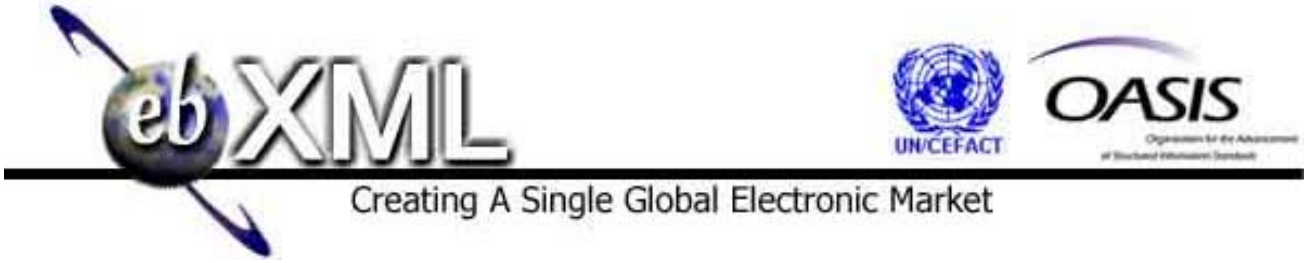


Message Service Specification

v1.0

Transport, Routing & Packaging Team

11 May 2001

(This document is the non-normative version formatted for printing, July 2001)

Table of Contents

1 Status of this Document

This document specifies an ebXML Technical Specification for the eBusiness community.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version

http://www.ebxml.org/specs/ebMS.pdf

Latest version

http://www.ebxml.org/specs/ebMS.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
David Craft	VerticalNet
Philippe De Smedt	Viquity
Lawrence Ding	WorldSpan
Rik Drummond	Drummond Group
Andrew Eisenberg	Progress Software
Colleen Evans	Progress / Sonic Software
David Fischer	Drummond Group
Christopher Ferris	Sun Microsystems
Robert Fox	Softshare
Brian Gibb	Sterling Commerce
Maryann Hondo	IBM
Jim Hughes	Fujitsu

John Ibbotson	IBM
Ian Jones	British Telecommunications
Ravi Kacker	Kraft Foods
Henry Lowe	OMG
Jim McCarthy	webXI
Bob Miller	GXS
Dale Moberg	Sterling Commerce
Joel Munter	Intel
Shumpei Nakagaki	NEC Corporation
Farrukh Najmi	Sun Microsystems
Akira Ochi	Fujitsu
Martin Sachs	IBM
Saikat Saha	Commerce One
Masayoshi Shimamura	Fujitsu
Prakash Sinha	Netfish Technologies
Rich Salz	Zolera Systems
Tae Joon Song	eSum Technologies, Inc.
Kathy Spector	Extricity
Nikola Stojanovic	Encoda Systems, Inc.
David Turner	Microsoft
Gordon Van Huizen	Progress Software
Martha Warfelt	DaimlerChrysler Corporation
Prasad Yendluri	Web Methods

3 Introduction

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

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

3.1 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 [HTTP] and [SMTP].

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 6)
- **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 7)
- **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

- **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 9)

- **Error Handling** – This section describes how one *ebXML Message Service* reports errors it detects to another ebXML Message Service Handler (section 10)

- **Security** – This provides a specification of the security semantics for ebXML Messages (section11).

Appendices to this specification cover the following:

- **Appendix A Schema** – This normative appendix contains [XMLSchema] for the ebXML SOAP **Header** and **Body**.

- **Appendix B Communication Protocol Envelope Mappings** – This normative appendix describes how to transport *ebXML Message Service* compliant messages over [HTTP] and [SMTP]

3.2 Document conventions

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

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

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

- *MUST: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.*

- *MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.*

- *SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.*

- *SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is*

acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

- *MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)*

3.3 Audience

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

3.4 Caveats and assumptions

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

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

3.5 Related documents

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

[ebTA] ebXML Technical Architecture Specification v1.04 – defines the overall technical architecture for ebXML

[secRISK] ebXML Technical Architecture Security Risk Assessment v1.0 – identifies the security risks associated with the ebXML technical architecture

[ebCPP] ebXML Collaboration Protocol Profile and Agreement Specification v1.0 - 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

[ebRS] ebXML Registry/Repository Services Specification v1.0 – defines a registry service for the ebXML environment

4 Design Objectives

The design objectives of this specification are to define a wire format and protocol for a Message Service 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 the exchange of 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 effort has been made to ensure that the REQUIRED functionality described in this specification has been prototyped by the ebXML Proof of Concept Team in order to ensure the clarity, accuracy and efficiency of this specification.

5 System Overview

This document defines the *ebXML Message Service* 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.

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

5.1 Message service purpose

The *ebXML Message Service* defines robust, yet basic, functionality to transfer messages between trading parties 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 [ebRS], Collaboration Protocol Profile/Agreement [ebCPP] and ebXML Technical Architecture [ebTA].

5.2 Message service overview

The *ebXML Message Service* may be conceptually broken down into following three parts: (1) an abstract *Service Interface*, (2) functions provided by the Message Service Handler (MSH), 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 Message Services* architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies.

- **Header Processing** - the creation of the SOAP **Header** elements for the *ebXML Message* uses input from the application, passed through the Message Service Interface, information from the *Collaboration Protocol Agreement* (*CPA* defined in [ebCPP]) that governs the

message, and generated information such as digital signature, timestamps and unique identifiers.

- **Header Parsing** - extracting or transforming information from a received SOAP **Header** or **Body** 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* (SOAP **Header** or **Body** elements and payload) into its SOAP Messages with Attachments [SOAPATTACH] 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.

