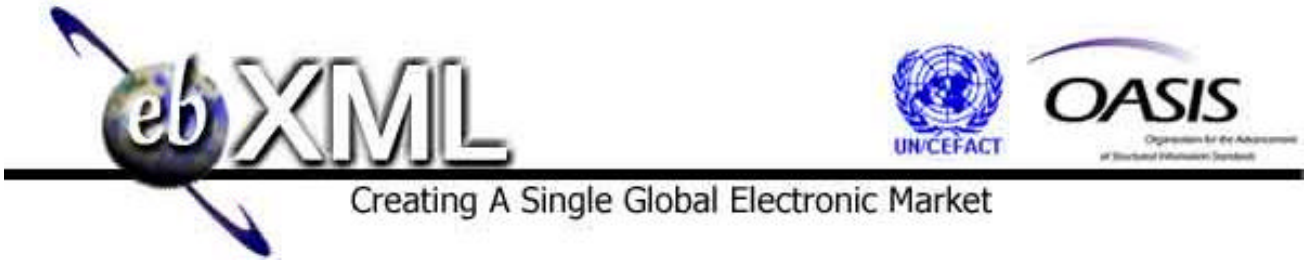


Messaging Service Specification

ebXML Transport, Routing & Packaging

Version 0.9a

21~~0~~ December 2000

1 Status of this Document

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

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

This version

http://www.ebxml.org/working/project_teams

Latest version

http://www.ebxml.org

Previous version

http://www.ebxml.org/…

2 ebXML Participants

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

Ralph Berwanger – bTrade.com
Jonathan Borden – Author of XMTP
Jon Bosak – Sun Microsystems
Marc Breissinger – webMethods
Dick Brooks – Group 8760
Doug Bunting – Ariba
David Burdett – Commerce One
Len Callaway – Drummond Group, Inc.
David Craft – VerticalNet
Philippe De Smedt – Viquity
Lawrence Ding – WorldSpan
Rik Drummond – Drummond Group, Inc. (Representing XML Solutions)
Christopher Ferris – Sun Microsystems
Maryann Hondo – IBM
Jim Hughes – Fujitsu
John Ibbotson – IBM
Ian Jones – British Telecommunications
Ravi Kacker – Kraft Foods
Nick Kassem – Sun Microsystems
Henry Lowe – OMG
Jim McCarthy – webXI
Bob Miller – GSX
Dale Moberg – Sterling Commerce
Joel Munter – Intel
Farrukh Najmi – Sun Microsystems
Akira Ochi – Fujitsu
Martin Sachs, IBM
Masayoshi Shimamura – Fujitsu
Kathy Spector – Extricity
Nikola Stojanovic – Columbine JDS Systems
Gordon Van Huizen – Process Software
Martha Warfelt – Daimler Chrysler
Prasad Yendluri – Vitria

3 Table of Contents

1 Status of this Document 2
2 ebXML Participants 3
3 Table of Contents 4
4 Introduction 8
4.1 Summary of Contents of Document 8
4.2 Document Conventions 8
4.3 Audience 9
4.4 Caveats and Assumptions 9
4.5 Related Documents 9
5 Design Objectives 10
6 System Overview 11
6.1 What the Message Service does 11
6.2 Message Service Overview 11
7 Packaging Specification 13
7.1 Introduction 13
7.1.1 ebXML Header Envelope and Payload Envelope 13
7.1.2 MIME usage Conventions 14
7.2 ebXML Message Envelope 14
7.2.1 Content-Type 14
7.2.1.1 type Attribute 14
7.2.1.2 boundary Attribute 14
7.2.1.3 version Attribute 14
7.2.2 ebXML Message Envelope Example 14
7.3 ebXML Header Container 15
7.3.1 Content-ID 15
7.3.2 Content-Type 15
7.3.2.1 version Attribute 15
7.3.2.2 charset Attribute 15
7.3.3 ebXML Header Envelope Example 15
7.4 ebXML Payload Container 16
7.4.1 Content-ID 16
7.4.2 Content-Type 16
7.4.3 Example of an ebXML MIME Payload Container 17
8 Header Document 18
8.1 XML Prolog 18
8.1.1 XML Declaration 18
8.1.2 Encoding Declaration 18
8.1.3 Standalone Document Declaration 18
8.1.4 Document Type Declaration 18
8.2 ebXMLHeader Element 19
8.2.1 ebXMLHeader attributes 19
8.2.1.1 Namespace attribute 19
8.2.1.2 version attribute 19
8.2.2 ebXMLHeader elements 19
8.2.3 Combining Principal Header Elements 19
8.2.3.1 Manifest element 20
8.2.3.2 Header element 20

8.2.3.3 RoutingHeaderList element 20
8.2.3.4 ApplicationHeaders element 20
8.2.3.5 StatusData element 20
8.2.3.6 ErrorList element 20
8.2.3.7 Acknowledgment element 20
8.2.3.8 Signature element 20
8.2.3.9 #wildcard element content 20
8.2.4 ebXMLHeader sample 21
8.3 Manifest element 21
8.3.1 Reference element 21
8.3.1.1 Description element 21
8.3.1.2 Schema element 22
8.3.2 Manifest sample 22
8.4 Header element 22
8.4.1 From and To elements 22
8.4.2 CPAId element 23
8.4.3 ConversationId element 23
8.4.4 Service element 23
8.4.4.1 Type attribute 23
8.4.4.2 ebXML Message Service Header namespace 23
8.4.5 Action element 23
8.4.6 Description element 23
8.4.7 ReliableMessagingInfo element 24
8.4.8 MessageData element 24
8.4.8.1 MessageId element 24
8.4.8.2 TimeStamp element 24
8.4.8.3 RefToMessageId element 24
8.4.9 #wildcard element 24
8.4.10 Header sample 24
8.5 RoutingHeaderList element 25
8.5.1 Routing Header Element 25
8.5.1.1 SenderURI element 25
8.5.1.2 ReceiverURI element 25
8.5.1.3 ErrorURI element 25
8.5.1.4 Timestamp element 26
8.5.1.5 #wildcard 26
8.5.2 Single Hop Routing Header Sample 26
8.5.3 Multi-hop Routing Header Sample 27
8.6 ApplicationHeaders Element 27
8.6.1 ApplicationHeaders sample 28
8.7 StatusData Element 28
8.8 ErrorList Element 28
8.8.1 id attribute 29
8.8.2 highestSeverity attribute 29
8.8.3 Error element 29
8.8.3.1 codeContext attribute 29
8.8.3.2 errorCode attribute 29
8.8.3.3 severity attribute 29
8.8.3.4 location attribute 29
8.8.3.5 errorMessage atttribute 30
8.8.3.6 softwareDetails attribute 30
8.8.4 Examples 30
8.8.5 errorCode values 30
8.8.6 Reporting Errors in the ebXML Header Document 31
8.8.7 Non-XML Document Errors 31
8.9 Acknowledgment Element 31
8.9.1 TimeStamp element 32
8.9.2 From element 32
8.9.3 type attribute 32

8.9.4 signed attribute 32
8.10 Signature Element 32
9 Message Service Handler Services 33
9.1 Message Status Request Service 33
9.1.1 Message Status Request Message 33
9.1.2 Message Status Response Message 33
9.1.3 Security Considerations 34
9.2 Message Service Handler Ping Service 34
9.2.1 Message Service Handler Ping Message 34
9.2.2 Message Service Handler Pong Message 34
9.2.3 Security Considerations 35
10 Reliable Messaging 36
10.1.1 Persistent Storage and System Failure 36
10.1.2 Methods of Implementing Reliable Messaging 36
10.2 ebXML Reliable Messaging Protocol 36
10.2.1 Single-hop Reliable Messaging 37
10.2.1.1 Resending Lost Messages 38
10.2.1.2 Duplicate Acknowledgment Messages 39
10.2.2 Multi-hop Reliable Messaging 40
10.2.2.1 Multi-hop Reliable Messaging without Intermediate Acknowledgments 40
10.2.2.2 Multi-hop Reliable Messaging with Intermediate Acknowledgments 41
10.3 ebXML Reliable Messaging using Queuing Transports 42
10.4 Failed Message Delivery 43
10.5 Reliable Messaging Parameters 44
10.5.1 Who sets Message Service Parameters 44
10.5.2 From Party Parameters 45
10.5.2.1 Delivery Semantics 45
10.5.2.2 Delivery Receipt Requested 46
10.5.2.3 Time To Live 46
10.5.3 To Party Parameters 47
10.5.3.1 DeliveryReceiptProvided 47
10.5.4 Sending MSH Parameters 47
10.5.4.1 Reliable Messaging Method 47
10.5.4.2 Intermediate Ack Requested 47
10.5.4.3 Timeout Parameter 47
10.5.4.4 Retries Parameter 48
10.5.4.5 RetryInterval Parameter 48
10.5.4.6 Deciding when to resend a message 48
10.5.5 Receiving MSH Parameters 48
10.5.5.1 Reliable Messaging Methods Supported 48
10.5.5.2 PersistDuration 48
10.5.5.3 msh Time Accuracy 49
11 Error Reporting and Handling 50
11.1 Definitions 50
11.2 Types of Errors 50
11.3 When to generate Error Messages 50
11.3.1 Security Considerations 50
11.4 Identifying the Error Reporting Location 50
12 Security 52
12.1 Security and Management 52
12.2 Collaboration Party Profiles 52
12.3 RISKS 52
12.3.1 Unauthorized Access 52

12.3.2 Data Integrity and Confidentiality 53
12.3.3 Denial-of Service 53
12.4 CounterMeasure Technologies 53
12.4.1 ebXML Message Countermeasures for Unauthorized Access and Data Integrity 53
12.4.2 Digital Certificates 53
12.4.3 ebXML Message Countermeasures for Denial of Service 53
12.4.4 ebXML Management Countermeasures for Denial of Service 54
12.5 Profiles 54
12.5.1 XML Digital Signature (XMLDSIG) 54
12.5.2 Profile - XMLSignature signing of header and/or payload 54
12.5.2.1 Risks 54
12.5.2.2 Benefits 54
12.5.3 S/MIME 54
12.5.4 Profile - S/MIME signing of message payload 55
12.5.4.1 Sample S/MIME signed payload 55
12.5.4.2 Risks 56
12.5.4.3 Benefits 56
12.5.5 Profile - S/MIME encryption of message payload 56
12.5.5.1 Risks 56
12.5.5.2 Benefits 56
12.5.6 PGP/MIME 56
12.5.7 Profile - PGP/MIME signing of message payload 56
12.5.7.1 Risks 56
12.5.7.2 Benefits 57
12.5.8 Profile - PGP/MIME encryption of message payload 57
12.5.8.1 Risks 57
12.5.8.2 Benefits 57
13 Synchronous and Asynchronous Responses 58
14 References 59
14.1 Normative References 59
14.2 Non-Normative References 59
15 Disclaimer 61
16 Contact Information 62
Appendix A Schema and Data Type Definitions 64
A.1 Schema Definition 64
A.2 Data Type Definition 68
Appendix B Examples 69
Appendix C Communication Protocol Interfaces 70
C.1 HTTP 70
C.1.1 Asynchronous HTTP 70
C.1.2 Synchronous HTTP 71
C.2 SMTP 72
C.3 FTP 73
C.4 Communication Protocol Errors during Reliable Messaging 73
Appendix D Reliable Messaging Processing Logic 74
Copyright Statement 75

4 Introduction

This is a draft standard for trial implementation. The specification is the first in a series of phased deliverables. This version of the specification does not address service interfaces. This is being developed as separate document and will be included in a later version or as additional specifications to the ebXML Message Services Specification.

4.1 Summary of Contents of Document

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

- the *ebXML Message* structure used to package ebXML Messages for transport between parties
- the behavior of the Message Service that sends and receives those messages.

No assumption or dependency is made relative to communication protocol or type of payload. The specifications contained here are both payload and communication protocol neutral.

This specification is organized around the following topics:

- **Packaging Specification** - A description of how to package an *ebXML Message* and its associated parts (section 7)
- **Message Headers** - A specification of the structure and composition of the information necessary for an ebXML Message Service to successfully generate or process an ebXML compliant message (section 8)
- **Message Service Handler Services** – A description of two services that enable one service to discover the status of another Message Service Handler or an individual message (section 9)
- **Reliable Messaging** - The Reliable Messaging function defines an interoperable protocol such that any two Message Service implementations can “reliably” exchange messages that are sent using “reliable messaging” semantics (section 9.2.3)
- **Error Handling** - This section describes how one ebXML Message Service reports errors it detects to another ebXML Message Service (section 11)
- **Security** - This version of the specification provides complete specification of the security requirements for ebXML Messages (section 12).

Appendices to this specification cover the following:

- Appendix A Schemas and DTD Definitions
- Appendix B Examples
- Appendix C Communication Protocol Envelope Mappings
- Appendix D Reliable Messaging Protocol Logic

4.2 Document Conventions

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

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

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

- MUST: This word, or the terms “REQUIRED” or “SHALL”, means that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase “SHALL NOT”, means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective “RECOMMENDED”, means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase “NOT RECOMMENDED”, means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

4.3 Audience

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

4.4 Caveats and Assumptions

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

4.5 Related Documents

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

- **ebXML Collaboration Protocol Profile and Agreement Specification** (under development) - defines how one party can discover and/or agree upon the information that party needs to know about another party prior to sending them a message that complies with this specification
- **ebXML Message Service Interface Specification** (to be developed) - defines an interface that may be used by software to interact with an ebXML Message Service
- **ebXML Message Services Security Specification** (under development) – defines the security mechanisms necessary to negate anticipated, selected threats
- **ebXML Message Services Requirements Specification** – defines the requirements of the Message Services

5 Design Objectives

The design objectives of this specification are to define a Message Service (MS) to support XML based electronic business between small, medium and large enterprises. This specification is intended to enable a low cost solution, while preserving a vendor's ability to add unique value through added robustness and superior performance. It is the intention of the Transport, Routing and Packaging Project Team to keep this specification as straightforward and succinct as possible.

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

6 System Overview

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

6.1 What the Message Service does

The ebXML Message Service defines robust, yet basic, functionality to transfer messages using various existing communication protocols. The ebXML Message Service will perform in a manner that will allow for reliability, persistence, security and extensibility.

The ebXML Message Service is provided for environments requiring a robust, yet low cost solution to enable electronic business. It is one of the three "infrastructure" components of ebXML that includes: Registry/Repository, Collaboration Protocol Profile/Agreement (CPP/CPA) and the ebXML Message Service.

