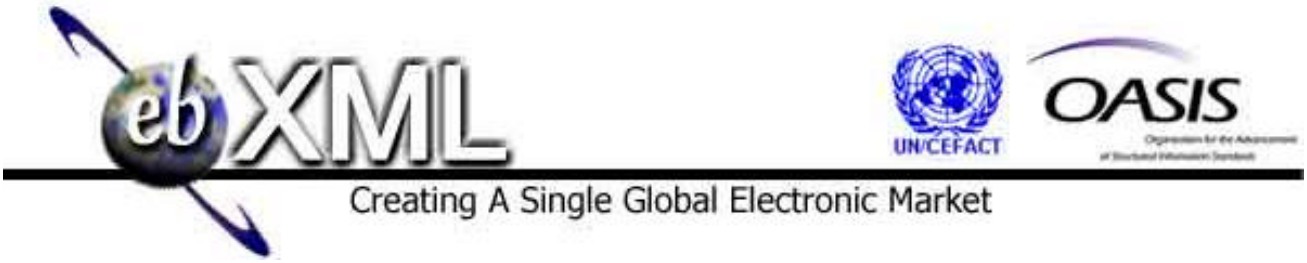


Messaging Service Specification

ebXML Transport, Routing & Packaging

Version 0.21d

16 October 2000

1 Status of this Document

This document specifies an ebXML DRAFT for the eBusiness community.

Distribution of this document is unlimited.

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

This version:

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

***Latest version*:**

http://www.ebxml.org/…

Previous version:

http://www.ebxml.org/…..

2 ebXML participants

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

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

3 Table of Contents

4 Introduction

4.1 Summary of Contents of Document

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

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

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

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

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

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

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

4.2 Audience

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

4.3 Related Documents

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

- **ebXML Messaging Service Specification** (this document) - defines the structure of the messages and the behavior of messaging services software. This will include:

- definitions of the messages
- behavior of the messaging service software
- reliable messaging
- message security
- extensibility and versioning

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

5 Design Objectives

5.1 Goals/Objectives/Requirements/Problem Description

The design objectives and goals are to define a Messaging Service (MS) to support XML based electronic business between small, medium and large enterprises. This specification is intended to enable a low cost solution, while preserving a vendor's ability to add unique value through added robustness and superior performance. It is the intention of the Transport, Routing and Packaging Project Team to keep this specification as simple and succinct as possible. Every item in this specification is being prototyped by the ebXML Proof of Concept Team in order to ensure the clarity and succinctness of this specification. This specification is organized around the following topics:

- **Packaging Specification** - A description of how to package an *ebXML Message* and associated parts. This section includes specifications for the various structures and containers. The Packaging Specification is a standard MIME multipart/related structure with two parts: XML Message Headers and Payload. The payload may be any type of data that MIME RFC 2045 and related IETF MIME extensions may support. The XML based Message Header elements and their structure were chosen after reviewing several current transports, both proprietary and non-proprietary, to ensure that the appropriate header elements were included in the specification
- **Message Headers** - A specification of the structure and composition of the information necessary for an ebXML Messaging Service to successfully generate or process an ebXML compliant message.
- **Reliable Messaging** - The Reliable Messaging function defines an interoperable protocol such that any two Messaging Service implementations can “reliably” exchange messages that are sent using “reliable messaging” semantics. Please see Section 7.11.
- **Error Handling** - This section describes how one ebXML Messaging Service reports errors it detects to another ebXML Messaging Service.
- **Security** - This version of the specification supports limited security services that is those security services that can be supported within the payload. The `multipart/related` payload may be encrypted using cryptographic techniques suitable for the payload type.

Appendices to this specification cover the following:

- Appendix A Schemas and DTD Definitions
- Appendix B Examples
- Appendix C Candidate Packaging Technologies and Selection Process
- Appendix D MIME Type discussion
- Appendix E Communication Protocol Envelope Mappings
- Appendix F Detailed list of the Messaging Services Requirement Phases

5.2 Caveats and Assumptions

The specification is the first in a series of phased deliverables. This version of the specification does not address complete message security, extensibility, service interface, reliability, and versioning. These are being developed as separate documents and will be included in later versions of this document or as additional service specifications to the ebXML Message Services Specification.

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

6 System Overview

This document defines the enveloping and *ebXML Message* header structure used to transfer *ebXML Message*s over a data communication mechanism. This document provides sufficient detail to develop software for the packaging, exchange and processing of *ebXML Message*s.

6.1 What ebXML Messaging Services does

ebXML Messaging Services (MS) defines, robust yet basic functionality necessary to transfer messages between two ebXML Message Services using various existing communication protocols. The ebXML Messaging Service will perform in a manner which will allow for reliability, persistence of messages, security, and extensibility.

6.2 Where ebXML Messaging Services May Be Implemented

The ebXML Messaging Services is expected to be implemented in environments requiring a robust, low cost solution to enable electronic business.

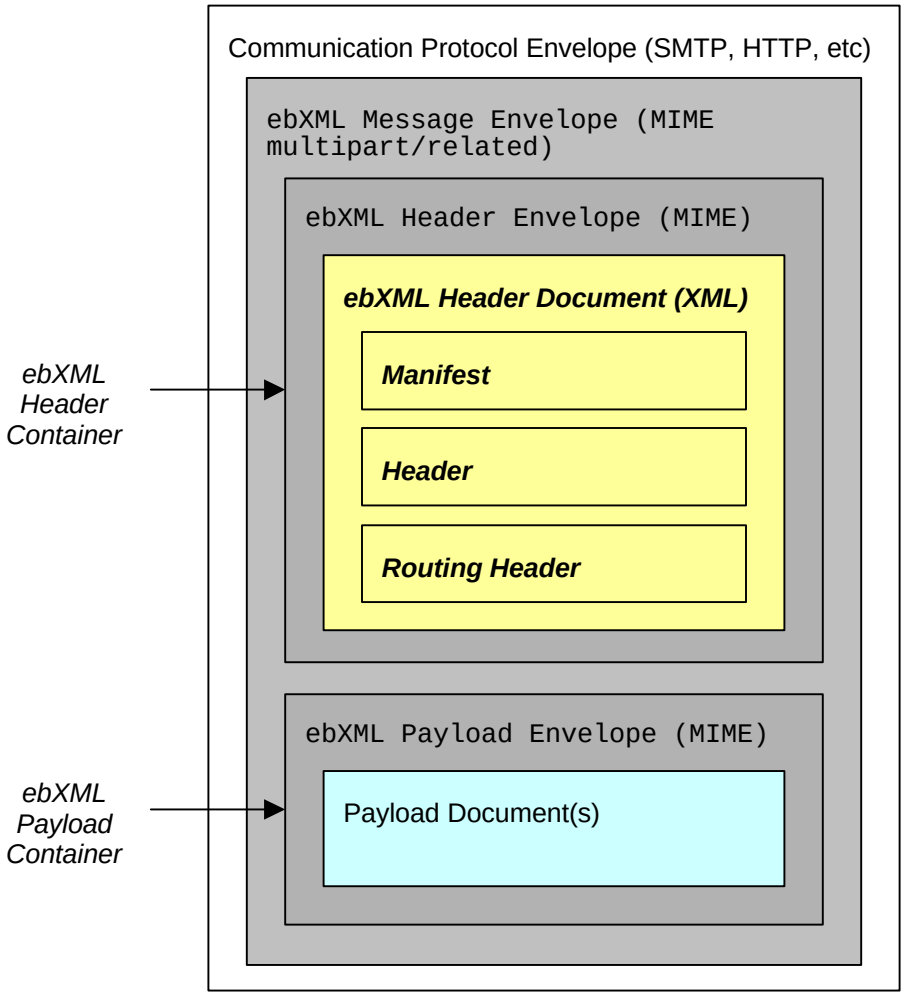
7 Definition and Scope

7.1 Packaging Specification

7.1.1 ebXML Message Structure

An *ebXML Message* consists of:

- an outer Communication Protocol Envelope, such as HTTP or SMTP,
- an inner communication “protocol independent” e*bXML 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, and
  - an optional, single *ebXML Payload Container* that MUST be used to envelope the actual payload (transferred data) of the Message



Note: The `Courier` font is used to represent MIME components. Items shown in ***bold italics*** represent XML items.

