Message Service Specification

DRAFT Version 2.0

OASIS ebXML Messaging Services Technical Committee

11 January 2002

Status of this Document

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

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

Note: Implementers of this specification should consult the OASIS ebXML Messaging Services Technical Committee web site for current status and revisions to the specification (http://www.oasis-open.org/committees/ebxml-msg/).

Specification
Version 1.0 of this Technical Specification document was approved by the ebXML Plenary in May 2001.

Version 2.0 of this Technical Specification document is presented to the OASIS Messaging Team as a Technical Committee(TC) Specification, January 4, 2002

Version 2.0 of this Technical Specification document is presented to the OASIS membership for consideration as an OASIS Technical Specification, April 2002.

This version

???

Previous version
V1.0 – *http://www.ebxml.org/specs/ebMS.doc*

ebXML Participants

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

Arvola Chan	RosettaNet/TIBCO	Himagiri Mukkamala	Sybase
Aynur Unal	E2Open	Ian Jones	British Telecom
Bob Miller	GE Global eXchange	Jeff Turpin	Cyclone Commerce
Brad Lund	Intel™ Corporation	Jim Hughes	Hewlett Packard
Brian Gibb	Sterling Commerce	Kazunori Iwasa	Fujitsu Limited
Bruce Pedretti	Hewlett-Packard	Martin Sachs	IBM Research
Cedrec Vessell	DISA	Pete Wenzel	RosettaNet/SeeBeyond
Chris Ferris	Sun Microsystems, Inc	Philippe DeSmedt	Agentis Software
Cliff Collins	Sybase	Prasad Yendluri	WebMethods
Colleen Evans	Sonic Software	Ralph Berwanger	BTrade
Jim Galvin	Drummond Group	Sanjay Cherian	Sterling Commerce
Dale Moberg	Cyclone Commerce	Scott Hinkelman	IBM
Daniel Weinreb	eXcelon	Sinisa Zimek	SAP
David Burdett	Commerce One	Yukinori Saito	Ecom
David Fischer	Drummond Group		
Doug Bunting	Sun Microsystems, Inc		

The UN/CEFACT-OASIS v1.0 Team – see Acknowledgments

Table of Contents

Introduction

This specification is one of a series of specifications realizing 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 electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type. This versatility ensures 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.

1 Summary of Contents of this Document

This specification defines the *ebXML Message Service Protocol* enabling 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 sending and receiving those messages over a data communications protocol.

This specification is independent of both the payload and the communications protocol used. Appendices to this specification describe how to use this specification with HTTP [RFC2616] and SMTP [RFC2821].

This specification is organized around the following topics:

Core Functionality

- **Packaging Specification** – A description of how to package an ebXML Message and its associated parts into a form that can be sent using a communications protocol such as HTTP or SMTP (section 2.1),
- **ebXML SOAP Envelope Extensions** – A specification of the structure and composition of the information necessary for an *ebXML Message Service* to generate or process an ebXML Message (section 2.3),
- **Error Handling** – A description of how one *ebXML Message Service* reports errors it detects to another ebXML Message Service Handler (section 4.1.5),
- **Security** – Provides a specification of the security semantics for ebXML Messages (section 4.1),
- **SyncReply** – Indicates to the Next MSH whether or not replies are to be returned synchronously (section 5).

Additional Elements

- **Reliable Messaging** – The Reliable Messaging function defines an interoperable protocol where any two Message Service implementations can reliably exchange messages sent using once-and-only-once delivery semantics (section 7),
- **Message Status Service** – A description of services enabling one service to discover the status of another Message Service Handler (MSH) or an individual message (section 8),
- **Message Order** – The Order of message receipt by the *To Party MSH* can be guaranteed (section 10),
- **Multi-Hop –** Messages may be sent through intermediary MSH nodes (section 10.1.2),

Appendices to this specification cover the following:

- **Appendix A Schema** – This normative appendix contains XML schema definition [XMLSchema] for the ebXML SOAP ***Header*** and ***Body*** Extensions,
- **Appendix B Communications Protocol Envelope Mappings** – This normative appendix describes how to transport *ebXML Message Service* compliant messages over HTTP and SMTP,
- **Appendix C Security Profiles** – a discussion concerning Security Service Profiles.

1.1.1 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). Attribute names begin with lowercase. Element names begin with Uppercase.

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:

- *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).*

1.1.2 Audience

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

1.1.3 Caveats and Assumptions

It is assumed the reader has an understanding of communications 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.

It is strongly RECOMMENDED implementors read and understand the Collaboration Protocol Profile/ Agreement [ebCPP] specification and its implications prior to implementation.

1.1.4 Related Documents

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

- **ebXML Technical Architecture Specification** [ebTA] – defines the overall technical architecture for ebXML
- **ebXML Technical Architecture Risk Assessment Technical Report** [secRISK] – defines the security mechanisms necessary to negate anticipated, selected threats
- **ebXML Collaboration Protocol Profile and Agreement Specification** [ebCPP] – defines how one party can discover and/or agree upon the information the party needs to know about another party prior to sending them a message that complies with this specification
- **ebXML Registry/Repository Services Specification** [ebRS] – defines a registry service for the ebXML environment

1.2 Concept of Operation

1.2.1 Scope

The ebXML Message Service(ebMS) defines the message enveloping and header document schema used to transfer ebXML messages over a communications protocol such as HTTP or SMTP and the behavior of software sending and receiving ebXML messages. The ebMS is defined as a set of layered extensions to the base Simple Object Access Protocol [SOAP] and SOAP Messages with Attachments [SOAPAttach] specifications. This document provides security and reliability features necessary to support international electronic business. These security and reliability features are not provided in the SOAP or SOAP with Attachments specifications.

The ebXML infrastructure is composed of several independent, but related, components. Specifications for the individual components are fashioned as stand-alone documents. The specifications are totally self-contained; nevertheless, design decisions within one document can and do impact the other documents. Considering this, the ebMS is a closely coordinated definition for an ebXML message service handler (MSH).

The ebMS provides the message packaging, routing and transport facilities for the ebXML infrastructure. The ebMS is not defined as a physical component, but rather as an abstraction of a process. An implementation of this specification could be delivered as a wholly independent software application or an integrated component of some larger business process.

1.2.2 Background and Objectives

Traditional business information exchanges have conformed to a variety of standards-based syntaxes. These exchanges were largely based on electronic data interchange (EDI) standards born out of mainframe and batch processing. Some of the standards defined bindings to specific communications protocols. These EDI techniques worked well; however, they were difficult and expensive to implement. Therefore, use of these systems was normally limited to large enterprises possessing mature information technology capabilities.

The proliferation of XML-based business interchanges served as the catalyst for defining a new global paradigm that ensured all business activities, regardless of size, could engage in electronic business activities. The prime objective of ebMS is to facilitate the exchange of electronic business messages within an XML framework. Business messages, identified as the 'payloads' of the ebXML messages, are not necessarily expressed in XML. XML-based messages, as well as traditional EDI formats, are transported by the ebMS. Actually, the ebMS payload can take any digital form—XML, ASC X12, HL7, AIAG E5, database tables, binary image files, etc.

The ebXML architecture requires that the ebXML Message Service protocol be capable of being carried over any available communications protocol. Therefore, this document does not mandate use of a specific communications protocol. This version of the specification provides bindings to HTTP and SMTP, but other protocols can, and reasonably will, be used.

The ebXML Requirements Specification [ebREQ] mandates the need for secure, reliable communications. The ebXML work focuses on leveraging existing and emerging technology—attempts to create new protocols are discouraged. Therefore, this document defines security within the context of existing security standards and protocols. Those requirements satisfied with existing standards are specified in the ebMS, others must be deferred until new technologies or standards are available, for example encryption of individual message header elements.

Reliability requirements defined in the ebREQ relate to delivery of ebXML messages over the communications channels. The ebMS provides mechanisms to satisfy the ebREQ requirements. The reliable messaging elements of the ebMS supply reliability to the communications layer; they are not intended as business-level acknowledgments to the applications supported by the ebMS. This is an important distinction. Business processes often anticipate responses to messages they generate. The responses may take the form of a simple acknowledgment of message receipt by the application receiving the message or a companion message reflecting action on the original message. Those messages are outside of the MSH scope. The acknowledgment defined in this specification does not indicate the payload of the ebXML message was syntactically correct. It does not acknowledge the accuracy of the payload information. It does not indicate business acceptance of the information or agreement with the content of the payload. The ebMS is designed to provide the sender with the confidence the receiving MSH has received the message securely and intact.

The underlying architecture of the MSH assumes messages are exchanged between two ebMS-compliant MSH nodes. This pair of MSH nodes provides a hop-to-hop model extended as required to support a multi-hop environment. The multi-hop environment allows the next destination of the message to be an intermediary MSH other than the 'receiving MSH' identified by the original sending MSH. The ebMS architecture assumes the sender of the message MAY be unaware of the specific path used to deliver a message. However, it MUST be assumed the original sender has knowledge of the final recipient of the message and the first of one or more intermediary hops.

The MSH supports the concept of 'quality of service.' The degree of service quality is controlled by an agreement existing between the parties directly involved in the message exchange. In practice, multiple agreements may be required between the two parties. The agreements might be tailored to the particular needs of the business exchanges. For instance, business partners may have a contract defining the message exchanges related to buying products from a domestic facility and another defining the message exchanges for buying from an overseas facility. Alternatively, the partners might agree to follow the agreements developed by their trade association. Multiple agreements may also exist between the various parties handling the message from the original sender to the final recipient. These agreements could include:

- an agreement between the MSH at the message origination site and the MSH at the final destination; and
- agreement between the MSH at the message origination site and the MSH acting as an intermediary; and
- an agreement between the MSH at the final destination and the MSH acting as an intermediary. There would, of course, be agreements between any additional intermediaries; however, the originating site MSH and final destination MSH MAY have no knowledge of these agreements.

An ebMS-compliant MSH shall respect the in-force agreements between itself and any other ebMS-compliant MSH with which it communicates. In broad terms, these agreements are expressed as Collaboration Protocol Agreements (CPA). This specification identifies the information that must be agreed. It does not specify the method or form used to create and maintain these agreements. It is assumed, in practice, the actual content of the contracts may be contained in initialization/configuration files, databases, or XML documents complying with the ebXML Collaboration Protocol Profile and Agreement Specification [ebCPP].

1.2.3 Operational Policies and Constraints

The ebMS is a service logically positioned between one or more business applications and a communications service. This requires the definition of an abstract service interface between the business applications and the MSH. This document acknowledges the interface, but does not provide a definition for the interface. Future versions of the ebMS MAY define the service interface structure.

Bindings to two communications protocols are defined in this document; however, the MSH is specified independent of any communications protocols. While early work focuses on HTTP for transport, no preference is being provided to this protocol. Other protocols may be used and future versions of the specification may provide details related to those protocols.

The ebMS relies on external configuration information. This information is determined either through defined business processes or trading partner agreements. These data are captured for use within a Collaboration Protocol Profile (CPP) or Collaboration Protocol Agreement (CPA). The ebXML Collaboration Protocol Profile and Agreement Specification [ebCPP] provides definitions for the information constituting the agreements. The ebXML architecture defines the relationship between this component of the infrastructure and the ebMS. As regards the MSH, the information composing a CPP/CPA must be available to support normal operation. However, the method used by a specific implementation of the MSH does not mandate the existence of a discrete instance of a CPA. The CPA is expressed as an XML document. Some implementations may elect to populate a database with the information from the CPA and then use the database. This specification does not prescribe how the CPA information is derived, stored, or used: it only states specific information items must be available for the MSH to achieve successful operations.

1.2.4 Modes of Operation

This specification does not mandate how the MSH will be installed within the overall ebXML framework. It is assumed some MSH implementations will not implement all functionality defined in this specification. For instance, a set of trading partners may not require reliable messaging services; therefore, no reliable messaging capabilities exist within their MSH. But, all MSH implementations shall comply with the specification with regard to the functions supported in the specific implementation and provide error notifications for functionality requested but not supported. Documentation for a MSH implementation SHALL identify all ebMS features not satisfied in the implementation.

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

Figure 1 depicts a logical arrangement of the functional modules existing 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 ebXML Header elements for the *ebXML Message* uses input from the application, passed through the Message Service Interface, information from the *Collaboration Protocol Agreement* governing the message, and generated information such as digital signature, timestamps and unique identifiers.

Header Parsing – extracting or transforming information from a received ebXML Header element into a form suitable for processing by the MSH implementation.

Security Services – digital signature creation and verification, encryption, 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. The service includes handling for persistence, retry, error notification and acknowledgment of messages requiring reliable delivery.

Message Packaging – the final enveloping of an *ebXML Message* (ebXML header elements and payload) into its SOAP Messages with Attachments [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 applications use to interact with the MSH to send and receive messages and which the MSH uses to interface with applications handling received messages (Delivery Module).

