element ref="Action"/>
2068         <xsd:element ref="MessageData"/>
2069         <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
2070         <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
2071         <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
2072       </xsd:sequence>
2073       <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2074       <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2075

```



```

2076     </xsd:complexType>
2077 </xsd:element>
2078 <xsd:element name="CPAId" type="xsd:string"/>
2079 <xsd:element name="ConversationId" type="xsd:string"/>
2080 <xsd:element name="Service" type="xsd:string"/>
2081 <xsd:element name="Action" type="xsd:string"/>
2082 <xsd:element name="MessageData">
2083   <xsd:complexType>
2084     <xsd:sequence>
2085       <xsd:element ref="MessageId"/>
2086       <xsd:element ref="Timestamp"/>
2087       <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
2088       <xsd:element ref="TimeToLive" minOccurs="0" maxOccurs="1"/>
2089     </xsd:sequence>
2090   </xsd:complexType>
2091 </xsd:element>
2092 <xsd:element name="MessageId" type="xsd:string"/>
2093 <xsd:element name="TimeToLive" type="xsd:timeInstant"/>
2094 <xsd:element name="QualityOfServiceInfo">
2095   <xsd:complexType>
2096     <xsd:attribute name="deliverySemantics" use="default" value="BestEffort">
2097       <xsd:simpleType>
2098         <xsd:restriction base="xsd:NMTOKEN">
2099           <xsd:enumeration value="OnceAndOnlyOnce"/>
2100           <xsd:enumeration value="BestEffort"/>
2101         </xsd:restriction>
2102       </xsd:simpleType>
2103     </xsd:attribute>
2104     <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed">
2105       <xsd:simpleType>
2106         <xsd:restriction base="xsd:NMTOKEN">
2107           <xsd:enumeration value="Guaranteed"/>
2108           <xsd:enumeration value="NotGuaranteed"/>
2109         </xsd:restriction>
2110       </xsd:simpleType>
2111     </xsd:attribute>
2112     <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
2113       <xsd:simpleType>
2114         <xsd:restriction base="xsd:NMTOKEN">
2115           <xsd:enumeration value="Signed"/>
2116           <xsd:enumeration value="Unsigned"/>
2117           <xsd:enumeration value="None"/>
2118         </xsd:restriction>
2119       </xsd:simpleType>
2120     </xsd:attribute>
2121   </xsd:complexType>
2122 </xsd:element>
2123 <!-- TRACE HEADER LIST -->
2124 <xsd:element name="TraceHeaderList">
2125   <xsd:complexType>
2126     <xsd:sequence>
2127       <xsd:element ref="TraceHeader" maxOccurs="unbounded"/>
2128     </xsd:sequence>
2129     <xsd:attribute name="id" type="xsd:ID"/>
2130     <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2131     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2132   </xsd:complexType>
2133 </xsd:element>
2134 <xsd:element name="TraceHeader">
2135   <xsd:complexType>
2136     <xsd:sequence>
2137       <xsd:element ref="SenderURI"/>
2138       <xsd:element ref="ReceiverURI"/>
2139       <xsd:element ref="Timestamp"/>
2140       <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
2141     </xsd:sequence>
2142     <xsd:attribute name="reliableMessagingMethod">
2143       <xsd:simpleType>
2144         <xsd:restriction base="xsd:NMTOKEN">
2145           <xsd:enumeration value="ebXML"/>
2146           <xsd:enumeration value="Transport"/>

```

```

2147     </xsd:restriction>
2148   </xsd:simpleType>
2149 </xsd:attribute>
2150 <xsd:attribute name="ackRequested">
2151   <xsd:simpleType>
2152     <xsd:restriction base="xsd:NMTOKEN">
2153       <xsd:enumeration value="Signed"/>
2154       <xsd:enumeration value="Unsigned"/>
2155       <xsd:enumeration value="None"/>
2156     </xsd:restriction>
2157   </xsd:simpleType>
2158 </xsd:attribute>
2159 </xsd:complexType>