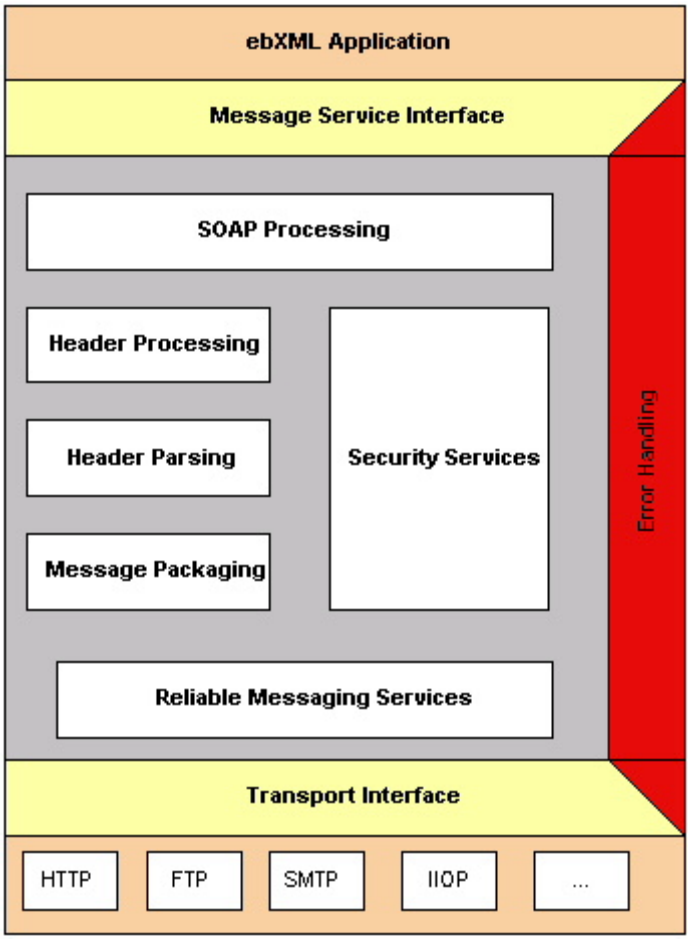


Figure 5-1 Typical Relationship between ebXML Message Service Handler Components

5.3 Use of version attribute

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

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

6 Packaging Specification

6.1 Introduction

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

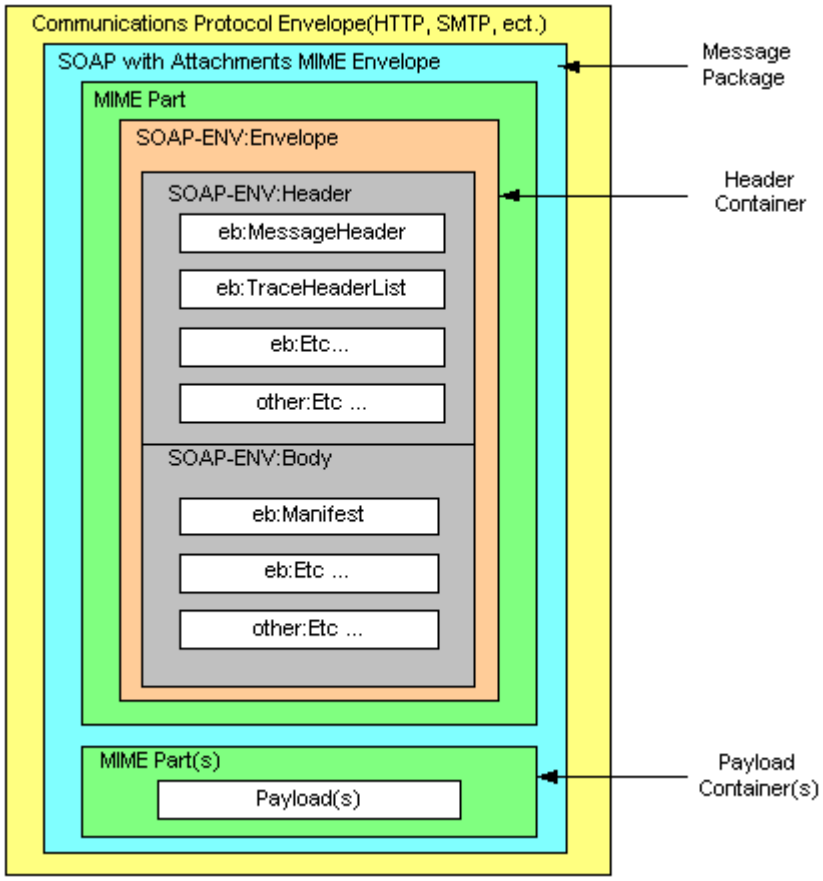


Figure 6-1 ebXML Message Structure

6.1.1 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.

6.2 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 the following fragment:

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

--boundaryValue
Content-ID: messagepackage-123@example.com
```

6.3 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.

6.3.1 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"
```

6.3.1.1 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"
```

6.3.2 Header container example

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

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

6.4 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 7.11).

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*.

6.4.1 Example of a payload container

The following fragment 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>                            -------------|                  |
```

6.5 Additional MIME parameters

Any MIME part described by this specification MAY contain additional MIME headers in conformance with the [RFC2045] specification. Implementations MAY ignore any MIME

header not defined in this specification. Implementations MUST ignore any MIME header 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.

6.6 Reporting MIME errors

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

7 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 ebXML SOAP extension elements,
- an ebXML SOAP extension element is not always present or,
- the data contained in the ebXML SOAP extension element MAY be digitally signed separately from the other ebXML SOAP extension elements.

7.1 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:

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

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

7.1.1 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.*

7.1.2 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 6.3).

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 SOAP **Header** Document.

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

7.2 ebXML SOAP Envelope extensions

In conformance with the [SOAP] specification, all extension element content MUST be namespace qualified. All of the ebXML SOAP extension element content defined in this specification MUST be namespace qualified to the ebXML SOAP **Envelope** extensions namespace as defined in section 7.2.1.

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

7.2.1 Namespace pseudo attribute

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

7.2.2 xsi:schemaLocation attribute

The SOAP namespace:

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

resolves to a schema that conforms to an early Working Draft version of the W3C XML Schema specification, specifically identified by the following URI:

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

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

In order to enable validating parsers and various schema validating tools to correctly process and parse ebXML SOAP *Messages*, it has been necessary that the ebXML TR&P team adopt an equivalent, but updated version of the SOAP schema that conforms to the W3C Candidate

Recommendation draft of the XML Schema specification[XMLSchema]. ebXML MSH implementations are strongly RECOMMENDED to include the XMLSchema-instance namespace qualified **schemaLocation** attribute in the SOAP **Envelope** element to indicate to validating parsers the location of the schema document that should be used to validate the document. Failure to include the **schemaLocation** attribute will possibly preclude *Receiving MSH* implementations from being able to validate messages received.

For example:

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

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

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

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

7.2.3 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(s) that sent and should receive the message. This element MAY be omitted.

- **ErrorList** – an 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. This element MAY be omitted.

- **Signature** – an element that contains a digital signature that conforms to [XMLDSIG] that signs data associated with the message. This element MAY be omitted.

- **Acknowledgment**– an element that is used by a *Receiving MSH* to acknowledge to the *Sending MSH* that a previous message has been received. This element MAY be omitted.

- **Via**– an element that is used to convey information to the next ebXML Message Service Handler that receives the message. This element MAY be omitted.

- SOAP **Body** extensions:
  - **Manifest** – an element that points to any data present either in the *Payload Container* or elsewhere, e.g. on the web. This element MAY be omitted.

  - **StatusRequest** – an element that is used to identify a message whose status is being requested. This element MAY be omitted.

  - **StatusResponse** – an element that is used by a MSH when responding to a request on the status of a message that was previously received. This element MAY be omitted.

  - **DeliveryReceipt** – an element used by the *To Party* that received a message, to let the *From Party* that sent the message know the message was received. This element MAY be omitted.

7.2.4 #wildcard element content

Some ebXML SOAP extension elements allow for foreign namespace-qualified element content to be added to provide for extensibility. The 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), the MSH SHALL report an error (see section 10) with **errorCode** set to **NotSupported** and **severity** set to **error**. 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.

7.2.5 id attributes

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

7.3 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.

7.4 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 ten subordinate elements:

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

- **SequenceNumber**
- **Description**

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

- SOAP **mustUnderstand**
- **Version**

In addition, the **MessageHeader** element MAY include an **id** attribute. See section 7.2.5 for details.

7.4.1 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 contain 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 contain one or more **PartyId** child elements.

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

Note This mechanism is particularly useful when transport of a message between the parties may involve multiple intermediaries (see Sections 7.5.4, Multi-hop TraceHeader Sample and 9.3, ebXML Reliable Messaging Protocol). More generally, the *From Party* should provide identification in all domains it knows in support of intermediaries and destinations that may give preference to particular identification systems.

7.4.1.1 PartyID element

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. It is further recommended that these values be taken from the EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registries.

If the **PartyId type** attribute is not present, the content of the **PartyId** element MUST be a URI [RFC2396], otherwise the *Receiving MSH* SHOULD report an error (see section 10) with **errorCode** set to **Inconsistent** and **severity** set to **Error**. It is strongly RECOMMENDED that the content of the **PartyID** element be a URI.

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

```
<eb:From>
  <eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
  <eb:PartyId eb:type="SCAC">RDWY</PartyId>
</eb:From>
<eb:To>
  <eb:PartyId>mailto:joe@example.com</eb:PartyId>
</eb:To>
```

7.4.2 CPAId element

The REQUIRED **CPAId** element is a string that identifies the parameters governing 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 [ebCPP]. An example of the **CPAId** element follows:

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

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

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

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

7.4.3 ConversationId element

The REQUIRED **ConversationId** element is a string identifying the set of related messages that make up a conversation between two *Parties*. It MUST be unique within the **From** and **To** party pair. The *Party* initiating 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 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.

7.4.4 Service element

The REQUIRED **Service** element identifies the *service* that acts on the message and 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 In the context of an ebXML business process model, an *action* equates to the lowest possible role based activity in the [ebBPSS] (requesting or responding role) and a *service* is a set of related actions for an authorized role within a party.

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

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

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

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

7.4.4.1 type attribute

If the **type** attribute is present, it indicates the parties 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 10).

7.4.5 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>
```

7.4.6 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>20001209-133003-28572@example.com</eb:MessageId>
  <eb:Timestamp>2001-02-15T11:12:12Z</eb:Timestamp>
  <eb:RefToMessageId>20001209-133003-28571@example.com</eb:RefToMessageId>
</eb:MessageData>
```

7.4.6.1 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.

7.4.6.2 Timestamp element

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

7.4.6.3 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 10).

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 7.13.4 and 9.

When **RefToMessageId** is contained inside either a **StatusRequest** or a **StatusResponse** element then it identifies a Message whose current status is being queried (see section 8.1)

7.4.6.4 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 9.

7.4.7 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 SHALL 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 9. The **deliverySemantics** attribute indicates whether or not a message is sent reliably.

7.4.7.1 deliveryReceiptRequested attribute

The **deliveryReceiptRequested** attribute is 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 a **DeliveryReceipt** element.