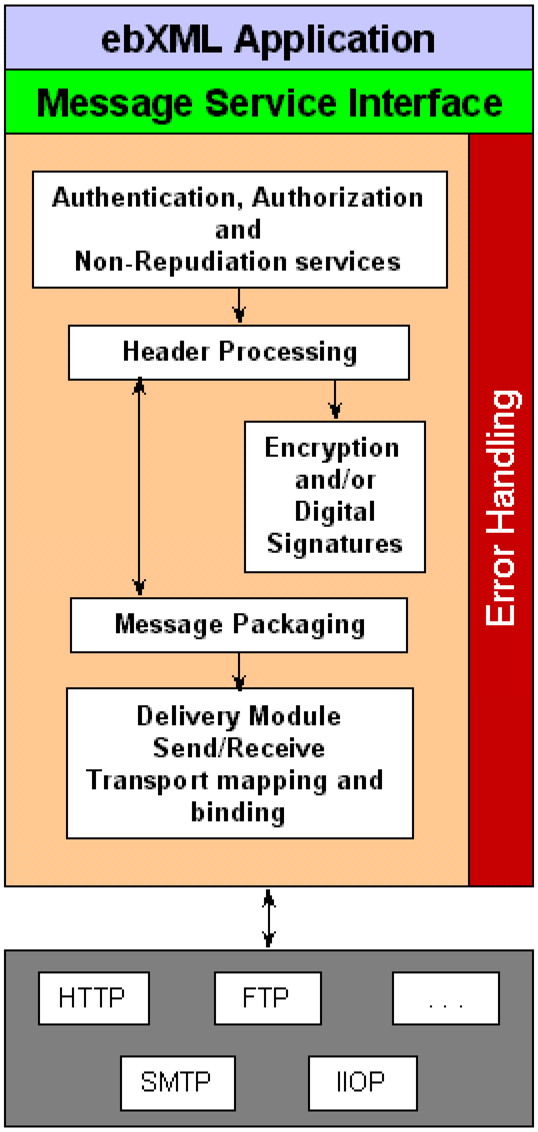
ebXML Application
Message Service Interface
Authentication, Authorization and Non-Repudiation services
Header Processing
Encryption and/or Digital Signatures
Message Packaging
Delivery Module Send/Receive Transport mapping and binding
Error Handling
HTTP
FTP
. . .
SMTP
IIOP

Figure 1.1 Typical Relationship between ebXML Message Service Handler Components

1.3 Minimal Requirements for Conformance

An implementation of this specification MUST satisfy ALL of the following conditions to be considered a conforming implementation:

- It supports all the mandatory syntax, features and behavior (as identified by the [RFC2119] key words MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT) defined in Part I – Core Functionality.
- It supports all the mandatory syntax, features and behavior defined for each of the additional module(s), defined in Part II – Additional Features, the implementation has chosen to implement.
- It complies with the following interpretation of the keywords OPTIONAL and MAY: When these keywords apply to the behavior of the implementation, the implementation is free to support these behaviors or not, as meant in [RFC2119]. When these keywords apply to message contents relevant to a module of features, a conforming implementation of such a module MUST be capable of processing these optional message contents according to the described ebXML semantics.
- If it has implemented optional syntax, features and/or behavior defined in this specification, it MUST be capable of interoperating with another implementation that has not implemented the optional syntax, features and/or behavior. It MUST be capable of processing the prescribed failure mechanism for those optional features it has chosen to implement.
- It is capable of interoperating with another implementation that has chosen to implement optional syntax, features and/or behavior, defined in this specification, it has chosen not to implement. Handling of unsupported features SHALL be implemented in accordance with the prescribed failure mechanism defined for the feature.

More details on Conformance to this specification – conformance levels or profiles and on their recommended implementation – are described in a companion document, "*Message Service Implementation Guidelines*" from the OASIS ebXML Implementation, Interoperability and Conformance (IIC) Technical Committee.

Part I. Core Functionality

2 ebXML with SOAP

The ebXML Message Service Specification defines a set of namespace-qualified SOAP ***Header*** and ***Body*** element extensions within the SOAP ***Envelope***. These are packaged within a MIME multipart to allow payloads or attachments to be included with the SOAP extension elements. In general, separate ebXML SOAP extension elements are used where:

- different software components may 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.

2.1 Packaging Specification

An ebXML Message is a communications 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*:

- The first 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 additional MIME parts, referred to as *Payload Containers*, containing application level payloads.

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

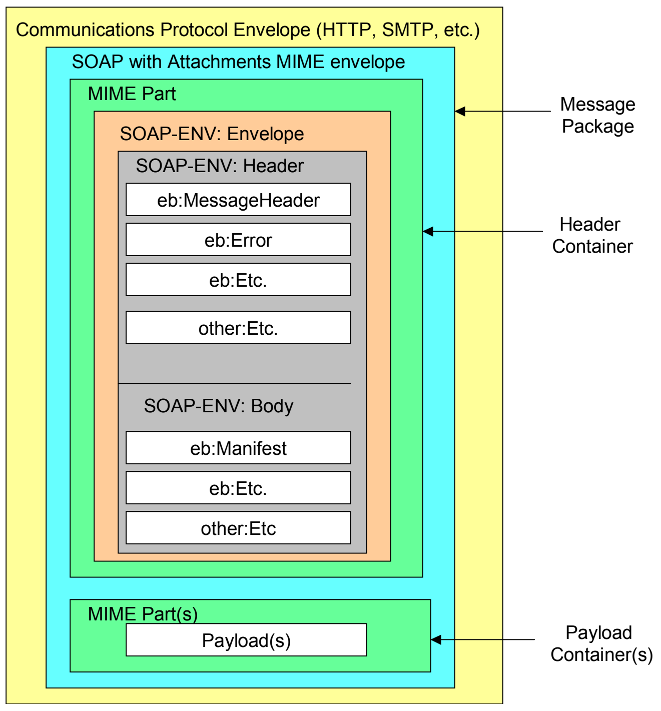

Figure 2.1 ebXML Message Structure

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

- 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.

2.1.1 SOAP Structural Conformance

The *ebXML Message* packaging complies 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 ebXML overrides existing semantics of SOAP, but rather the semantics of ebXML over SOAP maps directly onto SOAP semantics.

2.1.2 Message Package

All MIME header elements of the *Message Package* are in conformance with the SOAP Messages with Attachments [SOAPAttach] specification. In addition, the `Content-Type` MIME header in the *Message Package* contain a `type` attribute matching the MIME media type of the MIME body part containing the *SOAP Message* document. In accordance with the [SOAP] specification, the MIME media type of the *SOAP Message* has the value "`text/xml`".

It is strongly RECOMMENDED the initial headers contain a `Content-ID` MIME header structured in accordance with MIME [RFC2045], and in addition to the required parameters for the Multipart/Related media type, the `start` parameter (OPTIONAL in MIME Multipart/Related [RFC2387]) always be present. This permits more robust error detection. The following fragment is an example of the MIME headers for the multipart/related Message Package:

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

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

Implementations MUST support non-multipart messages, which may occur when there are no ebXML payloads. An ebXML message with no payload may be sent either as a plain SOAP message or as a [SOAPAttach] multipart message with only one body part.

2.1.3 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 consisting of one *SOAP Message* as defined in the SOAP Messages with Attachments [SOAPAttach] specification.

2.1.3.1 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"
```

2.1.3.2 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 XML [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 UTF-8 [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.

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

2.1.4 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 SHOULD 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 in the ebXML Message ***Manifest*** element within the SOAP ***Body*** (see section 3.2).

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

2.1.4.1 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>                            -------------|                 |
```

Note: It might be noticed the content-type used in the preceding example (application/XML) is different than the content-type in the example SOAP envelope in section 2.1.2 above (text/XML). The SOAP 1.1 specification states the content-type used for the SOAP envelope MUST be ‘text/xml’. However, many MIME experts disagree with the choice of the primary media type designation of 'text/*’ for XML documents as most XML is not "human readable" in the sense the MIME designation of ‘text’ was meant to infer. They believe XML documents should be classified as ‘application/XML’.

2.1.5 Additional MIME Parameters

Any MIME part described by this specification MAY contain additional MIME headers in conformance with the MIME [RFC2045] specification. Implementations MAY ignore any MIME header not defined in this specification. Implementations MUST ignore any MIME header 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.

2.1.6 Reporting MIME Errors

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

2.2 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 appearing in the XML prolog. For example:

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

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

2.2.1 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*.

2.2.2 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 SHALL be equivalent to the `charset` attribute of the MIME `Content-Type` of the *Header Container* (see section 2.1.3).

If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding used when creating the SOAP *Message*. It is RECOMMENDED 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 [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].

2.3 ebXML SOAP Envelope extensions

In conformance with the [SOAP] specification, all extension element content is namespace qualified. All of the ebXML SOAP extension element content defined in this specification is namespace qualified to the ebXML SOAP ***Envelope*** extensions namespace as defined in section 2.2.2.

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.

2.3.1 Namespace pseudo attribute

The namespace declaration for the ebXML SOAP ***Envelope*** extensions (***xmlns*** pseudo attribute) (see [XMLNS]) has a REQUIRED value of:

```
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
```

2.3.2 xsi:schemaLocation attribute

The SOAP namespace:

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

resolves to a schema conforming 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 ebXML SOAP extension element schema has been defined using the W3C Recommendation version 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 for the ebXML OASIS ebXML Messaging TC to adopt an equivalent, but updated version of the SOAP schema conforming to the W3C Recommendation version of the XML Schema specification [XMLSchema]. All 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 could prevent XML schema validation of received messages.

For example:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                      http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd ...">
```

In addition, ebXML SOAP ***Header*** and ***Body*** extension element content must be similarly qualified so as to identify the location where validating parsers can find the schema document containing 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.

The ***schemaLocation*** for the namespace described above in section 2.3.1 is:

```
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
```

Separate ***schemaLocation*** attribute are RECOMMENDED so tools, which 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:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                            http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
  <SOAP:Header
     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schemas/msg-header-2_0.xsd"
     xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
    <eb:MessageHeader ...>
       ...
    </eb:MessageHeader>
  </SOAP:Header>
  <SOAP:Body
     xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schemas/msg-header-2_0.xsd"
     xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schemas/msg-header-2_0.xsd">
    <eb:Manifest eb:version="2.0">
       ...
    </eb:Manifest>
  </SOAP:Body>
</SOAP:Envelope>
```

2.3.3 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/".

2.3.4 SOAP Body Element

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/".

2.3.5 ebXML SOAP Extensions

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

2.3.5.1 SOAP Header extensions:

- ***MessageHeader*** – a REQUIRED element containing routing information for the message (To/From, etc.) as well as other context information about the message.
- ***SyncReply*** – an element indicating the required transport state to the next SOAP node.

2.3.5.2 SOAP Body extension:

- ***Manifest*** – an element pointing to any data present either in the *Payload Container*(s) or elsewhere, e.g. on the web. This element MAY be omitted.

2.3.5.3 Core ebXML Modules:

- Error Handling Module
  - ***ErrorList*** – a SOAP Header element containing a list of the errors being reported against a previous message. The ***ErrorList*** element is only used if reporting an error or warning on a previous message. This element MAY be omitted.
- Security Module
  - ***Signature*** – an element that contains a digital signature that conforms to [XMLDSIG] that signs data associated with the message. This element MAY be omitted.

2.3.6 #wildcard Element Content

Some ebXML SOAP extension elements, as indicated in the schema, allow for foreign namespace-qualified element content to be added for extensibility. The extension element content MUST be namespace-qualified in accordance with XMLNS [XMLNS] and MUST belong to a foreign namespace. A foreign namespace is one that is NOT `http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd`. The wildcard elements are provided wherever extensions might be required for private extensions or future expansions to the protocol.

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

2.3.7 id attribute

Each of the ebXML SOAP extension elements defined in this specification 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.

2.3.8 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 SHOULD be "2.0". 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.

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.

2.3.9 SOAP mustUnderstand attribute

The REQUIRED SOAP ***mustUnderstand*** attribute on SOAP ***Header*** extensions, namespace qualified to the SOAP namespace (http://schemas.xmlsoap.org/soap/envelope/), indicates whether the contents of the element MUST be understood by a receiving process or else the message MUST be rejected in accordance with SOAP [SOAP]. This attribute with a value of '1' (***true***) indicates the element MUST be understood or rejected. This attribute with a value of '0' (***false***), the default, indicates the element may be ignored if not understood.

2.3.10 ebXML "Next MSH" actor URI

The urn:*oasis:names:tc:ebxml-msg:actor:nextMSH* when used in the context of the SOAP ***actor*** attribute value SHALL be interpreted to mean an entity that acts in the role of an instance of the ebXML MSH conforming to this specification.

This ***actor*** URI has been established to allow for the possibility that SOAP nodes that are NOT ebXML MSH nodes MAY participate in the message path of an *ebXML Message*. An example might be a SOAP node that digitally signs or encrypts a message.

All ebXML MSH nodes MUST act in this role.

2.3.11 ebXML "To Party MSH" actor URI

The urn:*oasis:names:tc:ebxml-msg:actor:toPartyMSH* when used in the context of the SOAP ***actor*** attribute value SHALL be interpreted to mean an instance of an ebXML MSH node, conforming to this specification, acting in the role of the Party identified in the ***MessageHeader/To/PartyId*** element of the same message. An ebXML MSH MAY be configured to act in this role. How this is done is outside the scope of this specification.

The MSH that is the ultimate destination of ebXML messages MUST act in the role of the *To Party MSH* actor URI in addition to acting in the default actor as defined by SOAP.

3 Core Extension Elements

3.1 MessageHeader Element

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

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

- an ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***mustUnderstand*** attribute with a value of '1' (see section 2.3.9 for details)
- ***From*** element
- ***To*** element
- ***CPAId*** element
- ***ConversationId*** element
- ***Service*** element
- ***Action*** element
- ***MessageData*** element
- ***DuplicateElimination*** element
- ***Description*** element

3.1.1 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 contains:

- ***PartyId*** elements – one or more
- ***Role*** element – zero or one.

If either the ***From*** or ***To*** elements contains multiple ***PartyId*** elements, all members of the list must identify the same organization. Unless a single ***type*** value refers to multiple identification systems, the value of any given ***type*** attribute MUST be unique within the list of ***PartyId*** elements contained within either the From or To element.

Note: This mechanism is particularly useful when transport of a message between the parties may involve multiple intermediaries. 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.

The ***From*** and ***To*** elements contain zero or one ***Role*** child element that, if present, SHALL immediately follow the last ***PartyId*** child element.

3.1.1.1 PartyId Element

The ***PartyId*** element has a single attribute, ***type*** and the content 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 4.1.5) with ***errorCode*** set to ***Inconsistent*** and ***severity*** set to ***Error***. It is strongly RECOMMENDED that the content of the ***PartyId*** element be a URI.