6.2 Message Service Overview

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

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

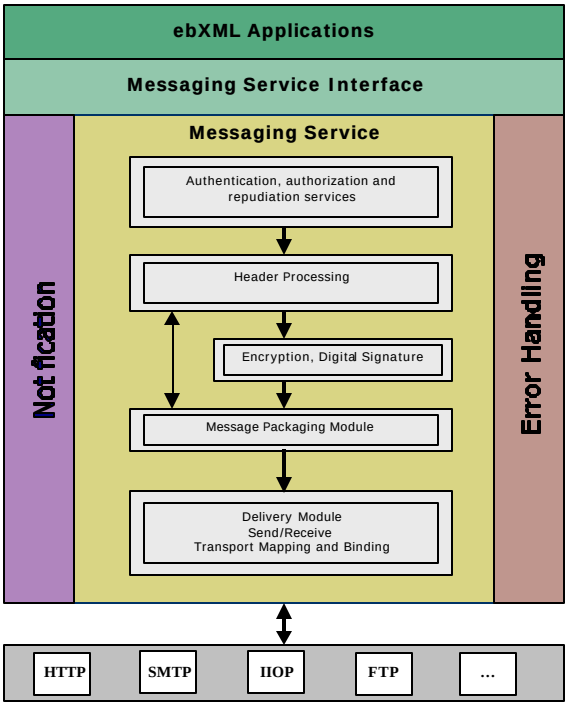


Figure 6-1 Typical Relationship between ebXML MSH Components

105 *<DB>An explanation of these components is needed</DB>*

7 Packaging Specification

7.1 Introduction

An *ebXML Message* consists of:

- an outer Communication Protocol Envelope, such as HTTP or SMTP,
- an inner communication "protocol independent" *ebXML Message Envelope*, specified using MIME multipart/related, that contains the two main parts of the Message:
  - an ebXML Header Container that is used to envelope one ebXML Header Document,
  - an optional, single *ebXML Payload Container* that MUST be used to envelope the actual payload (transferred data) of the Message Communication Protocol Envelope (SMTP, HTTP, etc)

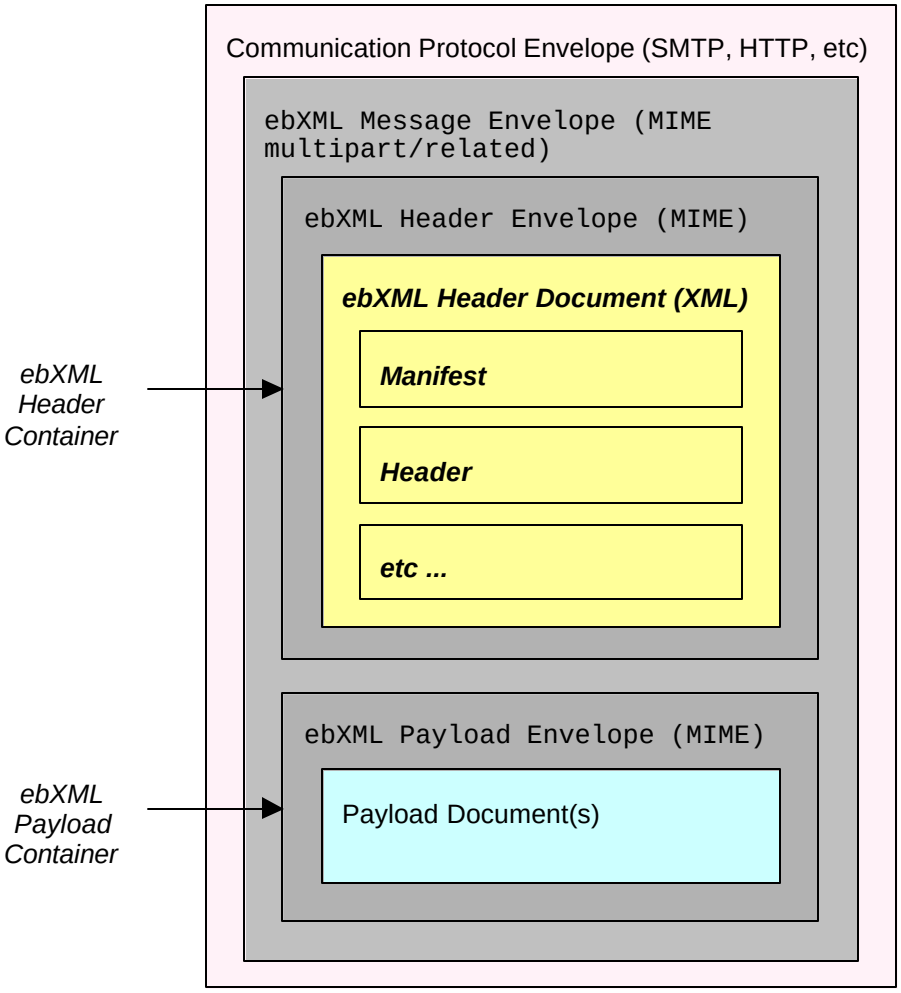


Figure 7-1 ebXML Message Structure

7.1.1 ebXML Header Envelope and Payload Envelope

An *ebXML Header Envelope* and an *ebXML Payload Envelope* are constructed of standard, MIME components.

An *ebXML Header (or Payload) Document* is the content of the standard MIME part and is:

- an XML document in an ***ebXML Heade***r, or
- an XML or some other document for the ebXML Payload

Any special considerations for the usage of the *ebXML Message Envelope* in TCP/IP, HTTP and SMTP transports are described in Appendix E.

7.1.2 MIME usage Conventions

Values associated with MIME header attributes are valid in both quoted and unquoted form. For example, the forms `type="ebxml"` and `type=ebxml` are both valid.

7.2 ebXML Message Envelope

The MIME structured *ebXML Message Envelope* is used to identify the message as an ebXML compliant structure and encapsulates the header and payload in MIME body parts. It MUST conform to [RFC2045] and MUST contain a Content-Type MIME header.

7.2.1 Content-Type

The MIME `Content-Type` MUST be set to `multipart/related` for all *ebXML Message Envelop*es. See Appendix C for selection rationale. For example:

```
Content-Type: multipart/related;
```

The MIME `Content-Type` header contains three attributes:

- `type`
- `boundary`
- `version`

7.2.1.1 type Attribute

The MIME `type` attribute is used to identify the *ebXML Message Envelope* as an ebXML compliant structure. It conforms to a MIME XML Media Type [XMLMedia] and MUST be set to `"application/vnd.eb+xml"`. This new media type is derived from the `application/xml` type and shares many semantics with that type. To that type, `application/vnd.eb+xml` adds a specific application context, the ebXML Message Service. For example:

```
type="application/vnd.eb+xml"
```

7.2.1.2 boundary Attribute

The MIME boundary attribute is used to identify the body part separator used to identify the start and end points of each body part contained in the message. The MIME boundary SHOULD be chosen carefully in order to ensure that it does not occur within the content area of a body part see [RFC 2045] for guidance on how to do this. For example:

```
boundary:="-------8760"
```

7.2.1.3 version Attribute

The MIME version attribute is used to identify the particular version of ebXML Message Envelope being used. All message headers SHOULD USE "0.8". For example:

```
version="0.8"
```

7.2.2 ebXML Message Envelope Example

An example of a compliant *ebXML Message Envelope* header appears as follows:

```
Content-Type: multipart/related; type="application/vnd.eb+xml";"boundary:="-------8760";
```

7.3 ebXML Header Container

The *ebXML Header Container* is a MIME body part that MUST consist of:

- one XML based ebXML Header Envelope, and
- one XML ***ebXML Header Document*** (described in section 8 of this document)

The following rules apply:

- the *ebXML Header Container* MUST be the first MIME body part in the *ebXML Message.*
- there MUST be one and only one *XML ebXML Header Document* in each *ebXML Message*. However, an *ebXML Payload Container* may be a completely encapsulated *ebXML Message*.

The MIME based *ebXML Header Envelope* conforms to [RFC 2045] and MUST consist of the following MIME headers:

- `Content-ID`
- `Content-Type`

7.3.1 Content-ID

The `Content-ID` MIME header identifies this instance of an ebXML Message header body part. The value for `Content-ID` SHOULD be a unique identifier, in accordance with RFC 2045. For example:

```
Content-ID: <2000-0722-161201-123456789@ebxmlhost.realm>
```

7.3.2 Content-Type

The MIME `Content-Type` for an ebXML header is identified with the value "`application/vnd.eb+xml`". `Content-Type` contains two attributes:

- `version`
- `charset`

7.3.2.1 version Attribute

The MIME `version` attribute indicates the version of the ebXML Message Service Specification to which the *ebXML Header Document* conforms. For example:

```
version="0.8";
```

7.3.2.2 `charset` Attribute

The MIME `charset` attribute identifies the character set used to create the ebXML Header Document (an XML document that is the content of this MIME entity). The semantics of this attribute are described in [`charset` parameter / encoding considerations] of `application/xml` as specified in [XML/Media]. 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 ebXML Header Document (see section 8). If provided, the MIME `charset` attribute MUST NOT contain a value conflicting with the encoding used when creating the ebXML Header Document. For maximum interoperability it is RECOMMENDED that [UTF-8] be used when encoding this document. Due to the processing rules defined for media types derived from `application/xml` [XMLMedia], this MIME attribute has no default. For example:

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

7.3.3 ebXML Header Envelope Example

The following represents an example of an *ebXML Header Envelope* and *ebXML Header*

Document:

```
Content-ID: ebxmlheader-123@ebxmlhost.realm --|    MIME ebXML    |
Content-Type: application/vnd.eb+xml;         | Header Envelope  |
version="0.9"; charset="UTF-8"             --|                  | ebXML
                                             |                  | Header
<ebXMLHeader>                    -------------|                  | Container
  <Manifest>........                         | XML ebXML Header |
  </Manifest>                                |     Document     |
  <Header>........                           |                  |
  </Header>                                  |                  |
  <Routing Header>........                   |                  |
  </Routing Header>                          |                  |
</ebXMLHeader>                   -------------|                  |
```

A complete example of an *ebXML Header Container* is presented in Appendix B. That example includes the `charset` attribute and portions of an *XML Prolog* (see sect 8.1), none of which is required to appear in an ebXML Header Container or ebXML Header Document. Appendix B also includes the outer ebXML Message Envelope and a complete (valid) `ebXMLHeader` element rather than the outline shown above.

7.4 ebXML Payload Container

If the *ebXML Message* contains a payload, then a single *ebXML Payload Container* MUST be used to envelop it.

If there is no payload within the *ebXML Message* then the *ebXML Payload Container* MUST not be present.

The contents of the *ebXML Payload Container* MUST be identified by the *Message Manifest* element within the *ebXML Header Document* (see section 8.3).

If the *Message Manifest* is an empty XML element, the ebXML Payload Container MUST NOT be present in the *ebXML Message*.

If an *ebXML Payload Container* is present, it MUST conform to MIME [RFC2045] and MUST consist of:

- a MIME header portion - the *ebXML Payload Envelope*, and
- a content portion - the payload itself that may be of any valid MIME type.

The *ebXML MIME Payload Envelope*, MUST consist of the following MIME headers:

- `Content-ID`
- `Content-Type`

The ebXML Message Service Specification makes no provision, nor limits in any way the structure or content of payloads. Payloads MAY be a simple-plain-text-object or complex nested multipart objects. This is the implementer's decision.

7.4.1 Content-ID

The `Content-ID` MIME Header is used to uniquely identify an instance of an *ebXML Message* payload body part. The value for `Content-ID` SHOULD be a unique identifier, in accordance with MIME [RFC 2045]. For example:

```
Content-ID: <2000-0722-161201-123456789@ebxmlhost.realm>
```

7.4.2 Content-Type

The MIME `Content-Type` for an ebXML payload is determined by the implementer and is used to identify the type of data contained in the content portion of the *ebXML Payload Container*. The MIME Content-Type MUST conform to [RFC2045]. For example:

```
Content-Type: application/xml
```

7.4.3 Example of an ebXML MIME Payload Container

The following represents an example of an *ebXML MIME Payload Envelope* and a payload:

```
Content-ID: ebxmlpayload-123@ebxmlhost.realm --|
ebXML MIME      |
Content-Type: application/xml     -------------| Payload Envelope | ebXML
                                                                   | Payload
<Invoice>                         -------------|                  | Container
  <Invoicedata>........                        | Payload          |
  </Invoicedata>                               |                  |
</Invoice>                        -------------|                  |
```

A complete example of the ebXML Payload Container is presented in Appendix XX.

8 Header Document

The ebXML Header Document is a single [XML] document with a number of principal header-elements. In general, separate principal-header elements are used where:

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

8.1 XML Prolog

The ebXML Header Document's XML Prolog MAY contain an XML declaration or a document type declaration. This specification has defined no additional comments or processing instructions that may appear in the XML prolog. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
<ebXMLHeader>...</ebXMLHeader>
```

8.1.1 XML Declaration

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

8.1.2 Encoding Declaration

If both are present, the XML prolog for the ebXML Header Document SHALL contain the encoding declaration that SHALL be equivalent to the `charset` attribute of the MIME Content-Type of the ebXML

Message Header Container (see section 7.3). If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding used when creating the ebXML Header Document. It is RECOMMENDED that UTF-8 be used when encoding the ebXML Header Document. If the character encoding cannot be determined by an XML processor using the rules specified in section 4.3.3 of [XML], the XML declaration and its contained encoding declaration SHALL be provided in the ebXML Header Document.

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

For example:

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

8.1.3 Standalone Document Declaration

The standalone document declaration, if present, MAY appear as standalone='yes' if and only if all of the validity requirements specified in section 2.9 of the XML Recommendation [XML] are met. It is RECOMMENDED that ebXML Header Documents omit this declaration.

8.1.4 Document Type Declaration

When the ebXML Header Document will or may be processed by an XML processor not complaint with the XML Schema Recommendation [XMLSchema], a document type declaration

containing a SYSTEM identifier of "level1-10122000.dtd" MUST be included. For example:

```
<!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
```

8.2 ebXMLHeader Element

The root element of the *XML ebXML Header Document* is named ***ebXMLHeader***. Its structure is described below.

8.2.1 ebXMLHeader attributes

There are two attributes associated with the ***ebXMLHeade***r, they are as follows:

- ***Namespace (xmlns)***
- ***version***

8.2.1.1 Namespace attribute

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

8.2.1.2 version attribute

The ***version*** attribute is required. Its purpose is to provide for future versioning capabilities. It has a default value of '0.9'.

8.2.2 ebXMLHeader elements

An ebXML Header Document consists of the following principal header elements:

- ***Manifest*** – an element that points any data present either in the *ebXML Payload Container* or elsewhere, e.g. on the web
- ***Header*** – a REQUIRED element that contains routing information for the message (To/From, etc.) as well as other context information about the message
- ***RoutingHeaderList*** – a REQUIRED element that contains entries that identify the Message Service Handler (MSH) that sent and should receive the message
- ***ApplicationHeaders*** – an element that can be used by a process or service to include additional information that needs to be associated with the data in the *ebXML Payload* but is not contained within it
- ~~? ***MessageStatusRequest*** – an element that is used by a MSH when requesting the status of a message that was previously sent~~
- ***MessageStatusResponse Status Data*** – an element that is used by a MSH when responding to a request on the status of a message that was previously received
- ***ErrorList*** – an element that contains a list of the errors that have been found in a message
- ***Acknowledgment*** – an element that is used by a MSH to indicate that a message has been received
- ***Signature*** – an element that contains a digital signature that conforms to [XMLDSIG] that signs data associated with the message
- ***#wildcard*** - any namespace-qualified element content belonging to a foreign namespace

8.2.3 Combining Principal Header Elements

This section describes how the various principal header elements may be used in combination.

8.2.3.1 Manifest element

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

8.2.3.2 Header element

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

8.2.3.3 RoutingHeaderList element

The ***RoutingHeaderList*** element MAY be present in any message. It MUST be present if the message is being sent reliably (see section 9.2.3).

8.2.3.4 ApplicationHeaders element

The ***ApplicationHeaders*** element MAY be present on any message except a message that contains one or more of the following:

- an ***ErrorList*** element with a ***highestSeverity*** attribute set to ***Error***. See also section 8.2.3.6
- a ***MessageStatusRequest*** element
- a ***MessageStatusResponse*** element

8.2.3.5 StatusData element

This element MUST NOT be present with the following elements:

- a ***Manifest*** element
- an ***ApplicationHeaders*** element,
- ~~***MessageStatusResponse*** element~~an ***ErrorList*** element with a ***highestSeverity*** attribute set to ***Error***. (See section 8.2.3.6.)

8.2.3.6 ErrorList element

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

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

- a ***Manifest*** element
- an ***ApplicationHeaders*** element
- a ***StatusData*** element

8.2.3.7 Acknowledgment element

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

8.2.3.8 Signature element

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

8.2.3.9 #wildcard element content

Any namespace-qualified element content MAY be added to provide for the extensibility of the ebXMLHeader. Extension element content MUST be namespace-qualified in accordance with

[XMLNamespaces] and MUST belong to a foreign namespace. A foreign namespace is one that is NOT http://www.ebxml.org/namespaces/messageHeader.

8.2.4 ebXMLHeader sample

The following is a sample ***ebXMLHeader*** document fragment demonstrating the overall structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<ebXMLHeader xmlns=”http://www.ebxml.org/namespaces/messageHeader” Version="0.8" >
<Manifest>...</Manifest>
<Header>...</Header>
<RoutingHeaderList>…</RoutingHeaderList>
</ebXMLHeader>
```

8.3 Manifest element

The ***Manifest*** element is a composite element consisting of zero or more ***Reference*** elements. Each ***Reference*** element identifies data associated with the message, whether included as part of the message, or remote resources accessible via a URL. The ***Manifest*** element, if present, SHALL be the first child element of the ***ebXMLHeade***r. It identifies the payload document(s) contained in the *ebXML Message Containe*r. The purpose of the ***Manifest*** is to make it easier to directly extract a particular document associated with the Message.

