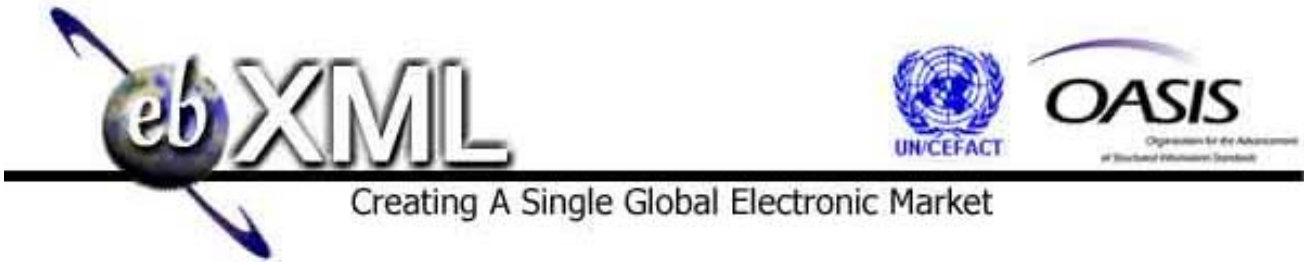


Message Service Specification

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.pdfLatest version

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

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

4 Introduction

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

4.1 Summary of Contents of Document

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

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

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

This specification is organized around the following topics:

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

Appendices to this specification cover the following:

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

4.2 Document Conventions

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

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

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

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

4.3 Audience

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

4.4 Caveats and Assumptions

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

4.5 Related Documents

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

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

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

5 Design Objectives

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

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

6 System Overview

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

6.1 What the Message Service does

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

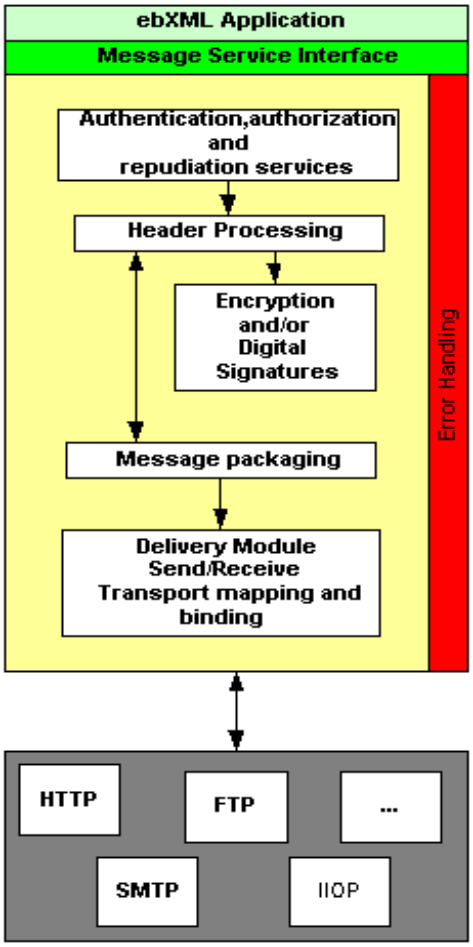
The ebXML Message Service is provided for environments requiring a robust, yet low cost solution to enable electronic business. It is one of the four "infrastructure" components of ebXML. The other three are: Registry/Repository [EBXMLRSS], Collaboration Protocol Profile/Agreement [EBXMLTP] and ebXML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

6.2 Message Service Overview

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

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

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



148

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

7 Packaging Specification

7.1 Introduction

An ebXML Message has the following structure:

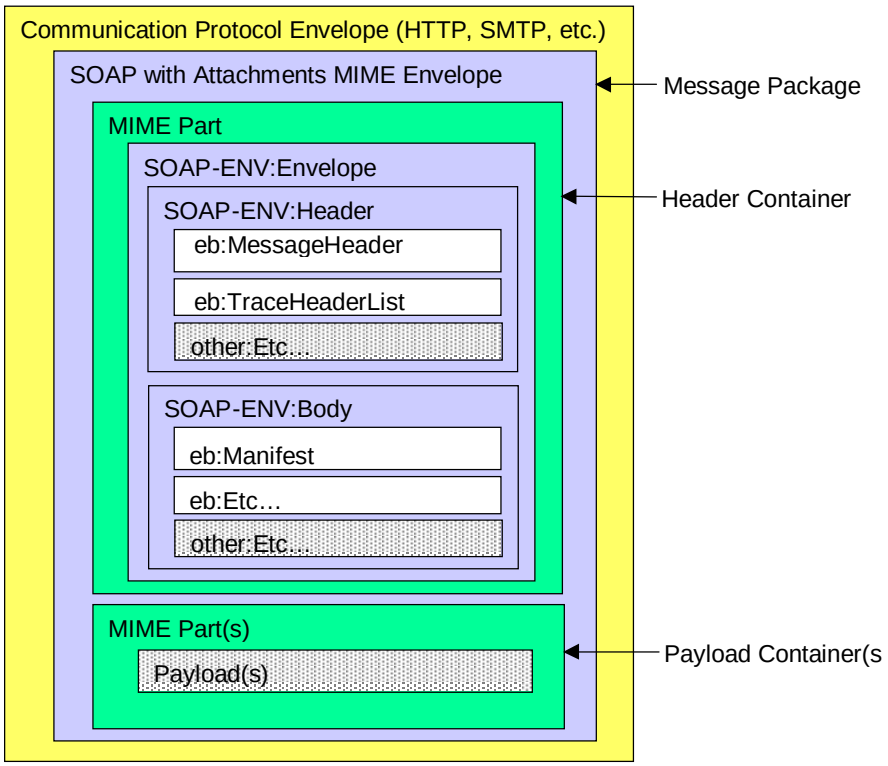
- a communication, “protocol independent” Multipart/MIME message envelope, structured in compliance with the Simple Object Access Protocol (SOAP) Messages with Attachments [SOAPATTACH] specification, referred to as the *Message Package*.
- there are two logical MIME parts within the *Message Package*: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

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

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

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

Figure 7-1 ebXML Message Structure



7.1.1 SOAP Structural Conformance

ebXML Message packaging SHALL comply with the following specifications:

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

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

7.2 Message Package

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

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

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

7.3 Header Container

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

7.3.1 Content-Type

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

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

7.3.1.1 charset Attribute

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

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

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

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

7.3.2 Header Container Example

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

```
Content-ID: messagepackage-123@example.com     --| Header Container
Content-Type: text/xml;                          |
              charset="UTF-8"                    |

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

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

7.4 Payload Container

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

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

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

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

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

7.4.1 Example of a Payload Container

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

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

7.5 Additional MIME Parameters

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

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

7.6 Reporting MIME Errors

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

8 ebXML SOAP Extensions

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

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

8.1 XML Prolog

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

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

8.1.1 XML Declaration

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

8.1.2 Encoding Declaration

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

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

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

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

For example:

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

8.2 ebXML SOAP Envelope Extensions

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

8.2.1 Namespace pseudo attribute

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

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

8.2.2 ebXML SOAP Extensions

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

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

8.2.3 #wildcard element content

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

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

8.3 SOAP Header element

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

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

```
</SOAP-ENV:Envelope>
```

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

8.4 MessageHeader element

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

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

- ***From***
- ***To***
- ***CPAId***
- ***ConversationId***
- ***Service***
- ***Action***
- ***MessageData***
- ***QualityOfServiceInfo***
- ***SequenceNumber***
- ***Description***

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

- SOAP ***mustUnderstand***
- ***version***

8.4.1 version attribute

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

8.4.2 SOAP mustUnderstand attribute

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

8.5 MessageHeader element description

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

8.5.1 From and To elements

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

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

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

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

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

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

```
<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>
```

8.5.2 CPAId element

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

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

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

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

8.5.3 ConversationId element

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

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

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

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

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

8.5.4 Service element

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

- a standards organization, or
- an individual or enterprise

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

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

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

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

8.5.4.1 type attribute

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

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

8.5.5 Action element

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

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

8.5.6 MessageData element

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

- ***MessageId***
- ***Timestamp***
- ***RefToMessageId***
- ***TimeToLive***

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

```
eb:MessageData>
                <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>

<eb:RefToMessageId>example.com.20001209-133003-28571</eb:RefToMessageId>
                <eb:Timestamp>20010215111212Z</Timestamp>
</eb:MessageData>
```

8.5.6.1 MessageId element

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

8.5.6.2 Timestamp element

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

8.5.6.3 RefToMessageId element

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

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

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

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

8.5.6.4 TimeToLive element

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

8.5.7 QualityOfServiceInfo element

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

- ***deliverySemantics***
- ***messageOrderSemantics***
- ***deliveryReceiptRequested***

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

8.5.7.1 messageOrderSemantics attribute

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

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

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

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

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

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

```
<eb:QualityOfServiceInfo deliverySemantics=”OnceAndOnlyOnce”
messageOrderSemantics=”Guaranteed”/>
```

8.5.7.2 deliveryReceiptRequested attribute

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

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

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

Valid values for ***deliveryReceiptRequested*** are:

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

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

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

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

An example of ***deliveryReceiptRequested*** follows:

```
<eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
                eb:messageOrderSemantics="Guaranteed"
```

8.5.8 eb:deliveryReceiptRequested="UnSigned"/> SequenceNumber element

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

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

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

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

1) First message from the within the Conversation

2) First message after resetting Sequence Number information by the From Party MSH

3) First message after wraparound (next value after 99999999)

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

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

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

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

8.5.9 Description element

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

8.5.10 MessageHeader sample

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

```
<eb:MessageHeader id="…" eb:version="1.0">
  <eb:From>uri:example.com</eb:From>
  <eb:To type="someType">QRS543</eb:To>
  <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
  <eb:ConversationId>987654321</eb:ConversationId>
  <eb:Service type="myservicetypes">QuoteToCollect</eb:Service>
  <eb:Action>NewPurchaseOrder</eb:Action>
  <eb:MessageData>
    <eb:MessageId>mid:UUID-2</eb:MessageId>
    <eb:Timestamp>20000725T121905.000Z</eb:Timestamp>
    <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
  </eb:MessageData>
  <eb:QualityOfServiceInfo
    deliverySemantics="OnceAndOnlyOnce"
    deliveryReceiptRequested="Signed"/>
</eb:MessageHeader>
```

8.6 TraceHeaderList element

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

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

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

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

- SOAP ***mustUnderstand***
- ***version***

8.6.1 SOAP mustUnderstand attribute

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

8.6.2 version attribute

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

8.6.3 TraceHeader Element

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

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

- ***SenderURI***
- ***ReceiverURI***
- ***Timestamp***
- ***#wildcard***

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

- ***reliableMessagingMethod***
- ***ackRequested***

8.6.3.1 SenderURI element

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

- responds to an earlier message
- acknowledges an earlier message
- reports an error in an earlier message.

8.6.3.2 ReceiverURI element

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

8.6.3.3 Timestamp element

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

8.6.3.4 #wildcard element

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

8.6.3.5 reliableMessagingMethod attribute

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

- ***ebXML***
- ***Transport***

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

8.6.3.6 ackRequested attribute

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

- ***Signed***
- ***UnSigned***
- ***None***

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

8.6.4 Single Hop TraceHeader Sample

A single hop message is illustrated by the diagram below.

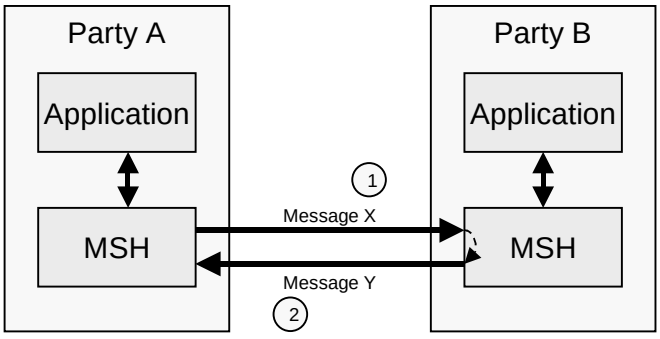


Figure 8-1 Single Hop Message

The content of the corresponding messages could include:

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