2160 </xsd:element>
2161 <xsd:element name="SenderURI" type="xsd:uriReference"/>
2162 <xsd:element name="ReceiverURI" type="xsd:uriReference"/>
2163 <xsd:element name="SequenceNumber" type="xsd:positiveInteger"/>
2164 <!-- ACKNOWLEDGEMENT -->
2165 <xsd:element name="Acknowledgment">
2166   <xsd:complexType>
2167     <xsd:sequence>
2168       <xsd:element ref="Timestamp"/>
2169       <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
2170     </xsd:sequence>
2171     <xsd:attribute name="id" type="xsd:ID"/>
2172     <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2173     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2174     <xsd:attribute name="type" use="default" value="DeliveryReceipt">
2175       <xsd:simpleType>
2176         <xsd:restriction base="xsd:NMTOKEN">
2177           <xsd:enumeration value="DeliveryReceipt"/>
2178           <xsd:enumeration value="IntermediateAck"/>
2179         </xsd:restriction>
2180       </xsd:simpleType>
2181     </xsd:attribute>
2182     <xsd:attribute name="signed" type="xsd:boolean"/>
2183   </xsd:complexType>
2184 </xsd:element>
2185 <!-- ERROR LIST -->
2186 <xsd:element name="ErrorList">
2187   <xsd:complexType>
2188     <xsd:sequence>
2189       <xsd:element ref="Error" maxOccurs="unbounded"/>
2190     </xsd:sequence>
2191     <xsd:attribute name="id" type="xsd:ID"/>
2192     <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2193     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2194     <xsd:attribute name="highestSeverity" use="default" value="Warning">
2195       <xsd:simpleType>
2196         <xsd:restriction base="xsd:string">
2197           <xsd:enumeration value="Warning"/>
2198           <xsd:enumeration value="Error"/>
2199         </xsd:restriction>
2200       </xsd:simpleType>
2201     </xsd:attribute>
2202   </xsd:complexType>
2203 </xsd:element>
2204 <xsd:element name="Error">
2205   <xsd:complexType>
2206     <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
2207     <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
2208     <xsd:attribute name="severity" use="default" value="Warning">
2209       <xsd:simpleType>
2210         <xsd:restriction base="xsd:NMTOKEN">
2211           <xsd:enumeration value="Warning"/>
2212           <xsd:enumeration value="Error"/>
2213         </xsd:restriction>
2214       </xsd:simpleType>
2215     </xsd:attribute>
2216     <xsd:attribute name="location" type="xsd:string"/>
2217     <xsd:attribute name="xml:lang" type="xsd:language"/>

```

```

2218     <xsd:attribute name="errorMessage" type="xsd:string"/>
2219   </xsd:complexType>
2220 </xsd:element>
2221 <!-- STATUS DATA -->
2222 <xsd:element name="StatusData">
2223   <xsd:complexType>
2224     <xsd:sequence>
2225       <xsd:element ref="RefToMessageId"/>
2226       <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
2227     </xsd:sequence>
2228     <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
2229     <xsd:attribute ref="soap:mustUnderstand" use="required"/>
2230     <xsd:attribute name="messageStatus">
2231       <xsd:simpleType>
2232         <xsd:restriction base="xsd:NMTOKEN">
2233           <xsd:enumeration value="Unauthorized"/>
2234           <xsd:enumeration value="NotRecognized"/>
2235           <xsd:enumeration value="Received"/>
2236           <xsd:enumeration value="Processed"/>
2237           <xsd:enumeration value="Forwarded"/>
2238         </xsd:restriction>
2239       </xsd:simpleType>
2240     </xsd:attribute>
2241   </xsd:complexType>
2242 </xsd:element>
2243 <!-- COMMON ELEMENTS -->
2244 <xsd:element name="PartyId">
2245   <xsd:complexType>
2246     <xsd:simpleContent>
2247       <xsd:extension base="xsd:string">
2248         <xsd:attribute name="type" type="xsd:string"/>
2249       </xsd:extension>
2250     </xsd:simpleContent>