Figure 7-1 ebXML Message Structure

7.1.2 ebXML Header Envelope and Payload Envelope

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

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

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

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

7.1.3 MIME usage Conventions

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

7.2 ebXML Message Envelope

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

- `Content-Type`
- `Content-Length`

7.2.1 Content-Type

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

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

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

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

7.2.1.1 type Attribute

The MIME `type` attribute is used to identify the *ebXML Message Envelope* as an ebXML compliant structure. It conforms to a MIME XML Media Type [XMLMedia] and MUST be set to `"application/vnd.eb+xml"`. For example:

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

7.2.1.2 boundary Attribute

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

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

7.2.1.3 version Attribute

The MIME `version` attribute is used to identify the particular version of *ebXML Message Envelope* being used. All message headers SHOULD USE "1.0". For example:

```
version="1.0"
```

7.2.2 Content-Length

The MIME `Content-Length` header is a decimal value used to identify the total number of `OCTETS` contained in all constituent message body parts, including body part boundaries.

The value of the Content-Length MIME header is computed by counting the total number of OCTETS starting with the first OCTET after the CRLF following the first MIME header and ending with the OCTET immediately before the MIME object's last boundary string.

Example:

```
Content-Length: 9841
```

7.2.3 ebXML Message Envelope Example

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

```
Content-Type: multipart/related; type="application/vnd.eb+xml" "boundary:="-------8760"
charset="iso-8859-1" Content-Length: 9841
```

7.3 ebXML Header Container

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

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

The following rules apply:

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

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

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

The *ebXML Header Document* within the content portion of the MIME container MAY be enhanced during transport, provided it has not been digitally signed. Any change in the size of the *ebXML Header Document* MUST be reflected in `Content-Length` attribute of the *ebXML Message Envelope* and *ebXML Header Envelope*.

7.3.1 Content-ID

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

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

7.3.2 Content-Length

The MIME `Content-Length` header is a decimal value used to identify the total number of OCTETS contained in the *ebXML Header Container* MIME body part. For example:

```
Content-Length: 4208
```

7.3.3 Content-Type

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

- `version`, and
- `charset`

7.3.3.1 version Attribute

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

```
version="1.0";
```

7.3.3.2 charset Attribute

The MIME `charset` attribute identifies the character set used to create the ebXML Header Document. The list of valid values can be found at http://www.iana.org/.

The MIME `charset` attribute SHALL be equivalent to the encoding attribute of the *ebXML Header Document* (see section 7.6). For maximum interoperability it is RECOMMENDED that [UTF-8] be used. Note: this is not the default for MIME. For example:

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

7.3.4 ebXML Header Envelope Example

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

```
Content-ID: ebxmlheader-123@ebxmlhost.realm --|                          |
Content-Length: 2048                          |   MIME ebXML             |
Content-Type: application/vnd.eb+xml;         | Header Envelope          |
     version=”1.0”; 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.

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 7.8).

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

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

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

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

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

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

7.4.1 Content-ID

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

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

7.4.2 Content-Length

The MIME `Content-Length` header is a decimal value used to identify the total number of `OCTETS` contained in the content portion of the *ebXML Payload Container*. For example:

```
Content-Length: 5012
```

7.4.3 Content-Type

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

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

7.4.4 Example of an ebXML MIME Payload Container

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

```
Content-ID: ebxmlpayload-123@ebxmlhost.realm --|                          |
Content-Length: 4096                           | 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 B.

7.5 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 is likely to be used to generate that header-element,
- 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.

7.6 XML Prolog

The XML prolog for the *ebXML Header Document* SHALL contain the encoding attribute which SHALL be equivalent to the `charset` attribute of the MIME `Content-Type` of the ebXML Message Header Container (see section 7.3.3.2). It is RECOMMENDED that UTF-8 be used explicitly although this is one of the default values assumed if none is specified.

NOTE: The encoding attribute is OPTIONAL in the XML version 1.0 specification [XML], however, it is mandatory for the ebXML message header to ensure no conflicts occur with the `charset` attribute of the MIME Content-Type of the container and to ensure maximum interoperability. For example:

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

7.7 ebXMLHeader Element

The root element of the *XML ebXML Header Document* is named ***ebXMLHeader***. It is comprised of three XML attributes and two subordinate elements.

7.7.1 ebXMLHeader attributes

There are three attributes associated with the ***ebXMLHeader***, which are:

- Namespace (xmlns)
- Version
- MessageType

7.7.1.1 Namespace

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

7.7.1.2 Version

The ***Version*** attribute is required. Its purpose is to provide for future versioning capabilities. It has a default value of '1.0'.

7.7.1.3 MessageType

The purpose of the ***MessageType*** attribute is to enable ebXML-aware software to distinguish between normal and communication protocol-specific messages, such as acknowledgment and error messages. The ***MessageType*** is an enumeration consisting of three possible values:

- ***Normal*** – the ebXML Payload Container contains data that has been provided to the ebXML Messaging Service by the software that called it
- ***Acknowledgment*** – a ebXML Messaging Service-specific acknowledgment message.

- ***Error*** – an ebXML Messaging Service-specific error message.

7.7.2 ebXMLHeader elements

The ***ebXMLHeader*** element MUST contain the following two elements:

- Manifest
- Header

7.7.2.1 Manifest

The **Manifest** is a REQUIRED element that contains a list of references to he other parts of the Message. This includes references to the documents, which comprise the *Payload* of the *Message.*

7.7.2.2 Header

The **Header** is a REQUIRED element that contains the information REQUIRED by the recipient to process the message. The message originator creates this information to which additional information MAY be added.

7.7.3 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="1.0" MessageType="Normal">
        <Manifest>...</Manifest>
        <Header>...</Header>
</ebXMLHeader>
```

7.8 XML Manifest

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

7.8.1 XML DocumentReference

The ***DocumentReference*** element is a composite element consisting of three subordinate elements as follows:

- ***DocumentDescription***
- ***DocumentLabel***
- ***DocumentId***

7.8.1.1 DocumentDescription

The ***DocumentDescription*** is an OPTIONAL textual description of the document/resource.

7.8.1.2 DocumentLabel

The ***DocumentLabel*** is a code that enables the purpose of the referenced document to be determined without retrieving the referenced document.

7.8.1.3 DocumentId

The ***DocumentId*** is the URL of the `Content-ID` of a MIME body part, as defined in [RFC2392], representing payload data, or a remote URL to some external resource.

7.8.2 Manifest sample

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

```
<Manifest>
    <DocumentReference>
        <DocumentLabel>PurchaseOrder</DocumentLabel>
        <DocumentId>cid:0987654321</DocumentId>
    </DocumentReference>
</Manifest>
```

7.9 XML Header

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 required subordinate elements:

- ***From***
- ***To***
- ***TPAInfo***
- ***MessageData***
- ***ReliableMessagingInfo***

7.9.1 From and To

The ***From*** element identifies the *Party* which originated the message. It is a logical identifier, which MAY take the form of a URN. An example of this would be a DUNS number. The ***From*** element consists of a ***PartyId*** element.

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

The ***PartyId*** element has a single attribute; ***context*** and a text value. The purpose of the context attribute is to provide a context for the text value of the ***PartyId*** element. The following fragment demonstrates usage of the ***From*** and ***To*** elements of the ***ebXMLHeader***.

```
<From>
    <PartyId context="DUNS">1234567890123</PartyId>
</From>
<To>
    <PartyId context="DUNS">3210987654321</PartyId>
```

```
</To>
```

7.9.2 TPAInfo

The ***TPAInfo*** element follows the ***From*** and ***To*** elements in the ***Header*** element structure. The ***TPAInfo*** element is a composite set of information that relates to the *Trading Partner Agreement* under which the message is governed. The ***TPAInfo*** element has four subordinate elements as follows:

- ***TPAId***
- ***ConversationId***
- ***ServiceInterface***
- ***Action***

7.9.2.1 TPAId

The **TPAId** is a URI which identifies the *Trading Partner Agreement* which governs the processing of the message.

7.9.2.2 ConversationId

The **ConversationId** is a URI which identifies the set of related messages that make up a conversation between two ***Parties***.

7.9.2.3 ServiceInterface

The **ServiceInterface** identifies the Service Interface that SHOULD act on the payload in the message. It is unique within the domain of the ***Party*** to which the message is being sent. UNR's MAY be considered suitable for the element content.

7.9.2.4 Action

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

7.9.2.5 TPAInfo sample

The following example fragment demonstrates the usage of the ***TPAInfo*** element.