```
<eb:MessageHeader id=”...” eb:version="1.0">
  <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
  <eb:To>urn:myscheme.com:id:PartyB-id</eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
```

```
  </eb:MessageData>
   ...
</eb:MessageHeader>

<eb:TraceHeaderList id=”...” eb:version="1.0">
  <eb:TraceHeader>
    <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
    <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
    <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

8.6.5 Multi-hop TraceHeader Sample

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

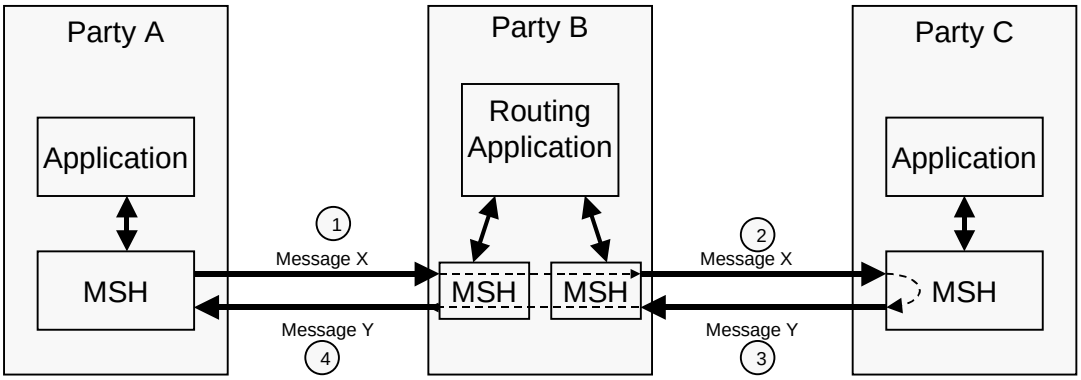


Figure 8-2 Multi-hop Message

The content of the corresponding messages could include:

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

```
<eb:MessageHeader id=”...” eb:version="1.0">
  <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
  <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
   ...
</eb:MessageHeader>

<eb:TraceHeaderList id=”...” eb:version="1.0">
  <eb:TraceHeader>
    <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
    <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
    <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

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

```
<eb:MessageHeader id=”...” eb:version="1.0">
  <eb:From>urn:myscheme.com:id:PartyA-id</eb:From>
  <eb:To>urn:myscheme.com:id:PartyC-id</eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
   ...
</eb:MessageHeader>
```

```
<eb:TraceHeaderList id=”...” eb:version="1.0">
  <eb:TraceHeader>
    <eb:SenderURI>url:PartyA.com/PartyAMsh</eb:SenderURI>
    <eb:ReceiverURI>url:PartyB.com/PartyBMsh</eb:ReceiverURI>
    <eb:Timestamp>20001216T21:19:35.145Z-8</eb:Timestamp>
  </eb:TraceHeader>
  <eb:TraceHeader>
    <eb:SenderURI>url:PartyB.com/PartyAMsh</eb:SenderURI>
    <eb:ReceiverURI>url:PartyC.com/PartyBMsh</eb:ReceiverURI>
    <eb:Timestamp>20001216T21:19:45.483Z-6</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

8.7 Via element

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

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

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

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

- ***syncReply*** attribute
- ***reliableMessagingMethod*** attribute
- ***ackRequested*** attribute
- ***CPAId*** element

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

- ***Service*** element
- ***Action*** element

8.7.1 SOAP mustUnderstand attribute

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

8.7.2 SOAP actor attribute

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

8.7.3 version attribute

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

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

8.7.4 syncReply attribute

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

8.7.5 reliableMessagingMethod attribute

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

- **ebXML**
- **Transport**

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

8.7.6 ackRequested attribute

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

- Signed
- UnSigned
- None

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

8.7.7 CPAId element

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

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

8.7.8 Service and Action elements

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

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

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

8.7.9 Sample Via element

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

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

8.8 ErrorList element

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

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

- ***id*** attribute
- SOAP ***mustUnderstand*** attribute
- ***version attribute***
- ***highestSeverity*** attribute

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

8.8.1 id attribute

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

8.8.2 SOAP mustUnderstand attribute

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

8.8.3 version attribute

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

8.8.4 highestSeverity attribute

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

8.8.5 Error element

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

- ***codeContext***
- ***errorCode***
- ***severity***
- ***location***
- ***xml:lang***

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

8.8.5.1 codeContext attribute

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

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

8.8.5.2 errorCode attribute

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

8.8.5.3 severity attribute

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

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

8.8.5.4 location attribute

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

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

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

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

8.8.5.5 Error element Content

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

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

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

8.8.6 Examples

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

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

8.8.7 errorCode values

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

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

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

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

8.8.8 Reporting Errors in the ebXML Elements

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

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

8.8.9 Non-XML Document Errors

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

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

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

8.9 Signature element

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

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

8.10 SOAP Body Extensions

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

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

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

- ***Manifest*** element
- ***StatusData*** element
- ***Acknowledgement*** element

Each is defined in the following sections.

8.11 Manifest element

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

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

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

- a REQUIRED ***id*** attribute
- a REQUIRED SOAP ***mustUnderstand*** attribute
- a REQUIRED ***version*** attribute
- one or more ***Reference*** elements
- ***#wildcard***

8.11.1 id attribute

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

8.11.2 SOAP mustUnderstand attribute

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

8.11.3 version attribute

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

8.11.4 Reference element

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

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

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

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

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

8.11.4.1 Schema element

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

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

8.11.4.2 Description element

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

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

8.11.4.3 #wildcard element

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

8.11.5 What References are Included in a Manifest

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

8.11.6 Manifest Validation

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

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

8.11.7 Manifest sample

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

```
<eb:Manifest id="Manifest" eb:version="1.0">
  <eb:Reference id="pay01"
    xlink:href="cid:payload-1"
    xlink:role="http://regrep.org/gci/purchaseOrder">
    <eb:Description>Purchase Order for 100,000 widgets</eb:Description>
    <eb:Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
      version="1.0"/>
  </eb:Reference>
</eb:Manifest>
```

