Creating A Single Global Electronic Market

Message Service Specification

ebXML Transport, Routing & Packaging

Version 0.93

8 February 2001

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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/…

Copyright © ebXML 2000 & 2001. All Rights Reserved.

2 ebXML Participants

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

Ralph Berwanger – bTrade.com
Jonathan Borden – Author of XMTP
Jon Bosak – Sun Microsystems
Marc Breissinger – webMethods
Dick Brooks – Group 8760
Doug Bunting – Ariba
David Burdett – Commerce One
Len Callaway – Drummond Group, Inc.
David Craft – VerticalNet
Philippe De Smedt – Viquity
Lawrence Ding – WorldSpan
Rik Drummond – Drummond Group, Inc. (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
Andrew Eisenberg – Progress Software
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 – Progress Software
Martha Warfelt – Daimler Chrysler
Prasad Yendluri – Web Methods

Copyright © ebXML 2000 & 2001. All Rights Reserved.

3 Table of Contents

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

4 Introduction

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

4.1 Summary of Contents of Document

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

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

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

This specification is organized around the following topics:

- Packaging Specification – A description of how to package an ebXML Message and its associated parts into a form that can sent using a communications protocol such as HTTP or SMTP (section 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. This is represented as an XML document called the ebXML Header document (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 10)
- Error Handling – This section describes how one ebXML Message Service reports errors it detects to another ebXML Message Service Handler (section 11)
- Security – This provides a specification of the security semantics for ebXML Messages (section12).

Appendices to this specification cover the following:

- Appendix A Schemas and DTD Definitions – This contains [XML Schema] and [XML] Data Type Definitions for the ebXML Header document. Section A.1 is normative while Section A.2 is non-normative.
- Appendix B Examples – This contains a non-normative sample message content
- Appendix C Communication Protocol Envelope Mappings – This normative appendix describes how to transport ebXML Message Service compliant messages over [HTTP] and [SMTP]

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- Appendix D Registration of MIME media type Application/Vendor Tree—vnd – This non-normative appendix contains the registration information that was forwarded to IANA to register the MIME subtype vnd.eb+xml.

4.2 Document Conventions

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

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

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

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

4.3 Audience

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

4.4 Caveats and Assumptions

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

4.5 Related Documents

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

- **ebXML Message Services Requirements Specification** [EBXMLMSREQ] – defines the requirements of these Message Services
- **ebXML Technical Architecture** [EBXMLTA] – defines the overall technical architecture for ebXML

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

5 Design Objectives

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

6 System Overview

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

6.1 What the Message Service does

The ebXML Message Service defines robust, yet basic, functionality to transfer messages using various existing communication protocols. The ebXML Message Service 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; the other two are Registry/Repository [ebXMLRegRep], Collaboration Protocol Profile/Agreement [ebXMLTP] and the ebXML Message Service.

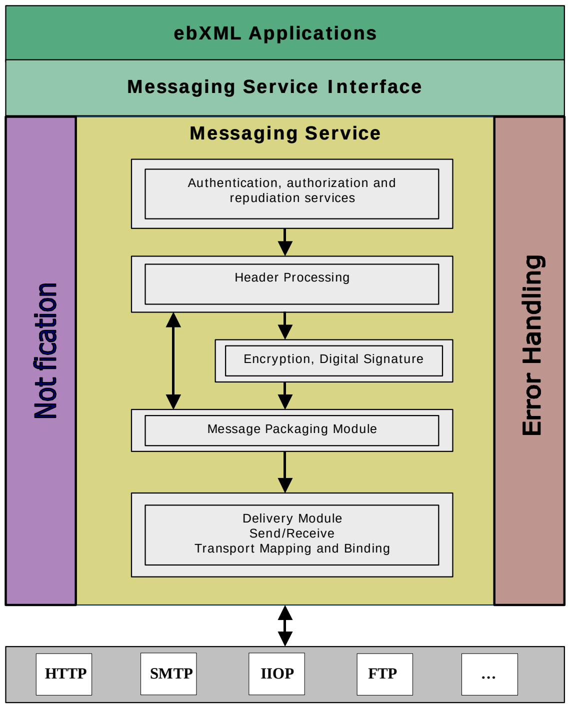
6.2 Message Service Overview

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

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

- Header Processing - the creation of the ebXMLHeader document for the ebXML Message uses input from the application, passed through the Message Service Interface, information from the CPA that governs the message, and generated information such as digital signature, timestamps and unique identifiers.
- Header Parsing - extracting or transforming information from a received ebXMLHeader into a form that is suitable for processing by the MSH implementation.
- Security Services - digital signature creation and verification, authentication and authorization. These services MAY be used by other components of the MSH including the Header Processing and Header Parsing components.
- Reliable Messaging Services - handles the delivery and acknowledgment of ebXML Messages sent with deliverySemantics of OnceAndOnlyOnce. The service includes handling for persistence, retry, error notification and acknowledgment of messages requiring reliable delivery.
- Message Packaging - the final enveloping of an ebXML Message (ebXMLHeader and payload) into its MIME multipart/related container
- Error Handling - this component handles the reporting of errors encountered during MSH or Application processing of a message as well as processing of received messages that have an ErrorList element detailing an error reported by a foreign MSH on a message previously sent by the MSH.
- Notification - *<rb>add additional text here for description </rb>*
- Message Service Interface - an abstract service interface that applications use to interact with the MSH to send and receive messages and which the MSH uses to interface with applications that handle received messages.

Copyright © ebXML 2000 & 2001. All Rights Reserved.



157

158 **Figure 6-1 Typical Relationship between ebXML MSH Components**

159 *<DB>Diagram needs to be simplified and an explanation of these components needs to be*
160 *provided. (Ralph & Chris)</DB>*

Copyright © ebXML 2000 & 2001. All Rights Reserved.

7 Packaging Specification

7.1 Introduction

An ebXML Message consists of two parts:

- 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,
  - at most one ebXML Payload Container that MUST be used to envelope the actual payload (transferred data) of the Message Communication Protocol Envelope (SMTP, HTTP, etc)

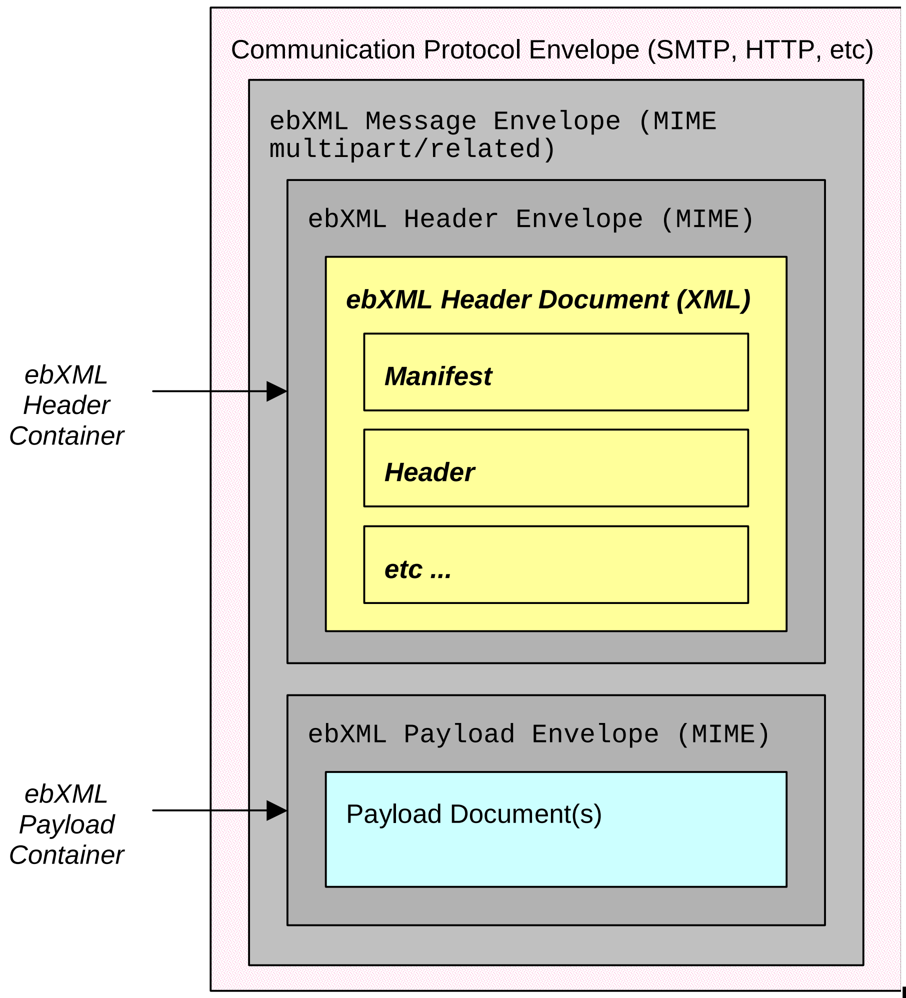

Communication Protocol Envelope (SMTP, HTTP, etc)
ebXML Message Envelope (MIME multipart/related)
ebXML Header Envelope (MIME)
ebXML Header Document (XML)
Manifest
Header
etc ...
ebXML Payload Envelope (MIME)
Payload Document(s)
ebXML Header Container
ebXML Payload Container

Figure 7-1 ebXML Message Structure

7.1.1 ebXML Header Envelope and ebXML Payload Envelope

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

The ebXML Header Envelope contains a single ebXMLHeader document (see section 8). The contents of the ebXML Payload are determined by the user of the ebXML service.