The Manifest element MAY have a single attribute: ***id*** that is an XML ID.

8.3.1 Reference element

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

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

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

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

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

8.3.1.1 Description element

The ***Description*** is an OPTIONAL textual description of the payload object referenced by the parent ***Reference*** element. The language of the description is defined by a required ***xml:lang***

attribute. The ***xml:lang*** attribute MUST comply with the rules for identifying languages specified in [XML]. This element is provided solely for the purpose of providing a human readable description of the payload object identified by the parent Reference element.

8.3.1.2 Schema element

The ***Schema*** element MAY be present as a child of the ***Reference*** element. It provides a means of identifying the schema, and its version, that defines the payload object identified by the parent ***Reference*** element. It has no element or text content. The Schema element contains the following attributes:

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

8.3.2 Manifest sample

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

```
<Manifest id="Manifest">
  <Reference id="pay01"
    xlink:href="cid:payload-1" xlink:label="PO"
    xlink:role="http://regrep.org/gci/purchaseOrder">
    <Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
      version="1.0"/>
  </Reference>
</Manifest>
```

8.4 Header element

The ***Header*** element immediately follows the ***Manifest*** element. It is REQUIRED in all ***ebXMLHeader*** documents. The ***Header*** element is a composite element comprised of the following subordinate elements:

- ***From***
- ***To***
- ***CPAId***
- ***ConversationId***
- ***Service***
- ***Action***
- ***ReliableMessagingInfo***
- ***MessageData***
- ***#wildcard***

The Header attribute MAY have an attribute: ***id*** that is of type XML ID.

8.4.1 From and To elements

The ***From*** element identifies the *Party* that originated the message. It is a logical identifier, that MAY take the form of an URI. The ***From*** element consists of a ***PartyId*** element.

The ***To*** element identifies the intended recipient of the message. As with ***From***, it is a logical identifier that is comprised of a ***PartyId*** element.

The ***PartyId*** element has a single attribute: ***type*** and a string value.

If the ***type*** attribute is present, then it indicates that the parties that are sending and receiving the message know, by some other means, how to interpret the content of the ***PartyId*** 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 PartyId element MUST be a URI [RFC 2396] otherwise there is an error.

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

```
<From>
  <PartyId type="MyNumberingScheme">1234567890123</PartyId>
</From>
<To>
  <PartyId">urn:dnb.com:duns:3210987654321</PartyId>
</To>
```

8.4.2 CPAId element

The ***CPAId*** is a string that identifies the *Collaboration Protocol Agreement* that governs the processing of the message. The CPAId MAY be an URI, possibly established by registering a CPA with the ebXML Registry, that identifies the CPA uniquely.

8.4.3 ConversationId element

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

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

8.4.4 Service element

The ***Service*** element identifies the service that SHOULD act on the payload in the message. It is specified by the designer of the service. The designer of the service may be:

- a standards organization, or
- an individual or enterprise

An URI MAY be used for the element content.

8.4.4.1 Type attribute

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

If the ***type*** attribute is not present, the content of the ***Service*** element MUST be a URI [RFC 2396] otherwise there is an error.

8.4.4.2 ebXML Message Service Header namespace

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

8.4.5 Action element

The ***Action*** identifies a process within a ***Service***, that processes the Message. ***Action*** SHALL be unique within the ***Service*** in which it is defined.

8.4.6 Description element

The ***Description*** element MAY be present as a child element of the Header. The language of the description is defined by a required ***xml:lang*** attribute. The ***xml:lang*** attribute MUST comply with the rules for identifying languages specified in [XML]. This element is provided solely for the purpose of providing a human readable description of the purpose or intent of the message.

8.4.7 ReliableMessagingInfo element

The last element of the ***Header*** is the ***ReliableMessagingInfo*** element. This element identifies the quality of service with which the message MUST be delivered. This element has a single attribute, ***deliverySemantics***. See section 9.2.3 for more details on how this element is used. An example of a ***ReliableMessagingInfo*** element follows.

```
<ReliableMessagingInfo deliverySemantics="OnceAndOnlyOnce"/>
```

8.4.8 MessageData element

The REQUIRED ***MessageData*** element follows the ***Action*** element within the Header element. The purpose of the ***MessageData*** element is to provide a means of uniquely identifying an *ebXML Message*. It is contains the following three elements:

- ***MessageID***
- ***TimeStamp***
- ***RefToMessageID***

8.4.8.1 MessageId element

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

8.4.8.2 TimeStamp element

The ***TimeStamp*** is a value representing the time that the message header was created conforming to [ISO-8601]. The format of CCYYMMDDTHHMMSS.SSSZ is REQUIRED to be used. This time format is Coordinated Universal Time (UTC).

8.4.8.3 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 message referenced, 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 8.8).

For Acknowledgment Messages, the ***RefToMessageId*** element is REQUIRED, and its value MUST be the ***MessageId value*** of the ebXML Message being acknowledged. See also sections 8.2.3.7 and 9.2.3.

8.4.9 #wildcard element

In support of allowing an ebXML Message to be extended to include element content from a foreign namespace, a ***#wildcard*** element has been provided. Additional element content MAY be added to the ***Header*** element immediately following the ***MessageData*** element. Such additional element content MUST be namespace-qualified in accordance with [XMLNamespaces].

8.4.10 Header sample

The following fragment demonstrates the structure of the ***Header*** element of the ***ebXMLHeader*** document:

```
<Header id="N01">
  <From>
    <PartyId type="uri">…</PartyId>
  </From>
  <To>
    <PartyId type="userType">...</PartyId>
  </To>
```

```
  <CPAId>http://www.ebxml.org/cpa/123456</CPAId>
  <ConversationId>987654321</ConversationId>
  <Service>urn:processdesigners.com:service:QuoteToCollect</Service>
  <Action>NewPurchaseOrder</Action>
  <ReliableMessagingInfo deliverySemantics="BestEffort"/>
  <MessageData>
    <MessageId>UUID-2</MessageId>
    <TimeStamp>20000725T121905.000Z</TimeStamp>
    <RefToMessageId>UUID-1</RefToMessageId>
  </MessageData>
</Header>
```

8.5 RoutingHeaderList element

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

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

- the message is being sent over a single hop (see section 8.5.2), and
- the message is not being sent reliably (see section 9.2.3)

8.5.1 Routing Header Element

The ***RoutingHeader*** element contains information about a single transmission of a message between two Parties. If a message traverses multiple hops by passing through some type of intermediate system between the *From Party* and the *To Party*, then each transmission over each hop results in the addition of a new Routing Header element.

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

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

8.5.1.1 SenderURI element

This element contains the URI of the messages' Sender Messaging Service Handler. The recipient of the message, unless there is another URI more specifically identified within the CPA, uses the URI to send a message, when required that:

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

8.5.1.2 ReceiverURI element

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

8.5.1.3 ErrorURI element

This URI, if present, identifies the URI that is used for reporting errors. If it is not present then errors are reported by sending a message to the ***SenderURI.***

8.5.1.4 Timestamp element

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

8.5.1.5 #wildcard

This MAY contain any namespace-qualified element content belonging to a foreign namespace.

8.5.2 Single Hop Routing Header Sample

A single hop message and its return is illustrated by the diagram below.

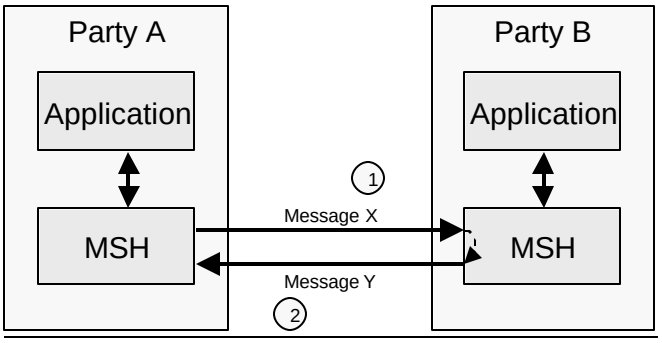


Figure 8-1 Single Hop Message

The content of the corresponding messages could include:

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

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyA-id</From>
  <To>urn:myscheme.com:id:PartyB-id</From>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

?? Transmission 2 - Message Y From Party B To Party A

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyB-id</From>
  <To>urn:myscheme.com:id:PartyA-id</From>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:20:05.274Z-6</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

8.5.3 Multi-hop Routing Header Sample

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

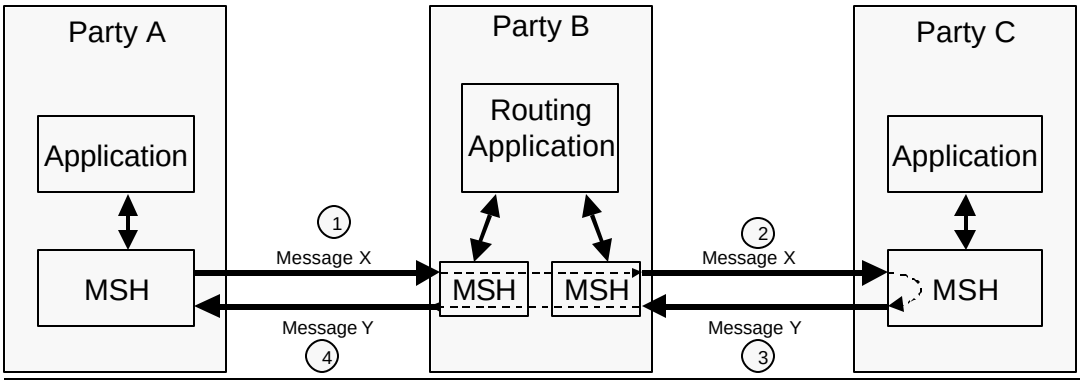


Figure 8-2 Multi-hop Message

The content of the corresponding messages could include:

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

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyA-id</From>
  <To>urn:myscheme.com:id:PartyC-id</From>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

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

```
<Header id="...">
  <From>urn:myscheme.com:id:PartyA-id</From>
  <To>urn:myscheme.com:id:PartyC-id</From>
  ...
</Header>
<RoutingHeaderList id="...">
  <RoutingHeader>
    <SenderURI>url:PartyA.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyB.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:35.145Z-8</Timestamp>
  </RoutingHeader>
  <RoutingHeader>
    <SenderURI>url:PartyB.com/PartyAMsh</SenderURI>
    <ReceiverURI>url:PartyC.com/PartyBMsh</ReceiverURI>
    <Timestamp>20001216T21:19:45.483Z-6</Timestamp>
  </RoutingHeader>
</RoutingHeaderList>
```

Message Y would be similar to Message X except that the direction of transmission is reversed.

8.6 ApplicationHeaders Element

In support of allowing an ebXML Message to be extended to include application or implementation specific information, an ***ApplicationHeaders*** container element has been provided. Element content from a foreign namespace MAY be added to the ***ApplicationHeaders*** element. Such additional element content MUST be namespace-qualified in accordance with [XMLNamespaces].

An MSH implementation MUST make the information content of the ***ApplicationHeaders*** element available to the application or application services layer of software as described in the Service Interface section.

The ***ApplicationHeaders*** element has a single attribute: ***mustUnderstand***. This attribute has two possible values: ***true*** and ***false***. The default value for the ***mustUnderstand*** attribute is ***false***. An ***ApplicationHeaders*** element that has a ***mustUnderstand*** set to a value of 'true' means that a receiving MSH MUST be capable of understanding the meaning of the namespace-qualified element content. If not, the receiving MSH MUST respond with a message that includes an error code of ***NotSupported*** in an ***ErrorData*** element as defined in section 8.8.

8.6.1 ApplicationHeaders sample

```
<ApplicationHeaders mustUnderstand="true">
  <foo:ProprietaryStuff
    xmlns:foo="http://www.example.com/ebxml-msh-extensions">…
  </foo:ProprietaryStuff>
</ApplicationHeaders>
```

8.7 StatusData Element

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

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

- a ***RefToMessageId*** element that contains the ***MessageId*** of the message whose status is being checked
- a ***ReceiptTimeStamp*** element. This contains the time that the message whose status is being checked was received. This MUST be omitted if the message whose status is being checked is ***NotRecognized*** or the request was ***UnAuthorized***
- a ***ForwardURI*** element. This MAY only be present if ***messageStatus*** is set to ***Forwarded***. If present it indicates the URI of the ***ReceiverURI*** to which the message was forwarded
- a ***messageStatus*** attribute that is set to one of the following values:
  - ***UnAuthorized*** – the Message Status Request is not authorized or accepted
  - ***NotRecognized*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element is not recognized
  - ***Received*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element has been received by the MSH, but has not been processed by an application or forwarded to another MSH
  - ***Processed*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element has been received by the MSH for the To Party on the original message, and has been passed to the application or other process that is to handle it
  - ***Forwarded*** – the message identified by the ***RefToMessageId*** element in the ***StatusData*** element has been received by the MSH, and has been forwarded to another MSH

8.8 ErrorList Element

The existence of an ***ErrorList*** element indicates that the message that is identified by the ***RefToMessageId*** in the header has an error.

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

- ***id*** attribute
- ***highestSeverity*** attribute

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

8.8.1 id attribute

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

8.8.2 highestSeverity attribute

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

8.8.3 Error element

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

- ***codeContext*** attribute
- ***errorCode*** attribute
- ***severity*** attribute
- ***location*** attribute
- ***xml:lang*** attribute
- ***errorMessage*** attribute
- ***softwareDetails*** attribute

8.8.3.1 codeContext attribute

The ***codeContext*** attribute identifies the namespace or scheme for the ***errorCodes***. It MUST be a URI. Its default value is `http://www.ebxml.org/messageServiceErrors`. If it is present then it indicates that an implementation of this specification has used its own ***errorCodes***.

Use of non ebXML values for ***errorCodes*** is NOT RECOMMENDED. In addition, an implementation of this specification MUST NOT use its own ***errorCodes*** if an existing ***errorCode*** as defined in section 8.8.5 has the same or very similar meaning.

8.8.3.2 errorCode attribute

The required ***errorCode*** attribute indicates the nature of the error in the *message in error*. Valid values for the ***errorCode*** and a description of the code's meaning are given in section 8.8.5.

8.8.3.3 severity attribute

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

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

8.8.3.4 location attribute

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

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

If the ebXML Header document is not "well formed" then the location attribute MUST be omitted.

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

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

8.8.3.5 errorMessage atttribute

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

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

The ***errorMessage*** attribute MAY be omitted.