8.12 StatusData Element

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

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

- a REQUIRED ***RefToMessageId*** element
- a ***Timestamp*** element
- a REQUIRED SOAP ***mustUnderstand*** attribute
- a REQUIRED ***version*** attribute
- a ***messageStatus*** attribute

8.12.1 RefToMessageId element

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

8.12.2 Timestamp element

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

8.12.3 SOAP mustUnderstand attribute

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

8.12.4 version attribute

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

8.12.5 messageStatus attribute

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

- ***UnAuthorized*** – the Message Status Request is not authorized or accepted
- ***NotRecognized*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element is not recognized
- ***Received*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element has been received by the MSH, but has not been processed by an application or forwarded to another MSH

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

8.13 Acknowledgment Element

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

-

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

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

- a ***Timestamp*** element
- a ***From*** element
- a REQUIRED SOAP ***mustUnderstand*** attribute
- a REQUIRED ***version*** attribute
- a ***type*** attribute
- a ***signed*** attribute

8.13.1 Timestamp element

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

8.13.2 From element

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

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

8.13.3 SOAP mustUnderstand attribute

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

8.13.4 version attribute

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

8.13.5 type attribute

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

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

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

8.13.6 signed attribute

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

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

The default value for ***signed*** is f***alse***.

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

8.14 Combining ebXML SOAP Extension Elements

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

8.14.1 Manifest element

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

8.14.2 MessageHeader element

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

8.14.3 TraceHeaderList element

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

8.14.4 StatusData element

This element MUST NOT be present with the following elements:

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

8.14.5 ErrorList element

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

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

- a ***Manifest*** element
- a ***StatusData*** element

8.14.6 Acknowledgment element

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

8.14.7 Signature element

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

9 Message Service Handler Services

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

- Message Status Request
- Message Service Handler Ping

Each service is described below:

9.1 Message Status Request Service

The Message Status Request Service consists of the following:

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

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

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

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

9.1.1 Message Status Request Message

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

- A ***Header*** element
- A ***TraceHeaderList*** element
- A ***Signature*** element

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

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

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

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

9.1.2 Message Status Response Message

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

- a ***Header*** element

- a ***TraceHeaderList*** element
- an ***Acknowledgment*** element
- a ***StatusData*** element (see section 8.12)
- a ***Signature*** element

The ***TraceHeaderList***, ***Acknowledgment*** and ***Signature*** elements MAY be omitted (see sections 8.6, 8.14.6 and8.14.7).

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

- a ***From*** element that identifies the sender of the Message Status Response message
- a ***To*** element that is set to the value of the ***From*** element in the Message Status Request message
- a ***Service*** element that contains the value: ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- an ***Action*** element that contains ***Response***
- a ***RefToMessageId*** that identifies the Message Status Request message.

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

9.1.3 Security Considerations

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

9.2 Message Service Handler Ping Service

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

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

9.2.1 Message Service Handler Ping Message

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

- A ***Header*** element
- A ***TraceHeaderList*** element
- A ***Signature*** element

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

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

- a ***From*** element that identifies the party creating the MSH Ping message
- a ***To*** element that identifies the party that is being sent the MSH Ping message
- a ***Service*** element that contains: ***http://www.ebxml.org/namespaces/messageService/MSHStatus***
- an ***Action*** element that contains ***Ping***

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

9.2.2 Message Service Handler Pong Message

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

- a ***Header*** element
- a ***TraceHeaderList*** element
- an ***Acknowledgment*** element
- a ***Signature*** element

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

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

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

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

9.2.3 Security Considerations

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

10 Reliable Messaging

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

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

10.1.1 Persistent Storage and System Failure

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

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

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

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

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

10.1.2 Methods of Implementing Reliable Messaging

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

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

10.2 Reliable Messaging Parameters

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

10.2.1 Delivery Semantics

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

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

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

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

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

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

10.2.2 Sync Reply

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

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

The default value is ***false***.

10.2.3 Time To Live

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

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

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

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

10.2.4 reliableMessagingMethod

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

- ***ebXML***
- ***Transport***

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

10.2.5 AckRequested

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

Valid values for ***AckRequested*** are:

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

The default value is ***None***.

10.2.6 Timeout Parameter

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

10.2.7 Retries

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

10.2.8 RetryInterval

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

10.2.9 PersistDuration

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

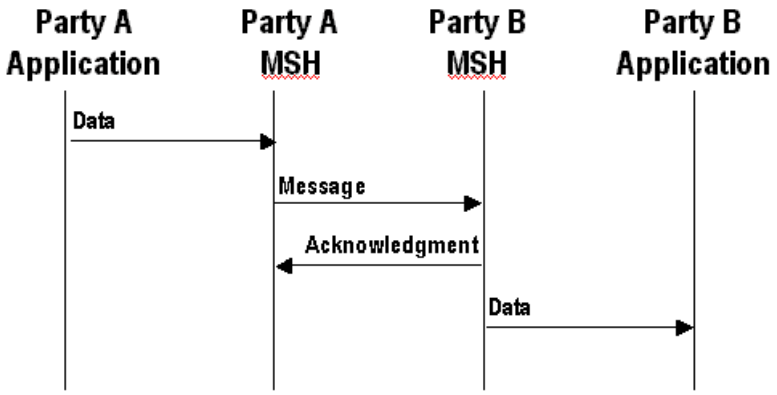
If the ***persistDuration*** has passed since the message was first sent, a Sending MSH SHOULD NOT resend a message with the same ***MessageId.*** .