```
<TPAInfo>
        <TPAId context = "tpadb">12345678</TPAId>
        <ConversationId>987654321</ConversationId>
        <ServiceInterface>QuoteToCollect</ServiceInterface>
        <Action>NewPurchaseOrder</Action>
</TPAInfo>
```

7.9.3 MessageData

The required ***MessageData*** element follows the ***TPAInfo*** element. The purpose of the ***MessageData*** element is to provide a means of identifying an *ebXML Message*. It is a composite element that contains the following three elements:

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

7.9.3.1 MessageId

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

7.9.3.2 TimeStamp

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

7.9.3.3 RefToMessageId

For **Normal** and **Error** Messages, the ***RefToMessageId*** is an optional reference to an earlier ebXML Message. If there is no earlier message, the element MUST be empty. If element is not empty then it MUST contain the value of the ***MessageId*** of the earlier related ebXML Message.

For **Acknowledgment** Messages, the ***RefToMessageId*** reference is mandatory, and its value MUST be the ***MessageId*** of the ebXML Message being acknowledged.

7.9.3.4 MessageId sample

The following example demonstrates the usage of the ***MessageData*** element.

```
<MessageData>
    <MessageId>UUID-2</MessageId>
    <TimeStamp>20000725T121905.000Z</TimeStamp>
    <RefToMessageId>UUID-1</RefToMessageId>
</MessageData>
```

7.9.4 ReliableMessagingInfo

The last element of the ***ebXMLHeader*** is the ***ReliableMessagingInfo*** element. This element identifies the degree of reliability with which the message will be delivered. This element has a single attribute, ***DeliverySemantics.*** This attribute is an enumeration, which may have one of the following values:

- "OnceAndOnlyOnce" – reliable messaging semantics: the receiving Service Interface handler will receive a given message no more than once, the sending Messaging Service will execute retry procedures in the event of failure and the sending Service Interface handler will be notified in the event of failure.
- "BestEffort" – reliable delivery semantics are not specified: the Sending Service Interface handler is not notified of failure to deliver the message, duplicate messages might be delivered and persistent storages are not required.

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

7.9.5 XML Header sample

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

```
<Header>
    <From>...</From>
    <To>...</To>
    <TPAInfo>...</TPAInfo>
    <MessageData>...</MessageData>
    <ReliableMessagingInfo>...</ReliableMessagingInfo>
```

```
</Header>
```

7.10 XML Routing Header

One ***RoutingHeader*** element immediately follows the ***Header*** element. It is required in all ***ebXMLHeader*** documents. The ***RoutingHeader*** element is a composite element comprised of at least the following four required subordinate elements:

- ***SenderURI*** – the Sender's Messaging Service Handler URI.
- ***ReceiverURI*** – the Receiver's Messaging Service Handler URI.
- ***ErrorURI*** – URI designated by the Sender for reporting errors.
- ***Timestamp*** – timestamp of the ***RoutingHeader*** creation, in the same format used for ***Timestamp*** in the ***XML Header MessageData*** element.

When the ***RoutingHeader*** is used for a message sent with Reliable Messaging functions (***DeliverySemantics*** is set to "OnceAndOnlyOnce" in the ***XML Header ReliableMessagingInfo*** element), the Sender SHALL add one additional ***RoutingHeader*** element to the ***RoutingHeader***:

- ***SequenceNumber*** – Integer value that is incremented (e.g. 1, 2, 3, 4...) for each Sender-prepared message sent to the Receiver. The Sequence Number consists of ASCII numerals in the range 1-999,999,999. In following cases, the Sequence Number takes the value "1":
  - First message from the Sender to a particular Receiver
  - First message after wraparound (next value after 999,999,999)
  - First message after removing Sequence Number information in the Sender (Sender MAY remove Sequence Number information when it has no messages which were sent to the Receiver for long time).

The following fragment demonstrates the structure of the ***RoutingHeader*** element of the ***ebXMLHeader*** document when Reliable Messaging is used:

```
<RoutingHeader>
    <SenderURI>...</SenderURI>
    <ReceiverURI>...</ReceiverURI>
    <ErrorURI>...</ErrorURI>
    <Timestamp>...</Timestamp>
    <SequenceNumber>...</SequenceNumber>
</RoutingHeader>
```

7.11 Reliable Messaging Flow

The Reliable Messaging function defines an interoperable protocol such that any two Messaging Service implementations can "reliably" exchange messages that are sent using "reliable messaging" semantics.

Reliably exchanging messages means that, with respect to Sending and Receiving Message Service implementations:

- For any given message provided to the Sending Messaging Service, the Receiving Messaging Service will deliver at most one copy of the message to the Receiver.

- A positive acknowledgement will be sent from the Receiving Messaging Service to the Sending Messaging Service to indicate receipt and storage in persistent storage, and if this acknowledgement is not received the Sending Messaging Service will notify the original Sending Party
- Both the Sending and Receiving Messaging Services will use persistent storage for recovery

Reliable Messaging is defined only for direct connections between Messaging Service implementations. At a later time, networks consisting of intermediate Messaging Service implementations will be supported.

All ebXML Messaging Service implementations SHALL support the Reliable Messaging function. With respect to a particular Sender and Receiver pair, transmission of one reliable message SHALL be completed before another reliable message may be sent.

The following figure shows the reliable messaging flow:

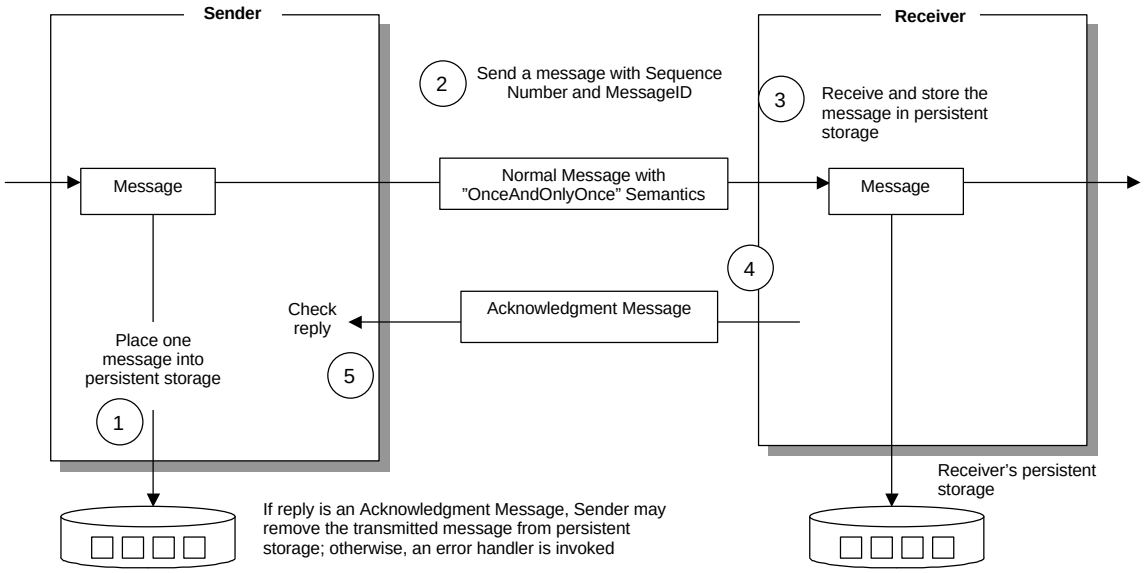


Figure 7-2: Reliable Message Transfer Sequence

Reliable Messaging processing is shown in the following sequence:

(1) Message preparation

Sender initially stores messages passed from the ebXML "From-Party" in persistent storage, and then prepare the stored message for message transfer.

(2) Sending message

A Reliable Message has ***DeliverySemantics*** = "*OnceAndOnlyOnce*", and receipt of a message with this value notifies the Receiver of Reliable Messaging semantics.

(3) Receiving, checking and storing message

The Receiver receives the reliable message and, if the message is not a duplicate message, stores the message in persistent storage and processes the message appropriately.

(4) Acknowledgment by Receiver

The Receiver returns an Acknowledgment Message to the Sender for every received reliable message, even if it is a duplicate message.

(5) Sender checks the acknowledgement and removes transferred message

Sender checks the Acknowledgement Message from the Receiver. If the reply is an appropriate Acknowledgement Message for the transferred message, Sender may remove the transferred message from Sender's persistent storage if the message is no longer needed for some other Messaging Service function or later failure recovery.

The Receiver's Messaging Service sends an Acknowledgement Message to the Sender's Messaging Service for every Normal Reliable Messaging message received. There is no reply to the Acknowledgement message from the Sender's Messaging.

In the Acknowledgement Message:

- The ***MessageType*** SHALL be "Acknowledgement"
- There is no Payload and no business level response information.
- ***From*** SHALL be the ***ReceiverURI*** as shown in the Routing Header Document
- ***To*** SHALL be the ***SenderURI*** as shown in the Routing Header Document
- ***TPAId*** and ***ConversationID*** as shown in the Header Document
- ***ServiceInterface*** and ***Action*** SHALL be empty
- ***RefToMessageId*** SHALL be the ***MessageId*** of the reliable message
- ***DeliverySemantics*** SHALL be "BestEffort"

7.12 Reliable Messaging Recovery Procedures

7.12.1 Messaging Service Parameters

In Reliable Messaging, the sending messaging service uses the following Messaging Service parameters during recovery procedures.

This information may be determined in a number of ways, such as the TPA or some other method.

Table 7-2 Messaging Service Parameters used in Recovery

Argument	Outline Description
Timeout	Wait time for any response from the Receiver. • Integer value specifying a number of seconds • After sending a Normal Message, the Sender SHALL wait for any response (MS Acknowledgement or Error Message) for the specified time before start of retry
Retries	Maximum number of retries. • Integer value specifying the number of retries • The Sender SHALL repeat retries the specified number of times until the Sender receives an MS Acknowledgement Message • If the Sender does not receive an MS Acknowledgement Message after the maximum number of retries, the Sender SHALL notify the incident to the higher level (application and/or system admin)

RetryInterval	Wait time between retries, if an Acknowledgement Message is not received • Integer value specifying a number of seconds • After a retry, the Sender SHALL wait for a response (MS Acknowledgement or Error Message) for specified time before start of the next retry

7.12.2 Recovery Sequence for Lost Messages

When the Sender detects a timeout while waiting for an Acknowledgement Message from the last sent message, the appropriate recovery handler in the Sender executes a Messaging Service recovery sequence.

The timeout value period is defined as ***Timeout***. The recovery sequence SHALL re-send the final message to the Receiver and SHALL use a retry interval, ***RetryInterval***, between attempts The retry sequence SHALL be attempted a ***Retries*** number of times.

The content of the re-sent message is exactly the same as the original message. In the recovery sequence or after the recovery sequence,

- If the Sender does not receive any error message or Acknowledgment Message in the retry interval, the recovery handler repeats the recovery sequence the ***Retries*** number of times.

- If the Sender detects or receives another Error Message, the recovery handler executes the appropriate recovery sequence for the error.

- If the Sender receives an Acknowledgment Message during the recovery sequence, the message transmission is completed.

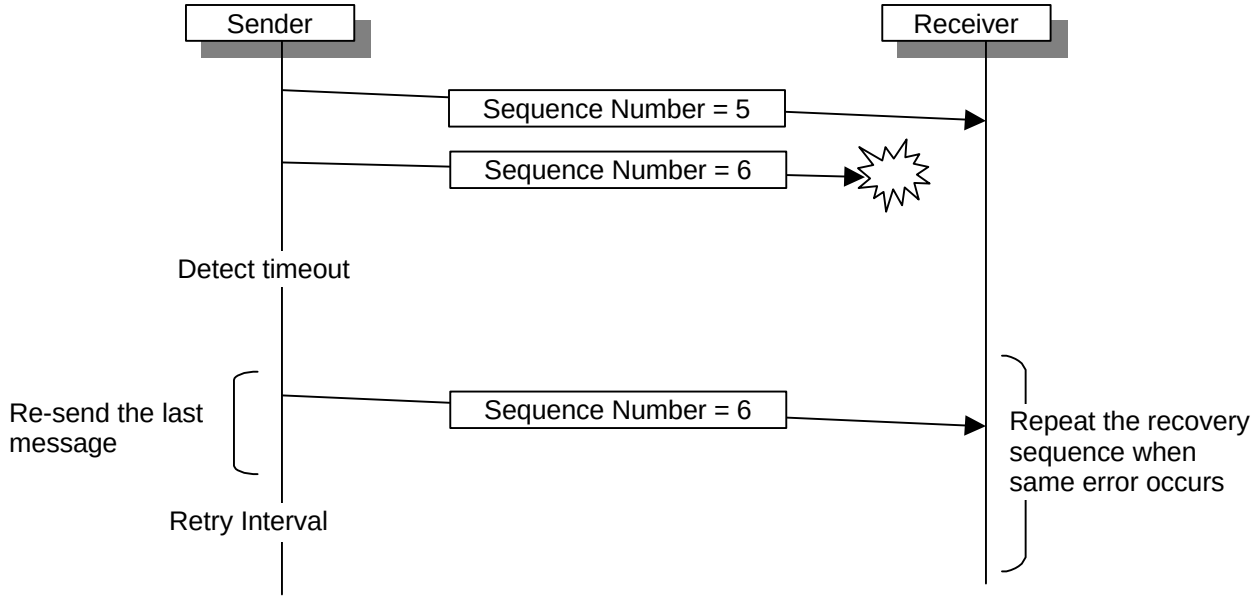


Figure 7-3 Recovery Sequence for Timeout

7.12.3 Maximum Number of Retries and Retry Interval

The retry interval is defined as ***RetryInterval***. When the total number of retries in a reliable message transmission reaches a maximum number, defined as ***Retries*****,** and the last error is still not resolved, the recovery handler will:

(1) Suspend sending messages to the Receiver

(2) Report this incident to a higher-level so that a system administrator can resolve this incident

When the system administrator resolves the incident, the recovery handler will reset the retry counter to zero and then re-start message transfer sequence from the uncompleted reliable message transmission.

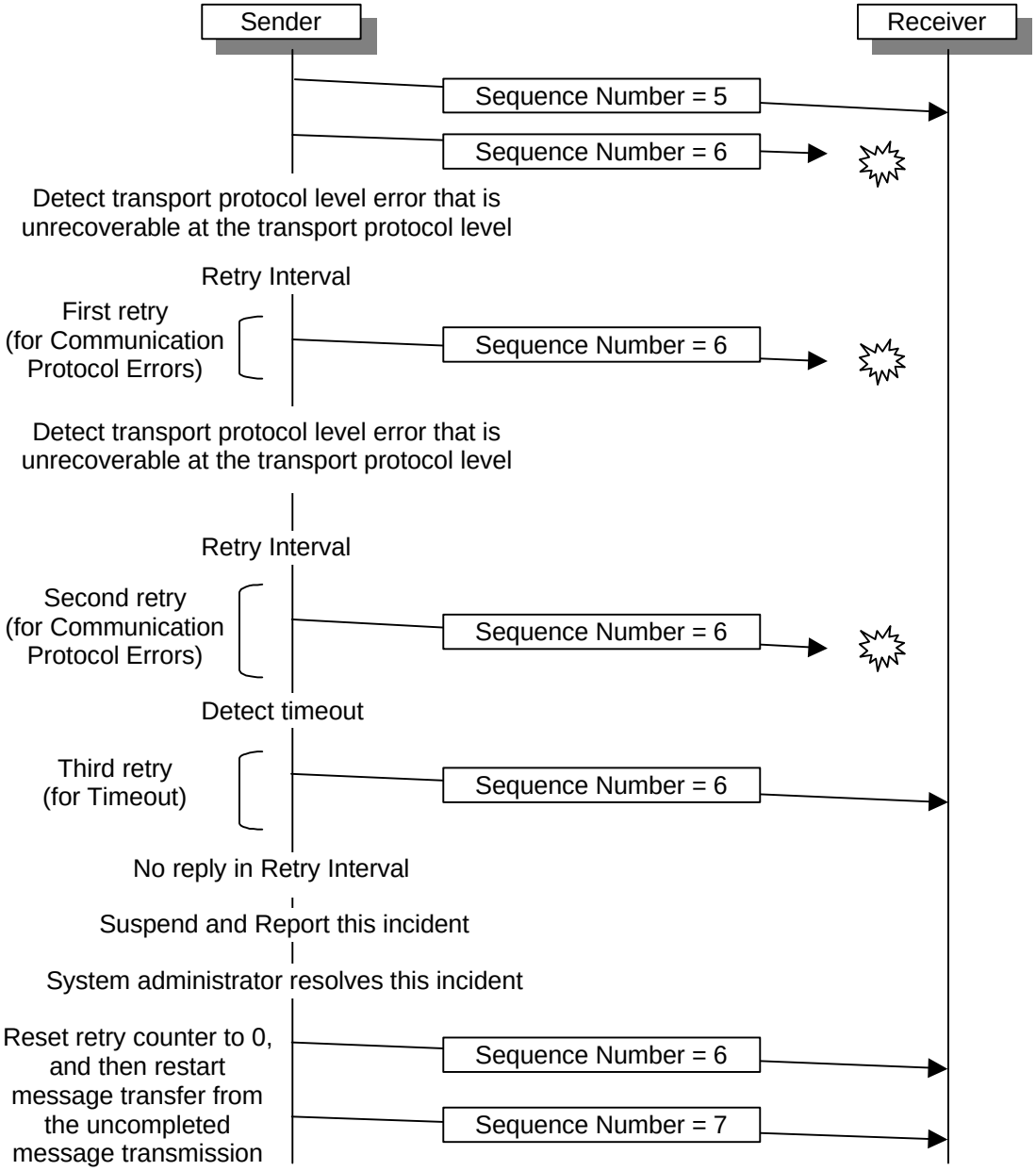


Figure 7-4 Repeat of Recovery Sequence (maximum number of retries specified is 3)

7.13 ebXML Error Reporting

This section describes how one ebXML Messaging Service reports errors it detects to another ebXML Messaging Service.

7.13.1 Definitions

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

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

7.13.2 Types of Errors

One ebXML Messaging Service needs to report to another ebXML Messaging Service errors in messa*ge in error* that are associated with:

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

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

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

7.13.3 When to generate Error Messages

When an ebXML Messaging Service detects an error in a *message in error*, a *message reporting the error* MUST be generated and delivered to the ebXML Messaging Service which sent the *message in error* for a normal ebXML message if:

- the Error Reporting Location (see section 7.13.4) to which the *message reporting the error* should be sent can be determined, and
- the message in error does not have a ***MessageType*** of ***Error***.

If the Error Reporting Location cannot be found or the *message in error* has a ***MessageType*** of ***Error***, it is recommended that:

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

7.13.4 Identifying the Error Reporting Location

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

- by reference, for example by using the ***TPAId*** to identify the Party Agreement which contains the Error Reporting Location, or
- by value, for example by using the ***ErrorURI*** contained within the Routing Header element.

If a *message* contains both an ***ErrorURI*** and other means of identifying the Error Reporting Location then the ***ErrorURI*** MUST be used.

If an ***ErrorURI*** is not used then the method used to determine the Error Reporting Location is outside of the scope of this version of the specification.

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

7.13.5 ebXML Error Message

This section defines the structure and content of an ebXML Error Message that is contained within a *message reporting an error*.

7.13.5.1 Message Structure

An ebXML Error Message is created using the rules for creating an ebXML Message contained within this specification. In addition:

- the ***MessageType*** in the ebXML header is set to ***Error***
- the payload consists of a single ebXML Error Document

7.13.5.2 ebXML Error Document

An ebXML Error Document has a root element that consists of:

- an ***ErrorHeader*** element that identifies the nature and severity of the error, and
- zero or more ***ErrorLocation*** elements, that identify the part(s) of the message(s) that are in error.

The structure of an ebXML Error Document is illustrated below.