Note To clarify the distinction between an acknowledgement message containing a **DeliveryReceipt** and a Reliable Messaging Acknowledgement: (1) An acknowledgement message containing a **Delivery Receipt** indicates the *To Party* has received the message. (2) The Reliable Messaging Acknowledgment indicates a MSH, possibly only an intermediate MSH, has received the message.

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

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* a message containing a **DeliveryReceipt** element.

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**.

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

- the **Service** element MUST be set to **uri:www.ebXML.org/messageService/**
- the **Action** element MUST be set to **DeliveryReceipt**

An example of **deliveryReceiptRequested** follows:

```
<eb:QualityOfServiceInfo eb:deliverySemantics="OnceAndOnlyOnce"
            eb:messageOrderSemantics="Guaranteed"
            eb:deliveryReceiptRequested="Unsigned"/>
```

7.4.7.2 messageOrderSemantics attribute

The **messageOrderSemantics** attribute is used to indicate whether the message is passed to the receiving application in the order 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 the sending application specified.

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

If **messageOrderSemantics** is set to **Guaranteed**, 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 **messageOrderSemantics** is set to **Guaranteed** the **SequenceNumber** element MUST be present.

If **deliverySemantics** is not **OnceAndOnlyOnce** and **messageOrderSemantics** is set to **Guaranteed** then report the error to the *From Party* with an **errorCode** of **Inconsistent** and a **severity** of **Error** (see sections 9 and 10).

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

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 eb:deliverySemantics="OnceAndOnlyOnce"
    eb:messageOrderSemantics="Guaranteed"/>
```

7.4.8 SequenceNumber element

The **SequenceNumber** element indicates the sequence in which messages MUST be processed by a *Receiving MSH*. The **SequenceNumber** is unique within the **ConversationId** and MSH. The *From Party* MSH and the *To Party* MSH each set an independent **SequenceNumber** as the *Sending MSH* within the **ConversationID**. 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** has a value of **OnceAndOnlyOnce** and **messageOrderSemantics** has a value of **Guaranteed**. If this criterion is not met, an error MUST be reported to the From Party MSH with an **errorCode** of **Inconsistent** and a **severity** of **Error**.

A 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 *Receiving MSH* MUST indicate a delivery failure to the *Sending MSH* with **errorCode** set to **DeliveryFailure** and **severity** set to **Error** (see section 10).

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

1. First message from the *Sending MSH* within the conversation

2. First message after resetting **SequenceNumber** information by the *Sending 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 **SequenceNumber** is reset as shown in 1 or 2 above

- **Continue** – the **SequenceNumber** continues sequentially (including 3 above)

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

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

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

7.4.9 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**.

7.4.10 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 "1.0". Future versions of this specification SHALL require other values of this attribute. The version attribute MUST be namespace qualified for the ebXML SOAP **Envelope** extensions namespace defined above.

7.4.11 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).

7.4.12 MessageHeader sample

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

```
<eb:MessageHeader id="…" eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From><eb:PartyId>uri:example.com</eb:PartyId></eb:From>
  <eb:To eb:type="someType">
     <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
  </eb:To>
  <eb:CPAId>http://www.ebxml.org/cpa/123456</eb:CPAId>
  <eb:ConversationId>987654321</eb:ConversationId>
  <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
  <eb:Action>NewPurchaseOrder</eb:Action>
  <eb:MessageData>
    <eb:MessageId>mid:UUID-2</eb:MessageId>
    <eb:Timestamp>2000-07-25T12:19:05Z</eb:Timestamp>
    <eb:RefToMessageId>mid:UUID-1</eb:RefToMessageId>
  </eb:MessageData>
  <eb:QualityOfServiceInfo
      eb:deliverySemantics="OnceAndOnlyOnce"
      eb:deliveryReceiptRequested="Signed"/>
</eb:MessageHeader>
```

7.5 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 section 7.5.3), and
- the message is not being sent reliably (see section 9)

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

- SOAP **mustUnderstand** (See section 7.4.11 for details)
- SOAP **actor** attribute with the value "http://schemas.xmlsoap.org/soap/actor/next"
- **Version** (See section 7.4.10 for details)

In addition, the **TraceHeaderList** element MAY include an **id** attribute. See section 7.2.5 for details.

7.5.1 SOAP actor attribute

The **TraceHeaderList** 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 **TraceHeaderList** element MUST be processed by

the MSH that receives the message and SHOULD NOT be forwarded to the next MSH. A MSH that handles the **TraceHeaderList** element is REQUIRED to perform the function of appending a new **TraceHeader** element to the **TraceHeaderList** and (re)inserting it into the message for the next MSH.

7.5.2 TraceHeader element

The **TraceHeader** element contains information about a single transmission of a message between two instances of a MSH. 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 by the *Sending MSH*.

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

- **Sender**
- **Receiver**
- **Timestamp**
- **#wildcard**

In addition, the **TraceHeader** element MAY include an **id** attribute. See section 7.2.5 for details.

7.5.2.1 Sender element

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

- **PartyId**
- **Location**

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

7.5.2.1.1 PartyId element

This element has the syntax and semantics described in Section 7.4.1.1, **PartyId** element. In this case, the identified party is the sender of the message. This element may be used in a later message addressed to this party by including it in the **To** element of that message.

7.5.2.1.2 Location element

This element contains the URL of the Sender's Message Service Handler. Unless there is another URL identified within the *CPA* or in **MessageHeader** (section 7.4.2), the recipient of the message uses the URL to send a message, when required that:

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

7.5.2.2 Receiver element

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

- **PartyId**
- **Location**

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

The descendant elements of the **Receiver** element (**PartyId** and **Location**) are implemented in the same manner as the Sender element (see section 7.5.2.1).

7.5.2.3 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 (section 7.4.6.2).

7.5.2.4 #wildcard element

Refer to section 7.2.4 for discussion of #wildcard element handling.

7.5.3 Single hop TraceHeader sample

A single hop message is illustrated by the diagram below.

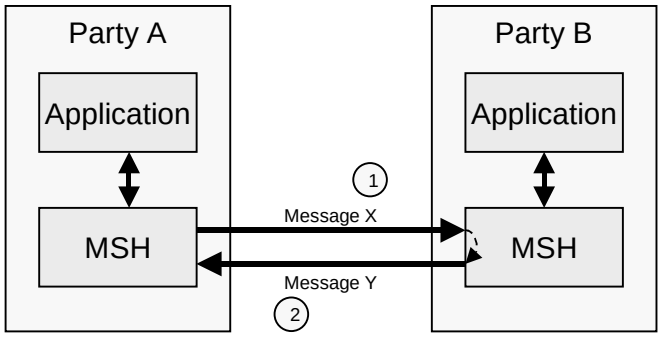


Figure 7-1 Single Hop Message

The content of the corresponding messages could include:

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

```
<eb:MessageHeader eb:id=”...” eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From>
     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
  </eb:From>
  <eb:To>
     <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
  </eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
   ...
</eb:MessageHeader>

<eb:TraceHeaderList eb:id=”...” eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:TraceHeader>
    <eb:Sender>
       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
      <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

7.5.4 Multi-hop TraceHeader sample

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

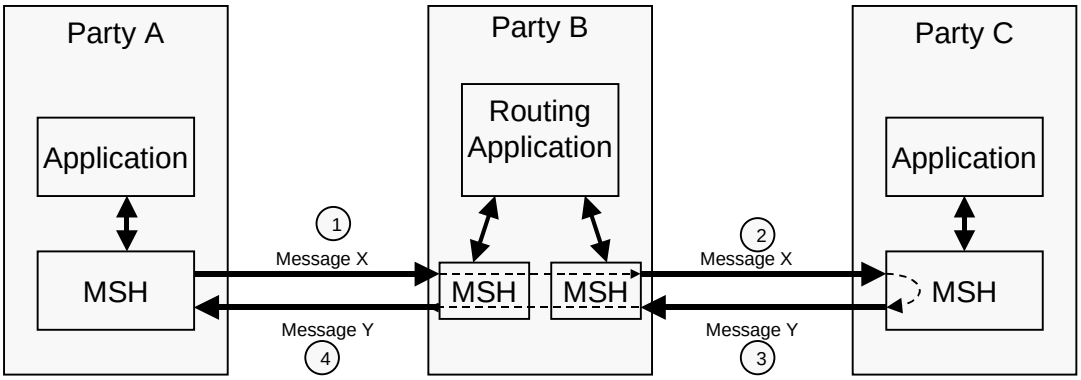


Figure 7-2 Multi-hop Message

The content of the corresponding messages could include:

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

```
<eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From>
     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
  </eb:From>
  <eb:To>
     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
  </eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
   ...
</eb:MessageHeader>

<eb:TraceHeaderList eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1"
       SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
  <eb:TraceHeader>
    <eb:Sender>
       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

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

```
<eb:MessageHeader eb:id="..." eb:version="1.0" SOAP-ENV:mustUnderstand="1">
  <eb:From>
     <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
  </eb:From>
  <eb:To>
     <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
  </eb:To>
  <eb:ConversationId>219cdj89dj2398djfjn</eb:ConversationId>
  ...
  <eb:MessageData>
    <eb:MessageId>29dmridj103kvna</eb:MessageId>
    ...
  </eb:MessageData>
   ...
```

```
</eb:MessageHeader>

<eb:TraceHeaderList eb:id=”...” eb:version="1.0" SOAP-ENV:mustUnderstand="1"
    SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
  <eb:TraceHeader>
    <eb:Sender>
       <eb:PartyId>urn:myscheme.com:id:PartyA-id</eb:PartyId>
       <eb:Location>http://PartyA.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
       <eb:Location>http://PartyB.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:35Z</eb:Timestamp>
  </eb:TraceHeader>
  <eb:TraceHeader>
    <eb:Sender>
       <eb:PartyId>urn:myscheme.com:id:PartyB-id</eb:PartyId>
       <eb:Location>http://PartyB.com/PartyAMsh</eb:Location>
    </eb:Sender>
    <eb:Receiver>
       <eb:PartyId>urn:myscheme.com:id:PartyC-id</eb:PartyId>
       <eb:Location>http://PartyC.com/PartyBMsh</eb:Location>
    </eb:Receiver>
    <eb:Timestamp>2000-12-16T21:19:45Z</eb:Timestamp>
  </eb:TraceHeader>
</eb:TraceHeaderList>
```

7.6 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
- zero or more **ds:Reference** element(s)
- a REQUIRED SOAP **mustUnderstand** attribute (See section 7.4.11 for details)
- a REQUIRED SOAP **actor** attribute
- a REQUIRED **version** attribute (See section 7.4.10 for details)
- an **id** attribute (See section 7.2.5 for details)

7.6.1 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 (section 7.4.6.2).

7.6.2 From element

This is the same element as the **From** element within **MessageHeader** element (see section 7.4.1). 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.

7.6.3 ds:Reference element

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

7.6.4 SOAP actor attribute

The **Acknowledgment** 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 **Acknowledgment** element MUST be processed by the MSH that receives the message and SHOULD NOT be forwarded to the next MSH.

7.6.5 Acknowledgement sample

An example of the **Acknowledgement** element is given below:

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

7.7 Via element

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

Note 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:

- **id** attribute (See section 7.2.5)
- **version** attribute (See section 7.4.10 for details)
- SOAP **MustUnderstand** attribute
- SOAP **actor** attribute

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

7.7.1 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**.

7.7.2 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 MSH that receives the message and SHOULD NOT be forwarded to the next MSH.

7.7.3 syncReply attribute

The **syncReply** attribute is used only if the data communication protocol is *synchronous* (e.g. HTTP). It is an [XMLSchema] 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 return the response from the application or business process in the payload of the *synchronous* reply message. See also the description of **syncReply** in the [ebCPP] specification.

7.7.4 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**.

7.7.5 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 9.2.5 for a complete discussion as to the use of this attribute.

7.7.6 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** (section 7.4.2). 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 identified parameter values SHOULD be used instead of the values identified by the **CPAId** in the **MessageHeader** element.

7.7.7 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 7.4.4 and 7.4.5) 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 Message 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.

7.7.8 Via element sample

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

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

7.8 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 (See section 7.4.11 for details)
- **version** attribute (See section 7.4.10 for details)
- **highestSeverity** attribute

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

7.8.1 id attribute

The **id** attribute uniquely identifies the **ErrorList** element within the document (See section 7.2.5).

7.8.2 highestSeverity attribute

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

7.8.3 Error element

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

- **codeContext**
- **errorCode**
- **severity**
- **location**
- **xml:lang**
- **id** (See section 7.2.5 for details)

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

7.8.3.1 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.

7.8.3.2 errorCode attribute

The REQUIRED **errorCode** attribute indicates the nature of the error in the message in error. Valid values for the **errorCode** and a description of the code's meaning are given in sections 7.8.5.1 and 7.8.5.2

7.8.3.3 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.

7.8.3.4 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`.

7.8.3.5 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** attribute must comply with the rules for identifying languages specified in [XML].

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

7.8.4 ErrorList sample

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

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

7.8.5 errorCode values

This section describes the values for the **errorCode** element (see section 7.8.3.2) 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 This narrative MUST NOT be used in the content of the **Error** element.

- the third column 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.

7.8.5.1 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 content of the **Error** element should be used to indicate the nature of the problem.

7.8.5.2 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** 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 **MessageHeader** element
SecurityFailure	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.

Error Code	Short Description	Long Description
Unknown	Unknown Error	Indicates that an error has occurred that is not covered explicitly by any of the other errors. The content of the **Error** element should be used to indicate the nature of the problem.

7.9 ds:Signature element

An ebXML Message may be digitally signed to provide security countermeasures. Zero or more **ds:Signature** elements, belonging to the [XMLDSIG] defined namespace MAY be present in the SOAP **Header**. The **ds:Signature** element MUST be namespace qualified in accordance with [XMLDSIG]. The structure and content of the **ds:Signature** element MUST conform to the [XMLDSIG] specification. If there is more than one **ds: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 11. Additional **ds:Signature** elements MAY be present, but their purpose is undefined by this specification.

Refer to section 11 for a detailed discussion on how to construct the **ds:Signature** element when digitally signing an ebXML Message.

7.10 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
- **StatusRequest** element
- **StatusResponse** element
- **DeliveryReceipt** element

Each is defined in the following sections.

7.11 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. It is RECOMMENDED that no payload data be present in the SOAP **Body**. The purpose of the **Manifest** is as follows:

- to make it easier to directly extract a particular payload associated with this 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:

- an **id** attribute
- a REQUIRED **version** attribute (See section 7.4.10 for details)
- one or more **Reference** elements
- **#wildcard**

7.11.1 id attribute

The **Manifest** element MUST have an **id** attribute that is an XML ID (See section 7.2.5).

7.11.2 #wildcard element

Refer to section 7.2.4 for discussion of #wildcard element handling.

7.11.3 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** - an 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.

7.11.3.1 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

7.11.3.2 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>
```

7.11.3.3 #wildcard element

Refer to section 7.2.4 for discussion of #wildcard element handling.

7.11.4 References 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**.

7.11.5 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**.

7.11.6 Manifest sample

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

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

7.12 StatusRequest element

The **StatusRequest** element is an immediate child of a SOAP **Body** and is used to identify an earlier message whose status is being requested (see section 8.1).

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

- a REQUIRED **RefToMessageId** element
- a REQUIRED **version** attribute (See section 7.4.10 for details)

- an **id** attribute (See section 7.2.5 for details)

7.12.1 StatusRequest sample

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

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

7.13 StatusResponse element

The **StatusResponse** 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 8.1).

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

- a REQUIRED **RefToMessageId** element
- a **Timestamp** element
- a REQUIRED **version** attribute (See section 7.4.10 for details)
- a **messageStatus** attribute
- an **id** attribute (See section 7.2.5 for details)

7.13.1 RefToMessageId element

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

7.13.2 Timestamp element

The **Timestamp** element contains the time that the message, whose status is being reported, was received (section 7.4.6.2.). This MUST be omitted if the message whose status is being reported is **NotRecognized** or the request was **UnAuthorized.**

7.13.3 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 **StatusResponse** element is not recognized

- **Received** – the message identified by the **RefToMessageId** element in the **StatusResponse** element has been received by the MSH

Note 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.

7.13.4 StatusResponse sample

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

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

7.14 DeliveryReceipt element

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

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

- an **id** attribute (See section 7.2.5)

- a REQUIRED **version** attribute (See section 7.4.10 for details)

- a **Timestamp** element

- zero or more **ds:Reference** element(s)

7.14.1 Timestamp element

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

7.14.2 ds:Reference element

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

7.14.3 DeliveryReceipt sample

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

```
<eb:DeliveryReceipt eb:version="1.0">
  <eb:Timestamp>2001-03-09T12:22:30Z</eb:Timestamp>
  <ds:Reference URI="cid://blahblahblah/">
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <ds:DigestValue>...</ds:DigestValue>
  </ds:Reference>
</eb:DeliveryReceipt>
```

7.15 Combining ebXML SOAP extension elements

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

7.15.1 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.