3.1.1.2 Role Element

The OPTIONAL ***Role*** element identifies the authorized role (***fromAuthorizedRole*** or ***toAuthorizedRole***) of the *Party* sending (when present as a child of the ***From*** element) and/or receiving (when present as a child of the ***To*** element) the message. The value of the ***Role*** element is a non-empty string, which is specified in the *CPA*.

Note: Role is better defined as a URI – e.g. http://rosettanet.org/roles/buyer.

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:Role>http://rosettanet.org/roles/Buyer</eb:Role>
    </eb:From>
    <eb:To>
      <eb:PartyId>mailto:joe@example.com</eb:PartyId>
      <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
    </eb:To>
```

3.1.2 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 mutually agreed by the two parties. This could be a concatenation of the ***From*** and ***To PartyId*** values, a URI 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*, the 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***.

3.1.3 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 context of the specified ***CPAId***. 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.

3.1.4 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 Business Process [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 ***urn:oasis:names:tc:ebxml-msg:service*** are reserved for use by this specification.

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

3.1.4.1 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 ***errorCode*** of ***Inconsistent*** and ***severity*** of ***Error*** (see section 4.1.5).

3.1.5 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. The value of the ***Action*** element is specified by the designer of the *service*. An example of the ***Action*** element follows:

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

3.1.6 MessageData Element

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

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

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

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

3.1.6.1 MessageId Element

The REQUIRED element ***MessageId*** is a globally unique identifier for each message conforming to MessageId [RFC2822]. The "local part" of the identifier as defined in MessageId [RFC2822] is implementation dependent.

Note: In the Message-Id and Content-Id MIME headers, values are always surrounded by angle brackets. However references in mid: or cid: scheme URI's and the MessageId and RefToMessageId elements MUST NOT include these delimiters.

3.1.6.2 Timestamp Element

The REQUIRED ***Timestamp*** is a value representing the time that the message header was created conforming to a dateTime [XMLSchema] and MUST be expressed as UTC. Indicating UTC in the ***Timestamp*** element by including the 'Z' identifier is optional.

3.1.6.3 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 4.1.5).

3.1.6.4 TimeToLive Element

If the ***TimeToLive*** element is present, it MUST be used to indicate the time, expressed as UTC, by which a message should be delivered to the *To Party MSH*. It MUST conform to an XML Schema dateTime.

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

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 with the ***errorCode*** attribute set to **TimeToLiveExpired** and the ***severity*** attribute set to **Error**.

The ***TimeToLive*** element is discussed further under Reliable Messaging in section 7.4.5.

3.1.7 DuplicateElimination Element

The ***DuplicateElimination*** element, if present, identifies a request by the sender for the receiving MSH to have a persistent store implemented (see section 7.4.1 for more details).

Valid values for ***DuplicateElimination***:

- ***DuplicateElimination*** present – this results in a delivery behavior of At-Most-Once.
- ***DuplicateElimination*** not present – this results in a delivery behavior of Best-Effort.

The ***DuplicateElimination*** element MUST NOT be present if there is a CPA with ***duplicateElimination*** set to ***never*** (see section 7.4.1 and section 7.6 for more details).

3.1.8 Description Element

The ***Description*** element may be present zero or more times. 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 [XML]. Each occurrence SHOULD have a different value for ***xml:lang***.

3.1.9 MessageHeader Sample

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

```
<eb:MessageHeader id="…" eb:version="2.0" SOAP:mustUnderstand="1">
  <eb:From><eb:PartyId>uri:example.com</eb:PartyId></eb:From>
  <eb:To>
      <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
      <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
  </eb:To>
  <eb:CPAId>http://www.oasis-open.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>UUID-2</eb:MessageId>
    <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp>
    <eb:RefToMessageId>UUID-1</eb:RefToMessageId>
  </eb:MessageData>
  <eb:DuplicateElimination/>
</eb:MessageHeader>
```

3.2 Manifest Element

The ***Manifest*** element MAY be present as a child of the SOAP ***Body*** element. The ***Manifest*** element is a composite element consisting of one or more ***Reference*** elements. Each ***Reference*** element identifies payload 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:

- 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:

- an ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- one or more ***Reference*** elements

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

3.2.1 Reference Element

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

- zero or more ***Schema*** elements – information about the schema(s) that define the instance document identified in the parent ***Reference*** element
- zero or more ***Description*** elements – a textual description of the payload object referenced by the parent ***Reference*** element

The ***Reference*** element itself is a simple link [XLINK]. 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 [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.

3.2.1.1 Schema Element

If the item being referenced has schema(s) of some kind that describe it (e.g. an XML Schema, DTD and/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

3.2.1.2 Description Element

See section 3.1.8 for more details. An example of a ***Description*** element follows.

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

3.2.2 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 corresponding *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, not a content id (URI scheme "`cid`"), and the URI cannot be resolved, it is an implementation decision whether to report the error. If the error is to be reported, it SHALL be reported to the *From Party* with an ***errorCode*** of ***MimeProblem*** and a ***severity*** of ***Error***.

Note: If a payload exists, which is not referenced by the ***Manifest***, that payload SHOULD be discarded.

3.2.3 Manifest Sample

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

```
<eb:Manifest eb:id="Manifest" eb:version="2.0">
  <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="2.0"/>
```

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

4 Core Modules

4.1 Security Module

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 Risk Assessment Technical Report [secRISK].

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. For this specification, a *Signed Message* is any message containing a ***Signature*** element. See Appendix C for a table of security profiles.

4.1.1 Signature Element

An ebXML Message MAY be digitally signed to provide security countermeasures. Zero or more ***Signature*** elements, belonging to the XML Signature [XMLDSIG] defined namespace, MAY be present as a child of the SOAP ***Header***. The ***Signature*** element MUST be namespace qualified in accordance with XML Signature [XMLDSIG]. The structure and content of the ***Signature*** element MUST conform to the XML Signature [XMLDSIG] specification. If there is more than one ***Signature*** element contained within the SOAP ***Header***, the first MUST represent the digital signature of the ebXML Message as signed by the *From Party MSH* in conformance with section 4.1. Additional ***Signature*** elements MAY be present, but their purpose is undefined by this specification.

Refer to section 4.1.3 for a detailed discussion on how to construct the ***Signature*** element when digitally signing an ebXML Message.

4.1.2 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 mechanisms, site (or physical) security procedures, cryptographic protocols, update implementations and apply fixes as appropriate. (See http://www.cert.org/ and http://ciac.llnl.gov/)

4.1.2.1 Collaboration Protocol Agreement

The configuration of Security for MSHs may be specified in the *CPA*. Two 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 Transport section addresses security applied to the entire ebXML Document, which includes the header and the payload.

4.1.3 Signature Generation

An ebXML Message is signed using [XMLDSIG] and following these steps:

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

The ***SignedInfo*** element SHALL have a ***CanonicalizationMethod*** element, a ***SignatureMethod*** element and one or more ***Reference*** elements, as defined in XML Signature [XMLDSIG].

The RECOMMENDED canonicalization method applied to the data to be signed is

```
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

described in [XMLC14N] for the *ebXML Message Service*. This algorithm excludes comments.

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

```
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
```

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

The [XMLDSIG] ***Reference*** element for the SOAP ***Envelope*** document SHALL have a URI attribute value of "" to provide for the signature to be applied to the document that contains the ***Signature*** element.

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

The [XMLDSIG] ***Reference*** element for the SOAP ***Envelope*** SHALL include a child ***Transforms*** element. The ***Transforms*** element SHALL include the following ***Transform*** child elements.

The first ***Transform*** element has an ***Algorithm*** attribute with a value of:

```
  <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
```

The result of this statement excludes the parent ***Signature*** element and all its descendants.

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

```
  <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
    <XPath> not(ancestor-or-self::()[@SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"] |
                ancestor-or-self::()[@SOAP:actor="http://schemas.xmlsoap.org/soap/actor/next"] )
    </XPath>
  <Transform/>
```

The result of this [XPath] statement excludes all elements within the SOAP ***Envelope*** which contain a SOAP:***actor*** attribute targeting the ***nextMSH***, and all their descendants. It also excludes all elements with ***actor*** attributes targeting the element at the next node (which may change en route). Any intermediate node or MSH MUST NOT change, format or in any way modify any element not targeted to the intermediary. Intermediate nodes MUST NOT add or delete white space. Any such change may invalidate the signature.

The last ***Transform*** element SHOULD have an ***Algorithm*** attribute with a value of:

```
    <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

The result of this algorithm is to canonicalize the SOAP ***Envelope*** XML and exclude comments.

Note: These transforms are intended for the SOAP Envelope and its contents. These transforms are NOT intended for the payload objects. The determination of appropriate transforms for each payload is left to the implementation.

Each payload object requiring signing SHALL be represented by a [XMLDSIG] ***Reference*** element that SHALL have a ***URI*** attribute resolving to the payload object. This can be either the `Content-Id` URI of the MIME body part of the payload object, or a URI matching the Content-Location of the MIME body part of the payload object, or a URI that resolves to a 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 the payload object.

Note: When a transfer encoding (e.g. base64) specified by a Content-Transfer-Encoding MIME header is used for the SOAP Envelope or payload objects, the signature generation MUST be executed before the encoding.

Example of digitally signed ebXML SOAP *Message*:

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                          http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
                          http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <SOAP:Header>
    <eb:MessageHeader eb:id="..." eb:version="2.0" SOAP:mustUnderstand="1">
          ...
    </eb:MessageHeader>
    <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
        <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
        <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
        <Reference URI="">
          <Transforms>
            <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
              <XPath> not(ancestor-or-self::()[@SOAP:actor=
                  &quot;urn:oasis:names:tc:ebxml-msg:actor:nextMSH&quot;]
                        | ancestor-or-self::()[@SOAP:actor=
                  &quot;http://schemas.xmlsoap.org/soap/actor/next&quot;])
              </XPath>
            </Transform>
            <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
          </Transforms>
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>...</DigestValue>
        </Reference>
        <Reference URI="cid://blahblahblah/">
          <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <DigestValue>...</DigestValue>
        </Reference>
      </SignedInfo>
      <SignatureValue>...</SignatureValue>
      <KeyInfo>...</KeyInfo>
    </Signature>
  </SOAP:Header>
  <SOAP:Body>
    <eb:Manifest eb:id="Mani01" eb:version="2.0">
      <eb:Reference xlink:href="cid://blahblahblah/" xlink:role="http://ebxml.org/gci/invoice">
        <eb:Schema eb:version="2.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
      </eb:Reference>
    </eb:Manifest>
  </SOAP:Body>
</SOAP:Envelope>
```

4.1.4 Countermeasure Technologies

4.1.4.1 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 Container(s) or data elsewhere on the web that relate to the message.

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 XML Signature [XMLDSIG].

4.1.4.2 Persistent Signed Receipt

An *ebXML Message* that has been digitally signed MAY be acknowledged with an *Acknowledgment Message* that itself is digitally signed in the manner described in the previous section. The *Acknowledgment Message* MUST contain a [XMLDSIG] ***Reference*** element list consistent with those contained in the [XMLDSIG] ***Signature*** element of the original message.

4.1.4.3 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, 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 TLS [RFC2246] or IPSEC [RFC2402] provides the sender of an *ebXML Message* with a way to authenticate the destination for the TCP/IP environment.

4.1.4.4 Non-persistent Integrity

A secure network protocol such as TLS [RFC2246] or IPSEC [RFC2402] MAY be configured to provide for integrity check CRCs of the packets transmitted via the network connection.

4.1.4.5 Persistent Confidentiality

XML Encryption is a W3C/IETF joint activity 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 for v1.0 of this specification 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 Payload Containers MAY be provided by functionality possessed by a MSH. Payload confidentiality MAY be provided by using XML Encryption (when available) or some other cryptographic process (such as S/MIME [S/MIME], [S/MIMEV3], or PGP MIME [PGP/MIME]) bilaterally agreed upon by the parties involved. The XML Encryption standard shall be the default encryption method when XML Encryption has achieved W3C Recommendation status.

Note: When both signature and encryption are required of the MSH, sign first and then encrypt.

4.1.4.6 Non-persistent Confidentiality

A secure network protocol, such as TLS [RFC2246] or IPSEC [RFC2402], provides transient confidentiality of a message as it is transferred between two ebXML adjacent MSH nodes.

4.1.4.7 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 Name Assertion and Entitlements, based on Security Assertion Markup Language [SAML]. Use of technology based on this anticipated specification may provide persistent authorization for an *ebXML Message* once it becomes available.

4.1.4.8 Non-persistent Authorization

A secure network protocol such as TLS [RFC2246] or IPSEC [RFC2402] 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 and to recognize the source as an authorized source of *ebXML Messages*.

4.1.4.9 Trusted Timestamp

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

4.1.5 Security Considerations

Implementors should take note, there is a vulnerability present even when an XML Digital Signature is used to protect to protect the integrity and origin of ebXML messages. The significance of the vulnerability necessarily depends on the deployed environment and the transport used to exchange ebXML messages.

The vulnerability is present because ebXML messaging is an integration of both XML and MIME technologies. Whenever two or more technologies are conjoined there are always additional (sometimes unique) security issues to be addressed. In this case, MIME is used as the framework for the message package, containing the SOAP ***Envelope*** and any payload containers. Various elements of the SOAP ***Envelope*** make reference to the payloads, identified via MIME mechanisms. In addition, various labels are duplicated in both the SOAP ***Envelope*** and the MIME framework, for example, the type of the content in the payload. The issue is how and when all of this information is used.

Specifically, the MIME Content-ID: header is used to specify a unique, identifying label for each payload. The label is used in the SOAP ***Envelope*** to identify the payload whenever it is needed. The MIME Content-Type: header is used to identify the type of content carried in the payload; some content types may contain additional parameters serving to further qualify the actual type. This information is available in the SOAP ***Envelope***.