```
<?xml version="1.0"?>
<ebXMLError xmlns="http://www.ebxml.org/namespaces/error"
    Version="1.0">
    <ErrorHeader>...</ErrorHeader>
    <ErrorLocation>...</ErrorLocation>
    <ErrorLocation>...</ErrorLocation>
    ...
</ebXMLError>
```

Later versions of this specification may define how to report more than one error within an ebXML Error Document.

7.13.5.3 Error Header Element

The ***ErrorHeader*** element identifies the nature and severity of the error. It consists of the following attributes/elements.

7.13.5.3.1 ID attribute

The optional ***ID*** attribute uniquely identifies the ***ErrorHeader*** Element within the document.

7.13.5.3.2 ErrorCode element

The required ***ErrorCode*** element indicates the nature of the error in the *message in error*. Valid values for the ***ErrorCode*** are given in section 7.13.5.8.

7.13.5.3.3 Severity element

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

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

7.13.5.3.4 Description element

The optional ***Description*** element provides a narrative description of the error in the language defined by the ***xml:lang*** attribute on the ***Description*** element. The content of this attribute is defined by the vendor/developer of the software, which generated the ebXML Error Document. ***xml:lang*** must comply with the rules for identifying languages specified in [XML].

7.13.5.3.5 SoftwareDetails element

The optional ***SoftwareDetails*** element contains a value that is set by the vendor/developer of the software, which generated the ebXML Error Document. It SHOULD contain data that enables the vendor/developer to identify the precise location in their software and the set of circumstances that caused the software to generate a *message reporting the error*. It is RECOMMENDED that this element include plain text 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 which can be used by the Software Vendor to identify the circumstances that caused the error

7.13.5.4 Examples

Two examples of an ***ErrorHeader*** element are given below.

```
<ErrorHeader ID='ab184832' >
  <ErrorCode>UnableToParse</ErrorCode>
  <Severity>Error</Severity>
  <Description xml:lang='en-uk'>The "MessageManifest" element is not well formed.</Description>
  <SoftwareDetails>Software Development Corp.; ebXML Connector!!; v2.7, build 2.7313; Ref
HA</SoftwareDetails>
</ErrorHeader>
```

```
<ErrorHeader ID='sdj2309823' >
  <ErrorCode>NotSupported</ErrorCode>
  <Severity>Error</Severity>
  <Description>xml:lang='en-us'>Delivery Semantics of "OnceAndOnlyOnce" are not
supported.</Description>
  <SoftwareDetails>Unreliable Software Development Corp.; ebXML Message Handler !!; v23.5, build 5751;
Ref: xapowekxd</SoftwareDetails>
</ErrorHeader>
```

7.13.5.5 Error Location Element

The ***ErrorLocation*** Element identifies the location of an error either within a message or elsewhere.

Frequently a single ***ErrorLocation*** element will be all that is required within an ebXML Error document. For example, an ***ErrorCode*** of ***ValueNotRecognized*** is likely to reference an element or attribute and no other ***ErrorLocation*** element will be needed.

Sometimes though, multiple ***ErrorLocation*** elements will be required to define where the problem is. For example, an error code with a value of ***Inconsistent*** would frequently have two or more ***ErrorLocation*** elements that point to the various items that are inconsistent. The number of ***ErrorLocation*** elements included in an ebXML Error document is an implementation decision.

An ***ErrorLocation*** element consists of the following attributes/elements:

- ***ID*** attribute
- ***RefToMessageId*** element
- ***Href*** element

7.13.5.5.1 ID Attribute

The optional ***ID*** attribute uniquely identifies the ***ErrorLocation*** element within the ebXML Error document.

7.13.5.5.2 RefToMessageId element

The optional ***RefToMessageId*** element contains the ***MessageId*** from the ebXML Header Document of the *message in error*. This must be present if a ***MessageId*** can be identified within the *message in error*.

7.13.5.5.3 Href element

The ***Href*** URI identifies either some other location within the *message in error*, or elsewhere, that helps identify the location of the error.

7.13.5.6 Examples

Two examples of an ***ErrorLocation*** element are given below. The first example is indicating that the referenced message is inconsistent with a previously agreed Party Agreement.

```
<ErrorLocation ID='4982hw'>
  <RefToMessageId>ab131982387123</RefToMessageId>
  <Href>url:example.com/tpa/471839<Href>
</ErrorLocation>
```

The second example is pointing to an error in an ebXML Header Document.

```
<ErrorLocation ID='120938uqwe'>
  <RefToMessageId>ac198327123098</RefToMessageId>
  <Href>cid:--------8760<Href>
</ErrorLocation>
```

7.13.5.7 ebXML Error Document Type Definition

The following is the DTD for the ebXML Error Document.