7.15.2 MessageHeader element

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

7.15.3 TraceHeaderList element

The **TraceHeaderList** element MAY be present in any message. It MUST be present if the message is being sent reliably (see section 9) or over multiple hops (see section 7.5.4).

7.15.4 StatusRequest element

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

- a **Manifest** element
- an **ErrorList** element

7.15.5 StatusResponse element

This element MUST NOT be present with the following elements:

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

7.15.6 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 **StatusResponse** element

7.15.7 Acknowledgment element

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

7.15.8 Delivery receipt element

A **DeliveryReceipt** element may be present on any message.

7.15.9 Signature element

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

7.15.10 Via element

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

8 Message Service Handler Services

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

- Message Status Request
- Message Service Handler Ping

If a *Receiving MSH* does not support the service requested, it SHOULD return a SOAP fault with a **faultCode** of **MustUnderstand**. Each service is described below.

8.1 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 for messages that have been sent reliably (see section 9) and the **MessageId** in the **RefToMessageId** is present in *persistent storage* (see section 9.1.1).

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

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

8.1.1 Message status request message

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

- a **MessageHeader** element
- a **TraceHeaderList** element

- a **StatusRequest** element
- a **ds:Signature** element

The **TraceHeaderList** and the **ds:Signature** elements MAY be omitted (see sections 7.5 and 7.15.8).

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

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

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

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

8.1.2 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 SOAP **Header** and **Body**.

- a **MessageHeader** element
- a **TraceHeaderList** element
- an **Acknowledgment** element
- a **StatusResponse** element (see section 7.13)
- a **ds:Signature** element

The **TraceHeaderList**, **Acknowledgment** and **ds:Signature** elements MAY be omitted (see sections 7.5, 7.15.7 and 7.15.8).

The **MessageHeader** 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: **uri:www.ebxml.org/messageService/**
- an **Action** element that contains **StatusResponse**
- a **RefToMessageId** that identifies the Message Status Request message.

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

8.1.3 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.

8.2 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.

8.2.1 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 SOAP **Header**:

- a **MessageHeader** element
- a **TraceHeaderList** element
- a **ds:Signature** element

The **TraceHeaderList** and the **ds:Signature** elements MAY be omitted (see sections 7.5 and 7.15.8).

The **MessageHeader** 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 **CPAId** element
- a **ConversationId** element
- a **Service** element that contains: **uri:www.ebxml.org/messageService/**
- an **Action** element that contains **Ping**

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

8.2.2 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 SOAP **Header**:

- a **MessageHeader** element
- a **TraceHeaderList** element
- an **Acknowledgment** element
- an OPTIONAL **ds:Signature** element

The **TraceHeaderList**, **Acknowledgment** and **ds:Signature** elements MAY be omitted (see sections 7.5, 7.15.7 and 7.15.8).

The **MessageHeader** 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 **CPAId** element
- a **ConversationId** element
- a **Service** element that contains the value: **uri:www.ebxml.org/messageService/**
- 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*.

8.2.3 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.

9 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*.

9.1.1 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, a 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 8.1)
- complete response message

9.1.2 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 SOAP structures together with commercial software products that are designed to provide reliable delivery of messages using alternative protocols.

9.2 Reliable messaging parameters

This section describes the parameters required to control reliable messaging. This parameter information can be specified in the *CPA* or in the **MessageHeader** (section 7.4.2).

9.2.1 Delivery semantics

The **deliverySemantics** value MUST be used by the *From Party* MSH to indicate 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 or in **MessageHeader** (section 7.4.2). The default value for **deliverySemantics** is **BestEffort**.

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

9.2.2 mshTimeAccuracy

The **mshTimeAccuracy** parameter indicates the minimum accuracy 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.

9.2.3 TimeToLive

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 **mshTimeAccuracy** parameter for the *Receiving MSH* indicating 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**.

9.2.4 reliableMessagingMethod

The **reliableMessagingMethod** attribute 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 7.7.4 for discussion of the use of this attribute.

9.2.5 ackRequested

The **ackRequested** value is used by the *Sending MSH* to request that the *Receiving MSH* returns an *acknowledgment message* with an **Acknowledgment** element.

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**.

9.2.6 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.

9.2.7 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.

9.2.8 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 9.4).

9.3 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.

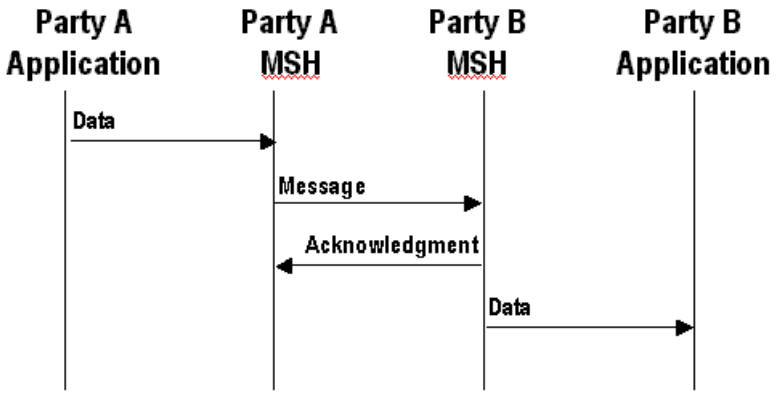


Figure 9-1 Indicating that a message has been received

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

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

9.3.1 Sending message behavior

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

1. Create a message from components received from the application that includes a **TraceHeader** element identifying the sender and the receiver as described in Section 7.5.2 **TraceHeader** element.

2. Save the message in *persistent storage* (see section 9.1.1)

3. Send the message to the *Receiver MSH*

4. Wait for the *Receiver MSH* to return an *Acknowledgment Message* and, if it does not or a transient error is returned, then take the appropriate action as described in section 9.3.4

9.3.2 Receiving message behavior

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

1. If the message is just an acknowledgement (i.e. the **Service** element is set to http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment and **Action** is set to **Acknowledgment**), then:

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

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

2. Otherwise, if the message is not just an acknowledgement, then check to see if the message is a duplicate (e.g. there is a **MessageId** held in *persistent storage* that was received earlier that contains the same value for the **MessageId**)

a.) 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 9.3.3).

b.) 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 if the CPA indicates 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 9.3.3).

9.3.3 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 **Via** 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 **syncReply** 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 **MessageHeader** 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 **MessageHeader** elements MUST be set as follows:

- The Service element MUST be set to: uri:www.ebxml.org/messageService/
- The Action element MUST be set to Acknowledgment.
- The From element MAY be populated with the To element extracted from the message received, or it MAY be set using the Receiver from the last TraceHeader in the message that has just been received. In either case, all PartyId elements from the message received SHOULD be included in this From element.
- The To element MAY be populated with the From element extracted from the message received, or it MAY be set using the Sender from the last TraceHeader in the message that has just been received. In either case, all PartyId elements from the message received SHOULD be included in this To element.
- The RefToMessageId element MUST be set to the MessageId of the message that has just been received

9.3.4 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:

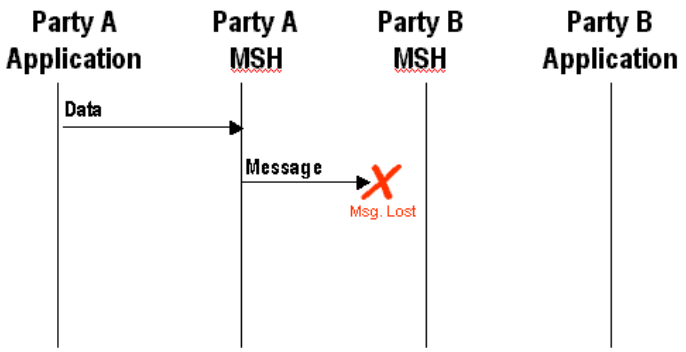


Figure 9-2 Undelivered Message

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

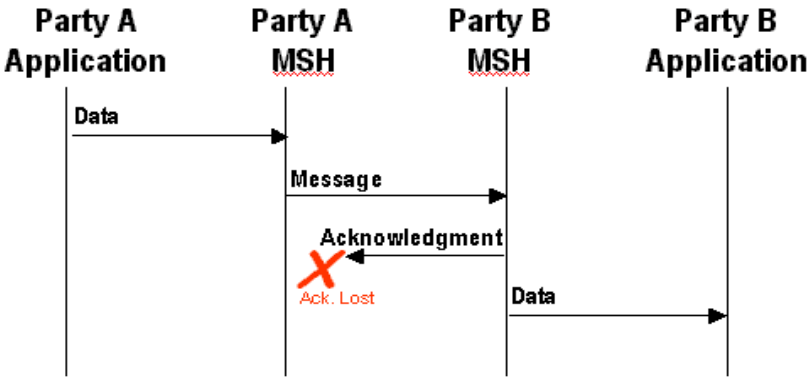


Figure 9-3 Lost Acknowledgment Message

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 the following are both true:

   a.) At least the time specified in the **retryInterval** has passed since the message was last sent, and

   b.) 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 an unrecoverable communications protocol error at the transport protocol level, the *Sending MSH* SHOULD resend the message.