8.8.3.6 softwareDetails attribute

The ***softwareDetails*** attribute contains a value that is set by the vendor/developer of the software that generated the ***Error*** element. It SHOULD contain data that enables the vendor/developer as well as the recipient of the message to identify the precise location in their software and the set of circumstances that caused the software to generate a *message reporting the erro*r. It is RECOMMENDED that this element include plain text separated by punctuation to identify:

- the name of the software vendor;
- the name, version and release number of the software that generated the ebXML Error Document
- the part of the software that caused the error to be generated that can be used by the Software Vendor to identify the circumstances that caused the error

If any part of the ***softwareDetails*** attribute contains text that is readable by a human, then it SHOULD be in the language identified by ***xml:lang***.

8.8.4 Examples

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

```
<ErrorList id='3490sdo9', highestSeverity="error">
  <Error errorCode='UnableToParse', severity="Error", location=cid:21398adhiwqe,
xml:lang="us-en", errorMessage='XSD parser error – document not parsable",
softwareDetails='Software Development Corp.; ebXML Connector!!; v2.7, build 2.7313; Ref
HA'/>
  <Error .... />
</ErrorListr>
```

8.8.5 errorCode values

This section describes the ***ErrorCodes*** (see section8.8.3.2) that are used in a *message reporting an erro*r. They are described as a list of bullet points. The following describes how to interpret this list:

- the first word is the actual ***errorCod***e, e.g. ***UnableToParse***
- the single sentence that immediately follows the error code is an "error code description" of the ***errorCode***. Note that this narrative MUST NOT be used in the ***errorMessage*** attribute.
- the sentence(s) that follow the "narrative", are the explanation of the meaning of the error and provide guidance on when the particular ***ErrorCode*** should be used.

It is RECOMMENDED that implementers of software that conforms to this specification make available to a user that is being informed of the error: the value of the ***errorCode***, the "error code description" and the "narrative".

It is also RECOMMENDED that the "error code description" and the "narrative" are translated into the preferred language of the user if this is known.

8.8.6 Reporting Errors in the ebXML Header Document

The following list contains error codes that can be associated with the ebXML Header Document:

- ***UnableToParse*** - XML not well formed or invalid. The XML document is not well formed or not valid and cannot be successfully parsed. See [XML] for the meaning of "well formed" and "not valid".
- ***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 recognized and therefore could not be used by the ebXML Message Service
- ***NotSupported*** - Element or attribute not supported. Although the document is well formed and valid, an element or attribute is present that:
  - is consistent with the rules and constraints contained in this specification, but
  - is not supported by the ebXML Message Service that is processing the message.
- ***Inconsistent*** - Element content or attribute value inconsistent with other elements or attributes. Although the document is well formed and valid, according to the rules and constraints contained in this specification the content of an element or attribute is inconsistent with the content of other elements or their attributes.
- ***OtherXml*** - Other error in an element content or attribute value. Although the document is well formed and valid, the element content or attribute value contains values that do not conform to the rules and constraints contained in this specification and is not covered by other error codes. The ***errorMessage*** attribute should be used to indicate the nature of the problem.

8.8.7 Non-XML Document Errors

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

- ***MessageTooLarge*** - Message too large. The message is too large to be processed by the ebXML Message Service.
- ***MimeProblem*** - A MIME error has occurred. An error has been detected in the structure or format of a MIME part of the message. For example:
  - Missing MIME Part. Although the MIME message is correctly structured, a MIME part is missing that should have been present if the rules and constraints contained in this specification are followed
  - Unexpected MIME Part. Unexpected MIME part. Although the MIME message is correctly structured, a MIME part is present that is not expected in the particular context according to the rules and constraints contained in this specification
- ***DeliveryFailure*** – Message Delivery Failure. A message has been received that either probably or definitely could not be sent to its next destination
- ***TimeToLiveExpired*** – Message Time To Live Expired. A message has been received that arrived after the time specified in the ***TimeToLive*** element of the ***Header*** element
- ***Unknown*** - Unknown Error. Indicates that an error has occurred that is not covered explicitly by any of the other errors. The ***errorMessage*** attribute should be used to indicate the nature of the problem.

Note this list will be expanded in future versions of this specification, for example to report errors on security. If the *message being acknowledged* was sent with ***deliverySemantics*** set to ***OnceAndOnlyOnce***, then the *acknowledgment message* from the *To Party* might still arrive.

8.9 Acknowledgment Element

The ***Acknowledgment*** element is an optional element that is used by one Party to indicate that another Party has received a message. <CF>The use of Party in this context is going to be confusing! Some will think that a Business ack would be handled in this manner that I gather it would not be based on our discussions. I would suggest replacing party with MSH.</CF>

For clarity two terms are defined:

- *message being acknowledged*. This is the *Message* that is has been received by a party that is now being acknowledged
- *acknowledgment message*. This is the message that acknowledges that *message being acknowledged* has been receivedf.

The *message being acknowledged* is identified by the ***RefToMessageId*** contained in the ***MessageData*** element contained within the ***Header Element*** of the *acknowledgment message* containing the value of the ***MessageId*** of the *message being acknowledged*.

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

- ***ReceiptTimeStamp*** element
- ***From*** element
- ***type*** attribute
- ***signed*** attribute

8.9.1 TimeStamp element

The ***TimeStamp*** element is a value representing the time that the *message being acknowledged* was received by the Party generating the *acknowledgment message*. It must conform to [ISO-8601]. The format of CCYYMMDDTHHMMSS.SSSZ is REQUIRED to be used. This time format is Coordinated Universal Time (UTC).

8.9.2 From element

This is the same element as the *From* element defined in section 8.4.1. However, when used in the context of an Acknowledgment Element, it contains the identifier of the *Party* that is generating the *acknowledgment message*.

8.9.3 type attribute

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

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

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

8.9.4 signed attribute

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

- ***True*** - indicates that the *acknowledgment message* is digitally signed, or
- ***False*** - indicates that the *acknowledgment message* is not digitally signed

The default value for ***signed*** is ***False***.

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

8.10 Signature Element

TBD

9 Message Service Handler Services

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

- Message Status Request
- Message Service Handler Ping

Each service is described below:

9.1 Message Status Request Service

The Message Status Request Service consists of the following:

- sending a Message Status Request message to a MSH about a message previously sent
- the Message Service Handler that receives the request sending a Message Status Response message in return.

9.1.1 Message Status Request Message

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

- A ***Header*** element
- A ***RoutingHeaderList*** element
- A ***Signature*** element

The ***RoutingHeaderList*** and the ***Signature*** elements MAY be omitted (see sections 8.5 and 8.10).

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

- a ***From*** element that identifies the creator of the message status request message
- a ***To*** that identifies a Party that received the message. If a ***RoutingHeader*** was present on the message whose status is being checked then this MUST be a ***ReceiverURI*** from that message.
- a ***Service*** element that contains: ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- an ***Action*** element that contains ***Request***

The message is then sent to the To Party.

9.1.2 Message Status Response Message

Once the To Party on the Message Status Request message receives the message, they MAY generate a Message Status Response message that consists of no ebXML Payload and the following elements in the ebXML Header.

- a ***Header*** element
- a ***RoutingHeaderList*** element
- a ***Acknowledgement*** element
- a ***StatusData*** element
- a ***Signature*** element

The ***RoutingHeaderList***, ***Acknowledgement*** and ***Signature*** elements MAY be omitted (see sections 8.5, 8.9 and 8.10).

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

- a ***From*** element that identifies the creator of the Message Status Response message

- a ***To*** element that identifies a Party that generated the Message Status Request message
- a ***Service*** element that contains: ***http://www.ebxml.org/namespaces/messageService/MessageStatus***
- an ***Action*** element that contains ***Response***
- a ***RefToMessageId*** that identifies the Message Status Request message.

The message is then sent to the To Party.

9.1.3 Security Considerations

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

<DB> Do we want to allow the Message Status Response to include the original response to the message in the Payload?</DB><CF> quite possibly.</CF>

9.2 Message Service Handler Ping Service

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

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

9.2.1 Message Service Handler Ping Message

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

- A ***Header*** element
- A ***RoutingHeaderList*** element
- A ***Signature*** element

The ***RoutingHeaderList*** and the ***Signature*** elements MAY be omitted (see sections 8.5 and 8.10).

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

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

The message is then sent to the To Party.

9.2.2 Message Service Handler Pong Message

Once the To Party on the MSH Ping message receives the message, they MAY generate a Message Service Handler Pong (MSH Pong) message that consists of no ebXML Payload and the following elements in the ebXML Header.

- a ***Header*** element
- a ***RoutingHeaderList*** element
- an ***Acknowledgement*** element
- a ***Signature*** element

The ***RoutingHeaderList***, ***Acknowledgement*** and ***Signature*** elements MAY be omitted (see sections 8.5, 8.9 and 8.10).

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

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

The message is then sent to the To Party.

9.2.3 Security Considerations

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

10 Reliable Messaging

Reliable Messaging defines an interoperable protocol such that the two Messaging Service Handlers operated by a *From Party* and a *To Party* can "reliably" exchange messages that are sent using "reliable messaging" semantics.

"Reliably" means that the *From Party* can be highly certain that the message sent will be delivered to the *To Party*. If there is a problem in sending a message then the sender resends the message until either the message is delivered, or the sender gives up. If the message cannot be delivered, for example because there has been a catastrophic failure of the *To Party's* system, then the *From Party* is informed.

10.1.1 Persistent Storage and System Failure

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

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

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

10.1.2 Methods of Implementing Reliable Messaging

ebXML support for Reliable Messaging can be implemented in one of two ways:

- using the ebXML Reliable Messaging protocol, or
- using commercial *queuing transport protocol* software products that are designed to provide reliable delivery of messages using proprietary protocols.

Each of these are described below.

10.2 ebXML Reliable Messaging Protocol

The ebXML Reliable Messaging Protocol is used to implement Reliable Messaging when either:

- no *queuing transport protocol* products are available, or
- a decision has been made to use the ebXML Reliable Messaging Protocol on top of a *queuing transport protocol.*

Use of the ebXML Reliable Messaging Protocol is identified by the ***ReliableMessagingMethod*** parameter being set to ***ebXML*** (the default).

The remainder of this section describes the ebXML Reliable Messaging Protocol. In outline it involves:

- a *From Party* sending a message to the *To Party*
- the *To Party* returning another messages that references the first

This is illustrated by the figure below.

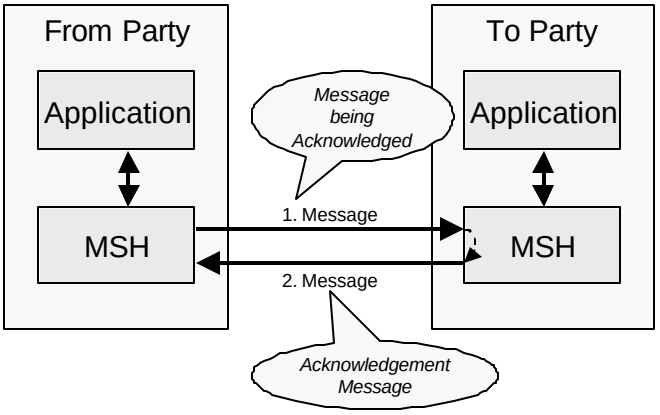


Figure 10-1 Indicating that a message has been received

The diagram above illustrates two terms that are used in the remainder of this section:

- *message being acknowledged*. This is the *Message* that needs to be sent reliably and therefore needs to be acknowledged
- *acknowledgment message*. This is the message that acknowledges that the *message being acknowledged* has been received.

The receipt of the *acknowledgment message* indicates that the *message being acknowledged* has been sent reliably.

A Message can be sent reliably either over:

- a *Single-hop* i.e. the sending of a message directly from the *From Party's* MSH to the *To Party's MSH* without passing through any intermediate MSHs.
- *Multi-hops* i.e. the sending of a message indirectly from the *From Party's* MSH to the *To Party's MSH* via an intermediate MSHs that is not owned by or operated by on behalf of either the *From Party* or the *Two Party.*

Multi-hop Reliable Messaging can work either with, or without, *Intermediate Acknowledgments*. See also section 8.5on Routing Headers

Single-hop Reliable Messaging is described first followed by Multi-hop Reliable Messaging. Note that Multi-hop Reliable Messaging is an extension of Single-hop reliable Messaging

10.2.1 Single-hop Reliable Messaging

Single-hop Reliable Messaging is illustrated by Figure 6-1 above.

In *Single-hop* Reliable Messaging:

- the *From Party* MSH sends to the *To Party* MSH, a message (the *message being acknowledged*) that contains:
  - a ***ReliableMessagingInfo*** element with ***deliverySemantics*** set to ***OnceAndOnlyOnce***
  - a ***RoutingHeader*** element that identifies the sender and the receipient URIs
- the *To Party* MSH returning to the *From Party* MSH, an *acknowledgment message* that contains:
  - a ***ReliableMessagingInfo*** element with ***deliverySemantics*** set to ***BestEfforts***
  - an ***Acknowledgment*** element with ***type*** set to ***deliveryReceipt***

10.2.1.1 Resending Lost Messages

This section describes the behavior that occurs when message are lost. For example, it is possible that a *message being acknowledged* was lost, for example:

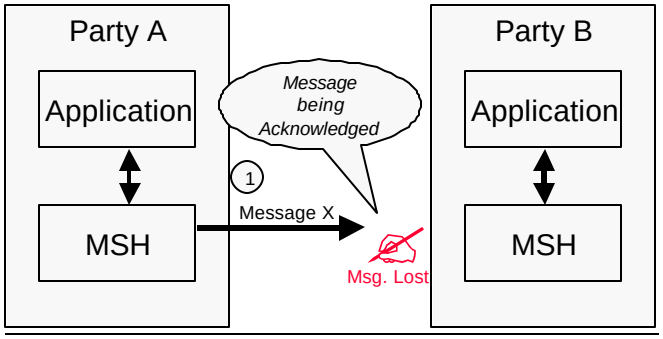


Figure 10-2 Lost "Message Being Acknowledged"

... or that the *Acknowledgment Message* was lost, for example ...

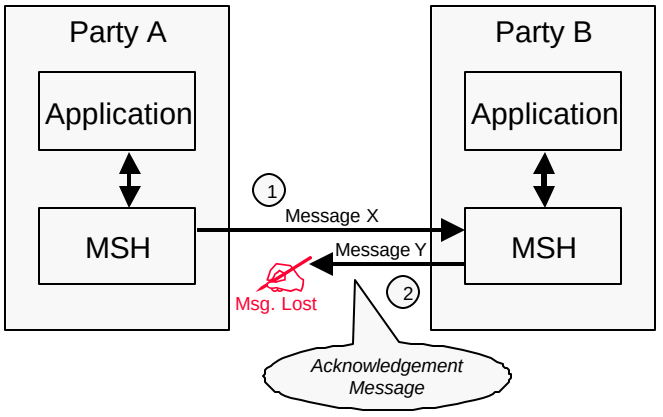


Figure 10-3 Lost Acknowledgment Message

In either case the sender of the original Message (Message X in the diagram) must resend the original message in the way illustrated by the diagram below if reliable messaging is being used.