```
<?xml version ="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema">
<!ELEMENT ebXMLError (ErrorHeader, ErrorLocation*) >
<!ELEMENT ErrorHeader (ErrorCode, Severity, Description?, SoftwareDetails>
<!ATTLIST ErrorHeader
  ID              NMTOKEN #IMPLIED
<!ELEMENT ErrorCode (#PCDATA) > <-- string max 20 char -->
<!ELEMENT Severity (#PCDATA) > <-- Either 'Warning' or 'Error' -->
<!ELEMENT Description (#PCDATA) > <-- string max 1024 (?) char -->
<!ATTLIST Description
  xml:lang        NMTOKEN #REQUIRED >
<!ELEMENT SoftwareDetails(#PCDATA) > <-- string max 16k (?) chars -->

<!ELEMENT ErrorLocation (RefToMessageId?, Href ) >
```

```
<!ATTLIST ErrorLocation
  ID              NMTOKEN #IMPLIED >
<!ELEMENT RefToMessageId (#PCDATA) >
<!ELEMENT Href (#PCDATA) >
```

7.13.5.8 Error Codes

This section describes the ***ErrorCodes*** (see section 7.13.5.3.2) that are used in a *message reporting an error*. They are described as a list of bullet points. The following describes how to interpret this list:

- the first word is the actual ***ErrorCode***, e.g. ***UnableToParse***
- the single sentence that immediately follows the error code is a "narrative" that describes the ***ErrorCode***, for example "XML not well formed or invalid".
- the sentence(s) that follow the narrative, are the explanation of the meaning of the error and provide guidance on when the particular ***ErrorCode*** should be used.
- It is RECOMMENDED that implementers:
  - use both the ***ErrorCode*** and the "narrative" to explain an error to, for example, a user
  - translate the "narrative" into the preferred language of the recipient of *the message in error* if this is known

7.13.5.9 Reporting Errors in the ebXML Header Document

The following list contains error codes that can be associated with XML documents, for example the ebXML Header Document:

- ***UnableToParse*** - XML not well formed or invalid. The XML document is not well formed or not valid and cannot be successfully parsed. See [XML] for the meaning of "well formed" and "not valid".
- ***ValueNotRecognized*** - Element content or attribute value not recognized. Although the document is well formed and valid, the element/attribute contains a value which could not recognized and therefore could not be used by the ebXML Messaging Service
- ***NotSupported*** - Element or attribute not supported. Although the document is well formed and valid, an element or attribute is present that:
  - is consistent with the rules and constraints contained in this specification, but
  - is not supported by the ebXML Messaging 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 which do not conform to the rules and constraints contained in this specification and is not covered by other error codes. The ***Description*** element should be used to indicate the nature of the problem.

7.13.5.10 Non-XML Document Errors

The following are error codes that identify errors that are not associated with an XML Document:

- ***MessageTooLarge*** - Message too large. The message is too large to be processed by the ebXML Messaging 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
- ***Unknown*** - Unknown Error. Indicates that an error has occurred that is not covered explicitly by any of the other errors. The ***Description*** element should be used to indicate the nature of the problem.

Note this list will be expanded in future versions of this specification, for example to report errors on security.

7.14 Security

This version of the specification supports limited security services, that is those security services that can be supported within the payload. The `multipart/related` payload may be encrypted using cryptographic techniques suitable for the payload type. Expanded definition of security will be addressed in the Phase 2.

8 References

8.1 Normative References

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

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

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

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

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

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

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

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

[XML] Extensible Mark Up Language. A W3C recommendation. See http://www.w3.org/TR/1998/REC-xml-19980210 for the 10 February 1998 version.

8.2 Non-Normative References

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

9 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.

10 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:	(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:	(781) 442-3063
EMail:	chris.ferris@sun.com

Team Editor

Name	David Burnett
Company	CommerceOne
Street	4400 Rosewood Drive 3rd Fl, Bldg 4
City, State, Postal Code	Pleasanton, CA 94588
Country	USA
Phone:	(925) 520-4422 or (650) 623-2888
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:	(205) 250-8053
E-mail:	dick@8760.com

Name	David Burdett
Company	CommerceOne
Street	4400 Rosewood Drive 3rd Fl, Bldg 4
City, State, Postal Code	Pleasanton, CA 94588
Country	USA
Phone:	(925) 520-4422 or (650) 623-2888
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:	(781) 442-3063
EMail:	chris.ferris@east.sun.com

Name	John Ibbotson
Company	IBM UK Ltd
Street	Hursley Park
City, State, Postal Code	Winchester SO21 2JN
Country	United Kingdom
Phone:	+44 (1962) 815188
Email:	john_ibbotson@uk.ibm.com

Name	Nicholas Kassem
Company	Java Software, Sun Microsystems
Street	901 San Antonio Road, MS CUP02-201
City, State, Postal Code	Palo Alto, CA 94303-4900
Phone:	(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:	(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, CF241XF
Country	UK
Phone:	+44 29 2072 4063
EMail:	ian.c.jones@bt.com

Name	Martha Warfelt
Company	DaimlerChrysler Corporation
Street	800 Chrysler Drive
City, State, Postal Code	Auburn Hills, MI
Country	USA
Phone:	(248) 944-5481
EMail:	maw2@daimlerchrysler.com

Appendix A Schemas and DTD Definitions

The following are definitions for validation of the *ebXML Message* header structure.

A.1 XML Header DTD

```
<?xml version ="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema">

<!ELEMENT ebXMLHeader  (Manifest , Header )>
<!ATTLIST ebXMLHeader  Version     CDATA  #FIXED '1.0'
                  MessageType CDATA  #FIXED 'Normal' >
<!ELEMENT Manifest  (DocumentReference )+>
<!ELEMENT DocumentReference  (Document Description?, DocumentLabel , DocumentId )>
<!ELEMENT DocumentDescription  (#PCDATA )>
<!ATTLIST DocumentDescription e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT DocumentLabel  (#PCDATA )>
<!ATTLIST DocumentLabel  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT DocumentId  (#PCDATA )>
<!ATTLIST DocumentId  e-dtype NMTOKEN  #FIXED 'uri' >
<!ELEMENT Header  (From , To , TPAInfo , MessageData , ReliableMessagingInfo )>
<!ELEMENT TPAInfo  (TPAId , ConversationId , ServiceInterface , Action )>
<!ELEMENT ServiceInterface  (#PCDATA )>
<!ATTLIST ServiceInterface  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT Action  (#PCDATA )>
<!ATTLIST Action  e-dtype NMTOKEN  #FIXED 'string' >
<!ELEMENT TPAId  (#PCDATA )>
<!ATTLIST TPAId  context CDATA    'Undefined'
            e-dtype NMTOKEN  #FIXED 'uri' >
<!ELEMENT ConversationId  (#PCDATA )>
<!ATTLIST ConversationId  context CDATA    'Undefined'
                  e-dtype NMTOKEN  #FIXED 'uri' >
<!ELEMENT MessageData  (MessageId , TimeStamp , RefToMessageId )>
<!ELEMENT RefToMessageId  (#PCDATA )>
<!ATTLIST RefToMessageId  e-dtype NMTOKEN  #FIXED 'uuid' >
<!ELEMENT MessageId  (#PCDATA )>
<!ATTLIST MessageId  e-dtype NMTOKEN  #FIXED 'uuid' >
<!ELEMENT From  (PartyId )>
<!ELEMENT To  (PartyId )>
<!ELEMENT PartyId  (#PCDATA )>
<!ATTLIST PartyId  context CDATA    'Undefined'
              e-dtype NMTOKEN  #FIXED 'uri' >
<!ELEMENT ReliableMessagingInfo EMPTY>
<!ATTLIST ReliableMessagingInfo  DeliverySemantics  (OnceAndOnlyOnce | BestEffort )  #FIXED 'BestEffort' >
<!ELEMENT RoutingHeader (SenderURI , ReceiverURI , ErrorURI, Timestamp,
```

```
   SequenceNumber )>
<!ELEMENT SenderURI (#PCDATA )>
<!ATTLIST SenderURI e-dtype NMTOKEN #FIXED 'uri' >
<!ELEMENT ReceiverURI (#PCDATA )>
<!ATTLIST ReceiverURI e-dtype NMTOKEN #FIXED 'uri' >
<!ELEMENT ErrorURI (#PCDATA )>
<!ATTLIST ErrorURI e-dtype NMTOKEN #FIXED 'uri' >
<!ELEMENT TimeStamp  (#PCDATA )>
<!ATTLIST TimeStamp  e-dtype NMTOKEN  #FIXED 'dateTime' >
<!ELEMENT SequenceNumber (#PCDATA )>
```

A.2 XML Header Schema Definition