Any special considerations for the usage of the ebXML Message Envelope in HTTP and SMTP transports are described in Appendix C.

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. 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 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:="-------boundaryValueHere"
```

7.2.1.3 version Attribute

The MIME `version` attribute indicates the version of the ebXML Message Service Specification to which the *ebXML Message Envelope* conforms. All message headers SHOULD USE "0.93". For example:

```
version="0.93"
```

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:="-----boundaryValue";
```

7.3 ebXML Header Container

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

- one *ebXML Header Envelope*, that contains

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- one XML *ebXML Header document* (see section 8).

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 *ebXML Header Document* in each *ebXML Message*.

Note that, 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@example.com>
```

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 Envelope* and *ebXML Header Document* conform. All message headers MUST USE “0.93”. Future versions of this specification may require other values of this attribute. However, the value specified here MUST match that specified in the `version` attribute of the *ebXML Header Document* for all versions of this specification. For example:

```
version="0.93";
```

7.3.2.2 charset Attribute

The MIME `charset` attribute identifies the character set used to create the *ebXML Header Document*. The semantics of this attribute are described in the “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@example.com     --|    MIME ebXML    |
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

```
Content-Type: application/vnd.eb+xml;          | Header Envelope  |
version=”0.93”; 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), neither 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 Messag*e.

-

If an ebXML Payload Container is present, it MUST conform to MIME [RFC2045] and MUST consist of a single payload MIME object that may be any valid MIME type including any of the MIME mulitipart/* types.

The *ebXML MIME Payload Envelop*e, 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. The specification of the structure and composition of payload objects is the prerogative of the organization that defines the business process or information exchange that uses the ebXML Message Service.

7.4.1 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@example.com>
```

7.4.2 Content-Type

The MIME Content-Type for an ebXML Payload Container is used to specify the media type and subtype of data in the body of the ebXML Payload Container. The value of this MIME parameter is determined by the organization that defines the business process or information exchange. The value selected SHOULD be chosen from the list of registered MIME media types found at: ftp://isi.edu/in-notes/iana/assignments/media-types/. The MIME Content-Type MUST conform to

Copyright © ebXML 2000 & 2001. All Rights Reserved.

[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: domainname.example.com-------------| 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.

7.5 Additional MIME Parameters

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

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

7.6 Reporting MIME Errors

If a MIME error is detected in the *ebXML Header Envelope* or the *ebXML Payload Envelope* then it MUST be reported by sending an ebXML message containing an ***ebXMLHeader*** element with an ***ErrorList*** element (see section 8.8) where ***errorCode*** is set to ***MimeProblem*** and a ***severity*** set to ***Error***. See section 11 for more details on how to indicate an error.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

8.1 XML Prolog

The 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 the encoding declaration and the MIME charset 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:

```
Content-Type:application/vnd.eb+eml; version “0.93”; charset=”UTF-8”
<?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 compliant with the XML Schema Recommendation [XMLSchema], a document type declaration

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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 *ebXML Header Document* is named the ***ebXMLHeader***. Its structure is described below.

8.2.1 ebXMLHeader attributes

There are two attributes defined for the ***ebXMLHeader*** element, they are as follows:

- Namespace (xmlns)
- Version

Additional namespace declarations and namespace-qualified attributes from foreign namespaces MAY be added to support extensions to the ***ebXMLHeader*** document.

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 required ***version*** attribute indicates the version of the ebXML Message Service Specification to which the *ebXML Header Document* conforms. Its purpose is to provide for future versioning capabilities. All *ebXML Header Documents* MUST USE “0.93”. Future versions of this specification SHALL require other values of this attribute. However, the value specified here MUST match that specified in the MIME `version` attribute of the *ebXML Header Envelope* for all versions of this specification.

8.2.2 ebXMLHeader elements

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

- ***Manifest*** – an element that points to 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*** – an element that contains entries that identifies the Message Service Handler (MSH) that sent and should receive the message. This element can be omitted.
- ***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* that the MSH MUST make available to the application processing the ebXML Payload Container
- ***StatusData*** – 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 are being reported against a previous message
- ***Acknowledgment*** – an element that is used by a receiving MSH to acknowledge to the sending MSH that a previous 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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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 10) or over multiple hops (see section 8.5.3).

8.2.3.4 ApplicationHeaders element

The ***ApplicationHeaders*** element MAY be present on any message except a message that contains an ***ErrorList*** element with a ***highestSeverity*** attribute set to ***Error***.

8.2.3.5 StatusData element

This element MUST NOT be present with the following elements:

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

8.2.3.6 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.

Any namespace-qualified element added SHOULD include the global ***mustUnderstand*** attribute. If the ***mustUnderstand*** attribute is NOT present, the default value implied is 'false'. If an

Copyright © ebXML 2000 & 2001. All Rights Reserved.

implementation of the MSH does not recognize the namespace of the element and the value of the ***mustUnderstand*** attribute is 'true' then the MSH SHALL respond with a message that includes an ***errorCode*** of ***NotSupported*** in an ***Error*** element as defined in section 8.8. If the value of the ***mustUnderstand*** attribute is 'false' or if the ***mustUnderstand*** attribute is not present then an implementation of the MSH MAY ignore the namespace-qualified element and its content.

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="0.93" >
<Manifest>...</Manifest>
<Header>...</Header>
<RoutingHeaderList>…</RoutingHeaderList>
</ebXMLHeader>
```

8.3 Manifest element

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

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

The ***Manifest*** element MUST 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:

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

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

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

- ***id*** - a REQUIRED XML ID for the ***Reference*** element,
- ***xlink:type*** - this attribute defines the element as being an XLINK simple link. It has a fixed value of 'simple',
- ***xlink:href*** - this REQUIRED attribute has a value that is the URI of the payload object referenced. It SHALL conform to the [XLINK] specification criteria for a simple link,
- ***xlink:role*** - this attribute identifies some resource that describes the payload object or its purpose. If present, then it SHALL have a value that is a valid URI in accordance with the [XLINK] specification,

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- 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 Schema element

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

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

8.3.1.2 Description element

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

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

8.3.1.3 #wildcard element

Refer to section 8.2.3.9 for discussion of #wildcard element handling.

8.3.2 What References are Included in a Manifest

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

8.3.3 Manifest Validation

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

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

8.3.4 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:role="http://regrep.org/gci/purchaseOrder">
    <Description>Purchase Order for 100,000 widgets</Description>
    <Schema location="http://regrep.org/gci/purchaseOrder/po.xsd"
      version="1.0"/>
  </Reference>
</Manifest>
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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***
- ***MessageData***
- ***QualityOfServiceInfo***
- ***SequenceNumber***
- ***Description***
- ***#wildcard***

8.4.1 From and To elements

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

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

The ***PartyId*** element has a single attribute, ***type*** and content that is a string value. If the ***type*** attribute is present, then it MUST be a URN. It indicates the domain of names to which the string, in the content of the ***From*** or ***To*** element, belongs.

If the ***PartyId type*** attribute is not present, the content of the ***PartyId*** element MUST be an URI [RFC 2396] otherwise report an error (see section 11) with ***errorCode*** set to ***Inconsistent*** and ***severity*** set to ***error***. It is strongly RECOMMENDED that the content be an URN.

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

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

8.4.2 CPAId element

The REQUIRED ***CPAId*** element is a string that identifies the *Collaboration Protocol Agreement* (CPA) that governs the processing of the message. The identifier MUST be unique within the domain of the names chosen by the Parties.

A *Party* that receives the message, must be able to resolve the ***CPAId*** to the CPA instance as information in the CPA is used, for example, by Reliable Messaging (see section 10). It is therefore RECOMMENDED that the ***CPAId*** is a URI.

8.4.3 ConversationId element

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

The value used for a ***ConversationId*** is implementation dependent.

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

8.4.4 Service element

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

- a standards organization, or
- an individual or enterprise

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

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

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]. If it is not a URI then report an error with an ***errorCode*** of ***Inconsistent*** and a ***Severity*** of ***Error*** (see section 11).

8.4.4.2 ebXML Message Service namespace

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

8.4.5 Action element

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

8.4.6 MessageData element

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

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

8.4.6.1 MessageId element

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

8.4.6.2 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).

8.4.6.3 RefToMessageId element

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

For Error messages, the ***RefToMessageId*** element is REQUIRED and its value MUST be the ***MessageId*** value of the *message in error* (as defined in section 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 10.

8.4.6.4 TimeToLive element

The TimeToLive element indicates the time by which a message should be delivered to and processed by the To Party.

In this context, the TimeToLive has expired if the time of t he internal clock of the MSH that receives a message is greater than the value of TimeToLive for the message.

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 MSH TimeAccuracy parameter for the Receiving MSH (see section 10.6.5.3) 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.

If the TO Party's MSH receives a message where TimeToLive has expired, it SHALL send a message to the From party MSH, reporting that the TimeToLive of the message has expired. This message SHALL be comprised of:

- A Payload that consists of the ebXML message that expired;
- An ErrorList containing an Error that has the errorCode attribute set to TimeToLiveExpired, and the severity attribute set to Error.

8.4.7 QualityOfServiceInfo element

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

- ***deliverySemantics***
- ***messageOrderSemantics***
- ***deliveryReceiptRequested***
- ***syncReplyMode***, and

The ***QualityOfServiceInfo*** element MAY be present if any of the attributes within the element need to be set to their non-default value.

8.4.7.1 deliverySemantics attribute

The ***deliverySemantics*** attribute, if present, over-rides the value of the same parameter in the CPA. If it is not present, the value in the CPA MUST be used.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

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

If ***deliverySemantics*** is set to ***BestEffort*** 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***.

8.4.7.1 messageOrderSemantics attribute

The ***messageOrderSemantics*** attribute, if present, over-rides the value of the same parameter in the CPA. If it is not present, the value in the CPA MUST be used.

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

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

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

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

If ***deliverySemantics*** is set to ***NotGuaranteed***, then the *To Party* MSH does not need to correct invalid order of messages. If the *To Party* is unable to support the type of ***MessageOrderSemantics*** requested, then the *To Party* MUST report the error to the *From Party* using an ***ErrorCode*** of ***NotSupported*** and a ***Severity*** of ***Error***. A sample of ***messageOrderSemantics*** follows.

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

8.4.7.2 DeliveryReceiptRequested attribute

The ***deliveryReceiptRequested*** attribute, if present, over-rides the value of the same parameter in the CPA. If not present then the value in the CPA MUST be used.

The ***deliveryReceiptRequested*** parameter/element MUST be used by a *From Party* MSH to indicate whether a message received by the *To Party* MSH should result in the *To Party* MSH

Copyright © ebXML 2000 & 2001. All Rights Reserved.

returning an acknowledgment message containing an ***Acknowledgment*** element with a ***type*** of ***deliveryReceipt***.

The ***deliveryReceiptRequested*** parameter/element is frequently used to help implement Reliable Messaging (see section 10) although it can be used independently.

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

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

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

When a *To Party* MSH receives a message with ***deliveryReceiptRequested*** not set to ***None*** then it should check if it is able to support the type of Delivery Receipt requested.

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

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

8.4.7.3 syncReplyMode attribute

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

- ***True*** which indicates that the MSH that receives the message MUST get the message processed by the application or other process that needs to process it before the MSH sends any response to the original message, or
- ***False*** which indicates that an acknowledgment to the message MAY be sent separately before processing of the message by the application or other process.

The default value is ***False***.

8.4.8 SequenceNumber element

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

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

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

The ***SequenceNumber*** element is an integer value that is incremented (e.g. 0, 1, 2, 3, 4...) for each From Party application-prepared message sent to the To Party application in the ***ConversationId***. The next value of 99999999 in the increment is “0”. The Sequence Number

Copyright © ebXML 2000 & 2001. All Rights Reserved.

consists of ASCII numerals in the range 0-99999999. In following cases, the Sequence Number takes the value “0”:

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

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

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

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