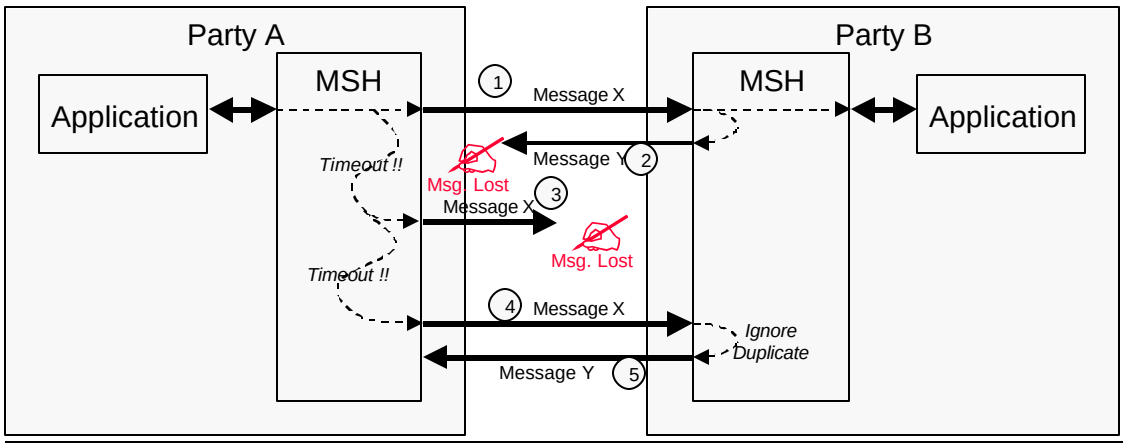


Figure 10-4 Resending Lost Messages

The diagram above shows the behavior that MUST be followed by the sender of the *message being acknowledged* (e.g. Message X) and the *acknowledgment message* (e.g. Message Y). Specifically:

1) The sender of the *message being acknowledged* (e.g. Party A) MUST re-send the *identical message* to the *To Party* MSH if no *Acknowledgment Message* is received

2) The recipient of the *message being acknowledged* (e.g. Party B), when it receives a *duplicate message*, MUST re-send to the sender of the *message being acknowledged* (e.g Party A), a message identical to the *most recent message* that was sent to the recipient (e.g. Party B)

3) The recipient of the *message being acknowledged* (e.g. Party A) MUST ignore *duplicate messages* and not forward them a second time to the application, the next MSH or other process that ultimately needs to receive them.

In this context:

- an *identical message* is a *message* that contains, apart from an additional ***RoutingHeader*** element, the same *ebXML Header* and *ebXML Payload* as the earlier *message* that was sent.
- a *duplicate message* is a *message* that contains the same ***MessageId*** as an earlier message that was received.
- the *most recent message* is the message with the latest ***Timestamp*** in the ***MessageData*** element that has the same ***RefToMessageId*** as the duplicate message that has just been received.

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

10.2.1.2 Duplicate Acknowledgment Messages

The uncertain nature of the delivery of messages over the Internet means that the Party MSH sending the *message being acknowledged* must check for duplicate *Acknowledgment Messages* as the diagram below illustrates.

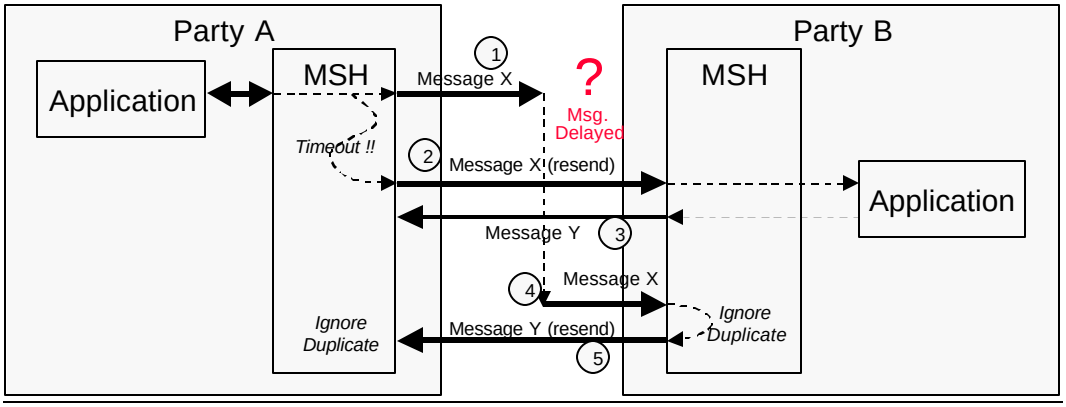


Figure 10-5 Handling Duplicate Acknowledgment Messages

In this example the *message being acknowledged* (message X) was delayed the first time it was sent (transmission 1). Meanwhile the *message being acknowledged* was resent by the sender (Party A) of the *message being acknowledged* (transmission 2) and processed by the recipient of the *message being acknowledged* (Party B) resulting in the *acknowledgment message* (message Y) being returned (tranmission 3). Later the original *message being acknowledged* from the sender of the *message being acknowledged* was received (transmission 4) and treated by the

recipient of the *message being acknowledged* as a duplicate that meant that the recipient of the *message being acknowledged* resent the *most recent message.*

To support this, the sender of a *Message being acknowledged* MUST ignore duplicate *Acknowledgment Messages* with the same ***MessageId*** and not process them a second time.

10.2.2 Multi-hop Reliable Messaging

Multi-hop reliable Messaging can occur either:

- without Intermediate Acknowledgment, or
- with Intermediate Acknowledgments

Each of these is described below.

10.2.2.1 Multi-hop Reliable Messaging without Intermediate Acknowledgments

Multi-hop Reliable Messaging without Intermediate Acknowledgment is identified by the ***IntermediateAckRequested*** of the *Routing Header* for the hop being set to ***False***.

The overall message flow is illustrated by the diagram below.

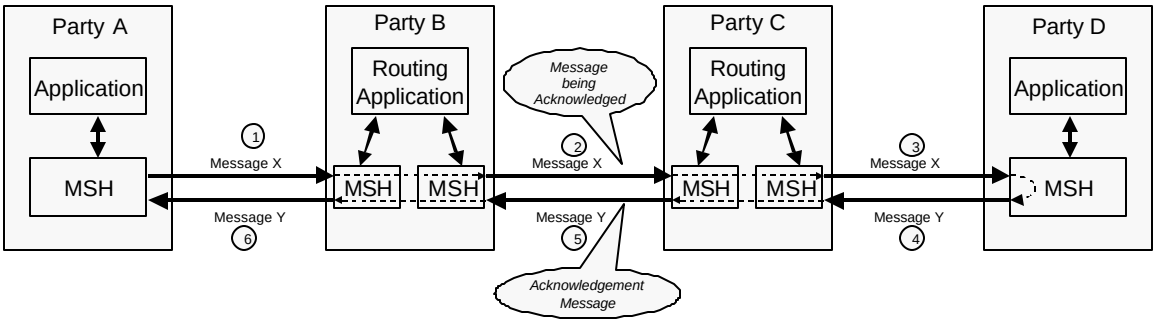


Figure 10-6 Multi-hop Reliable Messaging without Intermediate Acknowledgments

This is essentially the same a Single-hop Reliable Messaging except that the Message passes through multiple intermediate parties. This means that:

- the From Party (e.g. Party A) and the To Party (e.g. Party D) are the only parties that adopt the Reliable Messaging behavior described in this section
- the intermediate parties (e.g. Parties B and C), just forward the messages they receive, they do not undertake any Reliable Messaging behavior.

It is RECOMMENDED that Multi-hop Reliable Messaging without Intermediate Acknowledgments is used when the *From Party* that is sending a message is confident that the sum of the times taken for ...

- the *message being acknowledged* to be sent to the *To Party*, and
- the *acknowledgment message* to be returned

... is sufficiently short so that the *From Party* will not resend the *message being acknowledged.*

This is described in more detail below:

1) The *From Party* MSH (e.g. Party A) sends to an Intermediate Party (e.g. Party B) a message (the *message being acknowledged)* e.g. Message X in transmission 1, that contains
   a) a ReliableMessagingInfo element with deliverySemantics set to OnceAndOnlyOnce
   b) a RoutingHeader element that contains the SenderURI of the sender (e.g. the URL for Party A's MSH) and the ReceiverURI of the next recipient of the message (e.g. the URL of Party B's MSH)

2) Once the Intermediate Party (e.g. Party B or Party C) receives the message, they determine its next destination (in the example above this could be done by the Routing Application) and forward the message (e.g. Transmission 2 of Message X) to the next Party (e.g. either Party C or Party D). Before sending the message they:

   a) transfer elements in the ebXML Header and Payload unchanged from the inbound message to the outbound message except that, they

   b) add a ***RoutingHeader*** element to the ***RoutingHeaderList*** that contains the ***SenderURI*** of the next party to receive the message (e.g. the URL for Party C's or Party D's MSH) and the ***ReceiverURI*** (e.g. the URL for Party B's or Party C's MSH)

3) The previous step then repeats until eventually the message (e.g. Message X) reaches its final destination at the *To Party* (e.g. Party D)

4) Once the *To Party* receives the message (i.e. the *message being acknowledged)* they return an *acknowledgment message* to the *From Party* through the Intermediate Parties. The *acknowledgment message* (e.g. Message Y in transmission 4) contains:

   a) a **RefToMessageId** element that contains the **MessageId** of the message being acknowledged

   b) a **ReliableMessagingInfo** element with **deliverySemantics** set to **BestEfforts**

   c) an **Acknowledgment** element with type set to **deliveryReceipt**

   d) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL for Party D's MSH) and the **ReceiverURI** of the next recipient of the message (e.g. the URL of Party C's MSH)

5) Steps 2 and 3 above then repeat until the *acknowledgment message* reaches the *To Party* (e.g. Party A)

10.2.2.2 Multi-hop Reliable Messaging with Intermediate Acknowledgments

Multi-hop Reliable Messaging with Intermediate Acknowledgments is similar to Multi-hop Reliable Messaging without Intermediate Acknowledgment except that any of the Parties that are transmitting a Message may request that the recipient return an *Intermediate Acknowledgment.*

This is illustrated by the diagram below.

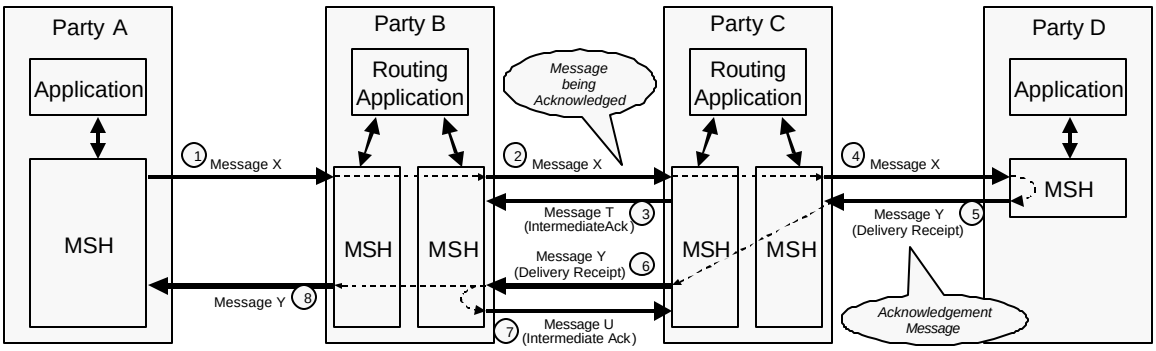


Figure 10-7 Multi-hop Reliable Messaging with Intermediate Acknowledgments

The main difference between Multi-Hop Reliable Messaging with Intermediate Acknowledgments and the without is:

- ?? any party may request an intermediate acknowledgment
- ?? any party that either sends or receives a message that requests an intermediate acknowledgment must adopt the reliable messaging behavior even if the ***ReliableMessagingInfo*** element indicates otherwise.

It is RECOMMENDED that Multi-hop Reliable Messaging with Intermediate Acknowledgments is used when the *From Party* that is sending a message is considers that the sum of the times taken for ...

- the *message being acknowledged* to be sent to the *To Party*, and
- the *acknowledgment message* to be returned

... is so long that the *From Party* will resend the *message being acknowledged*.

The rules that apply to Multi-hop Reliable Messaging with Intermediate Acknowledgment are as follows:

1) Any Party that is sending a message can request that the recipient send an *Acknowledgment Message* that is an *Intermediate Acknowledgment* by setting the ***IntermediateAckRequested*** of the ***RoutingHeader*** for the hop to ***True***. (e.g Transmission 2 of Message X, or Transmission 6 of Message Y)
2) If a MSH receives a message that is not the *To Party* that requires an Intermediate Acknowledgment (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y) then:
   a) If the MSH can identify itself as the ***ReceiverURI*** in the ***RoutingHeader*** for the hop, and
   b) an *Intermediate Acknowledgment* is requested, then
   c) the MSH must return an *Acknowledgment Message* (e.g. Transmission 3 of Message T, or Transmission 7 of Message U) with:
      i) a **RefToMessageId** element that contains the **MessageId** of the message being acknowledged
      ii) a **ReliableMessagingInfo** element with **deliverySemantics** set to **BestEfforts**
      iii) an **Acknowledgment** element with type set to **IntermediateAck**
      iv) a **RoutingHeader** element that contains the **SenderURI** of the sender (e.g. the URL for Party C's or Party B's MSH) and the **ReceiverURI** of the next recipient of the message (e.g. the URL of Party B's or Party C's MSH)
   d) Note. This applies even if the Reliable Messaging Info for the Message is set to ***BestEfforts***
3) If a MSH receives a message where it is *To Party* and it requires an Intermediate Acknowledgment (see step 2) then, unless the *To Party* is returning an *Acknowledgment Message* that is a *Delivery Receipt*, return an *Acknowledgment Message* as described in step 2c above.

10.3 ebXML Reliable Messaging using Queuing Transports

This section describes the differences that apply if a Queuing Transport is used to implement Reliable Messaging.

Use of the ebXML Reliable Messaging Protocol is identified by the ***ReliableMessagingMethod*** parameter being set to ***Transport*** for Transmission (either a Single-hop or a Multi-hop)

If Reliable Messaging using a Queuing Transport is being used then the following rules apply:

1) An Intermediate Ack SHOULD not be requested. If an Intermediate Ack is requested, then it is ignored.
2) No *message acknowledgments* with an ***Acknowledgment*** element with a ***type*** of ***IntermediateAck*** should be sent, even if requested
3) Implementations should use the facilities of the Queuing Transport to determine if the message was delivered

4) If an intermediate MSH cannot forward message to the next Party then the *From Party* should be notified using the procedure described in section 10.4.

5) An *acknowledgment message* with an ***Acknowledgment*** element with a ***type*** attribute set to ***deliveryReceipt*** may still be sent to inform the sender of the *message being acknowledged* that the message was delivered.

10.4 Failed Message Delivery

It is possible, that a Message cannot be delivered to its ultimate destination. This can be either:

- when the *To Party* MSH cannot deliver the message to the Application or other process that needs it, or
- when using Intermediate Acknowledgments and an Intermediate system determines that a message may have been lost. This is illustrated by the diagram below.

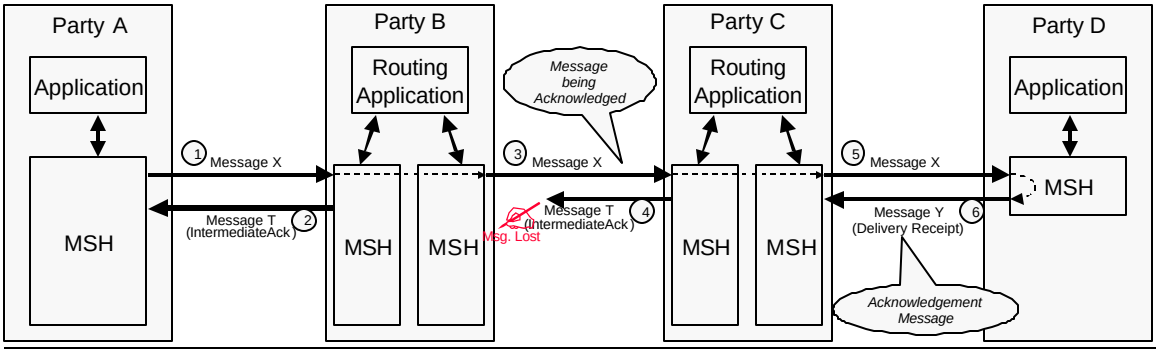


Figure 10-8 Failed Message Delivery using Intermediate Acknowledgments

In this example, Party B does not know if Party C (or Party D) has received the message since, even after resending, it has not received the *acknowledgment message* (Message T).