```
<?xml version ="1.0"?>
<schema xmlns = "http://www.w3.org/1999/XMLSchema">
        <element name = "ebXMLHeader">
                <complexType content = "elementOnly">
                        <sequence>
                                <element ref = "Manifest"/>
                                <element ref = "Header"/>
                        </sequence>
                        <attribute name="Version" use="fixed" value="1.0" type="string"/>
                        <attribute name="MessageType" use="fixed" value="Normal" type = "string"/>
                </complexType>
        </element>
        <element name = "Manifest">
                <complexType content = "elementOnly">
                        <sequence minOccurs = "0" maxOccurs = "unbounded">
                                <element ref = "DocumentReference"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "DocumentReference">
                <complexType content = "elementOnly">
                        <sequence minOccurs = "1" maxOccurs = "unbounded">
                                <element ref = "DocumentDescription" />
                                <element ref = "DocumentLabel"/>
                                <element ref = "DocumentId"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "DocumentLabel" type = "string">
        </element>
        <element name = "DocumentId" type = "uri">
        </element>
        <element name = "Header">
                <complexType content = "elementOnly">
                        <sequence>
                                <element ref = "From"/>
                                <element ref = "To"/>
                                <element ref = "TPAInfo"/>
                                <element ref = "MessageData"/>
                                <element ref = "ReliableMessagingInfo"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "BusinessServiceInterface" type = "string">
        </element>
        <element name = "Action" type = "string"/>
        <element name = "TPAId">
                <complexType base = "uri" content = "textOnly">
```

```
                        <attribute name="context" use="default" value="Undefined" type = "string"/>
                    </complexType>
                </element>
                <element name = "ConversationId">
                    <complexType base = "uri" content = "textOnly">
                        <attribute name = "context" use = "default" value = "Undefined" type = "string"/>
                    </complexType>
                </element>
                <element name = "MessageData">
                    <complexType content = "elementOnly">
                        <sequence>
                            <element ref = "MessageId"/>
                            <element ref = "TimeStamp"/>
                            <element ref = "RefToMessageId"/>
                        </sequence>
                    </complexType>
                </element>
                <element name = "RefToMessageId" type = "uuid">
                </element>
                <element name = "TimeStamp" type = "dateTime">
                </element>
                <element name = "MessageId" type = "uuid">
                </element>
                <element name = "From">
                    <complexType content = "elementOnly">
                        <sequence>
                            <element ref = "PartyId"/>
                        </sequence>
                    </complexType>
                </element>
                <element name = "To">
                    <complexType content = "elementOnly">
                        <sequence>
                            <element ref = "PartyId"/>
                        </sequence>
                    </complexType>
                </element>
                <element name = "PartyId">
                    <complexType base = "uri" content = "textOnly">
                        <attribute name = "context" use = "default" value = "Undefined" type = "string"/>
                    </complexType>
                </element>
                <element name = "ReliableMessagingInfo">
                    <complexType content = "empty">
                        <attribute name = "DeliverySemantics" use = "fixed" value = "Unspecified">
                            <simpleType base = "ENUMERATION">
                                <enumeration value = "OnceAndOnlyOnce"/>
                                <enumeration value = "BestEffort"/>
                            </simpleType>
                        </attribute>
                    </complexType>
                </element>
                <element name = "TPAInfo">
                    <complexType content = "elementOnly">
                        <sequence>
                            <element ref = "TPAId"/>
                            <element ref = "ConversationId"/>
                            <element ref = "BusinessServiceInterface"/>
                            <element ref = "Action"/>
                        </sequence>
                    </complexType>
                </element>
</schema>
```

Appendix B Examples

The following are complete examples of *ebXML Message*s showing the structure as defined in this specification.

B.1 Complete Example of an ebXML Message Envelope using multipart/related Content-Type sent via HTTP POST

```
POST http://127.0.0.1:9090/servlet/AS2Appl HTTP/1.0
Connection: Keep-Alive
User-Agent: Group 8760 Java MultiPost
Content-type: multipart/related; type="application/vnd.eb+xml"; version="0.21"; boundary=----------8760567890----
Content-Length: 2717

----------8760567890----
Content-ID: ebxmlheader-8760-901543739
Content-Length: 1357
Content-type: application/vnd.eb+xml; version="1.0"; charset="UTF-8"

<?xml version ="1.0" encoding="UTF-8"?>
<!DOCTYPE ebXMLHeader SYSTEM "level1-10122000.dtd">
<ebXMLHeader
        xmlns = "http://www.ebxml.org/namespaces/messageHeader"
        Version = "1.0"
        MessageType = "Normal">
 <Manifest>
  <DocumentReference>
        <DocumentLabel>Purchase Order Request Action</DocumentLabel>
        <DocumentId>cid:8760.com901543736</DocumentId>
        <DocumentDescription xml:lang="en-us">GCI Purchase Order</DocumentDescription>
  </DocumentReference>
 </Manifest>
 <Header>
  <From>
   <PartyId context = "DUNS">2059397184</PartyId>
  </From>
  <To>
   <PartyId context = "DUNS">943561654</PartyId>
  </To>
  <TPAInfo>
   <TPAId>/2059397184/943561654GCIPO-20000202</TPAId>
   <ConversationId>8760.com901543737</ConversationId>
   <ServiceInterface>OrderProcessing</ServiceInterface>
   <Action>ProcessNewOrder</Action>
  </TPAInfo>
```

```
    <MessageData>
      <MessageId>8760.com901543738</MessageId>
      <Timestamp>20001014175625510.000Z</Timestamp>
      <RefToMessageId>Not Applicable</RefToMessageId>
    </MessageData>
    <ReliableMessagingInfo DeliverySemantics ="OnceAndOnlyOnce"/>
  </Header>
  <RoutingHeader>
    <RouteInfo>
      <SenderURI>ford.com/ebXMLHandler</SenderURI>
      <ReceiverURI>gm.com/ebXMLHandler</ReceiverURI>
      <ErrorURI>mailto:ebxmlerrors@ford.com</ErrorURI>
      <Timestamp>20001014175625510.000Z</Timestamp>
      <SequenceNumber>00001</SequenceNumber>
    </RouteInfo>
  </RoutingHeader>
</ebXMLHeader>
--------------------------8760567890----
Content-Length: 1043
Content-ID: 8760.com901543736
Content-type: application/xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Order SYSTEM "OrderV0.1072400.dtd">
<Order actionRequestStatusIndicator="Create">
        <Document id="" status="COPY" type="" language="EN">
                <creationDate date="2000-02-02"/>
        </Document>
        <buyer>
                <PartyIdentification>
                        <GlobalLocationNumber>4325335000003</GlobalLocationNumber>
                </PartyIdentification>
                <BuyerAlignmentIdentification>Buyer</BuyerAlignmentIdentification>
        </buyer>
        <seller>
                <PartyIdentification>
                        <GlobalLocationNumber/>
                </PartyIdentification>
                <SellerAlignmentIdentification/>
        </seller>
        <shipTo>
                <PartyIdentification>
                        <GlobalLocationNumber/>
                </PartyIdentification>
                <ShipToAlignmentIdentification/>
        </shipTo>
```

```
        <requestedMovementType movement="requestedDeliveryDate">
                <movementDate date="2000-03-03"/>
        </requestedMovementType>
        <lineItem lineItemNumber="1">
                <itemId>
                        <GlobalTradeItemNumber/>
                </itemId>
                <requestedQuantity value="100"/>
                <price netPrice="1000.00" currencyOfNetPrice=""/>
        </lineItem>
</Order>

------------------------8760567890------
```

B.2 Complete Example of an ebXML Message Envelope using multipart/related Content-Type sent via SMTP

The default Content-transfer-encoding type of 7BIT is being used in this message.