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

10.3 ebXML Reliable Messaging Protocol

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

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:

i) Save the ***MessageId*** of the received message in *persistent storage*. As an implementation decision, the whole message MAY be stored if there are other reasons for doing so.

ii) If the received message contains a ***RefToMessageId*** element then do the following:

(1) Look for a message in *persistent storage* that has a ***MessageId*** that is the same as the value of ***RefToMessageId*** on the received Message

(2) If a message is found in *persistent storage* then mark the persisted message as delivered

iii) Generate an *Acknowledgement Message* in response (see section 10.3.1.3).

d) If the message is a duplicate, then do the following:

i) Look in persistent storage for the first response to the received message and resend it (i.e. it contains a ***RefToMessageId*** that matches the ***MessageId*** of the received message)

ii) If a message was found in *persistent storage* then resend the persisted message back to the MSH that sent the received message,

iii) If no message was found in *persistent storage,* then:

(1) if **syncReply** is set to ***True*** and the CPA indicates that an application response is included, ignore the received message (i.e. no message was generated in response to the message, or the processing of the earlier message is not yet complete)

(2) if ***syncReply*** is set to ***False*** then generate an *Acknowledgement Message* (see section 10.3.1.3).

10.3.1.3 Generating an Acknowledgement Message

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

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

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

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

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

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

1) The ***Service*** element MUST be set to: http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment

2) The ***Action*** element MUST be set to ***Acknowledgment***.

3) The ***From*** element MUST be set to the ***ReceiverURI*** from the last ***TraceHeader*** in the *message* that has just been received

4) The ***To*** element MUST be set to the ***SenderURI*** from the last ***TraceHeader*** in the *message* that has just been received

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

10.3.1.4 Resending Lost Messages and Duplicate Filtering

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

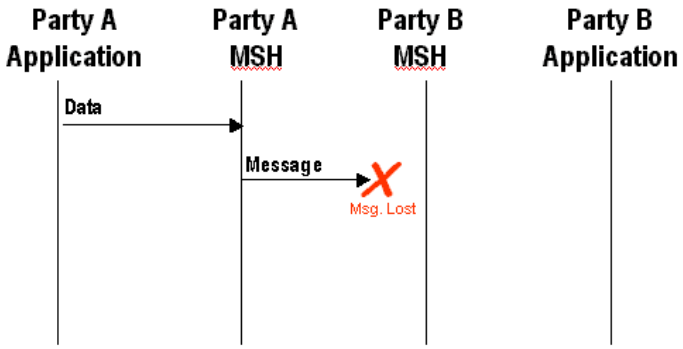


Figure 10-2 Undelivered Message

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

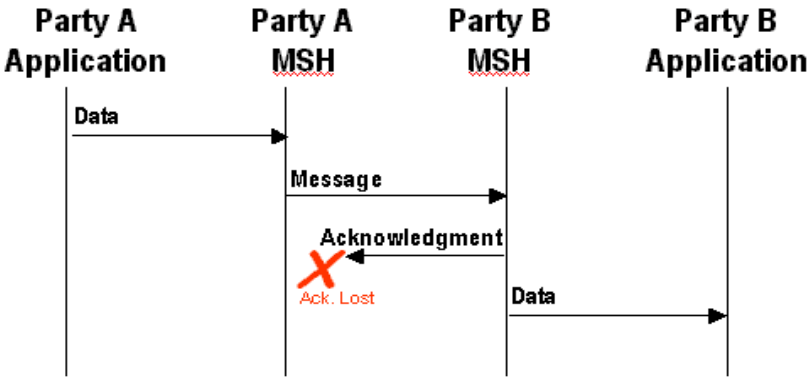


Figure 10-3 Lost Acknowledgment Message

The rules that apply are as follows:

1) The Sending MSH MUST resend the original message if an *Acknowledgment Message* has not been received from the Receiving MSH and either of the following are true:

   a) The message has not yet been resent and at least the time specified in the ***timeout*** parameter has passed since the first message was sent, or

   b) The message has been resent, and the following are both true:

      i) At least the time specified in the ***retryInterval*** has passed since the last time the message was resent, and

ii) The message has been resent less than the number of times specified in the ***retries*** Parameter

2) If the Sending MSH does not receive an *Acknowledgment Message* after the maximum number of retries, the Sending MSH SHOULD notify the application and/or system administrator function of the failure to receive an acknowledgement.

3) If the Sending MSH detects a communications protocol error that is unrecoverable at the transport protocol level, the Sending MSH SHOULD resend the message using the rules as defined in the CPA.

10.3.1.5 Duplicate Message Handling

In this context:

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

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

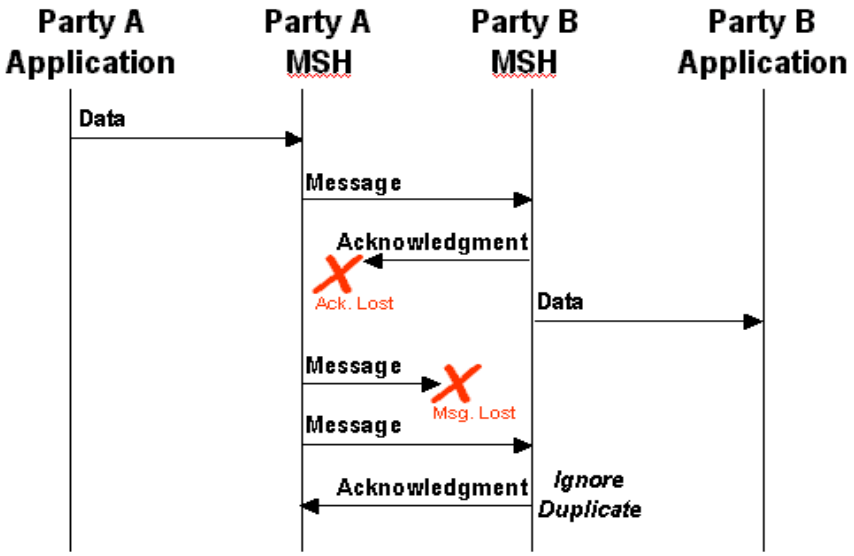


Figure 10-4 Resending Unacknowledged Messages

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

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

2) When the recipient (Party B) of the *message* receives a *duplicate message*, it MUST resend to the sender (Party A) a message identical to the *first message* that was sent to the sender Party A)

3) The recipient of the *message* (Party B) MUST NOT forward the message a second time to the application/process.

10.4 Failed Message Delivery

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

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

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

10.5 MSH Time Accuracy

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

11 Error Reporting and Handling

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

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

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

11.1 Definitions

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

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

11.2 Types of Errors

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

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

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

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

11.3 When to generate Error Messages

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

- the Error Reporting Location (see section11.4) to which the *message reporting the error* should be sent can be determined, and
- the *message in error* does not have an ***ErrorList*** element with ***highestSeverity*** set to ***Error***.