```
<SequenceNumber Status=”Reset”>0</SequenceNumber>
```

8.4.9 Description element

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

8.4.10 #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].

Refer to section 8.2.3.9 for discussion of #wildcard element handling.

8.4.11 Header sample

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

```
<Header>
  <From>example.com</From>
  <To type="userType">...</To>
  <CPAId>http://www.ebxml.org/cpa/123456</CPAId>
  <ConversationId>987654321</ConversationId>
  <Service type="myservicetypes">QuoteToCollect</Service>
  <Action>NewPurchaseOrder</Action>
  <MessageData>
    <MessageId>UUID-2</MessageId>
    <Timestamp>20000725T121905.000Z</Timestamp>
    <RefToMessageId>UUID-1</RefToMessageId>
  </MessageData>
  <QualityOfServiceInfo
    deliverySemantics=”OnceAndOnlyOnce”
    deliveryReceiptRequested=”Signed”/>
</Header>
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

8.5 RoutingHeaderList element

A ***RoutingHeaderList*** element 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 10)

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 one or more intermediate MSH modes as it travels between the From party MSH and the To Party MSH, then each transmission over each successive "hop" results in the addition of a new Routing Header element.

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

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

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

- ***reliableMessagingMethod***
- ***intermediateAckRequested***

8.5.1.1 reliableMessagingMethod attribute

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

- ebXML
- Transport

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

8.5.1.2 intermediateAckRequested attribute

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

- Signed
- UnSigned
- None

The default implied value for this attribute is "None". Refer to section 10.1.2 for discussion of the use of this attribute.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

8.5.1.3 SenderURI element

This element contains the URI of the message's 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.

8.5.1.4 ReceiverURI element

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

8.5.1.5 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.***

8.5.1.6 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.

8.5.1.7 SequenceNumber element

The ***SequenceNumber*** is an optional element that indicates the sequence in which messages must be processed by a receiving MSH. The SequenceNumber is unique within the ***ConversationId*** and Sender MSH. It is set to one on the first message from that MSH for a Conversation and then incremented by one for each subsequent message sent.

Preservation of message sequence MUST be used with ***deliverySemantics*** of ***OnceAndOnlyOnce*** otherwise there is an error.

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

If the implementation defined limit for saved out-of-sequence messages is reached, then the Receiving MSH MUST indicate a delivery failure to the Sending MSH with ***errorCode*** set to ***DeliveryFailure*** and ***severity*** set to ***Error*** (see section 10.5).

8.5.1.8 #wildcard element

Refer to section 8.2.3.9 for discussion of #wildcard element handling.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

8.5.2 Single Hop Routing Header Sample

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

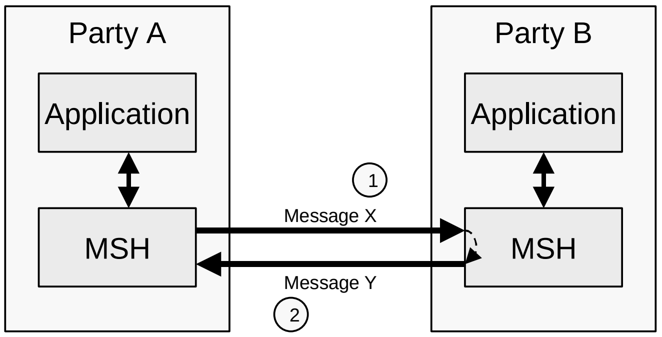


Figure 8-1 Single Hop Message

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</To>
  <ConversationId>219cdj89dj2398djfjn</ConversationId>
  ...
  <MessageData>
    <MessageId>29dmridj103kvna</MessageId>
    ...
  </MessageData>
   ...
</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>
```

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.

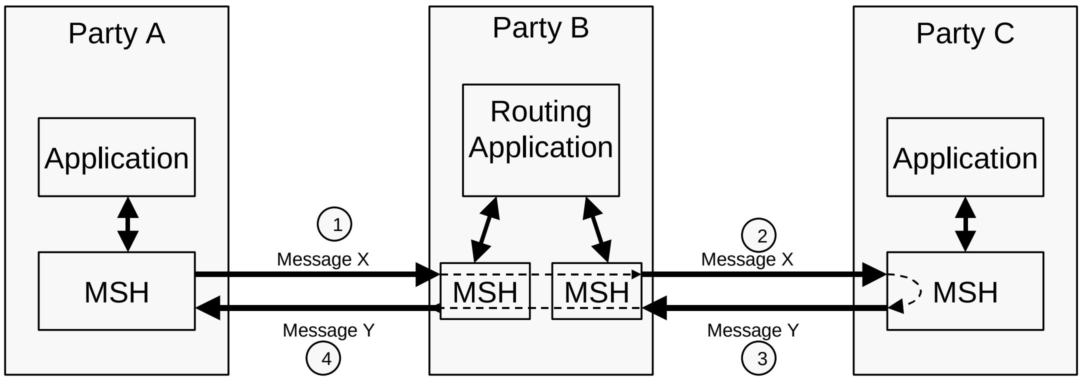


Figure 8-2 Multi-hop Message

The content of the corresponding messages could include:

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- 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>
  <ConversationId>219cdj89dj2398djfjn</ConversationId>
  ...
  <MessageData>
    <MessageId>29dmridj103kvna</MessageId>
    ...
  </MessageData>
   ...
</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>
  <ConversationId>219cdj89dj2398djfjn</ConversationId>
  ...
  <MessageData>
    <MessageId>29dmridj103kvna</MessageId>
    ...
  </MessageData>
   ...
</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>
```

8.6 ApplicationHeaders Element

The ***ApplicationHeaders*** element supports the extension of an ebXML Message through the inclusion of additional XML elements that belong to a foreign namespace, as child elements of the ***ApplicationHeaders*** element.