The MIME headers are not protected, even when an XML-based digital signature is applied. Although XML Encryption is not currently available and thus not currently used, its application is developing similarly to XML digital signatures. Insofar as its application is the same as that of XML digital signatures, its use will not protect the MIME headers. Thus, an ebXML message may be at risk depending on how the information in the MIME headers is processed as compared to the information in the SOAP ***Envelope***.

The Content-ID: MIME header is critical. An adversary could easily mount a denial-of-service attack by mixing and matching payloads with the Content-ID: headers. As with most denial-of-service attacks, no specific protection is offered for this vulnerability. However, it should be detected since the digest calculated for the actual payload will not match the digest included in the SOAP ***Envelope*** when the digital signature is validated.

The presence of the content type in both the MIME headers and SOAP ***Envelope*** is a problem. Ordinary security practices discourage duplicating information in two places. When information is duplicated, ordinary security practices require the information in both places to be compared to ensure they are equal. It would be considered a security violation if both sets of information fail to match.

An adversary could change the MIME headers while a message is en route from its origin to its destination and this would not be detected when the security services are validated. This threat is less significant in a peer-to-peer transport environment as compared to a multi-hop transport environment. All implementations are at risk if the ebXML message is ever recorded in a long-term storage area since a compromise of that area puts the message at risk for modification.

The actual risk depends on how an implementation uses each of the duplicate sets of information. If any processing beyond the MIME parsing for body part identification and separation is dependent on the information in the MIME headers, then the implementation is at risk of being directed to take unintended or undesirable actions. How this might be exploited is best compared to the common programming mistake of permitting buffer overflows: it depends on the creativity and persistence of the adversary.

Thus, an implementation could reduce the risk by ensuring that the unprotected information in the MIME headers is never used except by the MIME parser for the minimum purpose of identifying and separating the body parts. This version of the specification makes no recommendation regarding whether or not an implementation should compare the duplicate sets of information nor what action to take based on the results of the comparison.

4.2 Error Handling Module

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 module 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 a SOAP ***Fault*** message if it is unable to process the message. A *Sending MSH* MUST be prepared to accept and process these SOAP ***Fault***s.

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 ***Fault***s.

An ebXML *SOAP Message* reporting an error with a ***highestSeverity*** of ***Warning*** SHALL NOT be reported or returned as a SOAP ***Fault***.

4.2.1.1 Definitions:

For clarity, two phrases are defined for use in this section:

- "message in error" – A *message* containing or causing an error or warning of some kind
- "message reporting the error" – A *message* containing an ebXML ***ErrorList*** element that describes the warning(s) and/or error(s) found in a message in error (also referred to as an *Error Message* elsewhere in this document).

4.2.2 Types of Errors

One MSH needs to report errors to another MSH. For example, errors associated with:

- ebXML namespace qualified content of the *SOAP Message* document (see section 2.3.1)
- reliable messaging failures (see section 7.5.7)
- security (see section 4.1)

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 or defined elsewhere.

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

4.2.3 ErrorList Element

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

The ***ErrorList*** element consists of:

- ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***mustUnderstand*** attribute with a value of '1' (see section 2.3.9 for details)
- ***highestSeverity*** attribute
- one or more ***Error*** elements

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

4.2.3.1 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, highestSeverity*** MUST be set to ***Error***, otherwise, ***highestSeverity*** MUST be set to ***Warning***.

4.2.3.2 Error Element

An ***Error*** element consists of:

- ***id*** attribute (see section 2.3.7 for details)
- ***codeContext*** attribute
- ***errorCode*** attribute
- ***severity*** attribute
- ***location*** attribute
- ***Description*** element

The content of the ***Description*** element MAY contain error message text.

4.2.3.2.1 id attribute

If the error is a part of an ebXML element, the ***id*** of the element MAY be provided for error tracking.

4.2.3.2.2 codeContext attribute

The ***codeContext*** attribute identifies the namespace or scheme for the ***errorCode***s. It MUST be a URI. Its default value is ***urn:oasis:names:tc:ebxml-msg:service:errors***. If it does not have the default value, then it indicates that an implementation of this specification has used its own ***errorCode***s.

Use of a ***codeContext*** attribute value other than the default is NOT RECOMMENDED. In addition, an implementation of this specification should not use its own ***errorCode***s if an existing ***errorCode*** as defined in this section has the same or very similar meaning.

4.2.3.2.3 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 the next section.

4.2.3.2.4 severity attribute

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

- ***Warning*** – This indicates other messages in the conversation could be generated in the normal way in spite of this problem.
- ***Error*** – This indicates there is an unrecoverable error in the message and no further messages will be generated as part of the conversation.

4.2.3.2.5 location attribute

The ***location*** attribute points to the part of the message containing the error.

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

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

4.2.3.2.6 Description Element

The content of the ***Description*** element provides a narrative description of the error in the language defined by the ***xml:lang*** attribute. The XML parser or other software validating the message typically generates the message. The content is defined by the vendor/developer of the software that generated the ***Error*** element. The content of the ***Description*** element can be empty. (See section 3.1.8)

4.2.3.3 ErrorList Sample

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

```
<eb:ErrorList id="3490sdo9", eb:highestSeverity="error" eb:version="2.0" SOAP:mustUnderstand="1">
  <eb:Error eb:errorCode="SecurityFailure" eb:severity="Error" eb:location="URI_of_ds:Signature">
    <eb:Description xml:lang="en-US">Validation of signature failed<eb:Description>
  </eb:Error>
  <eb:Error ...> ... </eb:Error>
</eb:ErrorList>
```

4.2.3.4 errorCode values

This section describes the values for the ***errorCode*** attribute 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***. 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.

4.2.3.4.1 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, a module is present 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.

4.2.3.4.2 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.
MimeProblem	URI resolve error	If an xlink:href attribute contains a URI, not a content id (URI scheme "cid"), and the URI cannot be resolved, then it is an implementation decision whether to report the error.
Unknown	Unknown Error	Indicates that an error has occurred 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.

4.2.4 Implementing Error Reporting and Handling

4.2.4.1 When to Generate Error Messages

When a MSH detects an error in a message it is strongly RECOMMENDED the error is reported to the MSH that sent the message in error. This is possible when:

- the Error Reporting Location (see section 4.2.4.2) to which the message reporting the error should be sent can be determined
- 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:

- the error is logged
- the problem is resolved by other means
- no further action is taken.

4.2.4.1.1 Security Considerations

Parties receiving 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 the message received to be unauthorized or part of some security attack. The decision process resulting in this course of action is implementation dependent.

4.2.4.2 Identifying the Error Reporting Location

The Error Reporting Location is a URI 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. Otherwise, the recipient MAY resolve an ***ErrorURI*** using the ***From*** element of the message in error. If neither is possible, no error will be reported to the sending *Party*.

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

4.2.4.3 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*** 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: **urn:oasis:names:tc:ebxml-msg:service***
- The ***Action*** element MUST be set to ***MessageError***.

5 SyncReply Module

It may be necessary for the sender of a message, using a synchronous communications protocol, such as HTTP, to receive the associated response message over the same connection the request message was delivered. In the case of HTTP, the sender of the HTTP request message containing an ebXML message needs to have the response ebXML message delivered to it on the same HTTP connection.

If there are intermediary nodes (either ebXML MSH nodes or possibly other SOAP nodes) involved in the message path, it is necessary to provide some means by which the sender of a message can indicate it is expecting a response so the intermediary nodes can keep the connection open.

The ***SyncReply*** ebXML SOAP extension element is provided for this purpose.

5.1 SyncReply Element

The SyncReply element MAY be present as a direct child descendant of the SOAP Header element. It consists of:

- an ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***actor*** attribute with the REQUIRED value of "***http://schemas.xmlsoap.org/soap/actor/next***"
- a SOAP ***mustUnderstand*** attribute with a value of ***'true'*** (see section 2.3.9 for details)

If present, this element indicates to the receiving SOAP or ebXML MSH node the connection over which the message was received SHOULD be kept open in expectation of a response message to be returned via the same connection.

This element MUST NOT be used to override the value of ***syncReplyMode*** in the CPA. If the value of ***syncReplyMode*** is ***none*** and a ***SyncReply*** element is present, the *Receiving MSH* should issue an error with ***errorCode*** of ***Inconsistent*** and a ***severity*** of ***Error*** (see section 4.1.5).

An example of a ***SyncReply*** element:

```
<eb:SyncReply eb:id="3833kkj9", eb:version="2.0" SOAP:mustUnderstand="true"
              SOAP:actor="http://schemas.xmlsoap.org/soap/actor/next">
```

6 Combining ebXML SOAP Extension Elements

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

6.1.1 MessageHeader Element Interaction

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

6.1.2 Manifest Element Interaction

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

6.1.3 Signature Element Interaction

One or more XML Signature [XMLDSIG] ***Signature*** elements MAY be present on any message.

6.1.4 ErrorList Element Interaction

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

6.1.5 SyncReply Element Interaction

The ***SyncReply*** element MAY be present on any outbound message sent using synchronous communication protocol.

Part II. Additional Features

7 Reliable Messaging Module

Reliable Messaging defines an interoperable protocol such that two Message Service Handlers (MSH) can reliably exchange messages, using acknowledgment, retry and duplicate detection and elimination mechanisms, resulting in the *To Party* receiving the message Once-And-Only-Once. The protocol is flexible, allowing for both store-and-forward and end-to-end reliable messaging.

Reliability is achieved by a *Receiving MSH* responding to a message with an *Acknowledgment Message*. An *Acknowledgment Message* is any ebXML message containing an ***Acknowledgment*** element. Failure to receive an *Acknowledgment Message* by a *Sending MSH* MAY trigger successive retries until such time as an *Acknowledgment Message* is received or the predetermined number of retries has been exceeded at which time the *From Party* SHOULD be notified of the probable delivery failure.

Whenever an identical message may be received more than once, some method of duplicate detection and elimination is indicated, usually through the mechanism of a *persistent store*.

7.1 Persistent Storage and System Failure

A MSH that supports Reliable Messaging MUST keep messages 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 different degrees of resilience may be realized depending upon the technology used to store the data. However, at a minimum, persistent storage with the resilience characteristics of a hard disk (or equivalent) SHOULD be used. It is strongly RECOMMENDED that implementers of this specification use technology 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 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 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 needing to process it,
- the time the message was received, so the information can be used to generate the response to a *Message Status Request* (see section 8.1.1),
- the complete response message.

7.2 Methods of Implementing Reliable Messaging

Support for Reliable Messaging is implemented in one of the following ways:

- using the ebXML Reliable Messaging protocol,
- using ebXML SOAP structures together with commercial software products that are designed to provide reliable delivery of messages using alternative protocols,
- user application support for some features, especially duplicate elimination, or
- some mixture of the above options on a per-feature basis.

7.3 Reliable Messaging SOAP Header Extensions

7.3.1 AckRequested Element

The ***AckRequested*** element is an OPTIONAL extension to the SOAP ***Header*** used by the *Sending MSH* to request a *Receiving MSH,* acting in the role of the actor URI identified in the SOAP ***actor*** attribute, returns an *Acknowledgment Message*.

The ***AckRequested*** element contains the following:

- a ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***mustUnderstand*** attribute with a value of ***'true'*** (see section 2.3.9 for details)
- a SOAP ***actor*** attribute
- a ***signed*** attribute

This element is used to indicate to a *Receiving MSH,* acting in the role identified by the SOAP ***actor*** attribute, whether an *Acknowledgment Message* is expected, and if so, whether the message should be signed by the *Receiving MSH*.

An *ebXML Message* MAY have zero, one, or two instances of an ***AckRequested*** element. A single MSH node SHOULD only insert one ***AckRequested*** element. If there are two ***AckRequested*** elements present, they MUST have different values for their respective SOAP ***actor*** attributes. At most one ***AckRequested*** element can be targeted at the ***actor*** URI meaning *Next MSH* (see section 2.3.10) and at most one ***AckRequested*** element can be targeted at the ***actor*** URI meaning *To Party MSH* (see section 2.3.11) for any given message.

7.3.1.1 SOAP actor attribute

The ***AckRequested*** element MUST be targeted at either the Next MSH or the *To Party MSH* (these are equivalent for single-hop routing). This is accomplished by including a SOAP ***actor*** with a URN value with one of the two ebXML ***actor*** URNs defined in sections 2.3.10 and 2.3.11 or by leaving this attribute out. The default ***actor*** targets the *To Party MSH.*

7.3.1.2 signed attribute

The REQUIRED ***signed*** attribute is used by a *From Party* to indicate whether or not a message received by the *To Party MSH* should result in the *To Party* returning a signed *Acknowledgment Message* – containing a [XMLDSIG] ***Signature*** element as described in section 4.1. Valid values for ***signed*** are:

- ***true*** - a signed *Acknowledgment Message* is requested, or
- ***false*** - an unsigned *Acknowledgment Message* is requested.

Before setting the value of the ***signed*** attribute in ***AckRequested***, the *Sending MSH* SHOULD check if the *Receiving MSH* supports *Acknowledgment Messages* of the type requested (see also [ebCPP]).

When a *Receiving MSH* receives a message with ***signed*** attribute set to ***true*** or ***false*** then it should verify it is able to support the type of *Acknowledgment Message* requested.

- If the *Receiving MSH* can produce the *Acknowledgment Message* of the type requested, then it MUST return to the Sending *MSH* a message containing an ***Acknowledgment*** element.
- If the *Receiving MSH* cannot return an *Acknowledgment Message* as requested it MUST report the error to the *Sending MSH* using an ***errorCode*** of ***Inconsistent*** and a ***severity*** of either ***Error*** if inconsistent with the CPA, or ***Warning*** if not supported..