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

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

- no further action is taken.

11.3.1 Security Considerations

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

11.4 Identifying the Error Reporting Location

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

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

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

11.5 Service and Action Element Values

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

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

- *The **Service** element MUST be set to:*
  http://www.ebxml.org/namespaces/messageService/MessageStatus
- The ***Action*** element MUST be set to ***MessageError***.

12 Security

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

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

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

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

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

12.1 Security and Management

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

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

12.2 Collaboration Protocol Agreement

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

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

12.3 Countermeasure Technologies

12.3.1 Persistent Digital Signature

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

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

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

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

12.3.1.1 Signature Generation

1) Create a ***SignedInfo*** element with SignatureMethod, CanonicalizationMethod, and Reference(s) elements for the ebXML Header document and any required payload objects, as prescribed by [XMLDSIG].
2) Canonicalize and then calculate the SignatureValue over ***SignedInfo*** based on algorithms specified in SignedInfo as specified in [XMLDSIG].
3) Construct the Signature element that includes the ***SignedInfo***, ***KeyInfo*** (RECOMMENDED), and ***SignatureValue*** elements as specified in [XMLDSIG].
4) Include the namespace qualified ***Signatur***e element in the ebXML Header document just signed, following the ***TraceHeaderList*** element.

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

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

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

http://www.w3.org/2000/02/xmldsig#sha1

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

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

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

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

Example of digitally signed ebXML SOAP message:

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:eb="http://www.ebxml.org/namespaces/messageHeader"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <SOAP-ENV:Header>
    <eb:MessageHeader id="…" eb:version="1.0">
    ...
    </eb:MessageHeader>
    <eb:TraceHeaderList id="…" eb:version="1.0">
      <eb:TraceHeader>
        ...
      </eb:TraceHeader>
    </eb:TraceHeaderList>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlds#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/WD-xml-c14n-20001011"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmlds#dsa-sha1"/>
        <ds:Reference URI="">
          <ds:Transforms>
            <ds:Transform>
              <XPath>/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1'])
and not(ancestor-or-self::TraceHeaderList)]</XPath>
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="cid://blahblahblah/">
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmlds#sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo>...</ds:KeyInfo>
    </ds:Signature>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <eb:Manifest id="Mani01" eb:version="1.0">
      <eb:Reference xlink:href="cid://blahblahblah"
        xlink:role="http://ebxml.org/gci/invoice">
        <eb:Schema eb:version="1.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
      </eb:Reference>
    </eb:Manifest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

12.3.2 Persistent Signed Receipt

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

12.3.3 Non-persistent Authentication

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

12.3.4 Non-persistent Integrity

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

12.3.5 Persistent Confidentiality

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

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

12.3.6 Non-persistent Confidentiality

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

12.3.7 Persistent Authorization

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

12.3.8 Non-persistent Authorization

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

12.3.9 Trusted Timestamp

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

13 References

13.1 Normative References

[HTTP] IETF RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, January 1997

[RFC 2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E. Levinson, Published August 1998

[RFC 2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, Published August 1998

[RFC2045] IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein, Published November 1996

[SMTP] IETF RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982

[SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08 May 2000, http://www.w3.org/TR/SOAP

[SOAPATTACH] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000 http://www.w3.org/TR/SOAP-attachments

[UTF-8] UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage conventions.

[XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition), October 2000, http://www.w3.org/TR/2000/REC-xml-20001006

[XML Namespace] W3C Recommendation for Namespaces in XML, World Wide Web Consortium, 14 January 1999, http://www.w3.org/TR/REC-xml-names

[XLINK] W3C XML Linking Candidate Recommendation, http://www.w3.org/TR/xlink/

[XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification, http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/

[XMLMedia] IETF RFC 3023, XML Media Types. M. Murata, S. St.Laurent, January 2001

13.2 Non-Normative References

[Glossary] ebXML Glossary, see ebXML Project Team Home Page

[PGP/MIME] IETF RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins. October 1996.

[S/MIME] IETF RFC2311, “S/MIME Version 2 Message Specification”, S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.

[S/MIMECH] IETF RFC 2312, “S/MIME Version 2 Certificate Handling”, S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. March 1998.

[S/MIMEV3] IETF RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed.. June 1999.

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

14 Disclaimer

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

15 Contact Information

Team Leader

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

Vice Team Leader

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

Team Editor

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

Authors

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

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

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

Name	John Ibbotson
Company	IBM UK Ltd

Street		Hursley Park
City, State, Postal Code		Winchester SO21 2JN
Country		United Kingdom
Phone:		+44 (1962) 815188
Email:		john_ibbotson@uk.ibm.com
Name		Masayoshi Shimamura
Company		Fujitsu Limited
Street		Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
City, State, Postal Code		Kohoku-ku, Yokohama 222-0033, Japan
Phone:		+81-45-476-4590
E-mail:		shima@rp.open.cs.fujitsu.co.jp

Document Editing Team

Name		Ralph Berwanger
Company		bTrade.com
Street		2324 Gateway Drive
City, State, Postal Code		Irving, TX 75063
Country		USA
Phone:		+1 (972) 580-3970
EMail:		rberwanger@btrade.com
Name		Colleen Evans
Company		Progress/Sonic Software
Street		14 Oak Park
City,State,Postal Code		Bedford, MA 01730
Country		USA
Phone		+1 (720) 480-3919
Email		cevans@progress.com
Name		Ian Jones
Company		British Telecommunications
Street		Enterprise House, 84-85 Adam Street
City, State, Postal Code		Cardiff, CF24 2XF
Country		United Kingdom
Phone:		+44 29 2072 4063
EMail:		ian.c.jones@bt.com
Name		Martha Warfelt
Company		DaimlerChrysler Corporation
Street		800 Chrysler Drive
City, State, Postal Code		Auburn Hills, MI
Country		USA
Phone:		+1 (248) 944-5481
EMail:		maw2@daimlerchrysler.com
Name		David Fischer
Company		Drummond Group, Inc
Street		5008 Bentwood Ct
City, State, Postal Code		Fort Worth, TX 76132
Phone		+1 (817-294-7339
EMail		david@drummondgroup.com