2251   </xsd:complexType>
2252 </xsd:element>
2253 <xsd:element name="To">
2254   <xsd:complexType>
2255     <xsd:sequence>
2256       <xsd:element ref="PartyId"/>
2257     </xsd:sequence>
2258   </xsd:complexType>
2259 </xsd:element>
2260 <xsd:element name="From">
2261   <xsd:complexType>
2262     <xsd:sequence>
2263       <xsd:element ref="PartyId"/>
2264     </xsd:sequence>
2265   </xsd:complexType>
2266 </xsd:element>
2267 <xsd:element name="Description">
2268   <xsd:complexType>
2269     <xsd:simpleContent>
2270       <xsd:extension base="xsd:string">
2271         <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
2272       </xsd:extension>
2273     </xsd:simpleContent>
2274   </xsd:complexType>
2275 </xsd:element>
2276 <xsd:element name="RefToMessageId" type="xsd:string"/>
2277 <xsd:element name="Timestamp" type="xsd:timeInstant"/>
2278 <!-- VIA -->
2279 <xsd:element name="Via">
2280   <xsd:complexType>
2281     <xsd:sequence>
2282       <xsd:element ref="CPAId" minOccurs="0"/>
2283       <xsd:element ref="Service" minOccurs="0"/>
2284       <xsd:element ref="Action" minOccurs="0"/>
2285     </xsd:sequence>
2286     <xsd:attribute name="version" use="required" type="xsd:string"/>
2287     <xsd:attribute ref="soap:mustUnderstand"/>
2288     <xsd:attribute ref="soap:actor"/>

```

```
2289 <xsd:attribute name="syncReply" type="xsd:boolean"/>
2290 <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
2291   <xsd:simpleType>
2292     <xsd:restriction base="xsd:string">
2293       <xsd:enumeration value="Signed"/>
2294       <xsd:enumeration value="Unsigned"/>
2295       <xsd:enumeration value="None"/>
2296     </xsd:restriction>
2297   </xsd:simpleType>
2298 </xsd:attribute>
2299 <xsd:attribute name="reliableMessagingMethod">
2300   <xsd:simpleType>
2301     <xsd:restriction base="xsd:string">
2302       <xsd:enumeration value="ebXML"/>
2303       <xsd:enumeration value="Transport"/>
2304     </xsd:restriction>
2305   </xsd:simpleType>
2306 </xsd:attribute>
2307 <xsd:attribute name="ackRequested" type="xsd:boolean"/>
2308 </xsd:complexType>
2309 </xsd:element>
2310 </xsd:schema>
```

## 2311 **Appendix B Communication Protocol Bindings**

### 2312 **B.1 Introduction**

2313 One of the goals of ebXML's Transport, Routing and Packaging team is to design a message  
2314 handling service that is usable over a variety of network and application level communication  
2315 protocols. These protocols serve as the "carrier" of ebXML Messages and provide the underlying  
2316 services necessary to carry out a complete ebXML Message exchange between two parties.  
2317 HTTP, FTP, MQSeries and SMTP are examples of application level communication protocols.  
2318 TCP and SNA/LU6.2 are examples of network transport protocols. Communication protocols vary  
2319 in their support for data content, processing behavior and error handling and reporting. For  
2320 example, it is customary to send binary data in raw form over HTTP. However, in the case of  
2321 SMTP it is customary to "encode" binary data into a 7-bit representation. HTTP is equally  
2322 capable of carrying out synchronous or asynchronous message exchanges whereas it is likely  
2323 that message exchanges occurring over SMTP will be asynchronous. This section describes the  
2324 technical details needed to implement this abstract ebXML Message Handling Service over  
2325 particular communication protocols.

2326 This section specifies communication protocol bindings and technical details for carrying ebXML  
2327 Messaging Service messages for the following communication protocols:

- 2328 • Hypertext Transfer Protocol [HTTP], in both asynchronous and synchronous forms of  
2329 transfer.
- 2330 • Simple Mail Transfer Protocol [SMTP], in asynchronous form of transfer only.