7.3.1.3 AckRequested Sample

In the following example, an *Acknowledgment Message* is requested of a MSH node acting in the role of the *To Party* (see section 2.3.11). The ***Acknowledgment*** element generated MUST be targeted to the ebXML MSH node acting in the role of the *From Party* along the reverse message path (end-to-end acknowledgment).

```
<eb:AckRequested SOAP:mustUnderstand="true" eb:version="2.0" eb:signed="false">
```

7.3.1.4 AckRequested Element Interaction

An ***AckRequested*** element MUST NOT be included on a message with only an ***Acknowledgment*** element (no payload). This restriction is imposed to avoid endless loops of *Acknowledgement Messages.* An *Error Message* MUST NOT contain an ***AckReques*** element.

7.3.2 Acknowledgment Element

The ***Acknowledgment*** element is an OPTIONAL extension to the SOAP ***Header*** used by one Message Service Handler to indicate to another Message Service Handler that it has received a message. The ***RefToMessageId*** element in an ***Acknowledgment*** element is used to identify the message being acknowledged by its ***MessageId****.*

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

- an ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***mustUnderstand*** attribute with a value of '1' (see section 2.3.9 for details)
- a SOAP ***actor*** attribute
- a ***Timestamp*** element
- a ***RefToMessageId*** element
- a ***From*** element
- zero or more [XMLDSIG] ***Reference*** element(s)

7.3.2.1 SOAP actor attribute

The SOAP ***actor*** attribute of the ***Acknowledgment*** element SHALL have a value corresponding to the ***AckRequested*** element of the message being acknowledged. If there is no SOAP ***actor*** attribute present on an ***Acknowledgment*** element, the default target is the *To Party MSH* (see section for 11.1.3).

7.3.2.2 Timestamp Element

The REQUIRED ***Timestamp*** element is a value representing the time that the message being acknowledged was received by the *MSH* generating the acknowledgment message. It must conform to a dateTime [XMLSchema] and is expressed as UTC (section 3.1.6.2).

7.3.2.3 RefToMessageId Element

The REQUIRED ***RefToMessageId*** element contains the ***MessageId*** of the message whose delivery is being reported.

7.3.2.4 From Element

This is the same element as the ***From*** element within ***MessageHeader*** element (see section 3.1.1). However, when used in the context of an ***Acknowledgment*** element, it contains the identifier of the *Party* generating the A*cknowledgment Message*.

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

7.3.2.5 [XMLDSIG] Reference Element

An *Acknowledgment Message* MAY be used to enable non-repudiation of receipt by a MSH by including one or more ***Reference*** elements, from the XML Signature [XMLDSIG] namespace, derived from the *message being acknowledged* (see section 4.1.3 for details). The ***Reference*** element(s) MUST be namespace qualified to the aforementioned namespace and MUST conform to the XML Signature [XMLDSIG] specification. If the *message being acknowledged* contains an ***AckRequested*** element with a ***signed*** attribute set to ***true***, then the [XMLDSIG] ***Reference*** list is REQUIRED.

Receipt of an *Acknowledgment Message*, indicates the original message reached its destination. Receipt of a signed *Acknowledgment Message* validates the sender of the *Acknowledgment Message*. However, a signed *Acknowledgment Message* does not indicate whether the message arrived intact. Including a digest (see [XMLDSIG] section 4.3.3) of the original message in the *Acknowledgment Message* indicates to the original sender what was received by the recipient of the message being acknowledged. The digest contained in the *Acknowledgment Message* may be compared to a digest of the original message. If the digests match, the message arrived intact. Such a digest already exists in the original message, if it is signed, contained within the [XMLDSIG] ***Signature*** / ***Reference*** element(s).

If the original message is signed, the [XMLDSIG] ***Signature*** / ***Reference*** element(s) of the original message will be identical to the ***Acknowledgment*** / [XMLDSIG] ***Reference*** element(s) in the *Acknowledgment Message*. If the original message is not signed, the [XMLDSIG] ***Reference*** element must be derived from the original message (see section 4.1.3).

Upon receipt of an end-to-end *Acknowledgment Message*, the *From Party MSH* MAY notify the application of successful delivery for the referenced message. The MSH SHOULD ignore subsequent *Error* or *Acknowledgment Message*s with the same ***RefToMessageId*** value.

7.3.2.6 Acknowledgment Sample

An example ***Acknowledgment*** element targeted at the *To Party MSH*:

```
<eb:Acknowledgment SOAP:mustUnderstand="1" eb:version="2.0">
  <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
  <eb:RefToMessageId>323210:e52151ec74:7ffc@xtacy</eb:RefToMessageId>
  <eb:From> <eb:PartyId>uri:www.example.com</eb:PartyId> </eb:From>
</eb:Acknowledgment>
```

7.3.2.7 Sending an Acknowledgment Message by Itself

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

- the ***Service*** element MUST be set to ***urn:oasis:names:tc:ebxml-msg:service***
- the ***Action*** element MUST be set to ***Acknowledgment***

7.3.2.8 Acknowledgment Element Interaction

An ***Acknowledgment*** element MAY be present on any message, except as noted in section 7.3.1.4. An *Acknowledgment Message* MUST NOT be returned for an *Error Message*.

7.4 Reliable Messaging Parameters

This section describes the parameters required to control reliable messaging. Many of these parameters can be obtained from a CPA.

7.4.1 DuplicateElimination

The ***DuplicateElimination*** element MUST be used by the *From Party MSH* to indicate whether the *Receiving MSH* MUST eliminate duplicates (see section 7.6 for Reliable Messaging behaviors). If the value of ***duplicateElimination*** in the CPA is ***never***, ***DuplicateElimination*** MUST NOT be present.

- If ***DuplicateElimination*** is present – The *To Party MSH* must persist messages in a persistent store so duplicate messages will be presented to the *To Party* Application At-Most-Once, or
- If ***DuplicateElimination*** is not present – The *To Party MSH* is not required to maintain the message in persistent store and is not required to check for duplicates.

If ***DuplicateElimination*** is present, the *To Party MSH* must adopt a reliable messaging behavior (see section 7.6) causing duplicate messages to be ignored.

If ***DuplicateElimination*** is not present, a *Receiving MSH* is not required to check for duplicate message delivery. Duplicate messages might be delivered to an application and persistent storage of messages is not required – although elimination of duplicates is still allowed.

If the *To Party* is unable to support the requested functionality, or if the value of ***duplicateElimination*** in the CPA does not match the implied value of the element, the *To Party* SHOULD report the error to the *From Party* using an ***errorCode*** of ***Inconsistent*** and a ***Severity*** of ***Error***.

7.4.2 AckRequested

The ***AckRequested*** parameter is used by the *Sending MSH* to request a *Receiving MSH,* acting in the role of the actor URI identified in the SOAP ***actor*** attribute, return an *Acknowledgment Message* containing an ***Acknowledgment*** element (see section 7.3.1).

7.4.3 Retries

The ***Retries*** parameter, from a CPA, is an integer value specifying the maximum number of times a *Sending MSH* SHOULD attempt to redeliver an unacknowledged *message* using the same communications protocol.

7.4.4 RetryInterval

The ***RetryInterval*** parameter, from a CPA, is a time value, expressed as a duration in accordance with the ***duration*** [XMLSchema] data type. This value specifies the minimum time a *Sending MSH* SHOULD wait between ***Retries***, if an *Acknowledgment Message* is not received or if a communications error was detected during an attempt to send the message. ***RetryInterval*** applies to the time between sending of the original message and the first retry as well as the time between retries.

7.4.5 TimeToLive

TimeToLive is defined in section 3.1.6.4.

For a reliably delivered message, ***TimeToLive*** MUST conform to:

TimeToLive > ***Timestamp*** + ((***Retries + 1***) * ***RetryInterval***).

where ***TimeStamp*** comes from ***MessageData***.

7.4.6 PersistDuration

The ***PersistDuration*** parameter, from a CPA, is the minimum length of time, expressed as a ***duration*** [XMLSchema], 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 7.5.7).

TimeStamp for a reliably sent message (found in the message header), plus its ***PersistDuration*** (found in the CPA), must be greater than its ***TimeToLive*** (found in the message header).

7.4.7 syncReplyMode

The ***syncReplyMode*** parameter from the CPA is used only if the data communications protocol is *synchronous* (e.g. HTTP). If the communications protocol is not *synchronous*, then the value of ***syncReplyMode*** is ignored. If the ***syncReplyMode*** attribute is not present, it is semantically equivalent to its presence with a value of ***none***. If the ***syncReplyMode*** parameter is not ***none***, a ***SyncReply*** element MUST be present and the MSH must return any response from the application or business process in the payload of the *synchronous* reply message, as appropriate. See also the description of ***syncReplyMode*** in the CPPA [ebCPP] specification.

If the value of ***syncReplyMode*** is ***none*** and a ***SyncReply*** element is present, the Receiving MSH should issue an error with ***errorCode*** of ***Inconsistent*** and a ***severity*** of ***Error*** (see section 4.1.5).

7.5 ebXML Reliable Messaging Protocol

The ebXML Reliable Messaging Protocol is illustrated by the following figure.

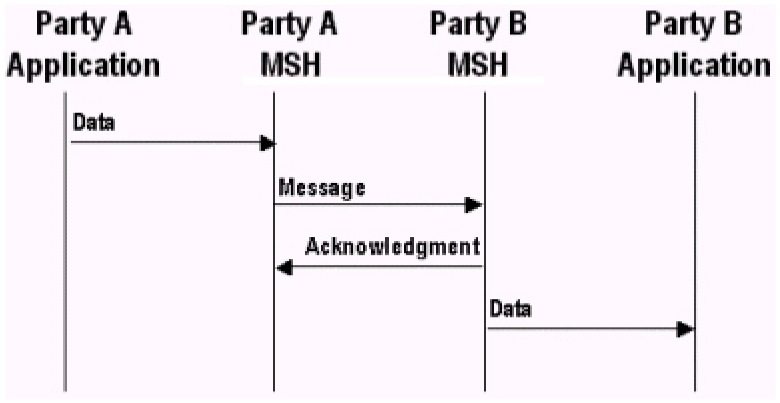

Figure 7-1 Indicating a message has been received

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

An *Acknowledgment Message* MUST contain an ***Acknowledgment*** element as described in section 7.3.1 with a ***RefToMessageId*** containing the same value as the ***MessageId*** element in the *message being acknowledged*.

7.5.1 Sending Message Behavior

If a MSH is given data by an application needing to be sent reliably, the MSH MUST do the following:

1. Create a message from components received from the application.
2. Insert an ***AckRequested*** element as defined in section 7.3.1.
3. Save the message in *persistent storage* (see section 7.1).
4. Send the message to the *Receiving MSH*.
5. Wait for the return of an *Acknowledgment Message* acknowledging receipt of this specific message and, if it does not arrive before ***RetryInterval*** has elapsed, or if a communications protocol error is encountered, then take the appropriate action as described in section 7.5.4.

7.5.2 Receiving Message Behavior

If this is an *Acknowledgment Message* as defined in section 7 then:

1 Look for a message in *persistent storage* with a ***MessageId*** 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.

If the *Receiving MSH* is NOT the *To Party MSH* (as defined in section 2.3.10 and 2.3.11), then see section 11.1.3 for the behavior of the ***AckRequested*** element.

If an ***AckRequested*** element is present (not an *Acknowledgment Message*) then:

1 If the message is a duplicate (i.e. there is a ***MessageId*** held in persistent storage containing the same value as the ***MessageId*** in the received message), generate an *Acknowledgment Message* (see section 7.5.3). Follow the procedure in section 7.5.5 for resending lost *Acknowledgment*

Messages. The *Receiving MSH* MUST NOT deliver the message to the application interface.

Note: The check for duplicates is only performed when ***DuplicateElimination*** is present.

2 If the message is not a duplicate or (there is no ***MessageId*** held in persistent storage corresponding to the ***MessageId*** in the received message) then:

a If there is a ***DuplicateElimination*** element, save the ***MessageId*** of the received message in persistent storage. As an implementation decision, the whole message MAY be stored.

b Generate an *Acknowledgment Message* in response (this may be as part of another message). The *Receiving MSH* MUST NOT send an *Acknowledgment Message* until the message has been safely stored in *persistent storage* or delivered to the application interface. Delivery of an *Acknowledgment Message* constitutes an obligation by the *Receiving MSH* to deliver the message to the application or forward to the next MSH in the message path as appropriate.

If there is no ***AckRequested*** element then do the following:

1 If there is a ***DuplicateElimination*** element, and the message is a duplicate, then do nothing.

2 Otherwise, deliver the message to the application interface

If the *Receiving MSH* node is operating as an intermediary along the message's message path, then it MAY use store-and-forward behavior. However, it MUST NOT filter out perceived duplicate messages from their normal processing at that node.

If an *Acknowledgment Message* is received unexpectedly, it should be ignored. No error should be sent.

7.5.3 Generating an Acknowledgment Message

An *Acknowledgment Message* MUST be generated whenever a message is received with an ***AckRequested*** element having a SOAP ***actor*** URI targeting the *Receiving MSH* node.

As a minimum, it MUST contain an ***Acknowledgment*** element with a ***RefToMessageId*** containing the same value as the ***MessageId*** element in the message being acknowledged. This message MUST be placed in persistent storage with the same ***PersistDuration*** as the original message.

The *Acknowledgment Message* can be sent at the same time as the response to the received message. In this case, the values for the ***MessageHeader*** elements of the *Acknowledgment Message* are determined by the ***Service*** and ***Action*** associated with the business response.

If an *Acknowledgment Message* 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: ***urn:oasis:names:tc:ebxml-msg:service***
- The ***Action*** element MUST be set to ***Acknowledgment***.
- The ***From*** element MAY be populated with the ***To*** element extracted from the message received and all child elements from the ***To*** element received SHOULD be included in this ***From*** element.
- The ***To*** element MAY be populated with the ***From*** element extracted from the message received and all child elements from the ***From*** element received SHOULD be included in this ***To*** element.
- The ***RefToMessageId*** element MUST be set to the ***MessageId*** of the message received.