9.3.5 Duplicate message handling

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

- an “identical message” is a *message* that contains, apart from an additional **TraceHeader** element, the same ebXML SOAP **Header**, **Body** 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.

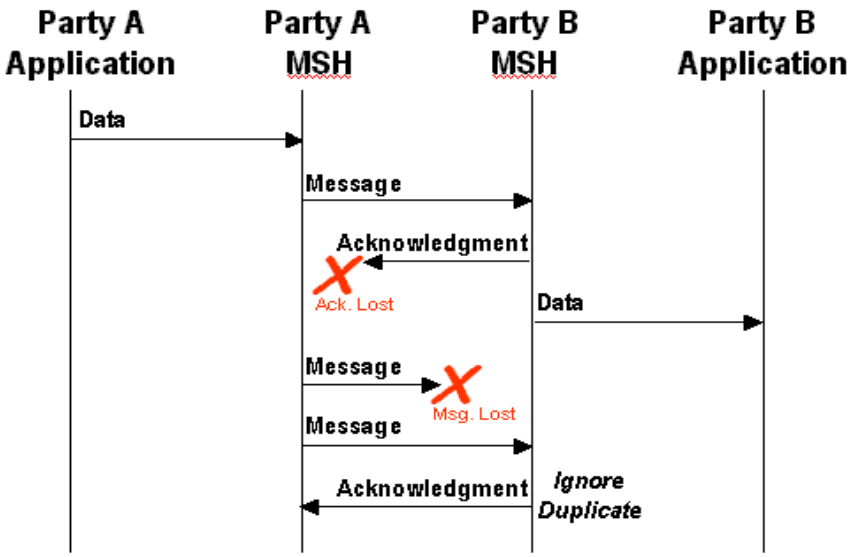


Figure 9-4 Resending Unacknowledged Messages

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.

9.4 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** element that identifies the *Party* who detected the problem
- a **To** element that identifies the *From Party* that created the message that could not be delivered
- a **Service** element and **Action** element set as described in 10.5
- 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 the message probably was not delivered although there is a small probability 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 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 a layer of processing above the SOAP processor layer. This means 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**.

10.1 Definitions

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

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

10.2 Types of errors

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

- ebXML namespace qualified content of the *SOAP Message* document (see section 7)
- reliable messaging failures (see section 9)
- security (see section 11)

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 do not use the error reporting mechanism described here.

10.3 When to generate error messages

When a 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 section 10.4) 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, and
- the problem is resolved by other means, and
- no further action is taken.

10.3.1 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 resulting in this course of action is implementation dependent.

10.4 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* and a **TraceHeaderList** is present in the message in error, the value of the **Location** element in the **Sender** of the topmost **TraceHeade**r MUST be used. Otherwise, the recipient MAY resolve an **ErrorURI** using the **From** element of the message in error. If this is not possible, no error will be reported to the sending *Party*.

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.

10.5 Service and action element values

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

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

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

11 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 and spoofing

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

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, 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.

11.1 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/)

11.2 Collaboration protocol agreement

The configuration of Security for MSHs may be 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.

11.3 Countermeasure technologies

11.3.1 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 SOAP **Header** and **Body** to the ebXML Payload or data elsewhere on the web that relates to the message. It is also strongly RECOMMENDED that XML Signature be 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 (the ebXML SOAP **Header** and **Body** 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 requiring a digital signature SHALL be signed following the process defined in this section of the specification and SHALL be in full compliance with [XMLDSIG].

11.3.1.1 Signature generation

1. Create a **ds:SignedInfo** element with **ds:SignatureMethod**, **ds:CanonicalizationMethod**, and **ds:Reference** elements for the SOAP **Header** and any required payload objects, as prescribed by [XMLDSIG].

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

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

4. Include the namespace qualified **ds:Signature** element in the SOAP **Header** 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/2000/09/xmldsig#dsa-sha1

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

The **ds:Reference** element for the SOAP **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 SOAP **Header**).

The **ds:Reference** element for the SOAP **Header** MAY include a **Type** attribute that has a value "http://www.w3.org/2000/09/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 SOAP **Header** SHALL include a child **ds:Transforms** element. The **ds:Transforms** element SHALL include two **ds:Transform** child elements. The first **ds:Transform** element SHALL have a **ds:Algorithm** attribute that has a value of:

http://www.w3.org/2000/09/xmldsig#enveloped-signature

The second **ds:Transform** element SHALL have a child **ds:XPath** element that has a value of:

```
not(ancestor-or-self::eb:TraceHeaderList or
    ancestor-or-self::eb:Via)
```

The result of the first [XPath] statement excludes the **ds:Signature** element within which it is contained, and all its descendants, and the second [XPath] statement excludes the **TraceHeaderList** and **Via** elements and all their descendants, as these elements are subject to change.