In both these circumstances the MSH that detects the problem MUST send a message to the *From Party* that sent the *message being acknowledged* (via the Intermediate Party if required). The message contains:

- a ***From Party*** that identifies the Party that detected the problem
- a ***To Party*** that identifies the ***From Party*** that created the message that could not be delivered
- a ***ReliableMessagingInfo*** element with ***deliverySemantics*** set to the same value as the ***deliverySemantics*** on the message that could not be delivered
- an ***ErrorData*** element with a severity of:
  - ***Error*** if the Party that detected the problem could not even transmit the message (e.g. Transmission 3 was impossible)
  - ***Warning*** if the message (e.g. Message X in Transmission 4) was transmitted, but no acknowledgment was received. This means that the message probably was not delivered although there is a small probability that it was
- an ***ErrorCode*** of ***DeliveryFailure***

This is illustrated by the diagram below by the text and arrows in red.

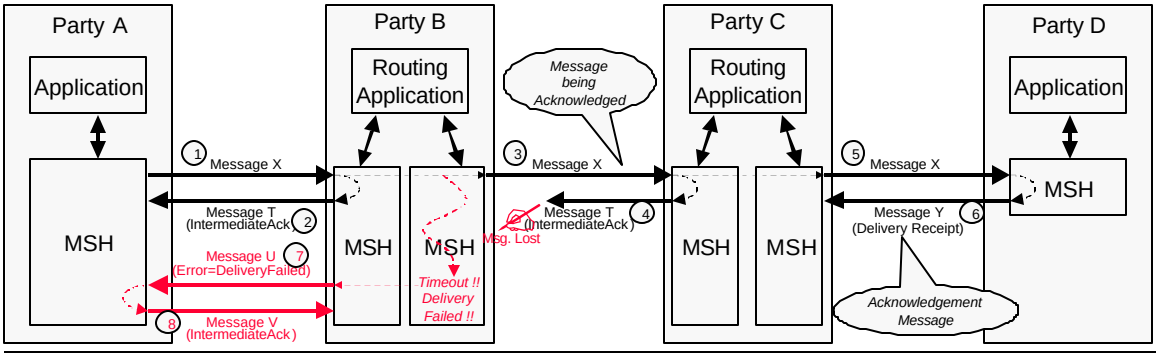


Figure 10-9 Reporting Failed Message Delivery

Note that the message that contains an ***ErrorData*** element with an ***ErrorCode*** of ***DeliveryFailure*** (e.g. Message U in Transmission 7) might be sent reliably. It is possible the *acknowledgment message* for this message (e.g. Message V in Transmission 8) is not received. In this case, the Party that detects the failed delivery (e.g. Party B) SHOULD inform the Party (e.g. Party A) that sent the *message being acknowledged* (e.g. Message X in Transmission 1) of the failure. How this is done is outside the scope of this specification.

10.5 Reliable Messaging Parameters

This section describes the parameters required to control reliable messaging. This parameter information may be contained:

- in the ebXML Message header, or
- in the CPA associated with the message.

If the information is in both the ebXML message header and the CPA, the information in the header over-rides the CPA.

10.5.1 Who sets Message Service Parameters

The values to be used in parameters can be specified by the following parties:

- the *From Party*
- the *To Party*
- the sending Message Service Handler (MSH)
- the receiving Message Service Handler

Parameters set by the *From Party* or the *To Party*, apply to the delivery of a message as a whole. Parameters set by the sending or receiving MSH apply to a single-hop.

Note that the *From Party* is the sending MSH and the *To Party* is the receiving MSH for the first/last MSH that handles the message.

The table below indicates where these parameters may be set.

Specified By	Parameter	CPA/ CPP	Message Header	Routing Header
From Party	`deliverySemantics`	Yes	Yes	N/A
From Party	`deliveryReceiptRequested`	Yes	Yes	N/A
From Party	`timeToLive`	Yes	Yes	N/A
From Party	`timeAccuracyRequired`	Yes	No	No

Specified By	Parameter	CPA/ CPP	Message Header	Routing Header
To Party	deliveryReceiptProvided	Yes	No	No
Sending MSH	reliableMessagingMethod	No	N/A	Yes
Sending MSH	intermediateAckRequested	No	N/A	Yes
Sending MSH	timeout	Yes	No	No
Sending MSH	retries	Yes	No	No
Sending MSH	retryInterval	Yes	No	No
Receiving MSH	reliableMessagingSupported	Yes	No	No
Receiving MSH	intermediateAckSupported	Yes	No	No
Receiving MSH	persistDuration	Yes	No	No
Receiving MSH	mshTimeAccuracy	Yes	No	No

In this table, the following interpretation of the columns should be used:

1) the ***Specified By*** columns indicates the Party that sets the value in the Collaboration Party Protocol, Message Header, or Routing Header

2) if the **CPA/CPP** column contains a **Yes** then it indicates that the party in the **Specified B** column specifies the value that is present in the CPP

3) if the **CPA/CPP** column contains a **No** then it indicates that the parameter value is never specified in the **CPP**

4) if the **Message Header/Routing Header** columns contain a **Yes** then it indicates that the parameter value may be specified in the **MessageHeader/Routing Header** and over-rides any value in the CPA. It the value is not specified in the **MessageHeader/Routing Header** then the value in the **CPA** must be used.

5) if the **Message Header/Routing Header** columns contain a **No** then it indicates that the value in the **CPA** is always used

6) if the **Message Header/Routing Header** columns contain a **N/A** then it indicates that the value may be specified in another header

These parameters are described below.

10.5.2 From Party Parameters

This section describes the parameters that are set by the *From Party*

10.5.2.1 Delivery Semantics

The ***deliverySemantics*** Parameter MUST used by the *From Party* to indicate whether the Message must be sent reliably. Valid Values are:

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

If no value is provided for ***deliverySemantics*** then the default value is ***BestEffort***.

If ***deliverySemantics*** is set to ***OnceAndOnlyOnce*** then if, for some reason a MSH cannot deliver the message to either directly or indirectly to the *To Party* MSH before ***TimeToLive*** has expired, then the MSH that detects the problem MUST send an Message to the *From Party*'s MSH indicating Delivery Failure. (See section 10.4)

If ***deliverySemantics*** is set to ***BestEffort*** then:

- a MSH that received a message that it is unable to deliver MUST NOT take any action to recover or otherwise notify anyone of the problem, and
- the MSH that sent the message must not attempt to recover from any failure.

This means that duplicate messages might be delivered to an application and persistent storage of messages is not required.

If the *To Party* is unable to support the type of Delivery Semantics requested, then the *To Party* SHOULD report the error to the *From Party* using an ***ErrorCode*** of ***NotSupported*** and a ***Severity*** of ***Error***.

10.5.2.2 Delivery Receipt Requested

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

The *From Party* SHOULD check the ***deliveryReceiptSupported*** parameter for the *To Party* before requesting a Delivery Receipt.

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

- ***Unsigned*** - requests that an unsigned Delivery Receipt is requeste4d
- ***Signed*** - requests that a signed Delivery Receipt is requested, or
- ***None*** - indicates that no Delivery Receipt is requested.

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

If the *To Party* is unable to support the type of Delivery Receipt requested, then the *To Party* SHOULD report the error to the *From Party* using an ***ErrorCode*** of ***NotSupported*** and a ***Severity*** of ***Error***.

10.5.2.3 Time To Live

TimeToLive is an optional element in the header that conforms to [ISO8601] and indicates the time by which a message should be delivered to the *To Party* MSH.

When setting a value for ***TimeToLive*** it is RECOMMENDED that the *From Party* takes into account the accuracy of its own internal clocks as well as the ***mshTimeAccuracy*** parameter for the Receiver MSH (see section 10.5.5.3) that indicates the accuracy to which a MSH will keep its internal clocks.

How a MSH ensures that its internal clocks are kept sufficiently accurate is an implementation decision.

Time To Live Expiry

If a MSH receives a Message where ***TimeToLive*** has expired the MSH MUST:

- send a *Message* to the *From Party* MSH, reporting that the ***TimeToLive*** of the message has passed
- NOT forward the message to another MSH or application/other system that should receive the message.

The *message* reporting the error MUST contain:

- an ***ErrorCode*** set to ***TimeToLiveExpired***
- a ***severity*** attribute set to ***Error***

In this context the ***TimeToLive*** has expired if the time of the internal clock of the MSH that receives a message is greater than the value ***TimeToLive*** for the *Message*

If ***TimeToLive*** is not present then it should be assumed that ***TimeToLive*** is infinite.

10.5.3 To Party Parameters

This section describes the parameters that are set by the *To Party*

10.5.3.1 DeliveryReceiptProvided

The ***DeliveryReceiptProvided*** parameter indicates whether a *To Party* can provide an *acknowledgment message* with a ***type*** attribute of ***deliveryReceipt*** in response to a message. Valid values are:

- ***Signed*** - indicates that only a signed Delivery Receipt can be provided
- ***Unsigned*** - indicates only an unsigned Delivery Receipt can be provided,
- ***Both*** - indicates that either a signed or an unsigned Delivery Receipt can be provided, or
- ***None*** - indicates that the *To Party* does not create Delivery Receipts

If a MSH receives a Message where ***deliveryReceiptRequested*** is in not compatible with the value of ***DeliveryReceiptProvided*** then the MSH MUST return an *Error Message* to the *From Party* MSH, reporting that the ***DeliveryReceiptProvided*** is not supported. This must contain:

- an ***ErrorCode*** set to ***NotSupported***
- a ***severity*** of ***Error***

10.5.4 Sending MSH Parameters

This section describes the parameters that are set by the *Party* that operates the Sending MSH.

10.5.4.1 Reliable Messaging Method

The ***ReliableMessagingMethod*** parameter indicates the requested method for Reliable Messaging that will be used when sending a Message. Valid values are:

- ***ebXML*** in this case the reliable messaging method as defined in section 10.2 is followed, or
- ***Transport***, in this case a Queuing Transport Protocol is used for reliable delivery of the message. See also section 10.3.

10.5.4.2 Intermediate Ack Requested

The ***intermediateAckRequested*** parameter is used by the Sending MSH to request that the Receiving MSH that receives the *Message* to return an *acknowledgment message* with an ***Acknowledgment*** element with a ***type*** of ***IntemediateAck*** to the Sending MSH.

Valid values for ***intermediateAckRequested*** are:

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

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

10.5.4.3 Timeout Parameter

The ***timeout*** parameter is an integer value that specifies the time in seconds that the Sending MSH MUST wait for an *Acknowledgment Message* before first resending a message to the Receiving MSH.

10.5.4.4 Retries Parameter

The ***retries*** Parameter is an integer value that specifies the maximum number of times the *message being acknowledged* must be resent to the Receiving MSH by the Sending MSH.

10.5.4.5 RetryInterval Parameter

The ***retryInterval*** parameter is an integer value specifying, in seconds, the time the Sending MSH MUST wait between retries, if an *Acknowledgment Message* is not received.

10.5.4.6 Deciding when to resend a message

The Sending MSH MUST resend the original message if an *Acknowledgment Message* has not been received from the Receiving MSH and either:

- the message has not yet been resent and at least the time specified in the ***timeout*** parameter has passed since the first message was sent, or
- the message has been resent, and
  - at least the time specified in the ***retryInterval*** has passed since the last time the message was resent, and
  - the message has been resent less than the number of times specified in the ***retries*** Parameter, and

If the Sending MSH does not receive an *Acknowledgment Message* after the maximum number of retries, the Sending MSH SHOULD notify either:

- the application and/or system administrator function if the Sending MSH is the *From Party* MSH, or
- send an message reporting the delivery failure, if the Sending MSH is operating by an Intermediate Party (see section 10.4)

10.5.5 Receiving MSH Parameters

This section describes the parameters that are set by the *Party* that operates the Receiving MSH.

10.5.5.1 Reliable Messaging Methods Supported

The ***reliableMessagingMethodsSupported*** parameter is a list of the methods that a MSH uses to support Reliable Messaging. It must be a URI. The URI for the ebXML Reliable Messaging Protocol described in section 10.2 is http://www.ebxml.org/namespaces/reliableMessaging

10.5.5.2 PersistDuration

PersistDuration is the minimum length of time in days that a Message that is sent reliably is kept in Persistent Storage by a MSH. The value used for ***PersistDuration*** is an implementation decision although it MUST be greater than the value of the ***TimeToLive*** parameter for any message that is sent.

If a duplicate message (i.e. with the same ***MessageId***) is received before the ***PersistDuration*** has passed, then the MSH that receives it MUST process it as a duplicate message as described in sections 10.2.1.1 and 10.2.1.2.

If a duplicate message is received after the ***PersistDuration*** has passed, then although it may be treated as a duplicate, the sender must realize that it will probably be treated by the MSH as if the message were a new message that had not been received before.

10.5.5.3 msh Time Accuracy

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

11 Error Reporting and Handling

This section describes how one ebXML Message Service Handler (MSH) reports errors it detects in an ebXML Message to another MSH.

11.1 Definitions

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

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

11.2 Types of Errors

One MSH needs to report to another MSH errors in *message in error* that are associated with:

- the structure or content of the *Message Envelope* (e.g. MIME),
- the ebXML Message Header document,
- security, or
- reliable messaging failures.

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

Errors associated with Data Communication protocols are detected and managed in an implementation specific way and are not part of this error reporting mechanism.

11.3 When to generate Error Messages

When an MSH detects an error in a *message in error*, a *message reporting the error* MUST be generated and delivered to the MSH that sent the *message in error* if:

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

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

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

11.3.1 Security Considerations

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

11.4 Identifying the Error Reporting Location

The Error Reporting Location is a URI that is specified by the sender of the *message in error* that indicates where to send a *message reporting the error*. This may be specified:

- by reference, for example by using the ***CPAId*** to identify the Party Agreement that contains the Error Reporting Location, or

- by value, for example by using the ***ErrorURI*** contained within the ***RoutingHeader*** element.

If a *message* contains both an ***ErrorURI*** then the ***ErrorURI*** MUST be used.

If an ***ErrorURI*** is not used then the ***ErrorURI*** implied by the CPA identified by the ***CPAid*** on the message SHOULD be used. If no ***ErrorURI*** is implied by the CPA, then the ***SenderURI*** MUST be used.

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

12 Security

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

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

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

Application of countermeasures SHOULD be balanced against an assessment of the inherent risks and the value of the asset(s) that might be placed at risk. <CF> need some reference to risk assessment sites/docs here.</CF>

12.1 Security and Management

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

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

12.2 Collaboration Party Profiles

The configuration of Security for MSH's is specified in the CPP. There are three areas of the CPP that have security definitions as follows:

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

12.3 RISKS

12.3.1 Unauthorized Access

One of the risks for Message Service Handlers is sending messages to or receiving messages from another message service handler that is not known or one that is being impersonated by a rogue MSH. Receiving a flood of requests from a known or unknown MSH can be considered a denial of service attack. Message Service Handlers need to be identified and need to be able to authenticate requests from other MSH's.

A message MAY carry information that authenticates the sending MSH in the message header. If authentication data is in the header, it MUST be protected from modification and inappropriate access.

Messages that are digitally signed MAY also be asserted as authenticated requests.

Authentication MAY also be provided by the underlying transport, such as through the use of [TLS].

12.3.2 Data Integrity and Confidentiality

Integrity protection is the term used to express a requirement that data MUST be protected from unauthorized modification while it is stored or passed over the network. The common technology for integrity protection is to generate a hash of the data and storing both the information and the hash securely. In a network protocol the hash is sent through a protected means and MAY be used to validate that the data received is the same data that was sent.