7.5.4 Resending Lost Application Messages

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

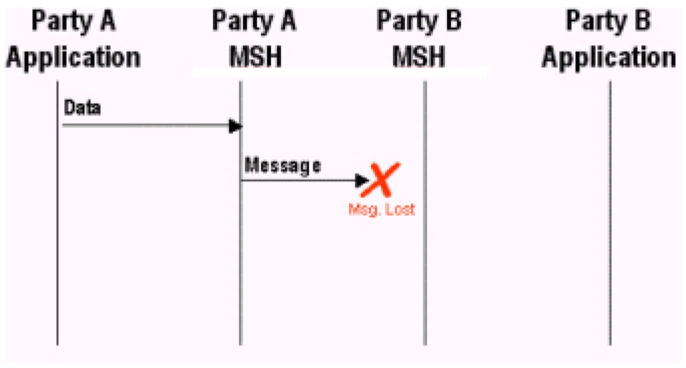

Figure 7-1 Undelivered Message

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

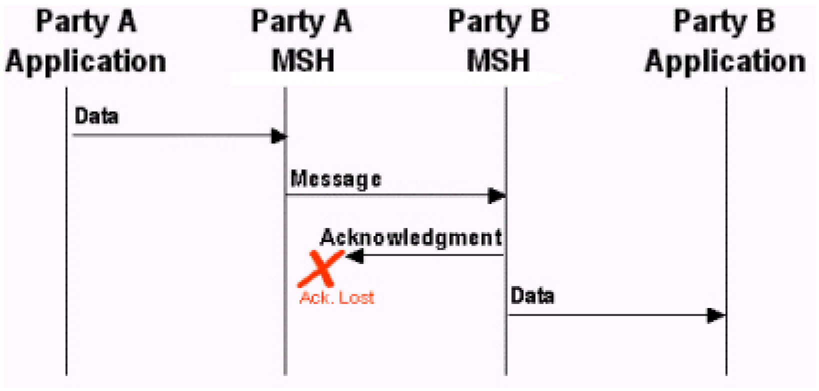

Figure 7-2 Lost Acknowledgment Message

Note: *Acknowledgment Messages* are never acknowledged.

The rules applying to the non-receipt of an anticipated Acknowledgment due to the loss of either the application message or the *Acknowledgment Message* are as follows:

- The *Sending MSH* MUST resend the original message if an *Acknowledgment Message* has been requested but has not been received and the following are both true:
  - At least the time specified in the ***RetryInterval*** parameter has passed since the message was last sent,
  - The message has been resent less than the number of times specified in the ***Retries*** parameter.
- If the *Sending MSH* does not receive an *Acknowledgment Message* after the maximum number of retries, the *Sending MSH* SHALL notify the application and/or system administrator function of the failure to receive an *Acknowledgment Message*.
- If the *Sending MSH* detects a communications protocol error, the *Sending MSH* MUST resend the message using the same algorithm as if it has not received an *Acknowledgment Message*.

7.5.5 Resending Acknowledgments

If the *Receiving MSH* receives a message it discovers to be a duplicate, it should resend the original *Acknowledgment Message* if the message is stored in *persistent store*. In this case, do the following:

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

If a response message was found in *persistent storage* then resend the persisted message back to the MSH that sent the received message. If no response message was found in *persistent storage,* then:

(1) If ***syncReplyMode*** is not set to ***none*** and if the CPA indicates an application response is included, then it must be the case that the application has not finished processing the earlier

copy of the same message. Therefore, wait for the response from the application and then return that response synchronously over the same connection that was used for the retransmission.

(2) Otherwise, generate an *Acknowledgment Message*.

7.5.6 Duplicate Message Handling

In the context of this specification:

- an "identical message" – a *message* containing the same ebXML SOAP ***Header***, ***Body*** and ebXML Payload Container(s) as the earlier sent *message*.
- a "duplicate message" – a *message* containing the same ***MessageId*** as a previously received message.
- the "first response message" – the message with the earliest ***Timestamp*** in the ***MessageData*** element having the same ***RefToMessageId*** as the duplicate message.

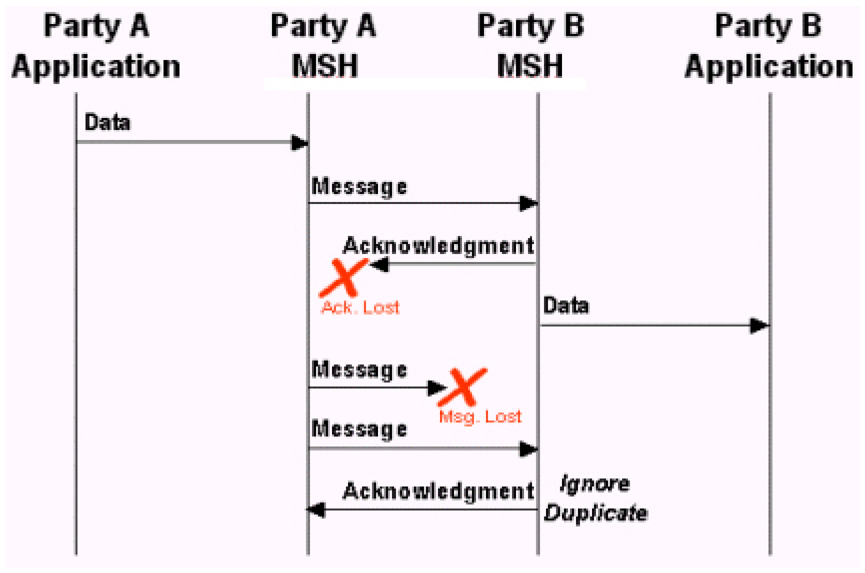

Figure 7-1 Resending Unacknowledged Messages

The diagram above shows the behavior to be followed by the *Sending* and *Receiving MSH* for messages sent with an ***AckRequested*** element and a ***DuplicateElimination*** element. Specifically:

1) The sender of the *message* (e.g. Party A MSH) MUST resend the "identical message" if no *Acknowledgment Message* is received.
2) When the recipient (Party B MSH) of the *message* receives a "duplicate message", it MUST resend to the sender (Party A MSH) an *Acknowledgment Message* identical to the *first response message* sent to the sender Party A MSH).
3) The recipient of the *message* (Party B MSH) MUST NOT forward the message a second time to the application/process.

7.5.7 Failed Message Delivery

If a message sent with an ***AckRequested*** element cannot be delivered, the MSH or process handling the message (as in the case of a routing intermediary) SHALL send a delivery failure notification to the *From Party.* The delivery failure notification message is an *Error Message* with ***errorCode*** of ***DeliveryFailure*** and 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.

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

Note: If the *From Party MSH* receives an *Acknowledgment Message* from the *To Party MSH*, it should ignore all other ***DeliveryFailure*** or *Acknowledgment Messages*.

7.6 Reliable Messaging Combinations

	Duplicate-Elimination[§]	*AckRequested ToPartyMSH*	*AckRequested NextMSH*	*Comment*
1	Y	Y	Y	**Once-And-Only-Once** Reliable Messaging at the End-To-End and At-Least-Once to the Intermediate. Intermediate and To Party can issue Delivery Failure Notifications if they cannot deliver.
2	Y	Y	N	**Once-And-Only-Once** Reliable Message at the End-To-End level only based upon end-to-end retransmission
3	Y	N	Y	**At-Least-Once Reliable** Messaging at the Intermediate Level – Once-And-Only-Once end-to-end if all Intermediates are Reliable. No End-to-End notification.
4	Y	N	N	**At-Most-Once** Duplicate Elimination only at the To Party No retries at the Intermediate or the End.
5	N	Y	Y	**At-Least-Once** Reliable Messaging with duplicates possible at the Intermediate and the To Party.
6	N	Y	N	**At-Least-Once** Reliable Messaging duplicates possible at the Intermediate and the To Party.
7	N	N	Y	**At-Least-Once** Reliable Messaging to the Intermediate and at the End. No End-to-End notification.
8	N	N	N	**Best Effort**

[§]*Duplicate Elimination is only performed at the To Party MSH,* not at the Intermediate Level.

8 Message Status 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 and the ***MessageId*** in the ***RefToMessageId*** is present in *persistent storage* (see section 7.1).

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.

If a *Receiving MSH* does not support the service requested, it SHOULD return an *Error Message* with an ***errorCode*** of ***NotSupported*** and a ***highestSeverity*** attribute set to ***Error***. Each service is described below.

8.1 Message Status Messages

8.1.1 Message Status Request Message

A Message Status Request message consists of an *ebXML Message* with no ebXML Payload Container and the following:

- a ***MessageHeader*** element containing:
  - a ***From*** element identifying the *Party* that created the Message Status Request message
  - a ***To*** element identifying a *Party* who should receive the message.
  - a ***Service*** element that contains: ***urn:oasis:names:tc:ebxml-msg:service***
  - an ***Action*** element that contains ***StatusRequest***
  - a ***MessageData*** element
- a ***StatusRequest*** element containing:
  - a ***RefToMessageId*** element in ***StatusRequest*** element containing the ***MessageId*** of the message whose status is being queried.
- an OPTIONAL [XMLDSIG] ***Signature*** element (see section 4.1 for more details)

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

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 with no ebXML Payload Container consisting of the following:

- a ***MessageHeader*** element containing:
  - a ***From*** element that identifies the sender of the Message Status Response message
  - a ***To*** element set to the value of the ***From*** element in the Message Status Request message
  - a ***Service*** element that contains ***uri:www.oasis-open.org/messageService/***
  - an ***Action*** element that contains ***StatusResponse***
  - a ***MessageData*** element containing:
    - a ***RefToMessageId*** that identifies the Message Status Request message.
- ***StatusResponse*** element (see section 8.2.3)
- an OPTIONAL [XMLDSIG] ***Signature*** element (see section 4.1 for more details)

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 the sender of the message to be unauthorized. The decision process resulting in this course of action is implementation dependent.

8.2 StatusRequest Element

The OPTIONAL ***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 8.3.5).

The ***StatusRequest*** element consists of the following:

- an **id** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a ***RefToMessageId*** element

8.2.1 RefToMessageId Element

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

8.2.2 StatusRequest Sample

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

```
<eb:StatusRequest eb:version="2.0" >
    <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
</eb:StatusRequest>
```

8.2.3 StatusRequest Element Interaction

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

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

8.3 StatusResponse Element

The OPTIONAL ***StatusResponse*** element is an immediate child of a SOAP ***Body*** and is used by one MSH to describe the status of processing of a message.

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

- an ***id*** attribute (see section 2.3.7 for details)
- a ***version*** attribute (see section 2.3.8 for details)
- a ***RefToMessageId*** element
- a ***Timestamp*** element
- a ***messageStatus*** attribute

8.3.1 RefToMessageId Element

A REQUIRED ***RefToMessageId*** element contains the ***MessageId*** of the message whose status is being reported. ***RefToMessageId*** element child of the ***MessageData*** element of a message containing a ***StatusResponse*** element SHALL have the ***MessageId*** of the message containing the ***StatusRequest*** element to which the ***StatusResponse*** element applies. The ***RefToMessageId*** child element of the ***StatusRequest*** or ***StatusResponse*** element SHALL contain the ***MessageId*** of the message whose status is being queried.

8.3.2 Timestamp Element

The ***Timestamp*** element contains the time the message, whose status is being reported, was received (section 3.1.6.2.). This MUST be omitted if the message, whose status is being reported, is ***NotRecognized*** or the request was ***UnAuthorized.***

8.3.3 messageStatus attribute

The REQUIRED ***messageStatus*** attribute identifies the status of the message 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
- ***Processed*** – the message identified by the ***RefToMessageId*** element in the ***StatusResponse*** element has been processed by the MSH

- ***Forwarded*** – the message identified by the ***RefToMessageId*** element in the ***StatusResponse*** element has been forwarded by the MSH to another 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, the Message Status Response may indicate the ***MessageId*** was ***NotRecognized*** – the ***MessageId*** is no longer in persistent storage.

8.3.4 StatusResponse Sample

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

```
<eb:StatusResponse eb:version="2.0" eb:messageStatus="Received">
  <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
  <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
</eb:StatusResponse>
```

8.3.5 StatusResponse Element Interaction

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***

9 Message Service Handler Ping Service

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

- one MSH sending a Message Service Handler Ping message to a MSH, and
- another MSH, receiving the Ping, responding with a Message Service Handler Pong message.

If a *Receiving MSH* does not support the service requested, it SHOULD return an *Error Message* with an ***errorCode*** of ***NotSupported*** and a ***highestSeverity*** attribute set to ***Error***.

9.1 Message Service Handler Ping Message

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

- a ***MessageHeader*** element containing the following:
  - a ***From*** element identifying the *Party* creating the MSH Ping message
  - a ***To*** element identifying the *Party* being sent the MSH Ping message
  - a ***CPAId*** element
  - a ***ConversationId*** element
  - a ***Service*** element containing: ***urn:oasis:names:tc:ebxml-msg:service***
  - an ***Action*** element containing ***Ping***
  - a ***MessageData*** element
- an OPTIONAL [XMLDSIG] ***Signature*** element (see section 4.1 for details).