### 2331 **B.2 HTTP**

#### 2332 **B.2.1 Minimum level of HTTP protocol**

2333 Hypertext Transfer Protocol Version 1.1 [HTTP] (<http://www.ietf.org/rfc2616.txt>) is the minimum  
2334 level of protocol that MUST be used.

#### 2335 **B.2.2 Sending ebXML Service messages over HTTP**

2336 Even though several HTTP request methods are available, this specification only defines the use  
2337 of HTTP POST requests for sending ebXML Message Service messages over HTTP. The identity  
2338 of the ebXML MSH (e.g. ebxmlhandler) may be part of the HTTP POST request:

```
2339 POST /ebxmlhandler HTTP/1.1  
2340
```

2341 Prior to sending over HTTP, an ebXML Message MUST be formatted according to ebXML  
2342 Message Service Specification sections 7 and 8. Additionally, the messages MUST conform to the  
2343 HTTP specific MIME canonical form constraints specified in section 19.4 of RFC 2616 [HTTP]  
2344 specification (see: <http://www.ietf.org/rfc2616.txt>).  
2345

2346 HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is OPTIONAL  
2347 for such parts in an ebXML Service Message prior to sending over HTTP. However, content-  
2348 transfer-encoding of such parts (e.g. using base64 encoding scheme) is not precluded by this  
2349 specification.  
2350

2351 The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- 2352 • The **Content-Type: multipart/related** MIME header with the associated  
2353 parameters, from the ebXML Service Message Envelope MUST appear as an HTTP  
2354 header.
  - 2355 • All other MIME headers that constitute the ebXML Message Envelope MUST also  
2356 become part of the HTTP header.
  - 2357 • The mandatory SOAPAction HTTP header field must also be included in the HTTP  
2358 header and must have a value of ebXML.
- 2359 **SOAPAction: ebXML**
- 2360 • Other headers with semantics defined by MIME specifications, such as Content-Transfer-  
2361 Encoding, SHALL NOT appear as HTTP headers. Specifically, the "MIME-Version: 1.0"  
2362 header MUST NOT appear as an HTTP header. However, HTTP-specific MIME-like  
2363 headers defined by HTTP 1.1 MAY be used with the semantic defined in the HTTP  
2364 specification.
  - 2365 • All ebXML Service Message parts that follow the ebXML Message Envelope, including  
2366 the MIME boundary string, constitute the HTTP entity body. This encompasses the SOAP  
2367 envelope and the constituent ebXML parts and attachments including the trailing MIME  
2368 boundary strings.

2369 The example below shows an example instance of an HTTP POST'ed ebXML Service Message:

```

2370 POST /servlet/ebXMLhandler HTTP/1.1
2371 Host: www.example2.com
2372 SOAPAction:
2373 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2374 start=" <ebxhmheader111@example.com>"
2375
2376 --Boundary
2377 Content-ID: <ebxhmheader111@example.com>
2378 Content-Type: text/xml
2379 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2380 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2381 <SOAP-ENV:Header>
2382 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2383 <eb:From>
2384 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2385 </eb:From>
2386 <eb:To>
2387 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2388 </eb:To>
2389 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2390 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2391 <eb:Service>OrderProcessing</eb:Service>
2392 <eb:Action>NewOrder</eb:Action>
2393 <eb:MessageData>
2394 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2395 <eb:Timestamp>20010215111212Z</Timestamp>
2396 </eb:MessageData>
2397 <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
2398 </eb:MessageHeader>
2399 </SOAP-ENV:Header>
2400 <SOAP-ENV:Body>
2401 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2402 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2403 xlink:role="XLinkRole"
2404 xlink:type="simple">
2405 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2406 </eb:Reference>
2407 </eb:Manifest>
2408 </SOAP-ENV:Body>
2409 </SOAP-ENV:Envelope>
2410 --Boundary
2411 Content-ID: <ebxmlpayload111@example.com>
2412 Content-Type: text/xml
2413 <?xml version="1.0" encoding="UTF-8"?>
2414

```

```
2415 <purchase_order>
2416   <po_number>1</po_number>
2417   <part_number>123</part_number>
2418   <price currency="USD">500.00</price>
2419 </purchase_order>
2420 --BoundaryY--
```

### 2421 B.2.3 HTTP Response Codes

2422 In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be  
2423 followed, for returning the HTTP level response codes. A 2xx code MUST be returned when the  
2424 HTTP Posted message is successfully received by the receiving HTTP entity. However, see  
2425 exception for SOAP error conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx  
2426 range MAY be returned for conditions corresponding to them. However, error conditions  
2427 encountered while processing an ebXML Service Message MUST be reported using the error  
2428 mechanism defined by the ebXML Message Service Specification.