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 MIME body part of the payload object, or a URI that matches the Content-Location of the MIME body part of the payload object, or a URI that resolves to an external payload object external to the Message Package. It is strongly RECOMMENDED that the URI attribute value match the xlink:href URI value of the corresponding **Manifest/Reference** element for 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 eb:id="…" eb:version="1.0">
    ...
    </eb:MessageHeader>
    <eb:TraceHeaderList eb:id="…" eb:version="1.0">
      <eb:TraceHeader>
        ...
      </eb:TraceHeader>
    </eb:TraceHeaderList>
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <ds:Reference URI="">
              <Transforms>
                 <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
                    <XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                               not(ancestor-or-self::eb:TraceHeaderList or
                               ancestor-or-self::eb:Via)
                    </XPath>
                 </Transform>
              </Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:DigestValue>...</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="cid://blahblahblah/">
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-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 eb: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>
```

11.3.2 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 a **ds:Reference** element contained in the **ds:Signature** element of the original message within the **Acknowledgment** element.

11.3.3 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* with a way to authenticate the destination for the TCP/IP environment.

11.3.4 Non-persistent Integrity

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

11.3.5 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 SOAP **Header**.

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]) 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.

11.3.6 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.

11.3.7 Persistent authorization

The OASIS Security Services Technical Committee (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 [SAML]. 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.*

11.3.8 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*.

11.3.9 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 to provide this capability.

11.3.10 Supported security services

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

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
✓	Profile 0										no security services are applied to data
✓	Profile 1	✓									*Sending MSH* applies XML/DSIG structures to message
	Profile 2		✓						✓		*Sending MSH* authenticates and *Receiving MSH* authorizes sender based on communication channel credentials.

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
	Profile 3		✓				✓				*Sending MSH* authenticates and both MSHs negotiate a secure channel to transmit data
	Profile 4		✓		✓						*Sending MSH* authenticates, the *Receiving MSH* performs integrity checks using communications protocol
	Profile 5		✓								*Sending MSH* authenticates the communication channel only (e.g., SSL 3.0 over TCP/IP)
	Profile 6	✓					✓				*Sending MSH* applies XML/DSIG structures to message and passes in secure communications channel
	Profile 7	✓		✓							*Sending MSH* applies XML/DSIG structures to message and *Receiving MSH* returns a signed receipt
	Profile 8	✓		✓			✓				combination of profile 6 and 7
	Profile 9	✓								✓	Profile 5 with a trusted timestamp applied
	Profile 10	✓		✓						✓	Profile 9 with *Receiving MSH* returning a signed receipt
	Profile 11	✓					✓			✓	Profile 6 with the *Receiving MSH* applying a trusted timestamp
	Profile 12	✓		✓			✓			✓	Profile 8 with the *Receiving MSH* applying a trusted timestamp
	Profile 13	✓				✓					*Sending MSH* applies XML/DSIG structures to message and applies confidentiality structures (XML-Encryption)
	Profile 14	✓		✓		✓					Profile 13 with a signed receipt
	Profile 15	✓		✓						✓	*Sending MSH* applies XML/DSIG structures to message, a trusted timestamp is added to message, *Receiving MSH* returns a signed receipt
	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 [SAML]

Present in baseline MSH		Persistent digital signature	Non-persistent authentication	Persistent signed receipt	Non-persistent integrity	Persistent confidentiality	Non-persistent confidentiality	Persistent authorization	Non-persistent authorization	Trusted timestamp	Description of Profile
	Profile 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)

12 References

12.1 Normative references

[RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task Force RFC 2119, March 1997

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

[RFC822] Standard for the Format of ARPA Internet text messages. D. Crocker. August 1982.

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

[RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N. Borenstein. November 1996.

[RFC2246] RFC 2246 - Dierks, T. and C. Allen, "The TLS Protocol", January 1999.

[RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.

[RFC2392] IETF RFC 2392. Content-ID and Message-ID Uniform Resource Locators. E. Levinson, Published August 1998

[RFC2396] IETF RFC 2396. Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, Published August 1998

[RFC2487] SMTP Service Extension for Secure SMTP over TLS. P. Hoffman. January 1999.

[RFC2554] SMTP Service Extension for Authentication. J. Myers. March 1999.

[RFC2616] RFC 2616 - Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", , June 1999.

[RFC2617] RFC2617 - Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", June 1999.

[RFC2817] RFC 2817 - Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", May 2000.

[RFC2818] RFC 2818 - Rescorla, E., "HTTP Over TLS", May 2000 [SOAP] Simple Object Access Protocol

[SMTP] IETF RFC 822, Simple Mail Transfer Protocol, D Crocker, 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

[SSL3] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol", Netscape Communications Corp., Nov 18, 1996.

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

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

[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

[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

12.2 Non-normative references

[ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 1.0, published 11 May, 2001

[ebBPSS] ebXML Business Process Specification Schema, version 1.0, published 11 May 2001.

[ebTA] ebXML Technical Architecture, version 1.04 published 16 February, 2001

[secRISK] ebXML Technical Architecture Risk Assessment Technical Report, version 1.0 published 11 May 2001

[ebRS] ebXML Registry Services Specification, version 1.0

[ebMSREQ] ebXML Transport, Routing and Packaging: Overview and Requirements, Version 0.96, Published 25 May 2000

[ebGLOSS] ebXML Glossary, http://www.ebxml.org, published 11 May, 2001.

[IPSEC] IETF RFC2402 IP Authentication Header. S. Kent, R. Atkinson. November 1998. RFC2406 IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November 1998.

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

[SAML] Security Assertion Markup Language, http://www.oasis-open.org/committees/security/docs/draft-sstc-use-strawman-03.html

[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.

[TLS] RFC2246, T. Dierks, C. Allen. January 1999.

[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

13 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
Email:	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
EMail:	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

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.

Appendix A ebXML SOAP Extension Elements Schema

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

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

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

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

XML Signature Core – http://ebxml.org/project_teams/transport/xmldsig-core-schema.xsd

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

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

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

Note If inconsistencies exist between the specification and this schema, the specification supersedes this example schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
xmlns:tns="http://www.ebxml.org/namespaces/messageHeader" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns="http://www.w3.org/2000/10/XMLSchema" version="1.0">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
schemaLocation="http://www.ebxml.org/project_teams/transport/xmldsig-core-schema.xsd"/>
  <import namespace="http://www.w3.org/1999/xlink"
schemaLocation="http://www.ebxml.org/project_teams/transport/xlink.xsd"/>
  <import namespace="http://schemas.xmlsoap.org/soap/envelope/"
schemaLocation="http://www.ebxml.org/project_teams/transport/envelope.xsd"/>
  <import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.ebxml.org/project_teams/transport/xml_lang.xsd"/>
  <!-- MANIFEST -->
  <element name="Manifest">
    <complexType>
      <sequence>
```

```
      <element ref="tns:Reference" maxOccurs="unbounded"/>
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
       processContents="lax"/>
  </complexType>
</element>
<element name="Reference">
  <complexType>
    <sequence>
      <element ref="tns:Schema" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="xlink:type" use="fixed" value="simple"/>
    <attribute ref="xlink:href" use="required"/>
    <attribute ref="xlink:role"/>
  </complexType>
</element>
<element name="Schema">
  <complexType>
    <attribute name="location" type="uriReference" use="required"/>
    <attribute name="version" type="tns:non-empty-string"/>
  </complexType>
</element>
<!-- MESSAGEHEADER -->
<element name="MessageHeader">
  <complexType>
    <sequence>
      <element ref="tns:From"/>
      <element ref="tns:To"/>
      <element ref="tns:CPAId"/>
      <element ref="tns:ConversationId"/>
      <element ref="tns:Service"/>
      <element ref="tns:Action"/>
      <element ref="tns:MessageData"/>
      <element ref="tns:QualityOfServiceInfo" minOccurs="0"/>
      <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="tns:SequenceNumber" minOccurs="0"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute ref="soap:mustUnderstand"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<element name="CPAId" type="tns:non-empty-string"/>
<element name="ConversationId" type="tns:non-empty-string"/>
<element name="Service">
  <complexType>
    <simpleContent>
      <extension base="tns:non-empty-string">
        <attribute name="type" type="tns:non-empty-string"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
<element name="Action" type="tns:non-empty-string"/>
<element name="MessageData">
  <complexType>
    <sequence>
      <element ref="tns:MessageId"/>
      <element ref="tns:Timestamp"/>
      <element ref="tns:RefToMessageId" minOccurs="0"/>
      <element ref="tns:TimeToLive" minOccurs="0"/>
```

```
    </sequence>
  </complexType>
</element>
<element name="MessageId" type="tns:non-empty-string"/>
<element name="TimeToLive" type="timeInstant"/>
<element name="QualityOfServiceInfo">
  <complexType>
    <attribute name="deliverySemantics" type="tns:deliverySemantics.type" use="default"
      value="BestEffort"/>
    <attribute name="messageOrderSemantics" type="tns:messageOrderSemantics.type"
      use="default" value="NotGuaranteed"/>
    <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type"
      use="default" value="None"/>
  </complexType>
</element>
<!-- TRACE HEADER LIST -->
<element name="TraceHeaderList">
  <complexType>
    <sequence>
      <element ref="tns:TraceHeader" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute ref="soap:mustUnderstand" use="required"/>
    <attribute ref="soap:actor" use="required"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<element name="TraceHeader">
  <complexType>
    <sequence>
      <element ref="tns:Sender"/>
      <element ref="tns:Receiver"/>
      <element ref="tns:Timestamp"/>
      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
  </complexType>
</element>
<element name="Sender" type="tns:senderReceiver.type"/>
<element name="Receiver" type="tns:senderReceiver.type"/>
<element name="SequenceNumber" type="positiveInteger"/>
<!-- DELIVERY RECEIPT -->
<element name="DeliveryReceipt">
  <complexType>
    <sequence>
      <element ref="tns:Timestamp"/>
      <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
        processContents="lax"/>
    <!-- <attribute name="signed" type="boolean"/> -->
  </complexType>
</element>
<!-- ACKNOWLEDGEMENT -->
<element name="Acknowledgment">
  <complexType>
    <sequence>
      <element ref="tns:Timestamp"/>
      <element ref="tns:From" minOccurs="0"/>
      <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute ref="soap:mustUnderstand" use="required"/>
    <attribute ref="soap:actor" use="required"/>
```

```
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<!-- ERROR LIST -->
<element name="ErrorList">
  <complexType>
    <sequence>
      <element ref="tns:Error" maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute ref="soap:mustUnderstand" use="required"/>
    <attribute name="highestSeverity" type="tns:severity.type"
      use="default" value="Warning"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
       processContents="lax"/>
  </complexType>
</element>
<element name="Error">
  <complexType>
    <attribute ref="tns:id"/>
    <attribute name="codeContext" type="uriReference" use="required"/>
    <attribute name="errorCode" type="tns:non-empty-string" use="required"/>
    <attribute name="severity" type="tns:severity.type" use="default" value="Warning"/>
    <attribute name="location" type="tns:non-empty-string"/>
    <attribute ref="xml:lang"/>
  </complexType>
</element>
<!-- STATUS RESPONSE -->
<element name="StatusResponse">
  <complexType>
    <sequence>
      <element ref="tns:RefToMessageId"/>
      <element ref="tns:Timestamp" minOccurs="0"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute name="messageStatus" type="tns:messageStatus.type"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<!-- STATUS REQUEST -->
<element name="StatusRequest">
  <complexType>
    <sequence>
      <element ref="tns:RefToMessageId"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<!-- VIA  -->
<element name="Via">
  <complexType>
    <sequence>
      <element ref="tns:CPAId" minOccurs="0"/>
      <element ref="tns:Service" minOccurs="0"/>
      <element ref="tns:Action" minOccurs="0"/>
    </sequence>
    <attribute ref="tns:id"/>
    <attribute ref="tns:version"/>
    <attribute ref="soap:mustUnderstand" use="required"/>
    <attribute ref="soap:actor" use="required"/>
    <attribute name="syncReply" type="boolean"/>
```

```
    <attribute name="deliveryReceiptRequested" type="tns:signedUnsigned.type"
      use="default" value="None"/>
    <attribute name="reliableMessagingMethod" type="tns:rmm.type"/>
    <attribute name="ackRequested" type="boolean"/>
    <anyAttribute namespace="http://www.w3.org/2000/10/XMLSchema-instance"
      processContents="lax"/>
  </complexType>
</element>
<!-- COMMON TYPES -->
<complexType name="senderReceiver.type">
  <sequence>
    <element ref="tns:PartyId" maxOccurs="unbounded"/>
    <element name="Location" type="uriReference"/>
  </sequence>
</complexType>
<simpleType name="messageStatus.type">
  <restriction base="NMTOKEN">
    <enumeration value="UnAuthorized"/>
    <enumeration value="NotRecognized"/>
    <enumeration value="Received"/>
    <enumeration value="Processed"/>
    <enumeration value="Forwarded"/>
  </restriction>
</simpleType>
<simpleType name="type.type">
  <restriction base="NMTOKEN">
    <enumeration value="DeliveryReceipt"/>
    <enumeration value="IntermediateAck"/>
  </restriction>
</simpleType>
<simpleType name="messageOrderSemantics.type">
  <restriction base="NMTOKEN">
    <enumeration value="Guaranteed"/>
    <enumeration value="NotGuaranteed"/>
  </restriction>
</simpleType>
<simpleType name="deliverySemantics.type">
  <restriction base="NMTOKEN">
    <enumeration value="OnceAndOnlyOnce"/>
    <enumeration value="BestEffort"/>
  </restriction>
</simpleType>
<simpleType name="non-empty-string">
  <restriction base="string">
    <minLength value="1"/>
  </restriction>
</simpleType>
<simpleType name="rmm.type">
  <restriction base="NMTOKEN">
    <enumeration value="ebXML"/>
    <enumeration value="Transport"/>
  </restriction>
</simpleType>
<simpleType name="signedUnsigned.type">
  <restriction base="NMTOKEN">
    <enumeration value="Signed"/>
    <enumeration value="Unsigned"/>
    <enumeration value="None"/>
  </restriction>
</simpleType>
<simpleType name="severity.type">
  <restriction base="NMTOKEN">
    <enumeration value="Warning"/>
    <enumeration value="Error"/>
  </restriction>
</simpleType>
<!-- COMMON ATTRIBUTES and ELEMENTS -->
<attribute name="id" type="ID" form="unqualified"/>
<attribute name="version" type="tns:non-empty-string" use="fixed" value="1.0"/>
```

```
  <element name="PartyId">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute name="type" type="tns:non-empty-string"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="To">
    <complexType>
      <sequence>
        <element ref="tns:PartyId" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="From">
    <complexType>
      <sequence>
        <element ref="tns:PartyId" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name="Description">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute ref="xml:lang"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="RefToMessageId" type="tns:non-empty-string"/>
  <element name="Timestamp" type="timeInstant"/>
</schema>
```