Privacy, or confidentiality, is the term used to express a requirement that data MUST be encrypted while it is stored or passed over the network. The common method for privacy protection is encryption via symmetric key algorithm like DES or triple-DES.

12.3.3 Denial-of Service

It is assumed that ebXML data and operations flow over the existing web infrastructure. All message services will implement their own web security infrastructure and practices. There are threats at all levels of the stack that need to be addressed through other means outside ebXML.

12.4 CounterMeasure Technologies

12.4.1 ebXML Message Countermeasures for Unauthorized Access and Data Integrity

12.4.2 Digital Certificates

The X.509 v3 standard describes an extensible framework within which basic certificate information MAY be extended. It also describes how such extensions MAY be used to control the process of issuing and validating certificates. Presently, there is no single view as to which certificate extensions must be present in an X.509 v3 digital certificate. The Collaboration Protocol Agreement identifies the particular X.509 v.3 certificate extensions that the parties to an agreement have agreed to use. An implementation of the ebXML Message Service MAY handle the subset of the certificate extensions listed in [S/MIME], but this capability is NOT REQUIRED. A Message Service that receives a message that contains critical extensions in an X509 v3 certificate that it is unable to handle MUST abandon processing of the message and return an Error XXX.

An implementation of the ebXML Message Service MAY implement a certificate-revocation list (CRL) retrieval mechanism. The purpose of a CRL is to gain access to certificate revocation information when validating certificate chains. It is RECOMMENDED that the Message Service retrieve and utilize CRL information each time a certificate is verified. The ultimate decision regarding use of the CRL information is left to the security policy of a party deploying an ebXML Message Service.

The use of a digital signature on an ebXML message satisfies the requirements for message integrity verification as well as authentication of the sender's identity. The digital signature also helps to establish the ebXML message non-repudiation property.

12.4.3 ebXML Message Countermeasures for Denial of Service

Message Service implementations SHOULD be able to immediately detect messages that MAY be a denial of service attack and take appropriate measures to reject these messages. Message Service implementations SHOULD be able to authenticate the claimed identity of a message sender when authentication is REQUIRED by the business.

12.4.4 ebXML Management Countermeasures for Denial of Service

It is STRONGLY RECOMMENDED that the site manager of an ebXML Message Service take appropriate measures to monitor announcements and descriptions of new attacks (See http://www.cert.org/) and apply updates and patches as appropriate.

12.5 Profiles

12.5.1 XML Digital Signature (XMLDSIG)

The joint W3C/IETF XMLDSIG Working Group has released the [XMLDSIG] specification as a Candidate Recommendation effective November, 2000. This means that the specification is made public for the purposes of encouraging implementations of the specification to validate that it can be successfully implemented. To date, there are at least three implementations of the specification in circulation, with others under development. It is anticipated that this specification, along with the recently initiated XML Encryption Working Group (also a joint W3C/IETF initiative) will be key technologies that MAY be employed by the ebXML Message Service.

The [XMLDSIG] specification defines how an XML document(s) MAY be signed, either in whole or as selective element content by means of a transformation such as [XPATH] or [XSLT].

12.5.2 Profile - XMLSignature signing of header and/or payload

An ebXML Message MAY be signed using technology that implements the [XMLDSIG] specification.

Blah blah blah blah, yadda yadda yadda.

12.5.2.1 Risks

This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED that this profile be used in conjunction with a secure transport that provides for authentication as well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S) would be such a transport mechanism.

12.5.2.2 Benefits

This profile provides the only means of signing both the header and payload objects. This profile also allows the message to be modified as it traverses through intermediary Message Service Handlers that MUST append ***RoutingHeader*** elements as the message is (re) sent on its path from the From Party to the To Party.

12.5.3 S/MIME

[S/MIME] names the message digest algorithms (md5, sha1), the public key encryption algorithm (RSA), and the bulk data encryption algorithms (RC2/40 and, optionally, Triple DES) that MUST be implemented in order to comply with the standard. An implementation of the ebXML Message Service that claims support for S/MIME SHALL conform to that standard.

The [S/MIME] specification REQUIRES that each MIME entity to be signed and/or encrypted MUST be converted to a canonical form that may be uniquely and unambiguously represented in both the environment where the signature is to be created and the environment where the signature is to be verified. MIME entities MUST be presented in a canonical format for enveloping as well as for signing.

The S/MIME specification RECOMMENDS transmitting entities such as 8-bit text and binary data to be encoded with quoted-printable or base-64 transfer encoding. This provision applies to formatting of the ebXML messages due to the transport independence property of the protocol.

Digital certificates are delivered as a part of the application/pkcs7-signature part of the multipart/signed message. [S/MIMECH] provides the guidelines for use of the digital certificates in S/MIME messages. The exact implementation of the certificate handling procedures and authentication semantics of the information in the digital certificate received with an ebXML message is left to the Trading Partner Agreement. *<CF> this needs work!</CF>*

12.5.4 Profile - S/MIME signing of message payload

The multipart/signed form defined by the [S/MIME] specification MAY be used to sign ebXML message payloads. This specification makes no claims as to how the signing and packaging of the payload object(s) is to be achieved. An implementation of the ebXML Message Service MAY choose to offer these services to the application or application service layers of software as described in the section on the Message Service Interface. However, this is not a REQUIRED feature of an ebXML Message Service.

This profile SHALL be uniquely identified by the following URI:

- http://www.ebxml.org/namespaces/security-profiles/smime-pkcs7-signed-payload

The [S/MIME] specification REQUIRES two parameters of the multipart/signed content type:

- protocol
- micalg

An ebXML message payload that is signed using this profile SHALL use the following values for these MIME parameters:

- protocol="application/pkcs7-signature
- micalg="rsa-sha1"

12.5.4.1 Sample S/MIME signed payload

```
Content-Type: multipart/related; type="application/vnd.eb+xml;
version="0.9"; boundary=ebxmlenvelopeuniquestring;
Content-Id: localpart@domain

--ebxmlenvelopeuniquestring
Content-Type: application/vnd.eb+xml; version="0.9"; charset="UTF-8";
Content-Id: localpart@domain

<?xml version="1.0" encoding="UTF-8"?>
<ebXMLHeader version="0.9"
xmlns=http://www.ebxml.org/namespaces/messageHeader>
…
</ebXMLHeader>

--ebxmlenvelopeuniquestring

Content-Type: multipart/signed; boundary="someuniquestring";
protocol="application/pkcs7-signature"; micalg="rsa-sha1";
Content-Id: localpart@domain

--someuniquestring
Content-Type: text/plain
Content-Id: localpart@domain

<Payload in the clear>

--someuniquestring
Content-Type: application/pkcs7-signed; name="smime.p7s";
Content-Id: localpart@domain
```

```
Content-Transfer-Encoding: base64

%^)*&TLYGSRKWHF

--someuniquestring--
--ebxmlenvelopeuniquestring--
```

12.5.4.2 Risks

This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED that this profile be used in conjunction with a secure transport that provides for authentication as well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S) would be such a transport mechanism.

The header document is unsigned and there is no binding of the header and payload.

12.5.4.3 Benefits

This is the simplest form of integrity, with application signing and authentication of the payload only.

12.5.5 Profile - S/MIME encryption of message payload

This profile SHALL be uniquely identified by the following URI:

- http://www.ebxml.org/namespaces/security-profiles/smime-pkcs7-encrypted-payload

The [S/MIME] specification REQUIRES two parameters of the multipart/signed content type:

- protocol
- micalg

An ebXML message payload that is signed using this profile SHALL use the following values for these MIME parameters:

- protocol="application/pkcs7-signature
- micalg="rsa-sha1"

12.5.5.1 Risks

The header document is unsigned and there is no binding of the header and payload.

12.5.5.2 Benefits

This is the simplest form of integrity, with application signing and authentication of the payload only.

12.5.6 PGP/MIME

[PGP/MIME] MAY be used to sign and/or encrypt an ebXML message payload object(s). An implementation of the ebXML Message Service that claims support for PGP/MIME SHALL conform to that standard.

12.5.7 Profile - PGP/MIME signing of message payload

TBD - Dick

12.5.7.1 Risks

This profile does not provide persistent privacy/confidentiality. It is STRONGLY RECOMMENDED that this profile be used in conjunction with a secure transport that provides for authentication as

well as encryption over the network such as is provided by [TLS]. HTTP over SSL (HTTP/S) would be such a transport mechanism.

The header document is unsigned and there is no binding of the header and payload.

12.5.7.2 Benefits

12.5.8 Profile - PGP/MIME encryption of message payload

TBD - Dick

12.5.8.1 Risks

12.5.8.2 Benefits

13 Synchronous and Asynchronous Responses

14 References

14.1 Normative References

[ISO 8601] International Standards Organization Ref. ISO 8601 Second Edition, Published 1997

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

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

[RFC 2396]

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

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

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

[XML] W3C XML 1.0 Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006

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

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

14.2 Non-Normative References

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

[XMTP] XMTP - Extensible Mail Transport Protocol http://www.openhealth.org/documents/xmtp.htm

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

[XMLMedia] IETF Internet Draft on XML Media Types. See http://www.imc.org/draft-murata-xml-08. Note. It is anticipated that this Internet Draft will soon become a RFC. Final versions of this specification will refer to the equivalent RFC.

[XMLSchema] W3C XML Schema Candidate Recommendation, http://www.w3.org/TR/xmlschema-0/ http://www. w3.org/TR/xmlschema-1/ http://www.w3.org/TR/xmlschema-2/

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

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

[S/MIME] RFC2311, “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.

1791 [PGP/MIME] RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins.
1792 October 1996.

1793

1794

15 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.

16 Contact Information

Team Leader

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

Vice Team Leader

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

Team Editor

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

Authors

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

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

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

Name	John Ibbotson
Company	IBM UK Ltd

Street		Hursley Park
City, State, Postal Code		Winchester SO21 2JN
Country		United Kingdom
Phone:		+44 (1962) 815188
Email:		john_ibbotson@uk.ibm.com
Name		Nicholas Kassem
Company		Java Software, Sun Microsystems
Street		901 San Antonio Road, MS CUP02-201
City, State, Postal Code		Palo Alto, CA 94303-4900
Phone:		+1 (408) 863-3535
E-mail:		Nick.Kassem@eng.sun.com
Name		Masayoshi Shimamura
Company		Fujitsu Limited
Street		Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
City, State, Postal Code		Kohoku-ku, Yokohama 222-0033, Japan
Phone:		+81-45-476-4590
E-mail:		shima@rp.open.cs.fujitsu.co.jp
Document Editing Team		
Name		Ralph Berwanger
Company		bTrade.com
Street		2324 Gateway Drive
City, State, Postal Code		Irving, TX 75063
Country		USA
Phone:		+1 (972) 580-2900
EMail:		rberwanger@btrade.com
Name		Ian Jones
Company		British Telecommunications
Street		Enterprise House, 84-85 Adam Street
City, State, Postal Code		Cardiff, CF24 2XF
Country		United Kingdom
Phone:		+44 29 2072 4063
EMail:		ian.c.jones@bt.com
Name		Martha Warfelt
Company		Daimler Chrysler Corporation
Street		800 Chrysler Drive
City, State, Postal Code		Auburn Hills, MI
Country		USA
Phone:		+1 (248) 944-5481 1210
EMail:		maw2@daimlerchrysler.com 1211

Appendix A Schema and Data Type Definitions

A.1 Schema Definition

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
xmlns:ds="http://www.w3.org/2000/10/xmldsig#" xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
    <xsd:import namespace="http://www.w3.org/2000/10/xmldsig#"
schemaLocation="http://www.w3.org/TR/2000/10/xmldsig-core-schema/xmldsig-core-
schema.xsd"/>

<!-- EBXML HEADER -->
    <xsd:element name="ebXMLHeader">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Manifest" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="Header"/>
                        <xsd:element ref="RoutingHeaderList" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="Acknowledgment" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="StatusData" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="ApplicationHeaders" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="ErrorList" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="ds:Signature" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="version" use="fixed" value="0.9" type="xsd:string"/>
                <xsd:anyAttribute namespace="##any" processContents="lax"/>
        </xsd:complexType>
    </xsd:element>

<!-- MANIFEST -->
    <xsd:element name="Manifest">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Reference" maxOccurs="unbounded"/>
                        <xsd:any namespace="##other" processContents="lax"/>
                </xsd:sequence>
                <xsd:attribute name="id" use="required" type="xsd:ID"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Reference">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Schema" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
                        <xsd:any namespace="##other" processContents="lax"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
                <xsd:attribute name="xlink:type" use="required" type="xsd:string"
value="simple"/>
                <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
                <xsd:attribute name="xlink:label" type="xsd:string"/>
                <xsd:attribute name="xlink:role" use="required" type="xsd:uriReference"/>
                <xsd:attribute name="xlink:title" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Schema">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:attribute name="location" use="required" type="xsd:string"/>
                        <xsd:attribute name="version" use="required" type="xsd:string"/>
```

```
                </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

<!-- HEADER -->
    <xsd:element name="Header">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="From"/>
                        <xsd:element ref="To"/>
                        <xsd:element ref="CPAId"/>
                        <xsd:element ref="ConversationId"/>
                        <xsd:element ref="Service"/>
                        <xsd:element ref="Action"/>
                        <xsd:element ref="MessageData"/>
                        <xsd:element ref="DeliveryReceiptRequested" minOccurs="0"
maxOccurs="1"/>
                        <xsd:element ref="TimeToLive" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="ReliableMessagingInfo" minOccurs="0"
maxOccurs="1"/>
                        <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
                        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="To">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                                <xsd:attribute name="type" type="xsd:string"/>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="CPAId" type="xsd:string"/>

    <xsd:element name="ConversationId" type="xsd:string"/>

    <xsd:element name="Service" type="xsd:string"/>

    <xsd:element name="Action" type="xsd:string"/>

    <xsd:element name="MessageData">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="MessageId"/>
                        <xsd:element ref="Timestamp"/>
                        <xsd:element ref="RefToMessageId" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="MessageId" type="xsd:string"/>

    <xsd:element name="DeliveryReceiptRequested" use="default" value="None"/>
        <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                        <xsd:enumeration value="Signed"/>
                        <xsd:enumeration value="UnSigned"/>
                        <xsd:enumeration value="None"/>
                </xsd:restriction>
        </xsd:simpleType>

    <xsd:element name="TimeToLive" type="xsd:timeInstant"/>

    <xsd:element name="ReliableMessagingInfo">
        <xsd:complexType>
```

```
                <xsd:simpleContent>
                        <xsd:attribute name="deliverySemantics" use="required"/>
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:NMTOKEN">
                                                <xsd:enumeration value="OnceAndOnlyOnce"/>
                                                <xsd:enumeration value="BestEffort"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

<!-- ROUTING HEADER LIST -->
    <xsd:element name="RoutingHeaderList">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="RoutingHeader" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="RoutingHeader">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="SenderURI"/>
                        <xsd:element ref="ReceiverURI"/>
                        <xsd:element ref="ErrorURI" minOccurs="0" maxOccurs="1"/>
                        <xsd:element ref="Timestamp"/>
                        <xsd:element ref="SequenceNumber" minOccurs="0" maxOccurs="1"/>
                        <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="reliableMessagingMethod"/>
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:NMTOKEN">
                                        <xsd:enumeration value="ebXML"/>
                                        <xsd:enumeration value="Transport"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                <xsd:attribute name="intermediateAckRequested"/>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:NMTOKEN">
                                <xsd:enumeration value="Signed"/>
                                <xsd:enumeration value="UnSigned"/>
                                <xsd:enumeration value="None"/>
                        </xsd:restriction>
                </xsd:simpleType>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="SenderURI" type="xsd:uriReference"/>