Any 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. How this is done is an implementation decision but conformance to the ebXML Service Interface specification (to be defined) is RECOMMENDED.

8.6.1 ApplicationHeaders sample

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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 reported
- a ***Timestamp*** element. This contains the time that the message, whose status is being reported, was received. This MUST be omitted if the message whose status is being reported is ***NotRecognized*** or the request was ***UnAuthorized***
- a ***ForwardURI*** element. This MUST 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 ***ErrorList*** 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 attributes:

- ***codeContext***
- ***errorCode***
- ***severity***
- ***location***

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- ***xml:lang***
- ***errorMessage***
- ***softwareDetails***

8.8.3.1 codeContext attribute

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

Use of non ebXML values for ***errorCodes*** is NOT RECOMMENDED. In addition, an implementation of this specification MUST NOT use its own ***errorCodes*** if an existing ***errorCode*** as defined in 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 erro*r. 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 an error, other messages in the conversation will still be generated in the normal way.
- ***Error*** - This indicates that there is an unrecoverable error in the message and no further messages will be generated as part of the conversation.

8.8.3.4 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 attribute

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.

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

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

<DB>Do we want to allow multiple errorMessage elements in different languages, e.g. so that if you send a message to Switzerland you could send it in French, German and Italian?</DB>

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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 .... />
</ErrorList>
```

8.8.5 errorCode values

This section describes the ***ErrorCodes*** (see section 8.8.3.2) that are used in a *message reporting an erro*r. They are described in a table with three headings:

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

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

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

8.8.6 Reporting Errors in the ebXML Header Document

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

Error Code	Short Description	Long Description
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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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:

Error Code	Short Description	Long Description
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. Note that if ***severity*** is set to ***Warning*** then there is a small probability that the message was delivered.
TimeToLiveExpired	Message Time To Live Expired	A message has been received that arrived after the time specified in the ***TimeToLive*** element of the ***Header*** element
SecurityFailure	Message Security Checks Failed	Validation of signatures or checks on the authenticity or authority of the sender of the message have failed.
Unknown	Unknown Error	Indicates that an error has occurred that is not

Copyright © ebXML 2000 & 2001. All Rights Reserved.

		covered explicitly by any of the other errors. The ***errorMessage*** attribute should be used to indicate the nature of the problem.

8.9 Acknowledgment Element

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

For clarity two terms are defined:

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

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:

- a ***Timestamp*** element
- a ***From*** element
- a ***type*** attribute
- a ***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]. *<DB>Do we make this conform to XML Schema timeInstant</DB>*

8.9.2 From element

This is the same element as the ***From*** element within ***Header*** element (see 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*.

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

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
- ***IntermediateAck*** - 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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

8.10 Signature Element

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

9 Message Service Handler Services

[The Message Service Handler Services section has not been agreed to by the membership of the TRP Project Team; however, it is being included to provide a basis for POC developers of MSH implementations. Implementers MUST be prepared for some change to the content of this section.]

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

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

9.1.2 Message Status Response Message

Once the To Party 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
- an ***Acknowledgment*** element
- a ***StatusData*** element
- a ***Signature*** element

Copyright © ebXML 2000 & 2001. All Rights Reserved.

The ***RoutingHeaderList***, ***Acknowledgment*** 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 is set to the value of the ***From*** element in 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.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

The ***RoutingHeaderList***, ***Acknowledgment*** 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.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

10 Reliable Messaging

[The Reliable Messaging section has not been agreed to by the membership of the TRP Project Team; however, it is being included to provide a basis for POC developers of MSH implementations. Implementers MUST be prepared for some change to the content of this section.]

Reliable Messaging defines an interoperable protocol such that the two Messaging Service Handlers (MSH) 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

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

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

Each of these are described below.

10.2 ebXML Reliable Messaging Protocol

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

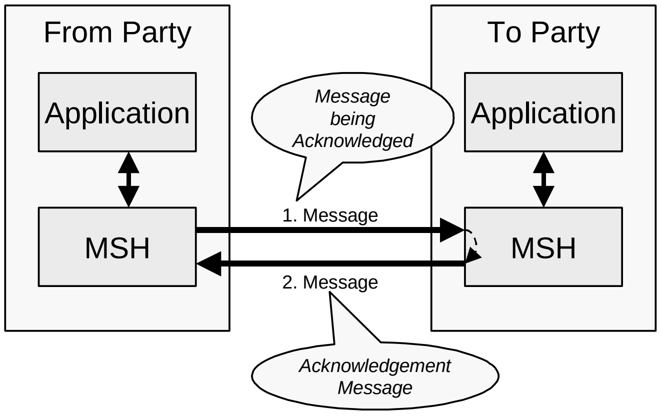


Figure 10-1 Indicating that a message has been received

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.

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

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 one or more intermediate MSHs.

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

This section describes the REQUIRED behavior of a Message Service Handler (MSH) that is sending and/or receiving messages that support the ebXML Reliable Messaging Protocol.

10.2.1.1 Sending Message Behavior

If a MSH is given data by an application that needs to be sent reliably then the MSH MUST do the following:

1) Create a message from components received from the application that includes:
   a) ***deliverySemantics*** set to ***OnceAndOnlyOnce***, and
   b) a ***RoutingHeader*** element that identifies the sender and the receiver URIs
2) Save the message in *persistent storage* (see section 10.1.1)
3) Send the message (the *message being acknowledged)* to the *Receiver* MSH
4) Wait for the *Receiver* MSH to return an *acknowledgment message* and, if it does not, then resend the *identical* message as described in section 10.2.1.3

Copyright © ebXML 2000 & 2001. All Rights Reserved.

It is RECOMMENDED that messages that are sent reliably include ***deliveryReceiptRequested*** set to ***Signed*** or ***UnSigned***.

If the message does not need to be sent reliably, then ***deliverySemantics*** MUST be set to ***BestEffort*** (the default).

10.2.1.2 Receiving Message Behavior

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

1) Check to see if the message is a duplicate (e.g. there is a message in *persistent storage* that was received earlier that contains the same value for the ***MessageId***)
2) If the message is not a duplicate then do the following:
    a) Save the ***MessageId*** of the received message in *persistent storage*. As an implementation decision, the whole message MAY be stored if there are other reasons for doing so.*<DB>Need to re-look at how duplicates are detected if sequence numbers are used. </DB>*
    b) If the received message contains a ***RefToMessageId*** element then do the following:
        i) Look for a message in *persistent storage* that has a ***MessageId*** that is the same as the value of ***RefToMessageId*** on the received Message
        ii) If a message is found in *persistent storage* then mark the persisted message as delivered
    c) If ***deliveryReceiptRequested*** is set to ***Signed*** or ***UnSigned*** then create an ***Acknowledgment*** element with ***type*** set to ***DeliveryReceipt*** that identifies the *received message*
    d) If ***syncReplyMode*** is set to ***True*** then pass the data in the received message to the application or other process that needs to process it and wait for the application to produce a response.
    e) If ***deliveryReceiptRequested*** is set to ***Signed*** or ***UnSigned***, or ***syncReplyMode*** is set to ***True*** then do the following:
        i) Create a ***RoutingHeader*** element that identifies the sender and the receiver URIs
        ii) Set the ***RefToMessageId*** to the value of the ***MessageId*** in the received message
        iii) Create a *message* from the response generated by the application (if any), the ***Acknowledgment*** element (if any) and the ***RoutingHeader*** that includes ***deliverySemantics*** set to ***OnceAndOnlyOnce***
        iv) Save the message in *persistent storage* for later resending
        v) Send the message back to the Sending MSH
    f) If ***syncReplyMode*** is set to ***False*** then pass the data in the received message to the application or other process that needs to process it. Note that, depending on the application, this can result in the application generating another message to be sent (see previous section).
3) If the message is a duplicate, then do the following:
    a) Look in persistent storage for a response to the received message (i.e. it contains a ***RefToMessageId*** that matches the ***MessageId*** of the received message) that was *most recently sent* to the MSH that sent the received message (i.e. it has a ***RoutingHeader*** element with the greatest value of the ***Timestamp***)

Copyright © ebXML 2000 & 2001. All Rights Reserved.

b) If no message was found in *persistent storage* then ignore the received message as either no message was generated in response to the message, or the processing of the earlier message is not yet complete

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

10.2.1.3 Resending Lost Messages and Duplicate Filtering

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

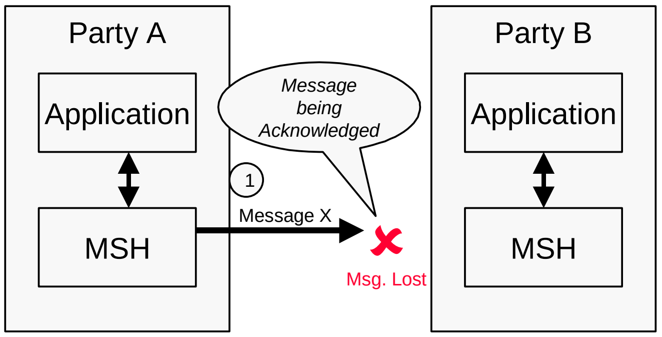


Figure 10-2 Lost “Message Being Acknowledged”

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

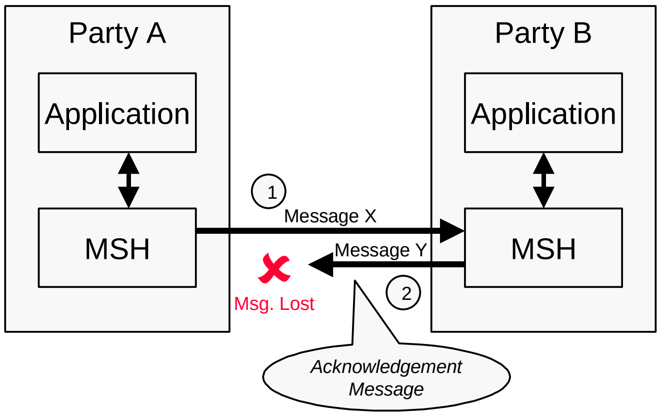


Figure 10-3 Lost Acknowledgment Message

The rules that apply are as follows:

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

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

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

3) If the Sending MSH detects a communications protocol error that is unrecoverable at the transport protocol level then the Sending MSH SHOULD first attempt to resend the message using the same transport protocol until the number of ***retries*** has been reached, and then again, using a different communications protocol, if the CPA allows this. If these are not successful, then notify the From Party of the failure to deliver as described in section 10.5.

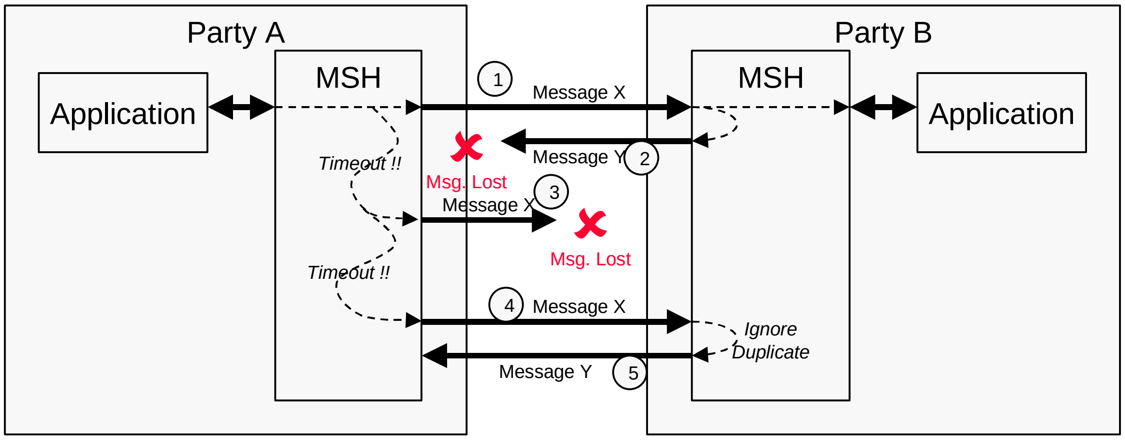


Figure 10-4 Resending Lost Messages

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 (e.g. Party B) 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 (i.e. Party A)

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 *<DB>next MSH is multi-hop, should not be here. </DB>*or other process that ultimately needs to receive them.

<DB>The above also includes recipient behavior which is not part of sending behavior. Should be in a separate section. </DB>

In this context:

- an *identical message* is a *message* that contains, apart from perhaps 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.*<DB>Chris Ferris, disagrees with resending the latest message. DB & CF need to go through this. </DB>*

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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.

10.2.2 Multi-hop Reliable Messaging

Multi-hop reliable Messaging can occur either:

- without Intermediate Acknowledgment, or
- with Intermediate Acknowledgments

One reason for using Multi-hop Reliable Messaging with Intermediate Acknowledgments is when the *From Party* that is sending a message is confident that the total time taken for ...

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

... is likely to result in the *From Party* resending the *message being acknowledged*. *<DB>Chris thinks this is superfluous, David thinks it useful as it explains why you should do multi-hop and helps an implementer decide when to use it. This requires further discussion. </DB>*

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 default).

The overall message flow is illustrated by the diagram below.

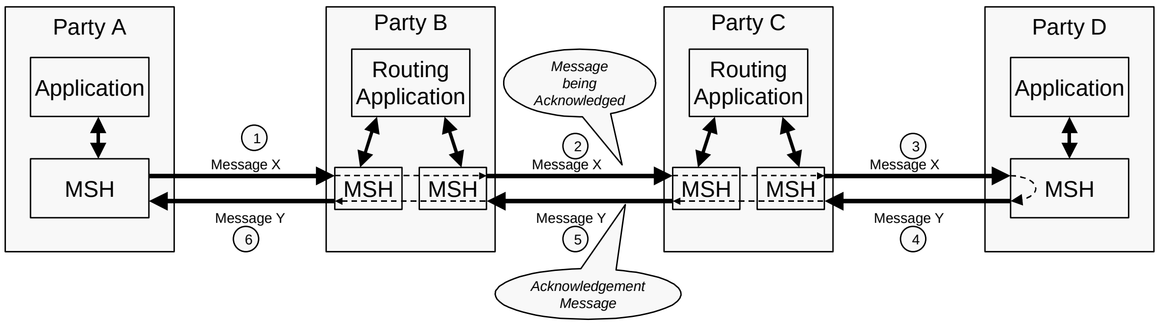


Figure 10-5 Multi-hop Reliable Messaging without Intermediate Acknowledgments