### 2429 B.2.4 SOAP Error conditions and Synchronous Exchanges

2430 The SOAP 1.1 specification states:

2431 *"In case of a SOAP error while processing the request, the SOAP HTTP server MUST issue an*  
2432 *HTTP 500 "Internal Server Error" response and include a SOAP message in the response*  
2433 *containing a SOAP Fault element indicating the SOAP processing error. "*

2434 However, the scope of the SOAP 1.1 specification is limited to synchronous mode of message  
2435 exchange over HTTP, whereas the ebXML Message Service Specification specifies both  
2436 synchronous and asynchronous modes of message exchange over HTTP. Hence, the SOAP 1.1  
2437 specification MUST be followed for synchronous mode of message exchange, where the SOAP  
2438 message containing a SOAP Fault element indicating the SOAP processing error MUST be  
2439 returned in the HTTP response with a response code of "HTTP 500 Internal Server Error". When  
2440 asynchronous mode of message exchange is being used, a HTTP response code in the range  
2441 2xx MUST be returned when the message is received successfully and any error conditions  
2442 (including SOAP errors) must be returned via a separate HTTP Post.

### 2443 B.2.5 Synchronous vs. Asynchronous

2444 When the **syncReplyMode** parameter in the ebXML Header is set to "true", the response  
2445 message(s) MUST be returned on the same HTTP connection as the inbound request, with an  
2446 appropriate HTTP response code, as described in section 1.2.3. See also section 1.2.3.1 on  
2447 SOAP error conditions and synchronous exchanges.

2448  
2449 When the **syncReplyMode** parameter in the ebXML Header is set to "false", the response  
2450 messages are not returned on the same HTTP connection as the inbound request, but using an  
2451 independent HTTP Post request. An HTTP response with a response code as defined in section  
2452 1.2.3 above and with an empty HTTP body MUST be returned in response to the HTTP Post,  
2453 however.

### 2454 B.2.6 Access Control

2455 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access  
2456 through the use of an access control mechanism. The HTTP access authentication process  
2457 described in "HTTP Authentication: Basic and Digest Access Authentication" [RFC2617] defines  
2458 the access control mechanisms allowed to protect an ebXML Message Service Handler from  
2459 unauthorized access.

2460 Implementers MAY support all of the access control schemes defined in [RFC2617] however they  
2461 MUST support the Basic Authentication mechanism, as described in section 2, when Access  
2462 Control is used.

2463 Implementers that use basic authentication for access control SHOULD also use communication  
2464 protocol level security, as specified in the section titled "Confidentiality and Communication  
2465 Protocol Level Security" in this document.

## 2466 **B.2.7 Confidentiality and Communication Protocol Level Security**

2467 An ebXML Message Service Handler MAY use transport layer encryption to protect the  
2468 confidentiality of ebXML Messages and HTTP transport headers. The IETF Transport Layer  
2469 Security specification [RFC2246] provides the specific technical details and list of allowable  
2470 options, which may be used by ebXML Message Service Handlers. ebXML Message Service  
2471 Handlers MUST be capable of operating in backwards compatibility mode with SSL [SSL3], as  
2472 defined in Appendix E of [RFC2246].

2473 ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key  
2474 sizes specified within [RFC2246]. At a minimum ebXML Message Service Handlers MUST  
2475 support the key sizes and algorithms necessary for backward compatibility with [SSL3].

2476 The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that  
2477 stronger encryption keys/algorithms SHOULD be used.