    <xsd:element name="ReceiverURI" type="xsd:uriReference"/>

    <xsd:element name="SequenceNumber" type="xsd:positiveInteger" minOccurs="0"
maxOccurs="1"/>

    <xsd:element name="ErrorURI" type="xsd:uriReference" minOccurs="0" maxOccurs="1"/>

<!-- APPLICATION HEADERS -->
    <xsd:element name="ApplicationHeaders" type="ApplicationHeaders"/>
    <xsd:complexType name="ApplicationHeaders">
        <xsd:sequence>
                <xsd:any namespace="##other" processContents="lax"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>

<!-- ACKNOWLEDGEMENT -->
    <xsd:element name="Acknowledgment">
```

```
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Timestamp"/>
                        <xsd:element ref="From" minOccurs="0" maxOccurs="1"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
                <xsd:attribute name="type" use="default" value="DeliveryReceipt"/>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:NMTOKEN">
                                <xsd:enumeration value="DeliveryReceipt"/>
                                <xsd:enumeration value="IntermediateAck"/>
                        </xsd:restriction>
                </xsd:simpleType>
                <xsd:attribute name="signed" type="xsd:boolean"/>
        </xsd:complexType>
    </xsd:element>

<!-- ERROR LIST -->
    <xsd:element name="ErrorList">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="Error" maxOccurs="unbounded"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="xsd:ID"/>
                <xsd:attribute name="highestSeverity" use="default" value="Warning"/>
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:string">
                                        <xsd:enumeration value="Warning"/>
                                        <xsd:enumeration value="Error"/>
                                </xsd:restriction>
                        </xsd:simpleType>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Error">
        <xsd:complexType>
                <xsd:attribute name="codeContext" use="required" type="xsd:uriReference"/>
                <xsd:attribute name="errorCode" use="required" type="xsd:string"/>
                <xsd:attribute name="severity" use="default" value="Warning"/>
                        <xsd:simpleType>
                                <xsd:restriction base="xsd:NMTOKEN">
                                        <xsd:enumeration value="Warning"/>
                                        <xsd:enumeration value="Error"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                <xsd:attribute name="location" type="xsd:string"/>
                <xsd:attribute name="xml:lang" type="xsd:language"/>
                <xsd:attribute name="errorMessage" type="xsd:string"/>
                <xsd:attribute name="softwareDetails" type="xsd:string"/>
        </xsd:complexType>
    </xsd:element>

<!-- STATUS DATA -->
    <xsd:element name="StatusData">
        <xsd:sequence>
                <xsd:element ref="RefToMessageId"/>
                <xsd:element ref="Timestamp" minOccurs="0" maxOccurs="1"/>
                <xsd:element name="ForwardURI" type="xsd:uriReference" minOccurs="0"
maxOccurs="1"/>
        </xsd:sequence>
        <xsd:attribute name="messageStatus"/>
                <xsd:simpleType>
                        <xsd:restriction base="xsd:NMTOKEN">
                                <xsd:enumeration value="UnAuthorized"/>
                                <xsd:enumeration value="NotRecognized"/>
                                <xsd:enumeration value="Received"/>
                                <xsd:enumeration value="Processed"/>
                                <xsd:enumeration value="Forwarded"/>
                        </xsd:restriction>
                </xsd:simpleType>
    </xsd:element>
```

```
<!-- COMMON ELEMENTS -->
    <xsd:element name="From">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                                <xsd:attribute name="type" type="xsd:string"/>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="Description">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                                <xsd:attribute name="xml:lang" type="xsd:NMTOKEN"/>
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="RefToMessageId" type="xsd:string"/>

    <xsd:element name="Timestamp" type="xsd:timeInstant"/>
<!-- Does timeInstant conform to ISO 2601? -->

</xsd:schema>
```

A.2 Data Type Definition

Appendix BExamples

Appendix CCommunication Protocol Interfaces

This Appendix describes how the ebXML Message Service messages are carried by Communication Protocols. Two protocols are supported:

- Hypertext Transfer Protocol – HTTP/1.1, in both asynchronous and synchronous forms, and
- SMTP – Simple Mail Transfer Protocol

C.1 HTTP

This section describes how to transport ebXML compliant messages of [HTTP]. This can work in one of the following two ways:

- asynchronously, where the response to a message is sent using a separate HTTP POST, and
- synchronously, where the response to a message is sent on the HTTP RESPONSE returned from an HTTP POST

These are described below.

C.1.1 Asynchronous HTTP

In Asynchronous HTTP, all ebXML Message Service messages are carried by an HTTP Request Message (POST method). The HTTP Response Message to an HTTP Request Message has no entity body. This is illustrated by the figure below.

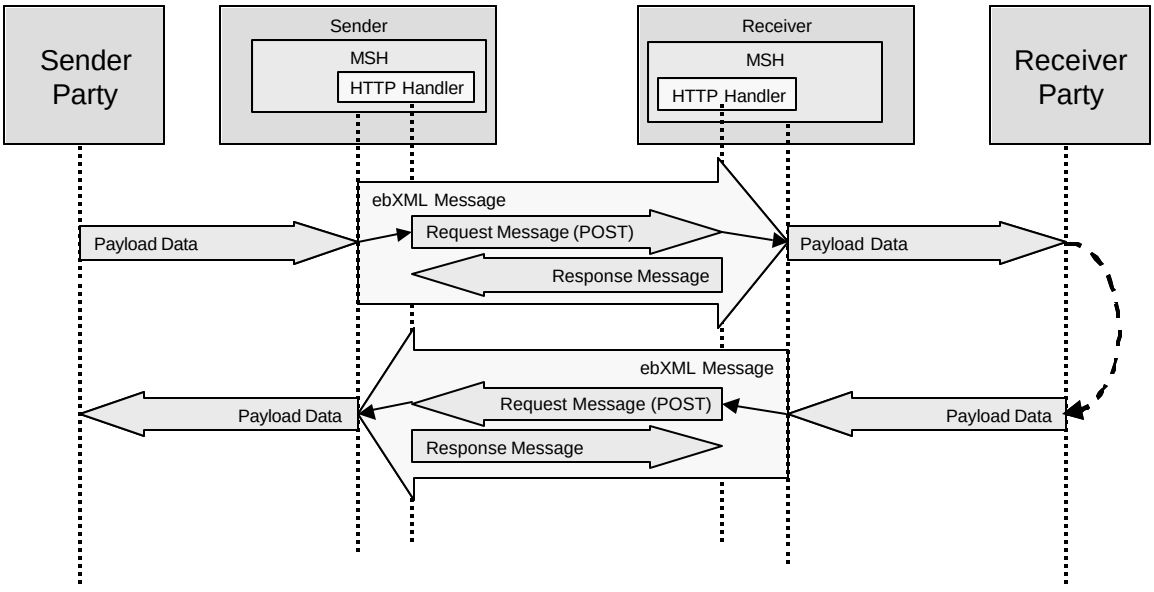


Figure C.1 Asynchronous HTTP Message Flow

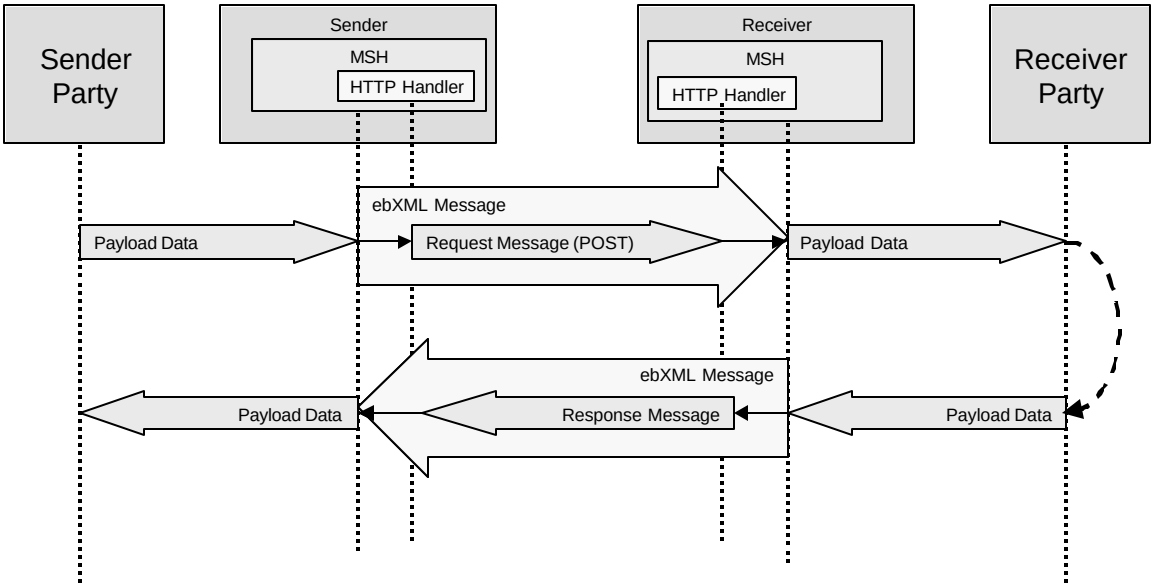
A message that is being sent asynchronously MAY be identified by the following HTTP header:

```
ebxmlresponse=asynchronous
```

2226 If the `ebXMLresponse` HTTP parameter is omitted then it MUST be assumed that the response
2227 is sent asynchronously.

2228 C.1.2 Synchronous HTTP

2229 In Synchronous HTTP, one ebXML Message Service message is carried by an HTTP Request
2230 Message (POST method) with the ebXML Message that is a response to the first message sent
2231 in the HTTP Response Message to the HTTP Request Message. This is illustrated by the figure
2232 below.



2233

2234 **Figure C.2 Synchronous HTTP Message Flow**

2235 If a response is being sent synchronously, the following HTTP header MUST be included in the
2236 HTTP envelope:

2237 `ebxmlresponse=synchronous`

C.2 SMTP

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

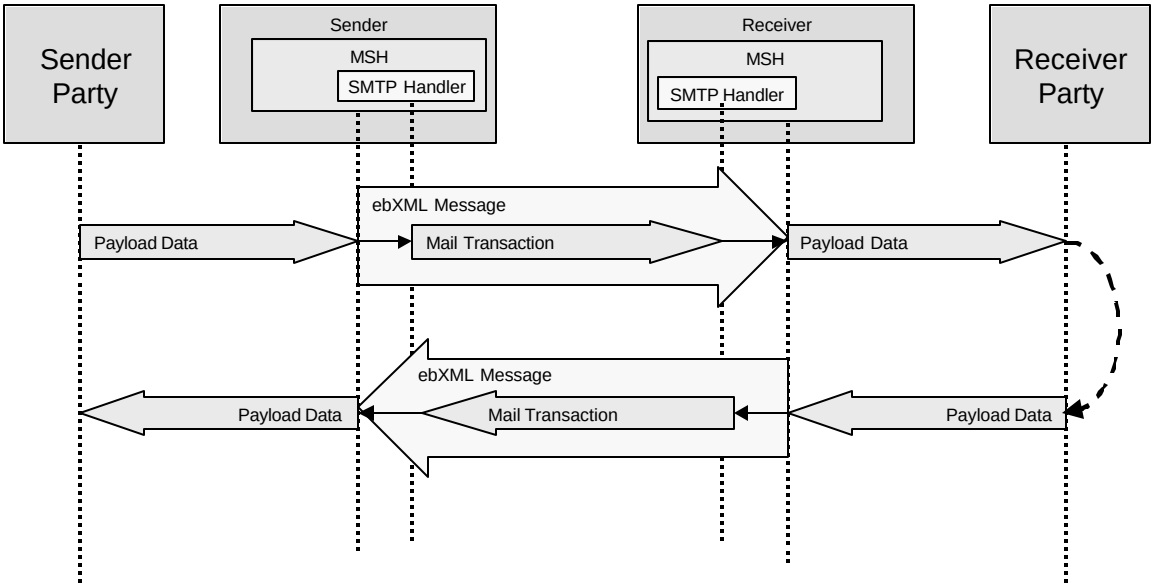


Figure C.3 SMTP Message Flow

The Mail Transaction follows RFC 821, “SIMPLE MAIL TRANSFER PROTOCOL”, as shown in the following Figure:

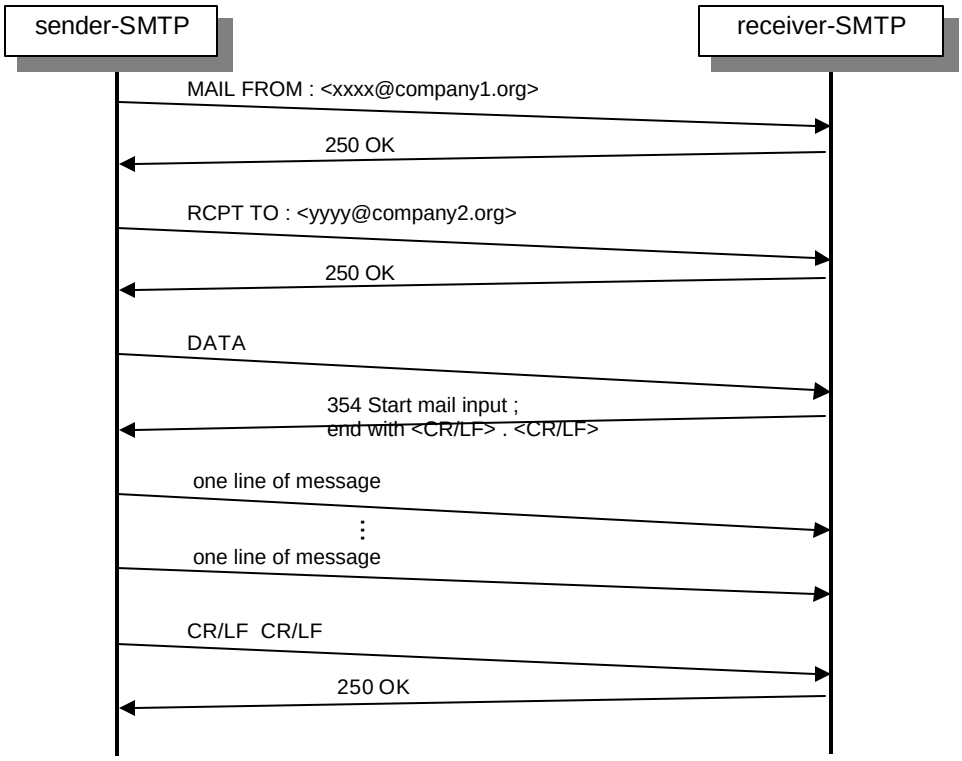


Figure C.4 SMTP Sequence

C.3 FTP

This section will describe how ebXML Messages may be sent using the File Transfer protocol as defined in RFC 959

This section to be completed.

C.4 Communication Protocol Errors during Reliable Messaging

When the Sender or the Receiver detects a transport protocol level error (such as an HTTP, SMTP or FTP error), the appropriate transport recovery handler will execute a recovery sequence. No Reliable Messaging functions (see section 9.2.3) are involved in this recovery sequence, since it happens at a lower level.

However, if the Sender detects a transport protocol level error that is unrecoverable at the transport protocol level, the appropriate recovery handler in the Sender will execute a Messaging Service recovery sequence as described in section 9.2.3.

<DB>Do we need to provide more detail for HTTP errors that are synchronous. For example if there is an ebXML error, do you respond with a 400?</DB>

Appendix D Reliable Messaging Processing Logic

This section will contain non-normative reference processing logic to describe the behavior of a MSH that is taking part in reliable messaging. It's purpose is to assist implementers in developing consistent interoperable solutions.

Copyright Statement