Appendix A ebXMLHeader Schema

A.1

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" version="0.98"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  <xsd:import namespace="http://www.w3.org/1999/xlink"
schemaLocation="http://www.w3.org/1999/xlink"/>
  <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>
  <!-- MANIFEST -->
  <xsd:element name="Manifest">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Reference" maxOccurs="unbounded"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" use="required" type="xsd:ID"/>
      <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
      <xsd:attribute ref="soap:mustUnderstand" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Reference">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Schema" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" use="required" type="xsd:ID"/>
      <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
      <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
      <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Schema">
    <xsd:complexType>
        <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
        <xsd:attribute name="version" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <!-- HEADER -->
  <xsd:element name="MessageHeader">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="From"/>
        <xsd:element ref="To"/>
        <xsd:element ref="CPAId"/>
        <xsd:element ref="ConversationId"/>
        <xsd:element ref="Service"/>
        <xsd:element ref="Action"/>
        <xsd:element ref="MessageData"/>
        <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
      <xsd:attribute ref="soap:mustUnderstand" use="required"/>
```

```
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="CPAId" type="xsd:string"/>
  <xsd:element name="ConversationId" type="xsd:string"/>
  <xsd:element name="Service" type="xsd:string"/>
  <xsd:element name="Action" type="xsd:string"/>
  <xsd:element name="MessageData">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="MessageId"/>
        <xsd:element ref="Timestamp"/>
        <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="TimeToLive" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="MessageId" type="xsd:string"/>
  <xsd:element name="TimeToLive" type="xsd:timeInstant"/>
  <xsd:element name="QualityOfServiceInfo">
    <xsd:complexType>
        <xsd:attribute name="deliverySemantics" use="default" value="BestEffort">
          <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
              <xsd:enumeration value="OnceAndOnlyOnce"/>
              <xsd:enumeration value="BestEffort"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed">
          <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
              <xsd:enumeration value="Guaranteed"/>
              <xsd:enumeration value="NotGuaranteed"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
          <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
              <xsd:enumeration value="Signed"/>
              <xsd:enumeration value="UnSigned"/>
              <xsd:enumeration value="None"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:attribute>
    </xsd:complexType>
  </xsd:element>
  <!-- TRACE HEADER LIST -->
  <xsd:element name="TraceHeaderList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="TraceHeader" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
      <xsd:attribute ref="soap:mustUnderstand" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="TraceHeader">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="SenderURI"/>
        <xsd:element ref="ReceiverURI"/>
        <xsd:element ref="Timestamp"/>
        <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="reliableMessagingMethod">
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="ebXML"/>
            <xsd:enumeration value="Transport"/>
```

```
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="ackRequested">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="Signed"/>
          <xsd:enumeration value="UnSigned"/>
          <xsd:enumeration value="None"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<xsd:element name="SenderURI" type="xsd:uriReference"/>
<xsd:element name="ReceiverURI" type="xsd:uriReference"/>
<xsd:element name="SequenceNumber" type="xsd:positiveInteger"/>
<!-- ACKNOWLEDGEMENT -->
<xsd:element name="Acknowledgment">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Timestamp"/>
      <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
    <xsd:attribute ref="soap:mustUnderstand" use="required"/>
    <xsd:attribute name="type" use="default" value="DeliveryReceipt">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="DeliveryReceipt"/>
          <xsd:enumeration value="IntermediateAck"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="signed" type="xsd:boolean"/>
  </xsd:complexType>
</xsd:element>
<!-- ERROR LIST -->
<xsd:element name="ErrorList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Error" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
    <xsd:attribute ref="soap:mustUnderstand" use="required"/>
    <xsd:attribute name="highestSeverity" use="default" value="Warning">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="Warning"/>
          <xsd:enumeration value="Error"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Error">
  <xsd:complexType>
    <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
    <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
    <xsd:attribute name="severity" use="default" value="Warning">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="Warning"/>
          <xsd:enumeration value="Error"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="location" type="xsd:string"/>
    <xsd:attribute name="xml:lang" type="xsd:language"/>
```

```
    <xsd:attribute name="errorMessage" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
<!-- STATUS DATA -->
<xsd:element name="StatusData">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="RefToMessageId"/>
      <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="version" use="fixed" type="xsd:string" value="1.0"/>
    <xsd:attribute ref="soap:mustUnderstand" use="required"/>
    <xsd:attribute name="messageStatus">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="UnAuthorized"/>
          <xsd:enumeration value="NotRecognized"/>
          <xsd:enumeration value="Received"/>
          <xsd:enumeration value="Processed"/>
          <xsd:enumeration value="Forwarded"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
<!-- COMMON ELEMENTS -->
<xsd:element name="PartyId">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="type" type="xsd:string"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="To">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PartyId"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="From">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="PartyId"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Description">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="RefToMessageId" type="xsd:string"/>
<xsd:element name="Timestamp" type="xsd:timeInstant"/>
<!-- VIA  -->
<xsd:element name="Via">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="CPAId" minOccurs="0"/>
      <xsd:element ref="Service" minOccurs="0"/>
      <xsd:element ref="Action" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="version" use="required" type="xsd:string"/>
    <xsd:attribute ref="soap:mustUnderstand"/>
    <xsd:attribute ref="soap:actor"/>
```

```
      <xsd:attribute name="syncReply" type="xsd:boolean"/>
      <xsd:attribute name="deliveryReceiptRequested" use="default" value="None">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Signed"/>
            <xsd:enumeration value="Unsigned"/>
            <xsd:enumeration value="None"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="reliableMessagingMethod">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="ebXML"/>
            <xsd:enumeration value="Transport"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:attribute>
      <xsd:attribute name="ackRequested" type="xsd:boolean"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Appendix B Communication Protocol Bindings