Appendix B Communication Protocol Bindings

Introduction

One of the goals of ebXML's Transport, Routing and Packaging team is to design a message handling service 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, Java Message Service (JMS) 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 Message 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.

HTTP

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.

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 6 and **7**. 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 MAY 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 trailing MIME boundary strings.

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: "ebXML"
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
        start=" <ebxhmheader111@example.com>"

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

<?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'>
<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>urn:services:SupplierOrderProcessing</eb:Service>
    <eb:Action>NewOrder</eb:Action>
    <eb:MessageData>
      <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
      <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo eb: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--
```

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 (see section 10).

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.

Synchronous vs. asynchronous

When the **syncReply** parameter in the **Via** element 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 above. When the **syncReply** parameter 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 "HTTP response codes," above, and with an empty HTTP body MUST be returned in response to the HTTP Post.

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.

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 hierarchical and peer-to-peer trust models.

SMTP

The Simple Mail Transfer Protocol [SMTP] 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 eMail 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.

Minimum level of supported protocols

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

Sending ebXML messages over SMTP

Prior to sending messages over SMTP an ebXML Message MUST be formatted according to ebXML Message Service Specification sections 6 and 7. 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[SMTP], 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 document part 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 an ebXML message, 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 `SOAPAction` MIME header field must also be included in the eMail MIME header and MAY 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 eMail header "To:" MUST contain the [RFC822] compliant eMail address of the ebXML Message Service Handler.
- The eMail header "From:" MUST contain the [RFC822] compliant eMail address of the senders ebXML Message Service Handler.
- Construct a "Date:" eMail header in accordance with [RFC822]
- Other headers MAY occur within the eMail 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 eMail 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: "ebXML"
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
        start="<ebxhmheader111@example.com>"

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

<?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'>
<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>urn:services:SupplierOrderProcessing</eb:Service>
    <eb:Action>NewOrder</eb:Action>
    <eb:MessageData>
      <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
      <eb:Timestamp>2001-02-15T11:12:12Z</Timestamp>
    </eb:MessageData>
    <eb:QualityOfServiceInfo eb: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--
```

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. A 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 causing 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.

Access control

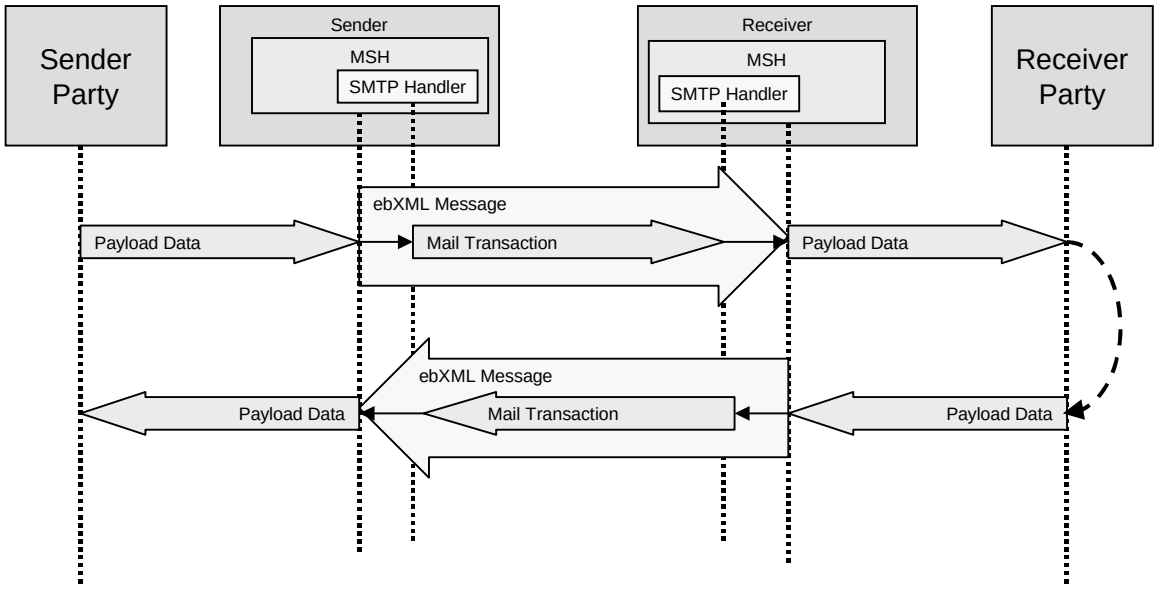
Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the use of an access control mechanism. The SMTP access authentication process described in "SMTP Service Extension for Authentication" [RFC2554] defines the ebXML recommended access control mechanism to protect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

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.

SMTP model

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



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 9).