2478 Both [RFC2246] and [SSL3] require the use of server side digital certificates. In addition client  
2479 side certificate based authentication is also permitted. ebXML message service handlers MUST  
2480 support 3rd party signed certificates as well as "self signed" certificates.

## 2481 **B.3 SMTP**

2482 The Simple Mail Transfer Protocol [RFC821] and its companion documents [RFC822] and  
2483 [ESMTP] makeup the suite of specifications commonly referred to as Internet Electronic Mail.  
2484 These specifications have been augmented over the years by other specifications, which define  
2485 additional functionality "layered on top" of these baseline specifications. These include:

- 2486 • Multipurpose Internet Mail Extensions (MIME) [RFC2045], [RFC2046], [RFC2387]
- 2487 • SMTP Service Extension for Authentication [RFC2554]
- 2488 • SMTP Service Extension for Secure SMTP over TLS [RFC2487]

2489

2490 Typically, Internet Electronic Mail Implementations consist of two "agent" types:

- 2491 • Message Transfer Agent (MTA): Programs that send and receive mail messages with  
2492 other MTA's on behalf of MUA's. Microsoft Exchange Server is an example of a MTA
- 2493 • Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail  
2494 messages and communicate with an MTA to send/retrieve mail messages. Microsoft  
2495 Outlook is an example of a MUA.

2496 MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

2497

2498 MUA's are responsible for constructing electronic mail messages in accordance with the Internet  
2499 Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML  
2500 compliant message for transport via e-mail from the perspective of a MUA. No attempt is made to  
2501 define the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.



### 2502 **B.3.1 Minimum level of supported protocols**

- 2503 • Simple Mail Transfer Protocol [RFC821] and [RFC822]
- 2504 • MIME [RFC2045] and [RFC2046]
- 2505 • Multipart/Related MIME [RFC2387]

2506

### 2507 **B.3.2 Sending ebXML Messages over SMTP**

2508

2509 Prior to sending messages over SMTP an ebXML Message MUST be formatted according to  
2510 ebXML Message Service Specification sections 7 and 8. Additionally the messages must also  
2511 conform to the syntax, format and encoding rules specified by MIME [RFC2045], [RFC2046] and  
2512 [RFC2387].

2513 Many types of data that a party might desire to transport via email are represented as 8bit  
2514 characters or binary data. Such data cannot be transmitted over SMTP [RFC821], which  
2515 restricts mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including  
2516 any trailing CRLF line separator. If a sending Message Service Handler knows that a receiving

2517 MTA, or ANY intermediary MTA's, are restricted to handling 7-bit data then any ebXML  
2518 header or payload data that uses 8 bit (or binary) representation must be "transformed"  
2519 according to the encoding rules specified in section 6 of [RFC2045]. In cases where a  
2520 Message Service Handler knows that a receiving MTA and ALL intermediary MTA's are  
2521 capable of handling 8-bit data then no transformation is needed on any part of the ebXML  
2522 Message.  
2523

2524 The rules for forming an ebXML Message for transport via SMTP are as follows:

- 2525 • **If using [RFC821] restricted transport paths**, apply transfer encoding to all 8-bit data  
2526 that will be transported in a ebXML header or payload body part, according to the  
2527 encoding rules defined in section 6 of [RFC2045]. The Content-Transfer-Encoding MIME  
2528 header MUST be included in the MIME envelope portion of any body part that has been  
2529 transformed (encoded).
- 2530 • The Content-Type: `Multipart/Related` MIME header with the associated parameters,  
2531 from the ebXML Message Envelope MUST appear as an email MIME header.
- 2532 • All other MIME headers that constitute the ebXML Message Envelope MUST also  
2533 become part of the email MIME header.
- 2534 • The mandatory SOAPAction MIME header field must also be included in the e-mail MIME  
2535 header and must have the value of ebXML:

2536 SOAPAction: **ebXML**

2537 Where Service and Action are values of the corresponding elements from  
2538 the ebXML MessageHeader.

- 2539 • The "MIME-Version: 1.0" header must appear as an email MIME header.