B.1 Introduction

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

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

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

B.2 HTTP

B.2.1 Minimum level of HTTP protocol

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

B.2.2 Sending ebXML Service messages over HTTP

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

POST /ebxmlhandler HTTP/1.1

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

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

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

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

SOAPAction: **ebXML**

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

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

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

--BoundarY
Content-ID: <ebxhmheader111@example.com>
Content-Type: text/xml
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
<SOAP-ENV:Header>
  <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:From>
      <eb:PartyId>urn:duns:123456789</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>urn:duns:912345678</eb:PartyId>
    </eb:To>
    <eb:CPAId>20001209-133003-28572</eb:CPAId>
    <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
    <eb:Service>OrderProcessing</eb:Service>
    <eb:Action>NewORder</eb:Action>
    <eb:MessageData>
      <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
      <eb:Timestamp>20010215111212Z</Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
  </eb:MessageHeader>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
         xlink:role="XLinkRole"
         xlink:type="simple">
        <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--BoundarY
Content-ID: <ebxmlpayload111@example.com>
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
```

```
<purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price currency="USD">500.00</price>
</purchase_order>
--BoundarY--
```

B.2.3 HTTP Response Codes

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

B.2.4 SOAP Error conditions and Synchronous Exchanges

The SOAP 1.1 specification states:

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

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

B.2.5 Synchronous vs. Asynchronous

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

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

B.2.6 Access Control

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

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

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

B.2.7 Confidentiality and Communication Protocol Level Security

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

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

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

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

B.3 SMTP

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

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

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

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

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

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

B.3.1 Minimum level of supported protocols

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

B.3.2 Sending ebXML Messages over SMTP

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

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

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

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

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

  SOAPAction: **ebXML**

  Where Service and Action are values of the corresponding elements from the ebXML MessageHeader.
- The "MIME-Version: 1.0" header must appear as an email MIME header.
- The e-mail header "To:" MUST contain the [RFC822] compliant e-mail address of the ebXML Message Service Handler.
- The e-mail header "From:" MUST contain the [RFC822] compliant e-mail address of the senders ebXML message service handler.
- Construct a "Date:" e-mail header in accordance with [RFC822]

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

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

```
From: ebXMLhandler@example.com
To: ebXMLhandler@example2.com
Date: Thu, 08 Feb 2001 19:32:11 CST
MIME-Version: 1.0
SOAPAction:
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
        start=" <ebxhmheader111@example.com>"

--BoundarY
Content-ID: <ebxhmheader111@example.com>
Content-Type: text/xml

<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
  xmlns:eb='http://www.ebxml.org/namespaces/messageHeader'>
<SOAP-ENV:Header>
  <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:From>
      <eb:PartyId>urn:duns:123456789</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>urn:duns:912345678</eb:PartyId>
    </eb:To>
    <eb:CPAId>20001209-133003-28572</eb:CPAId>
    <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
    <eb:Service>OrderProcessing</eb:Service>
    <eb:Action>NewORder</eb:Action>
    <eb:MessageData>
      <eb:MessageId>example.com.20001209-133003-28572</eb:MessageId>
      <eb:Timestamp>20010215111212Z</Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo deliverySemantics="BestEffort"/>
  </eb:MessageHeader>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
    <eb:Reference xlink:href="cid:ebxmlpayload111@example.com"
         xlink:role="XLinkRole"
         xlink:type="simple">
        <eb:Description xml:lang="en-us">Purchase Order 1</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--BoundarY
Content-ID: <ebxhmheader111@example.com>
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price currency="USD">500.00</price>
</purchase_order>
--BoundarY--
```

B.3.3 Response Messages

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

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

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

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

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

B.3.4 Access Control

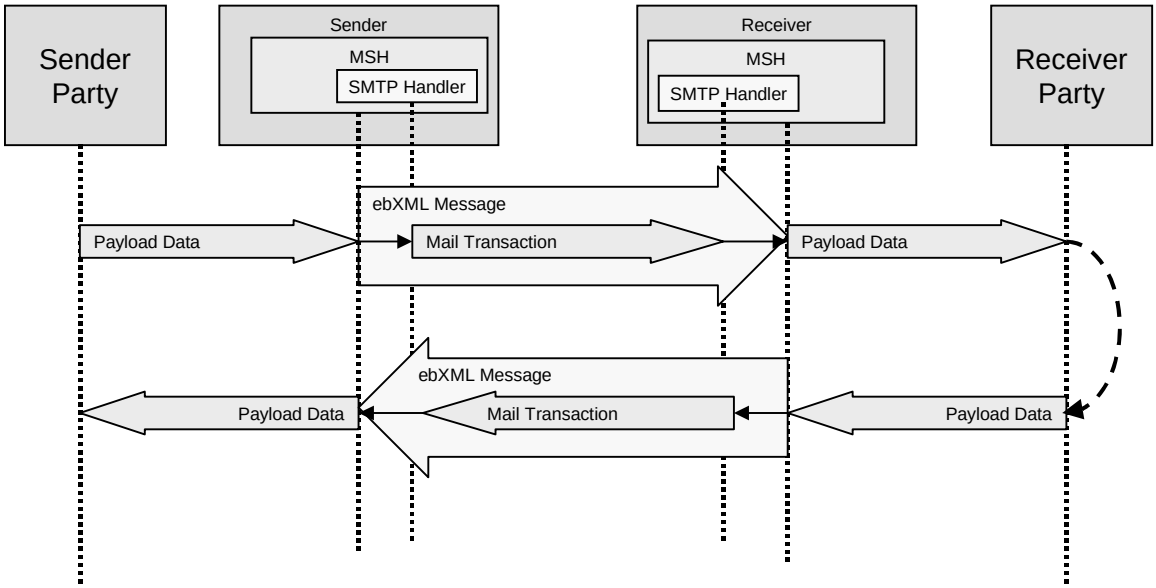
Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the use of an access control mechanism. The SMTP access authentication process described in "SMTP Service Extension for Authentication" [RFC2554] defines the ebXML recommended access control mechanism to protect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

B.3.5 Confidentiality and Communication Protocol Level Security

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

B.3.6 SMTP Model

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



B.4 Communication Errors during Reliable Messaging

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

Copyright Statement