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

An example Ping:

```
. . .Transport Headers
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="ebXMLBoundary"

--ebXMLBoundary
Content-Type: text/xml

```

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
<SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <eb:MessageHeader version="2.0" SOAP:mustUnderstand="true"
        xmlns=eb:"http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
        xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
      <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>20010215-111213-28572</eb:ConversationId>
      <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
      <eb:Action>Ping</eb:Action>
      <eb:MessageData>
          <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
          <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
      </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body/>
</SOAP:Envelope>

--ebXMLBoundary--
```

Note: The above example shows a Multipart/Related MIME structure with only one bodypart.

9.2 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 Container and the following:

- a ***MessageHeader*** element containing the following:
  - a ***From*** element identifying the creator of the MSH Pong message
  - a ***To*** element identifying a *Party* that generated the MSH Ping message
  - a ***CPAId*** element
  - a ***ConversationId*** element
  - a ***Service*** element containing the value: ***urn:oasis:names:tc:ebxml-msg:service***
  - an ***Action*** element containing the value ***Pong***
  - a ***MessageData*** element containing:
    - a ***RefToMessageId*** identifying the MSH Ping message.
- an OPTIONAL [XMLDSIG] ***Signature*** element (see section 4.1.1 for details).

An example Pong:

```
. . .Transport Headers
SOAPAction: "ebXML"
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                        http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
<SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
        xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <eb:MessageHeader eb:version="2.0" SOAP:mustUnderstand="true">
      <eb:From> <eb:PartyId>urn:duns:912345678</eb:PartyId> </eb:From>
      <eb:To>   <eb:PartyId>urn:duns:123456789</eb:PartyId> </eb:To>
      <eb:CPAId>20001209-133003-28572</eb:CPAId>
      <eb:ConversationId>20010215-111213-28572</eb:ConversationId>
      <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
```

```
        <eb:Action>Pong</eb:Action>
        <eb:MessageData>
            <eb:MessageId>20010215-111213-395884@example2.com</eb:MessageId>
            <eb:Timestamp>2001-02-15T11:12:13</eb:Timestamp>
            <eb:RefToMessageId>20010215-111212-28572@example.com</eb:RefToMessageId>
        </eb:MessageData>
    </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body/>
</SOAP:Envelope>
```

Note: This example shows a non-multipart MIME structure.

9.3 Security Considerations

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

10 MessageOrder Module

The ***MessageOrder*** module allows messages to be presented to the *To Party* in a particular order. This is accomplished through the use of the ***MessageOrder*** element. Reliable Messaging MUST be used when a ***MessageOrder*** element is present.

MessageOrder module MUST only be used in conjunction with the ebXML Reliable Messaging Module (section 7) with a scheme of Once-And-Only-Once (sections 7.6). If a sequence is sent and one message fails to arrive at the *To Party MSH*, all subsequent messages will also fail to be presented to the *To Party* Application (see ***status*** attribute section 10.1.1).

10.1 MessageOrder Element

The ***MessageOrder*** element is an OPTIONAL extension to the SOAP ***Header*** requesting the preservation of message order in this conversation.

The ***MessageOrder*** element contains the following:

- a ***id*** attribute (see section 2.3.7)
- a ***version*** attribute (see section 2.3.8 for details)
- a SOAP ***mustUnderstand*** attribute with a value of ***'true'*** (see section 2.3.9 for details)
- a ***SequenceNumber*** element

When the ***MessageOrder*** element is present, ***DuplicateElimination*** MUST also be present and ***SyncReply*** MUST NOT be present.

10.1.1 SequenceNumber Element

The REQUIRED ***SequenceNumber*** element indicates the sequence a *Receiving MSH* MUST process messages. 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 within a conversation and then incremented by one for each subsequent message sent.

A MSH that receives a message with a ***SequenceNumber*** element MUST NOT pass the message to an application until all the messages with a lower ***SequenceNumber*** have been 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 4.1.5).

The ***SequenceNumber*** element is an integer value 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 after 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***. The default value of the ***status*** attribute is ***Continue***.

A *Sending MSH* MUST wait before resetting the ***SequenceNumber*** of a conversation until it has received confirmation of all the messages previously sent for the conversation. Only when all the sent Messages are accounted for, can the *Sending MSH* reset the ***SequenceNumber***.

10.1.2 MessageOrder Sample

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

```
<eb:MessageOrder eb:version="2.0" SOAP:mustUnderstand="true">
    <eb:SequenceNumber>00000010</eb:SequenceNumber>
</eb:MessageOrder>
```

10.2 MessageOrder Element Interaction

For this version of the ebXML Messaging Specification, the ***MessageOrder*** element MUST NOT be present with the ***SyncReply*** element. If these two elements are received in the same message, the *Receiving MSH* SHOULD report an error (see section 4.1.5) with ***errorCode*** set to ***Inconsistent*** and ***severity*** set to ***Error***.

11 Multi-Hop Module

Multi-hop is the process of passing the message through one or more intermediary nodes or MSH's. An Intermediary is any node or MSH where the message is received, but is not the *Sending* or *Receiving MSH*. This node is called an Intermediary.

Intermediaries may be for the purpose of Store-and-Forward or may be involved in some processing activity such as a trusted third-party timestamp service. For the purposes of this version of this specification, Intermediaries are considered only as Store-and-Forward entities.

Intermediaries MAY be involved in removing and adding SOAP extension elements or modules targeted either to the ***Next*** SOAP node or the ***NextMSH***. SOAP rules specify, the receiving node must remove any element or module targeted to the ***Next*** SOAP node. If the element or module needs to continue to appear on the SOAP message destined to the ***Next*** SOAP node, or in this specification the ***NextMSH***, it must be reapplied. This deleting and adding of elements or modules poses potential difficulties for signed ebXML messages. Any Intermediary node or MSH MUST NOT change, format or in any way modify any element not targeted to the Intermediary. Any such change may invalidate the signature.

11.1 Multi-hop Reliable Messaging

Multi-hop (hop-to-hop) Reliable Messaging is accomplished using the ***AckRequested*** element (section 7.3.1) and an *Acknowledgment Message* containing an ***Acknowledgment*** element (section 7.3.1.4) each with a SOAP ***actor*** of ***Next MSH*** (section 2.3.10) between the *Sending MSH* and the *Receiving MSH*. This MAY be used in store-and-forward multi-hop situations.

The use of the duplicate elimination is not required for Intermediate nodes. Since duplicate elimination by an intermediate MSH can interfere with End-to-End Reliable Messaging Retries, the intermediate MSH MUST know it is an intermediate and MUST NOT perform duplicate elimination tasks.

At this time, the values of ***Retry*** and ***RetryInterval*** between Intermediate MSHs remains implementation specific. See section 7.4 for more detail on Reliable Messaging.

11.1.1 AckRequested Sample

An example of the ***AckRequested*** element targeted at the ***NextMSH*** is given below:

```
<eb:AckRequested SOAP:mustUnderstand="true" eb:version="2.0" eb:signed="false"
    SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH"/>
```

In the preceding example, an *Acknowledgment Message* is requested of the next ebXML MSH node (see section 2.3.10) in the message. The ***Acknowledgment*** element generated MUST be targeted at the next ebXML MSH node along the reverse message path (the *Sending MSH*) using the SOAP ***actor*** with a value of ***NextMSH*** (section 2.3.10).

Any Intermediary receiving an ***AckRequested*** with SOAP ***actor*** of ***NextMSH*** MUST remove the ***AckRequested*** element before forwarding to the next MSH. Any Intermediary MAY insert a single ***AckRequested*** element into the SOAP ***Header*** with a SOAP ***actor*** of ***NextMSH***. There SHALL NOT be two ***AckRequested*** elements targeted at the next MSH.

When the ***SyncReply*** element is present, an ***AckRequested*** element with SOAP ***actor*** of ***NextMSH*** MUST NOT be present. If the ***SyncReply*** element is not present, the Intermediary MAY return the Intermediate *Acknowledgment Message* synchronously with a synchronous transport protocol. If these two elements are received in the same message, the *Receiving MSH* SHOULD report an error (see section 4.1.5) with ***errorCode*** set to ***Inconsistent*** and ***severity*** set to ***Error***.

11.1.2 Acknowledgment Sample

An example of the ***Acknowledgment*** element targeted at the ***NextMSH*** is given below:

```
<eb:Acknowledgment SOAP:mustUnderstand="true" eb:version="2.0"
      SOAP:actor="urn:oasis:names:tc:ebxml-msg:actor:nextMSH">
  <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
  <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
  <eb:From> <eb:PartyId>uri:www.example.com</eb:PartyId> </eb:From>
</eb:Acknowledgment>
```

11.1.3 Multi-Hop Acknowledgments

There MAY be two ***AckRequested*** elements on the same message. An ***Acknowledgement*** MUST be sent for each ***AckRequested*** using an identical SOAP ***actor*** attribute as the ***AckRequested*** element.

If the *Receiving MSH* is the *To Party MSH*, then see section 7.5.2. If the *Receiving MSH* is the *To Party MSH* and there is an ***AckRequested*** element targeting the Next MSH (the *To Party MSH* is acting in both roles), then perform both procedures (this section and section 7.5.2) for generating *Acknowledgment Messages*. This MAY require sending two ***Acknowledgment*** elements, possibly on the same message, one targeted for the *Next MSH* and one targeted for the *To Party MSH*.

There MAY be multiple ***Acknowledgements*** elements, on the same message or on different messages, returning from either the Next MSH or from the *To Party MSH*. A MSH supporting Multi-hop MUST differentiate, based upon the ***actor***, which ***Acknowledgment*** is being returned and act accordingly.

If this is an *Acknowledgment Message* as defined in section 7 then:

1. Look for a message in *persistent storage* with a ***MessageId*** 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.

If an ***AckRequested*** element is present (not an *Acknowledgment Message*) then generate an *Acknowledgment Message* in response (this may be as part of another message). The *Receiving MSH* MUST NOT send an *Acknowledgment Message* until the message has been persisted or delivered to the *Next MSH*.

11.1.4 Signing Multi-Hop Acknowledgments

When a signed Intermediate *Acknowledgment Message* is requested (i.e. a signed *Acknowledgment Message* with a SOAP ***actor*** of ***NextMSH***), it MUST be sent by itself and not bundled with any other message. The XML Signature [XMLDSIG] ***Signature*** element with ***Transforms***, as described in section 4.1.3, will exclude this ***Acknowledgment*** element. To send a signed *Acknowledgment Message* with SOAP ***actor*** of ***NextMSH***, create a message with no payloads, including a single ***Acknowledgment*** element (see section 7.3.2.6), and a [XMLDSIG] ***Signature*** element with the following ***Transforms***:

```
<Transforms>
  <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
  <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
</Transforms>
```

11.1.5 Multi-Hop Security Considerations

SOAP messaging allows intermediaries to add or remove elements targeted to the intermediary node. This has potential conflicts with end-to-end signatures since the slightest change in any character of the SOAP ***Envelope*** or to a payload will invalidate the ***ds:Signature*** by changing the calculated digest. Intermediaries MUST NOT add or remove elements unless they contain a SOAP ***actor*** of ***next*** or ***nextMSH***. Intermediaries MUST NOT disturb white space – line terminators (CR/LF), tabs, spaces, etc. – outside those elements being added or removed.

11.2 Message Ordering and Multi-Hop

Intermediary MSH nodes MUST NOT participate in Message Order processing as specified in section 10.

Part III. Normative Appendices

Appendix A The ebXML SOAP Extension Elements Schema

The ebXML SOAP extension elements schema has been specified using the Recommendation version 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 URL was written to an earlier draft of the XML Schema specification, the OASIS ebXML Messaging Technical Committee has created a version of the SOAP 1.1 envelope schema specified using the schema vocabulary that conforms to the W3C XML Schema Recommendation specification [XMLSchema].

In addition, it was necessary to craft a schema for the XLINK [XLINK] attribute vocabulary and for the XML xml:lang attribute to conform to the W3C XML Schema Recommendation [XMLSchema].

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://www.oasis-open.org/committees/ebxml-msg/schema/xmldsig-core-schema.xsd

Xlink - http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd

xml:lang - http://www.oasis-open.org/committees/ebxml-msg/schema /xml_lang.xsd

SOAP1.1- http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
  xmlns:tns="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
  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/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="qualified"
  version="1.0">
  <import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/xmldsig-core-schema.xsd"/>
  <import namespace="http://www.w3.org/1999/xlink"
    schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd"/>
  <import namespace="http://schemas.xmlsoap.org/soap/envelope/"
    schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd"/>
  <import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/xml_lang.xsd"/>
  <!-- MANIFEST, for use in soap:Body element -->
  <element name="Manifest">
    <complexType>
      <sequence>
        <element ref="tns:Reference" maxOccurs="unbounded"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:bodyExtension.grp"/>
    </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" fixed="simple"/>
      <attribute ref="xlink:href" use="required"/>
      <attribute ref="xlink:role"/>
    </complexType>
  </element>
  <element name="Schema">
    <complexType>
      <attribute name="location" type="anyURI" use="required"/>
      <attribute name="version" type="tns:non-empty-string"/>
    </complexType>
  </element>
  <!-- MESSAGEHEADER, for use in soap:Header element -->
  <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:DuplicateElimination" minOccurs="0"/>
        <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
    </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="dateTime"/>
  <element name="DuplicateElimination">
  </element>
  <!-- SYNC REPLY, for use in soap:Header element -->
  <element name="SyncReply">
    <complexType>
      <sequence>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
      <attribute ref="soap:actor" use="required"/>
    </complexType>
  </element>
  <!-- MESSAGE ORDER, for use in soap:Header element -->
  <element name="MessageOrder">
    <complexType>
      <sequence>
        <element ref="tns:SequenceNumber"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
    </complexType>
  </element>
  <element name="SequenceNumber" type="tns:sequenceNumber.type"/>
  <!-- ACK REQUESTED, for use in soap:Header element -->
  <element name="AckRequested">
    <complexType>
      <sequence>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
      <attribute ref="soap:actor"/>
      <attribute name="signed" type="boolean" use="required"/>
    </complexType>
  </element>
  <!-- ACKNOWLEDGMENT, for use in soap:Header element -->
  <element name="Acknowledgment">
    <complexType>
      <sequence>
        <element ref="tns:Timestamp"/>
        <element ref="tns:RefToMessageId"/>
        <element ref="tns:From" minOccurs="0"/>
        <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
      <attribute ref="soap:actor"/>
    </complexType>
  </element>
  <!-- ERROR LIST, for use in soap:Header element -->
  <element name="ErrorList">
    <complexType>
      <sequence>
        <element ref="tns:Error" maxOccurs="unbounded"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:headerExtension.grp"/>
      <attribute name="highestSeverity" type="tns:severity.type" use="required"/>
    </complexType>
  </element>
  <element name="Error">
    <complexType>
      <sequence>
        <element ref="tns:Description" minOccurs="0"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute ref="tns:id"/>
      <attribute name="codeContext" type="anyURI"
            default="urn:oasis:names:tc:ebxml-msg:service:errors"/>
      <attribute name="errorCode" type="tns:non-empty-string" use="required"/>
      <attribute name="severity" type="tns:severity.type" use="required"/>
      <attribute name="location" type="tns:non-empty-string"/>
    </complexType>
  </element>
  <!-- STATUS RESPONSE, for use in soap:Body element -->
  <element name="StatusResponse">
    <complexType>
      <sequence>
        <element ref="tns:RefToMessageId"/>
        <element ref="tns:Timestamp" minOccurs="0"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:bodyExtension.grp"/>
      <attribute name="messageStatus" type="tns:messageStatus.type" use="required"/>
    </complexType>
  </element>
  <!-- STATUS REQUEST, for use in soap:Body element -->
  <element name="StatusRequest">
    <complexType>
      <sequence>
```

```
        <element ref="tns:RefToMessageId"/>
        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attributeGroup ref="tns:bodyExtension.grp"/>
    </complexType>
  </element>
  <!-- COMMON TYPES -->
  <complexType name="sequenceNumber.type">
    <simpleContent>
      <extension base="positiveInteger">
        <attribute name="status" type="tns:status.type" default="Continue"/>
      </extension>
    </simpleContent>
  </complexType>
  <simpleType name="status.type">
    <restriction base="NMTOKEN">
      <enumeration value="Reset"/>
      <enumeration value="Continue"/>
    </restriction>
  </simpleType>
  <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="non-empty-string">
    <restriction base="string">
      <minLength value="1"/>
    </restriction>
  </simpleType>
  <simpleType name="severity.type">
    <restriction base="NMTOKEN">
      <enumeration value="Warning"/>
      <enumeration value="Error"/>
    </restriction>
  </simpleType>
  <!-- COMMON ATTRIBUTES and ATTRIBUTE GROUPS -->
  <attribute name="id" type="ID"/>
  <attribute name="version" type="tns:non-empty-string"/>
  <attributeGroup name="headerExtension.grp">
    <attribute ref="tns:id"/>
    <attribute ref="tns:version" use="required"/>
    <attribute ref="soap:mustUnderstand" use="required"/>
  </attributeGroup>
  <attributeGroup name="bodyExtension.grp">
    <attribute ref="tns:id"/>
    <attribute ref="tns:version" use="required"/>
  </attributeGroup>
  <!-- COMMON ELEMENTS -->
  <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"/>
        <element name="Role" type="tns:non-empty-string" minOccurs="0"/>
      </sequence>
    </complexType>
  </element>
  <element name="From">
```

```
    <complexType>
      <sequence>
        <element ref="tns:PartyId" maxOccurs="unbounded"/>
        <element name="Role" type="tns:non-empty-string" minOccurs="0"/>
      </sequence>
    </complexType>
  </element>
  <element name="Description">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute ref="xml:lang" use="required"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name="RefToMessageId" type="tns:non-empty-string"/>
  <element name="Timestamp" type="dateTime"/>
</schema>
```