- 2540 • The e-mail header "To:" MUST contain the [RFC822] compliant e-mail address of the  
2541 ebXML Message Service Handler.
- 2542 • The e-mail header "From:" MUST contain the [RFC822] compliant e-mail address of the  
2543 senders ebXML message service handler.
- 2544 • Construct a "Date:" e-mail header in accordance with [RFC822]

- 2545 • Other headers MAY occur within the e-mail message header in accordance with  
 2546 [RFC822] and [RFC2045], however ebXML Message Service Handlers MAY choose to  
 2547 ignore them.

2548 The example below shows a minimal example of an e-mail message containing an ebXML  
 2549 Message:

```

2550 From: ebXMLhandler@example.com
2551 To: ebXMLhandler@example2.com
2552 Date: Thu, 08 Feb 2001 19:32:11 CST
2553 MIME-Version: 1.0
2554 SOAPAction:
2555 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
2556 start=" <ebxmhheader111@example.com>"
2557
2558 --Boundary
2559 Content-ID: <ebxmhheader111@example.com>
2560 Content-Type: text/xml
2561
2562 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
2563 xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
2564 <SOAP-ENV:Header>
2565 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2566 <eb:From>
2567 <eb:PartyId>urn:duns:123456789</eb:PartyId>
2568 </eb:From>
2569 <eb:To>
2570 <eb:PartyId>urn:duns:912345678</eb:PartyId>
2571 </eb:To>
2572 <eb:CPAId>20001209-133003-28572</eb:CPAId>
2573 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
2574 <eb:Service>OrderProcessing</eb:Service>
2575 <eb:Action>NewOrder</eb:Action>
2576 <eb:MessageData>
2577 <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
2578 <eb:Timestamp>20010215111212Z</Timestamp>
2579 </eb:MessageData>
2580 <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
2581 </eb:MessageHeader>
2582 </SOAP-ENV:Header>
2583 <SOAP-ENV:Body>
2584 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2585 <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
2586 xlink:role="XLinkRole"
2587 xlink:type="simple">
2588 <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
2589 </eb:Reference>
2590 </eb:Manifest>
2591 </SOAP-ENV:Body>
2592 </SOAP-ENV:Envelope>
2593 --Boundary
2594 Content-ID: <ebxmhheader111@example.com>
2595 Content-Type: text/xml
2596 <?xml version="1.0" encoding="UTF-8"?>
2597 <purchase_order>
2598 <po_number>1</po_number>
2599 <part_number>123</part_number>
2600 <price currency="USD">500.00</price>
2601 </purchase_order>
2602 --Boundary--
  
```

### 2604 B.3.3 Response Messages

2605 All ebXML response messages, including errors and acknowledgements, are delivered  
 2606 asynchronously between ebXML Message Service Handlers. Each response message MUST be

2607 constructed in accordance with the rules specified in the section titled "Sending ebXML  
2608 messages over SMTP" elsewhere in this document.

2609 ebXML Message Service Handlers MUST be capable of receiving a delivery failure notification  
2610 message sent by an MTA. An MSH that receives a delivery failure notification message  
2611 SHOULD examine the message to determine which ebXML message, sent by the MSH, resulted  
2612 in a message delivery failure. The MSH SHOULD attempt to identify the application responsible  
2613 for sending the offending message that caused the failure. The MSH SHOULD attempt to notify  
2614 the application that a message delivery failure has occurred. If the MSH is unable to determine  
2615 the source of the offending message the MSH administrator should be notified.

2616 MSH's which cannot identify a received message as a valid ebXML message or a message  
2617 delivery failure SHOULD retain the unidentified message in a "dead letter" folder.

2618 A MSH SHOULD place an entry in an audit log indicating the disposition of each received  
2619 message.