This is essentially the same as 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.

This is described in more detail below:

1) The *From Party* and the *To Party* adopt the sending message and receiving message behavior described in sections 10.2.1.1 and 10.2.1.2 except that 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 ***QualityOfServiceInfo*** element with ***deliverySemantics*** set to ***OnceAndOnlyOnce***

Copyright © ebXML 2000 & 2001. All Rights Reserved.

   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 do the following:

   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) If the Sending MSH (either at the From Party or at an Intermediate Party) does not receive an *Acknowledgment Message* after the maximum number of retries, the Sending MSH SHOULD notify the following of the delivery failure:

   a) The application and/or system administrator function if the Sending MSH is the *From Party* MSH, or

   b) The Sending MSH of the *From Party,* if the Sending MSH is operated by an Intermediate Party (see section 10.5)

4) 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)

5) 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.)

6) 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 can request that the recipient return an *Intermediate Acknowledgment.*

This is illustrated by the diagram below.

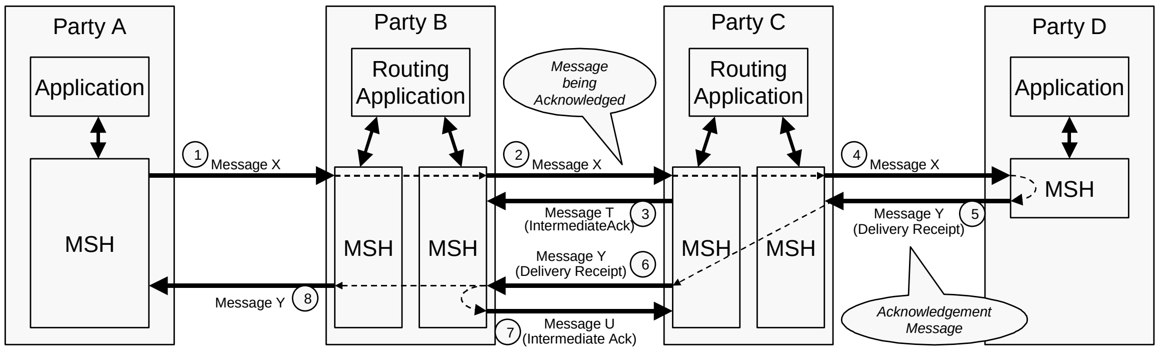


Figure 10-6 Multi-hop Reliable Messaging with Intermediate Acknowledgments

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

- any party may request an intermediate acknowledgment

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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 ***Signed*** or ***Unsigned***. (e.g. Transmission 2 of Message X, or Transmission 6 of Message Y)

2) If a MSH that is not the *To Party* receives a message 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 an *Intermediate Acknowledgment* is requested, then the MSH must return an *Acknowledgment Message* (e.g. Transmission 3 of Message T, or Transmission 7 of Message U) with:

      i) The ***Service*** and ***Action*** elements set as in defined in section 10.4

      ii) The ***From*** element contains the ***ReceiverURI*** from the last ***RoutingHeader*** in the message that has just been received

      iii) The ***To*** element contains the ***SenderURI*** from the last ***RoutingHeader*** in the message that has just been received

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

      v) a ***QualityOfServiceInfo*** element with ***deliverySemantics*** set to ***OnceAndOnlyOnce***

      vi) an ***Acknowledgment*** element with type set to ***IntermediateAck***

      vii) 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)

3) If a MSH that is the *To Party* receives a message 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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

4) If an intermediate MSH cannot forward a message to the next Party then the From Party should be notified using the procedure described in section 10.5.

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

10.4 Service and Action Element Values

An ***Acknowledgment*** element can be included in an ***ebXMLHeader*** that is part of a *message* that is being sent as a result of processing of an earlier message. In this case the values for the ***Service*** and ***Action*** elements are set by the designer of the Service (see section 8.4.4).

An ***Acknowledgment*** element also can be included in an ***ebXMLHeader*** that does not include any results from the processing of an earlier message. In this case, the values of the ***Service*** and ***Action*** elements MUST be set as follows:

- The ***Service*** element MUST be set to: ***http://www.ebxml.org/namespaces/messageService/MessageAcknowledgment***
- The ***Action*** element MUST be set to the value of the ***type*** attribute in the ***Acknowledgment*** element.

Note that ***deliveryReceiptRequested*** must be set to ***None*** on a message that is only an acknowledgment.

10.5 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.

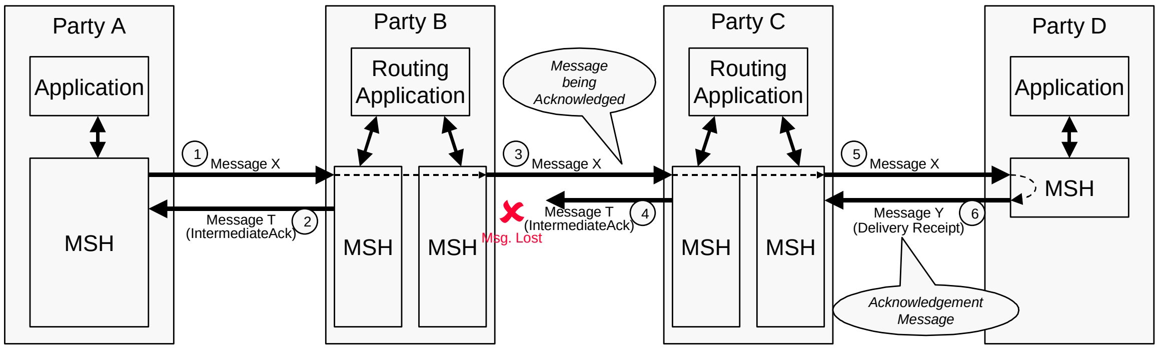


Figure 10-7 Failed Message Delivery using Intermediate Acknowledgments

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 ***Service*** element and ***Action*** element set as described in 11.5

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- a ***QualityOfServiceInfo*** element with ***deliverySemantics*** set to the same value as the ***deliverySemantics*** on the message that could not be delivered
- an ***Error*** 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 3) 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.

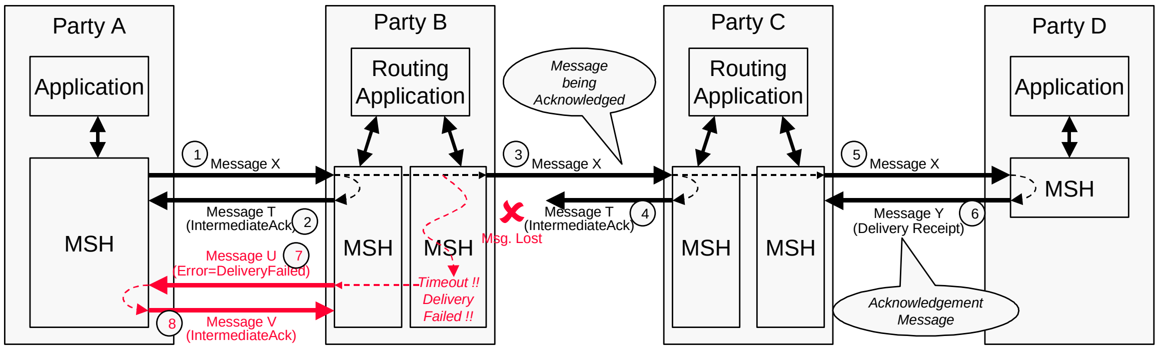

Party A
Application
MSH
Party B
Routing Application
MSH
MSH
Message being Acknowledged
Party C
Routing Application
MSH
MSH
Party D
Application
MSH
1 Message X
Message T (IntermediateAck) 2
Message U 7 (Error=DeliveryFailed)
8 Message V (IntermediateAck)
Timeout !! Delivery Failed !!
3 Message X
Msg. Lost
Message T (IntermediateAck) 4
5 Message X
Message Y (Delivery Receipt) 6
Acknowledgement Message

Figure 10-8 Reporting Failed Message Delivery

Note that the message that contains an ***Error*** 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.

10.6 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.

10.6.1 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.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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	syncReplyMode	Yes	Yes	N/A
From Party	timeToLive	Yes	Yes	N/A
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:

7) the **Specified By** columns indicates the Party that sets the value in the Collaboration Party Protocol, Message Header, or Routing Header
8) if the **CPA/CPP** column contains a **Yes** then it indicates that the party in the **Specified By** column specifies the value that is present in the CPP
9) if the **CPA/CPP** column contains a **No** then it indicates that the parameter value is never specified in the **CPP**
10) if the **Message Header** or **Routing Header** columns contain a **Yes** then it indicates that the parameter value may be specified in the **Header** element or **Routing Header** and over-rides any value in the CPA. It the value is not specified in the **Header element** or **Routing Header** then the value in the **CPA** must be used.
11) if the **Message Header/Routing Header** columns contain a **No** then it indicates that the value in the **CPA** is always used
12) 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.

10.6.2 From Party Parameters

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

10.6.2.1 Delivery Semantics

The ***deliverySemantics*** parameter may be present as either an element within the ***ebXMLHeader*** element or as a parameter within the CPA. See section 8.4.7.1 for more information.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

10.6.2.2 Delivery Receipt Requested

The ***deliveryReceiptRequested*** parameter may be present as either an element within the ***ebXMLHeader*** element or as a parameter within the CPA. See section 8.4.7.2 for more information.

10.6.2.3 Sync Reply Mode

The ***syncReplyMode*** parameter may be present as either an element within the ***ebXMLHeader*** element or as a parameter within the CPA. See section 8.4.7.3 for more information.

10.6.2.4 Time To Live

The ***TimeToLive*** element may be presented within the ***ebXMLHeader*** element see section 8.4.6.4 for more information.

10.6.3 To Party Parameters

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

10.6.3.1 Delivery Receipt Provided

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*** and a ***severity*** of Error.

10.6.4 Sending MSH Parameters

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

10.6.4.1 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 ebXML Reliable Messaging Protocol 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 section10.3.

10.6.4.2 Intermediate Ack Requested

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

Valid values for ***IntermediateAckRequested*** are:

- ***Unsigned*** - requests that an unsigned Delivery Receipt is requested

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

10.6.4.3 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.

10.6.4.4 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 using the same Communications Protocol by the Sending MSH.

10.6.4.5 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.

10.6.4.6 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 10.5)

10.6.5 Receiving MSH Parameters

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

10.6.5.1 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***

10.6.5.2 PersistDuration

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

- the complete message, at least until the information in the message has been passed to the application or other process that needs to process it
- the time the message was received, so that the information can be used to generate the response to a Message Status Request (see section 9.1.1)

persistDuration is specified in the CPA.

A MSH SHOULD NOT resend a message with the same ***MessageId*** to a receiving MSH if the elapsed time indicated by ***persistDuration*** has passed since the message was first sent as the receiving MSH will probably not treat it as a duplicate.

If a message cannot be sent successfully before ***persistDuration*** has passed, then the MSH should report a delivery failure (see section 10.5).

Note that implementations may determine that a message is persisted for longer than the time specified in ***persistDuration***, for example in order to meet legal requirements or the needs of a business process. This information is recorded separately within the CPA.

In order to ensure that persistence is continuous as the message is passed from the receiving MSH to the process or application that is to handle it, it is RECOMMENDED that a message is not removed from *persistent storage* until the MSH knows that the data in the message has been received by the process/application.

10.6.5.3 MSH Time Accuracy

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

11.2 Types of Errors

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

- the structure or content of the *Message Envelope* (e.g. MIME) (see section 7),
- the ebXML Message Header document (see section 8),
- reliable messaging failures (see section 10), or
- security (see section 12).

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

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

11.3 When to generate Error Messages

When an MSH detects an error in a *message in erro*r, 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 section11.4) to which the *message reporting the error* should be sent can be determined, and
- the *message in error* does not have an ***ErrorList*** element with ***highestSeverity*** set to ***Error***.

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

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

11.3.1 Security Considerations

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 erro*r. This may be specified:

Copyright © ebXML 2000 & 2001. All Rights Reserved.

- 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 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.

11.5 Service and Action Element Values

An ***ErrorList*** element can be included in an ***ebXMLHeader*** that is part of a *message* that is being sent as a result of processing of an earlier message. In this case, the values for the ***Service*** and ***Action*** elements are set by the designer of the Service (see section 8.4.4).

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

12 Security

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

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

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

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

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

12.1 Security and Management

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

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

12.2 Collaboration Protocol Agreement

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

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

12.3 Countermeasure Technologies

12.3.1 Persistent Digital Signature

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

12.3.1.1 Signature Generation

13) Create a SignedInfo element with SignatureMethod, CanonicalizationMethod, and Reference(s) elements for the ebXMLHeader document and any required payload objects, as prescribed by [XMLDSIG].

14) Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo as specified in [XMLDSIG].

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

16) Include the namespace qualified Signature element in the ebXMLHeader document just signed, following the RoutingHeaderList element.

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

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

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

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

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

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

The ds:Reference element for the ebXMLHeader document SHALL include a child ds:Transform element that excludes the containing ds:Signature element and all its descendants as well as the RoutingHeaderList element and all its descendants as these elements are subject to change. The ds:Transform element SHALL include a child ds:XPath element that has a value of:

```
/descendant-or-self::node()[not(ancestor-or-self::ds:Signature[@id='S1']) and not(ancestor-or-
self::RoutingHeaderList)]
```

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Example of digitally signed ebXMLHeader document:

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

12.3.2 Persistent Signed Receipt

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

12.3.3 Non-persistent Authentication

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

12.3.4 Non-persistent Integrity

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

12.3.5 Persistent Confidentiality

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

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

Section xx (TBD) describes RECOMMENDED bindings for providing persistent confidentiality using MIME-based encryption schemes.

12.3.6 Non-persistent Confidentiality

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

12.3.7 Persistent Authorization

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

12.3.8 Non-persistent Authorization

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

12.3.9 Trusted Timestamp

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

13 Synchronous and Asynchronous Responses

This section may not be needed.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

14 References

<DB>What's the difference between normative and non-normative</DB>

14.1 Normative References

[HTTP]	RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, January 1997
[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
[RFC 2396]	IETF Request For Comments 2396. Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, Published August 1998
[RFC2045]	IETF RFC 2045. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein, Published November 1996
[SMTP]	RFC 821, Simple Mail Transfer Protocol, J Postel, August 1982
[TLS]	RFC2246, T. Dierks, C. Allen. January 1999.
[UTF-8]	UTF-8 is an encoding that conforms to ISO/IEC 10646. See [XML] for usage conventions.
[XML]	W3C XML 1.0 Recommendation, http://www.w3.org/TR/2000/REC-xml-20001006
[XML Namespace]	Recommendation for Namespaces in XML, World Wide Web Consortium, 14 January 1999, http://www.w3.org/TR/REC-xml-names

14.2 Non-Normative References

[Glossary]	ebXML Glossary, see ebXML Project Team Home Page
[PGP/MIME]	RFC2015, "MIME Security with Pretty Good Privacy (PGP)", M. Elkins. October 1996.
[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.
[TRPREQ]	ebXML Transport, Routing and Packaging: Overview and Requirements, Version 0.96, Published 25 May 2000
[XLINK]	W3C Xlink Candidate Recommendation, http://www.w3.org/TR/xlink/
[XMLDSIG]	Joint W3C/IETF XML Digital Signature specification, http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/
[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.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Appendix A ebXMLHeader Schema and Data Type Definitions

A.1 Schema Definition

The following is the definition of the ***ebXMLHeader*** element as a schema that conforms to [XMLSchema].

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns="http://www.ebxml.org/namespaces/messageHeader"
targetNamespace="http://www.ebxml.org/namespaces/messageHeader"
xmlns: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="0.93" 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="unbounded"/>
        <xsd:element ref="Description" minOccurs="0" maxOccurs="1"/>
        <xsd:any namespace="##other" processContents="lax"/>
      </xsd:sequence>
      <xsd:attribute name="id" use="required" type="xsd:ID"/>
<!-- Changed required to fixed on xlink:type -->
      <xsd:attribute name="xlink:type" use="fixed" type="xsd:string" value="simple"/>
      <xsd:attribute name="xlink:href" use="required" type="xsd:uriReference"/>
<!-- Changed to optional on xlink:role -->
      <xsd:attribute name="xlink:role" type="xsd:uriReference"/>
    </xsd:complexType>
  </xsd:element>

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