Appendix B Communications Protocol Bindings

B.1 Introduction

One of the goals of this specification is to design a message handling service usable over a variety of network and application level transport 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 transport protocols. TCP and SNA/LU6.2 are examples of network transport protocols. Transport 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 transport protocols.

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

- Hypertext Transfer Protocol [RFC2616], in both *asynchronous* and *synchronous* forms of transfer.
- Simple Mail Transfer Protocol [RFC2821], in *asynchronous* form of transfer only.

B.2 HTTP

B.2.1 Minimum level of HTTP protocol

Hypertext Transfer Protocol Version 1.1 [RFC2616] is the minimum level of protocol that MUST be used.

B.2.2 Sending ebXML Service messages over HTTP

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

```
        POST /ebxmlhandler HTTP/1.1
```

Prior to sending over HTTP, an ebXML Message MUST be formatted according to ebXML Message Service Specification. Additionally, the messages MUST conform to the HTTP specific MIME canonical form constraints specified in section 19.4 of RFC 2616 [RFC2616] specification.

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 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:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                        http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
                        http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
<SOAP:Header>
  <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
    <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>2001-02-15T11:12:12</eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body>
  <eb:Manifest eb:version="2.0">
    <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:Body>
</SOAP:Envelope>

--BoundarY
Content-ID: <ebxmlpayload111@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price currency="USD">500.00</price>
</purchase_order>

--BoundarY—
```

B.2.3 HTTP Response Codes

In general, semantics of communicating over HTTP as specified in the [RFC2616] MUST be followed, for returning the HTTP level response codes. A 2xx code MUST be returned when the HTTP Posted message is successfully received by the receiving HTTP entity. However, see exception for SOAP error conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx range MAY be returned for conditions corresponding to them. However, error conditions encountered while processing an ebXML Service Message MUST be reported using the error mechanism defined by the ebXML Message Service Specification (see section 4.1.5).

B.2.4 SOAP Error conditions and Synchronous Exchanges

The SOAP 1.1 specification states:

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

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

B.2.5 Synchronous vs. Asynchronous

When a synchronous transport is in use, the MSH response message(s) SHOULD be returned on the same HTTP connection as the inbound request, with an appropriate HTTP response code, as described above. When the ***syncReplyMode*** parameter is set to values other than ***none***, the application response messages, if any, are also returned on the same HTTP connection as the inbound request, rather than using an independent HTTP Post request. If the ***syncReplyMode*** has a value of ***none***, an HTTP response with a response code as defined in section B.2.3 above and with an empty HTTP body MUST be returned in response to the HTTP Post.

B.2.6 Access Control

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

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

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

B.2.7 Confidentiality and Transport 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 TLS [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 TLS [RFC2246].

ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key sizes specified within TLS [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 TLS [RFC2246] and SSL [SSL3] require the use of server side digital certificates. Client side certificate based authentication is also permitted. All ebXML Message Service handlers MUST support hierarchical and peer-to-peer or direct-trust trust models.

B.3 SMTP

The Simple Mail Transfer Protocol (SMTP) [RFC2821] specification is commonly referred to as Internet Electronic Mail. This specifications has been augmented over the years by other specifications, which define additional functionality "layered on top" of this 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.

B.3.1 Minimum Level of Supported Protocols

Simple Mail Transfer Protocol [RFC2821]

MIME [RFC2045] and [RFC2046]

Multipart/Related MIME [RFC2387]

B.3.2 Sending ebXML Messages over SMTP

Prior to sending messages over SMTP an ebXML Message MUST be formatted according to the ebXML Message Service Specification. 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 [RFC2821], 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 MIME [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 SMTP [RFC2821] 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 MIME [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"**
- The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- The eMail header "To:" MUST contain the SMTP [RFC2821] compliant eMail address of the ebXML Message Service Handler.
- The eMail header "From:" MUST contain the SMTP [RFC2821] compliant eMail address of the senders ebXML Message Service Handler.
- Construct a "Date:" eMail header in accordance with SMTP [RFC2821]
- Other headers MAY occur within the eMail message header in accordance with SMTP [RFC2821] and MIME [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>"

     This is an ebXML SMTP Example

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

<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
               http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd">
<SOAP:Header   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
        xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <eb:MessageHeader SOAP:mustUnderstand="true" eb:version="2.0">
    <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>2001-02-15T11:12:12</eb:Timestamp>
    </eb:MessageData>
    <eb:DuplicateElimination/>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
      xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
  <eb:MessageHeader SOAP:mustUnderstand="1" eb:version="2.0">
  <eb:Manifest eb:version="2.0">
    <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:Body>
</SOAP:Envelope>

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

<?xml version="1.0" encoding="UTF-8"?>
<purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price currency="USD">500.00</price>
</purchase_order>

--BoundarY--
```

B.3.3 Response Messages

All ebXML response messages, including errors and acknowledgments, are delivered *asynchronously* between ebXML Message Service Handlers. Each response message MUST be constructed in accordance with the rules specified in the section B.3.2.

All 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.

B.3.4 Access Control

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

B.3.5 Confidentiality and Transport Protocol Level Security

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

B.3.6 SMTP Model

All *ebXML Message Service* messages carried as mail in an SMTP [RFC2821] Mail Transaction as shown in the figure below.

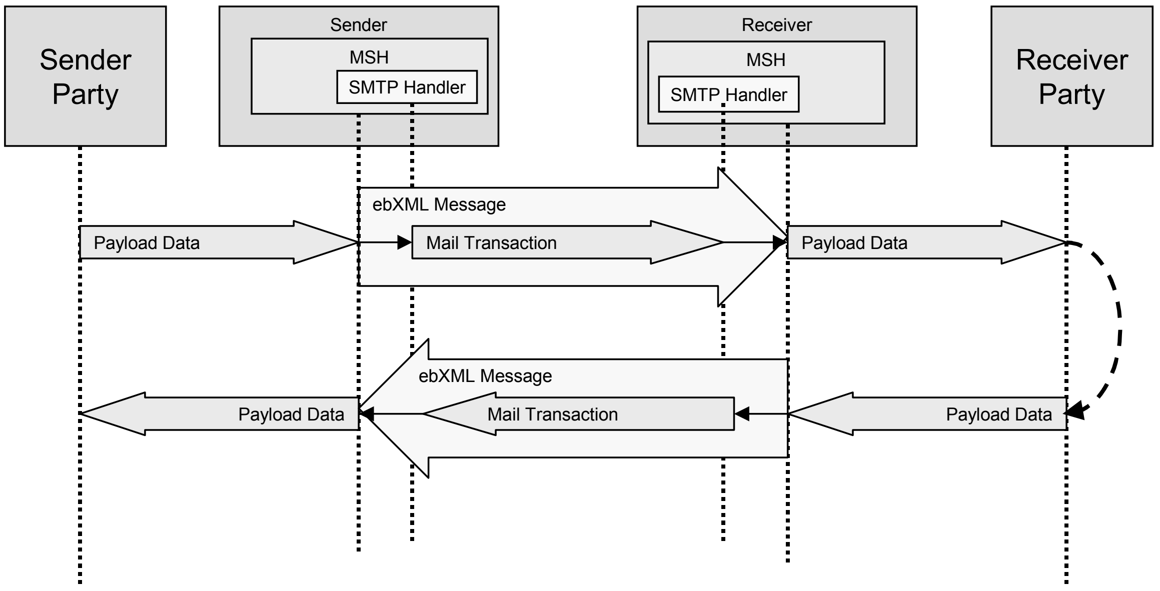

Figure B-1 SMTP Mail Depiction

B.4 Communication Errors during Reliable Messaging

When the Sender or the Receiver detects a communications 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 7).

Appendix C 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.
	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

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 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]
	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)

2742

References

Normative References

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

[RFC2045] 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] Dierks, T. and C. Allen, "The TLS Protocol", January 1999.

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

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

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

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

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

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

[RFC2821] Simple Mail Transfer Protocol, J. Klensin, Editor, April 2001 Obsoletes RFC 821

[RFC2616] 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] 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] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", May 2000.

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

[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

[XMLC14N] Joint W3C/IETF XML-Signature Syntax and Processing specification, http://www.w3.org/TR/xml-c14n.

[XMLNS] 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/2001/PR-xmldsig-core-20010820/.
[XMLMedia]	RFC 3023, XML Media Types. M. Murata, S. St. Laurent, January 2001
[XPointer]	XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11 September 2001, http://www.w3.org/TR/xptr/

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 27 April 2001.
[ebTA]	ebXML Technical Architecture, version 1.04 published 16 February, 2001
[ebRS]	ebXML Registry Services Specification, version 0.84
[ebREQ]	ebXML Requirements Specification, http://www.ebxml.org/specs/ebREQ.pdf, published 8 May 2001.
[ebGLOSS]	ebXML Glossary, http://www.ebxml.org/specs/ebGLOSS.doc, published 11 May, 2001.
[PGP/MIME]	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]	RFC 2311, "S/MIME Version 2 Message Specification", S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.
[S/MIMECH]	RFC 2312, "S/MIME Version 2 Certificate Handling", S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. March 1998.
[S/MIMEV3]	RFC 2633 S/MIME Version 3 Message Specification. B. Ramsdell, Ed June 1999.
[secRISK]	ebXML Technical Architecture Risk Assessment Technical Report, version 0.36 published 20 April 2001
[XMLSchema]	W3C XML Schema 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

Contact Information

Team Leader

Name	Ian Jones
Company	British Telecommunications
Address	Enterprise House, 84-85 Adam Street
	Cardiff, CF24 2XF United Kingdom
Phone:	+44 29 2072 4063
EMail:	ian.c.jones@bt.com

Vice Team Leader

Name	Brian Gibb
Company	Sterling Commerce
Address	750 W. John Carpenter Freeway
	Irving, Texas 75039 USA
Phone:	+1 (469) 524.2628
EMail:	brian_gibb@stercomm.com

Team Editor

Name	David Fischer
Company	Drummond Group, Inc
Address	P.O. Box 101567
	Fort Worth, Texas 76105 USA
Phone	+1 (817) 294-7339
EMail	david@drummondgroup.com

Acknowledgments

The OASIS ebXML-MS Technical Committee would like to thank the members of the original joint UN/CEFACT-OASIS ebXML Messaging Team for their work to produce v1.0 of this specification.

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 –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

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.