#### 2620 **B.3.4 Access Control**

2621 Implementers MAY protect their ebXML Message Service Handlers from unauthorized access  
2622 through the use of an access control mechanism. The SMTP access authentication process  
2623 described in "SMTP Service Extension for Authentication" [RFC2554] defines the ebXML  
2624 recommended access control mechanism to protect a SMTP based ebXML Message Service  
2625 Handler from unauthorized access.

#### 2626 **B.3.5 Confidentiality and Communication Protocol Level Security**

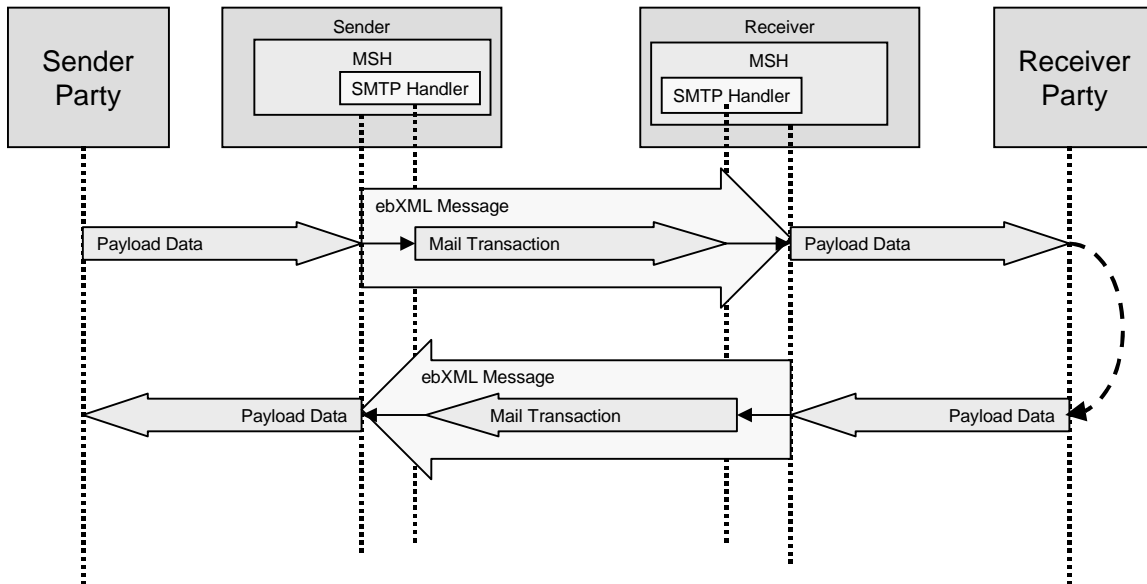
2627

2628 An ebXML Message Service Handler MAY use transport layer encryption to protect the  
2629 confidentiality of ebXML messages. The IETF "SMTP Service Extension for Secure SMTP over  
2630 TLS" specification [RFC2487] provides the specific technical details and list of allowable options,  
2631 which may be used.

#### 2632 **B.3.6 SMTP Model**

2633 All ebXML Message Service messages carried as mail in an [SMTP] Mail Transaction as shown  
2634 in the figure below.

2635



2636

2637

2638 **B.4 Communication Errors during Reliable Messaging**

2639 When the Sender or the Receiver detects a transport protocol level error (such as an HTTP,  
 2640 SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport  
 2641 recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does  
 2642 Reliable Messaging recovery take place (see section 10).

2643

**2644 Copyright Statement**

2645 Copyright © ebXML 2001. All Rights Reserved.

2646

2647 This document and translations of it MAY be copied and furnished to others, and derivative  
2648 works that comment on or otherwise explain it or assist in its implementation MAY be prepared,  
2649 copied, published and distributed, in whole or in part, without restriction of any kind, provided that  
2650 the above copyright notice and this paragraph are included on all such copies and derivative  
2651 works. However, this document itself MAY not be modified in any way, such as by removing the  
2652 copyright notice or references to the ebXML, UN/CEFACT, or OASIS, except as required to  
2653 translate it into languages other than English.

2654

2655 The limited permissions granted above are perpetual and will not be revoked by ebXML or its  
2656 successors or assigns.

2657

2658 This document and the information contained herein is provided on an  
2659 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
2660 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION  
2661 HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF  
2662 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

2663