```
      <xsd:simpleContent>
        <xsd:attribute name="location" use="required" type="xsd:uriReference"/>
        <xsd:attribute name="version" 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"/>
<!-- Changed Reliable Messaging Inf to Quality Of Service Info. -->
<!-- Removed DeliveryReceiptRequested and TimeToLive and made them optional attributes of
Quality of Service Info -->
        <xsd:element ref="QualityOfServiceInfo" minOccurs="0" maxOccurs="1"/>
<!-- Changed description from maxOccurs 1 to unbounded -->
        <xsd:element ref="Description" minOccurs="0" maxOccurs="unbounded"/>
<!-- Added SequenceNumber element -->
        <xsd:element ref="SequenceNumber" 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="QualityOfServiceInfo">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:attribute name="deliverySemantics" use="default" value="BestEffort"/>
          <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
              <xsd:enumeration value="OnceAndOnlyOnce"/>
              <xsd:enumeration value="BestEffort"/>
            </xsd:restriction>
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

```
          </xsd:simpleType>
<!-- Added in messageOrderSemantics attribute -->
        <xsd:attribute name="messageOrderSemantics" use="default" value="NotGuaranteed"/>
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="Guaranteed"/>
            <xsd:enumeration value="NotGuaranteed"/>
          </xsd:restriction>
        </xsd:simpleType>
<!-- Added in deliveryReceiptRequested attribute -->
        <xsd:attribute name="deliveryReceiptRequested" use="default" value="None"/>
        <xsd:simpleType>
          <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="Signed"/>
            <xsd:enumeration value="UnSigned"/>
            <xsd:enumeration value="None"/>
          </xsd:restriction>
        </xsd:simpleType>
<!-- Added in timeToLive attribute -->
        <xsd:attribute name="timeToLive" type="xsd:timeInstant"/>
      </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"/>

```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

```
<!-- 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"/>
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

```
      <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

This section will contain a [XML] DTD that is equivalent to the schema defined in section A.1.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Appendix B Examples

To be completed.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Appendix C Communication 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.

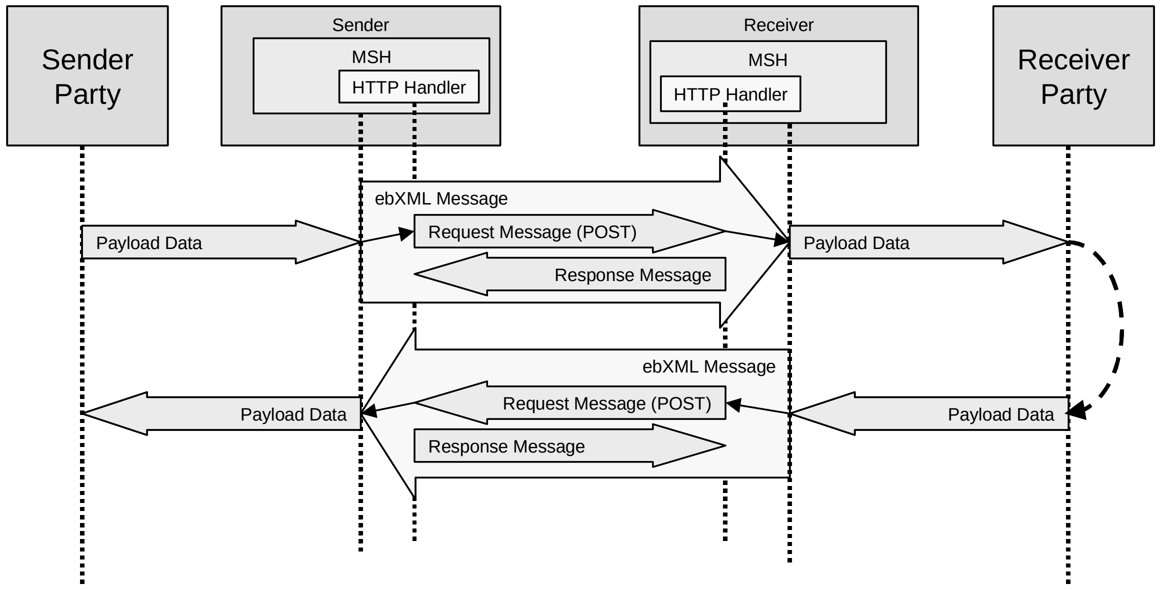


Figure C.1 Asynchronous HTTP Message Flow

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

```
ebxmlresponse=asynchronous
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

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

C.1.2 Synchronous HTTP

[The Synchronous HTTP section has not been agreed to by the membership of the TRP Project Team; however, it is being included to provide a basis for POC developers of MSH implementations. Implementers MUST be prepared for some change to the content of this section.]

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

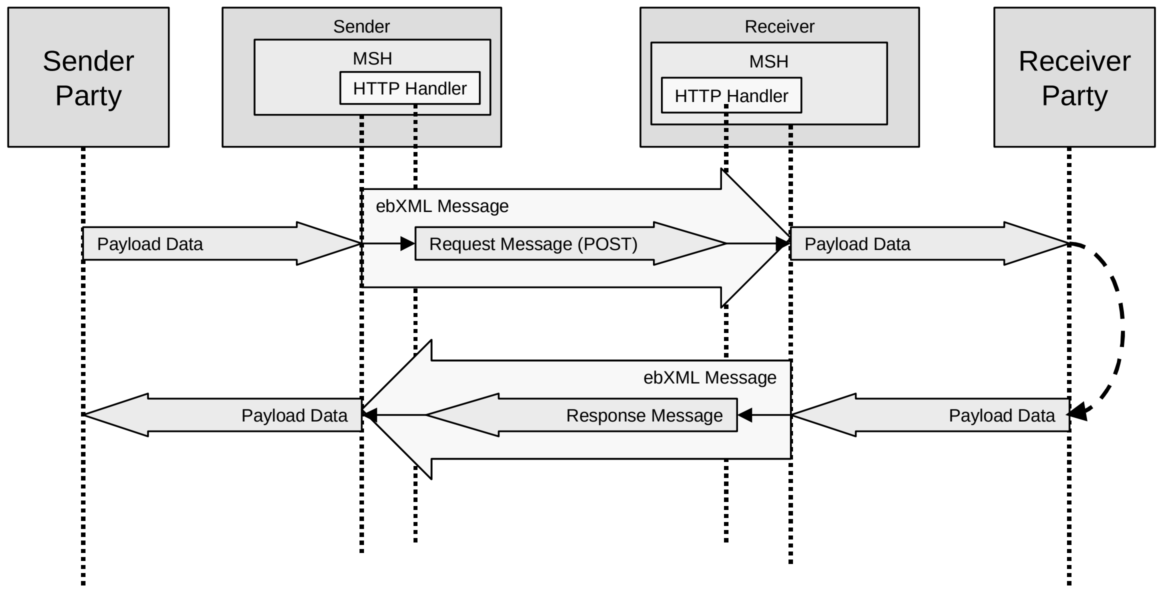


Figure C.2 Synchronous HTTP Message Flow

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

```
ebxmlresponse=synchronous
```

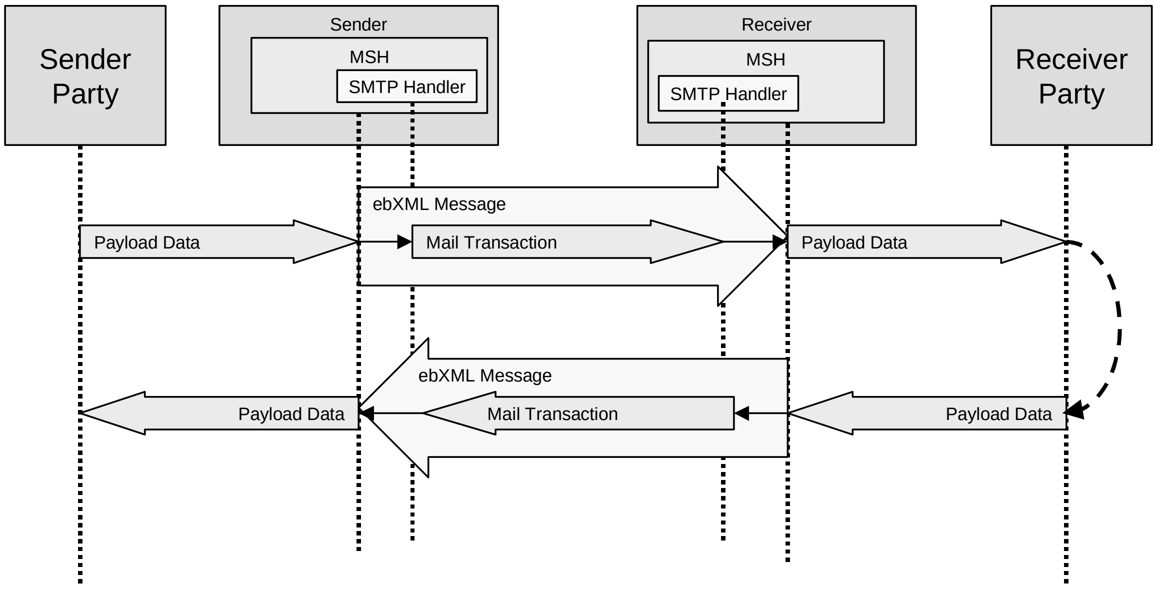
C.1.3 Use of Error Codes

Communication Protocol Error Codes are used only to report errors in the communication protocol envelope (see section 7.1). A normal OK Response (e.g. an HTTP code 200) is used even if there are errors in the MIME envelope, the ebXML Header document or the payload.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

2520 ## C.2 SMTP

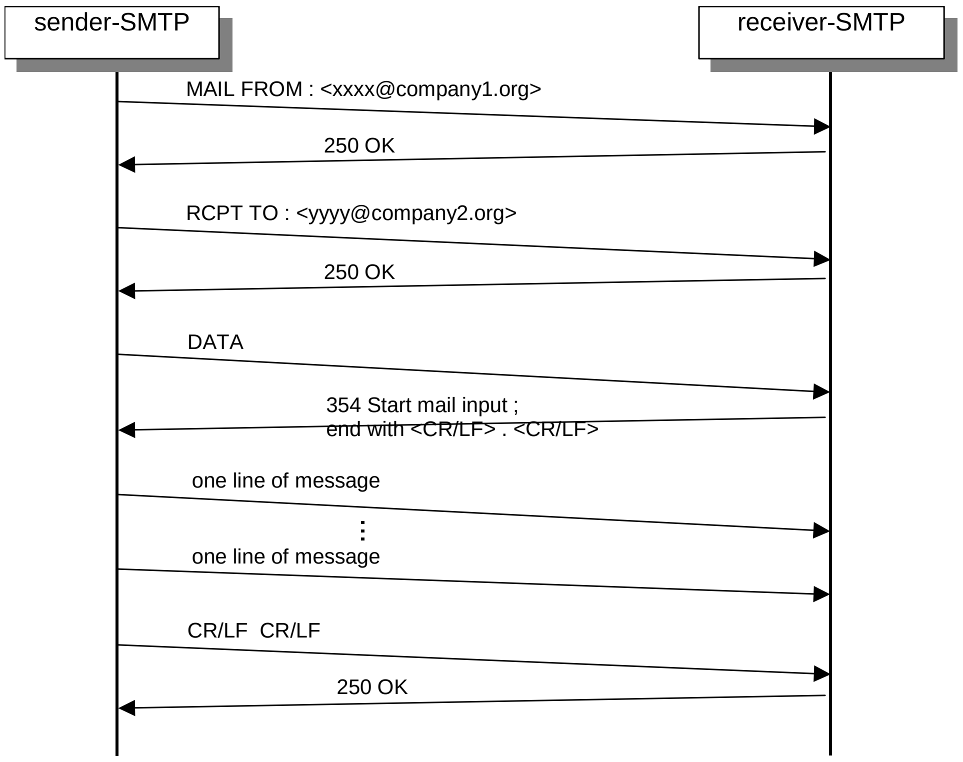
2521 All ebXML Message Service messages are carried as mail in an [SMTP] Mail Transaction as
2522 shown in the figure below.



2523

2524 **Figure C.3 SMTP Message Flow**

2525 The Mail Transaction follows RFC 821, “SIMPLE MAIL TRANSFER PROTOCOL”, as shown in
2526 the following Figure:



2527

2528 **Figure C.4 SMTP Sequence**

Copyright © ebXML 2000 & 2001. All Rights Reserved.

C.3 Communication Errors during Reliable Messaging

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Appendix D Request for MIME media type Application/Vendor Tree - vnd

This section is non-normative. It contains the information forwarded to IANA to register the MIME subtype vnd.be+xml. The information was extracted verbatim from the e-mail message forward by Dick Brooks, Group8760, on behalf of the ebXML Transportation, Routing, and Packaging Project Team.

```
From: Dick Brooks [mailto:dick@8760.com]
Sent: Thursday, February 01, 2001 2:00 PM
To: iana@iana.org; Dick Brooks
Subject: Request for MIME media type Application/Vendor Tree - vnd.

Name: Richard Brooks (on behalf of OASIS and UN/CEFACT)
E-mail: dick@8760.com
MIME media type name: Application
MIME subtype name: Vendor Tree - vnd.eb+xml
Required parameters: version
Optional parameters: charset
Encoding considerations: N/A

Security considerations: N/A
Interoperability considerations: N/A
Published specification: Message Service Specification ebXML Transport,
Routing and Packaging
Applications that use this media: ebXML Message Handling Services
Additional information:
      1. Magic number(s): N/A
      2. File extension(s): .ebx
      3. Macintosh file type code: N/A
      4. Object Identifiers: N/A

This media type is owned jointly by OASIS, UN/CEFACT and ebXML
Person to contact for further information:
1. Name: Richard Brooks
2. E-mail: dick@8760.com
Intended usage: Common
Identifies ebXML header documents
Author/Change controller:
      Christopher Ferris chris.ferris@east.sun.com
      Rik Drummond rvd2@worldnet.att.net
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Copyright © ebXML 2000 & 2001. All Rights Reserved.

Copyright Statement

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

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

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

Copyright © ebXML 2000 & 2001. All Rights Reserved.