```
From dick@8760.com Sun May  7 17:01:14 2000
Received: from granger.mail.mindspring.net by alpha2000.tech-comm.com; (8.8.5/1.1.8.2/05Jun95-1217PM)
        id RAA32702; Sun, 7 May 2000 17:01:13 -0500 (CDT)
Received: from gamma (user-33qt10l.dialup.mindspring.com [199.174.132.21])
        by granger.mail.mindspring.net (8.9.3/8.8.5) with SMTP id SAA11942
        for <ebxmlhandler@8760.com>; Sun, 7 May 2000 18:11:14 -0400 (EDT)
From: "Dick Brooks (E)" <dick@8760.com>
To: <ebxmlhandler@8760.com>
Subject: OTA Commission Event
Date: Sun, 7 May 2000 17:07:38 -0500
Message-ID: <NDBBIOBLMLCDOHCHIKMGKEEIDAAA.dick@8760.com>
MIME-Version: 1.0
X-Priority: 3 (Normal)
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook IMO, Build 9.0.2416 (9.0.2910.0)
Importance: Normal
X-MimeOLE: Produced By Microsoft MimeOLE V5.00.2314.1300
Content-Length: 8081
Content-Type: multipart/related; type="application/vnd.eb+xml"; version="0.21";  boundary="---
=_NextPart_000_0005_01BFB846.BF7FABA0"

------=_NextPart_000_0005_01BFB846.BF7FABA0
Content-Type: application/vnd.eb+xml; charset=”UTF-8”
Content-ID: ebxmlheader-9000
Content-Length: 272

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

```
<ebXMLHeader xmlns = “http://www.ebxml.org/namespaces/messageHeader"
          Version = "1.0"
          MessageType = "Normal">
   <Manifest>
      <DocumentReference>
         <DocumentLabel>Purchase Order Request Action</DocumentLabel>
         <DocumentId>
            cid:uid@originator-domain [C-ID of the payload MIME part]
         </DocumentId>
      </DocumentReference>
   </Manifest>
   <Header>
      <From>
         <PartyId context = "DUNS">requester-DUNS-number</PartyId>
      </From>
      <To>
         <PartyId context = "DUNS">responder-DUNS-number</PartyId>
      </To>
      <TPAInfo>
         <TPAId context = "tpadb">
            /requester-DUNS-number/responder-DUNS-number/PIP3A4/1.1
         </TPAId>
         <ConversationId context = “CreatePurchaseOrder">
            uid@requester-domain
         </ConversationId>
         <BusinessServiceInterface>
            Seller Service
         </BusinessServiceInterface>
         <Action version=”1.1”>Purchase Order Request Action</Action>
      </TPAInfo>
      <MessageData>
         <MessageId>uid@requester-domain</MessageId>
         <TimeStamp>CCYYMMDDThhmmss.sssZ</TimeStamp>
         <RefToMessageId>Not Applicable</RefToMessageId>
      </MessageData>
      <ReliableMessagingInfo DeliverySemantics = "Unspecified"/>
   </Header>
</ebXMLHeader>
------=_NextPart_000_0005_01BFB846.BF7FABA0
Content-Type: text/xml
Content-ID: ebxmlpayload-9000
Content-Length: 7515

<?xml version="1.0" encoding="UTF-8"?>
<HITISMessage xmlns="" Version="1.0">
          <Header OriginalBodyRequested="false" ImmediateResponseRequired="true">
```

```
                    <FromURI>http://www.pms.com/HITISInterface</FromURI>
                    <ToURI>http://www.crs.com/HITISInterface</ToURI>
                    <ReplyToURI>http://www.pms.com/HITISInterface</ReplyToURI>
                    <MessageID>1234567890</MessageID>
                    <OriginalMessageID>1234567890</OriginalMessageID>
                    <TimeStamp>1999-11-10T10:23:44</TimeStamp>
                    <Token>1234-567-8901</Token>
          </Header>
          <Body>
                    <HITISOperation OperationName="CommissionEventsUpdate">
                              <BodyPartstuffgoeshere/>
                    </HITISOperation>
          </Body>
</HITISMessage>
------=_NextPart_000_0005_01BFB846.BF7FABA0--
```

Appendix C Candidate Packaging Technologies and Selection Process

The packaging sub-group began its investigation of packaging technologies by identifying the technologies currently used for business-to-business message exchange or were being developed for this purpose. The following packaging technologies were identified:

- MIME - currently in use by companies exchanging business transactions using E-mail and HTTP
- XML - currently used by RosettaNet and Microsoft (BizTalk and SOAP) and others

C.1 Selection Process

Each candidate technology was evaluated based on its ability to meet the requirements listed in the section titled "Packaging and other Requirements" in this document. When necessary, specific parties were contacted to provide details describing how a technology was being used to meet specific requirements. The following parties were contacted to provide expert insight:

- Microsoft - David Turner, regarding use of XML packaging in BizTalk
- Develop Mentor - Don Box, regarding use of XML packaging in SOAP
- Vitria - Prasad Yendluri, regarding use of XML packaging in RosettaNet
- Jonathan Borden - author of [XMTP], an XML to MIME transformation tool

The packaging sub-group considered the inputs of people from the ebXML Transport mailing list as well as the parties listed above, before making a selection.

C.2 MIME

Multipurpose Internet Mail Extensions (MIME) is an international standard created by the Internet Engineering Task Force. It has been implemented by numerous software vendors across the globe and has been used to exchange mixed type payloads, including XML, for several years. MIME was designed purely as a packaging (enveloping) solution to allow the transport of mixed

payloads using Internet E-mail (SMTP). MIME is also being used by other transport technologies as a packaging technology, most notably HTTP.

C.3 XML

eXtensible Markup Language (XML) version 1.0 is a technical specification holding a RECOMMENDED status created by the World Wide Web Consortium. It has been implemented by numerous software vendors across the globe and has been used to describe a broad spectrum of document structures from very simple to very complex. XML is a very flexible markup language that can be used to represent virtually any type of document. XML can be used solely for packaging (enveloping) documents of any type, providing the data can be "transformed" into "legal" XML.

In some cases, XML documents MUST be placed into transport specific "envelopes" before being transported. For example, XML data MUST be placed in a MIME envelope when being transported via SMTP or HTTP.

C.4 Conclusion

The packaging sub-group examined the capabilities of both XML and MIME relative to the list of packaging requirements above. It's important to note that neither technology met all of the ebXML requirements and in the end it was the packaging sub-groups assessment of which technology came closest to meeting ALL of the ebXML requirements that determined which technology SHOULD be used.

MIME was chosen to serve as the ebXML packaging technology, over XML, based on the information contained in the following table:

Reason	Requirement(s) Satisfied
There is no formal packaging recommendation within IETF or W3C, based on XML. If ebXML were to choose XML as a packaging technology it would be required to define an XML packaging specification and submit this to IETF or W3C for adoption as a formal standard.	to not reinvent the wheel - re-use where possible [TRPREQ]
XML requires that binary and other types of payload data including XML documents be base64 encoded in order to be encapsulated within a XML root document. Base64 encoding ensures that no illegal XML characters exist within a document and recursive XML documents are "hidden". Base64 encoding imposes a significant processing overhead and results in larger messages, which affect both transmission and processing times. Base64 encoding of binary data is required of MIME content when being transported by SMTP, but this is a transport level requirement, not a requirement imposed by MIME. Binary data can be packaged and transported without alteration when using MIME over HTTP	Minimize intrusion to payload (special encoding or alteration) Low processing overhead
At the time of defining this specification there is no industry standard way to package an encrypted message, or portion of a message, using XML.	All or part of the documents in a message MAY be encrypted prior to sending [TRPREQ]
MIME could be used in conformance within existing IETF recommendations, no additions or changes are	to not reinvent the wheel - re-use where possible [TRPREQ]

initially required to produce a functional envelope.	

Appendix D MIME Type discussion

Three MIME media types were considered to serve as Content-Type for the *ebXML Message Envelope*:

- Multipart/related
- Multipart/Mixed
- Multipart/form-data
-

The group selected the multipart/related media type to serve as the preferred message envelope Content-Type.

Note:
There was some discussion over the similarities of multipart/related and multipart/mixed, both of which appear to offer similar capabilities and both could meet stated requirements. However, the group converged on multipart/related, believing it to be more semantically appropriate for ebXML. There was significant discussion over whether to support multipart/form-data as an alternate Content-Type for message-envelope, due to the large installed base of web browsers that support this Content-Type.

It was determined that multipart/related was a more generic Content-Type than multipart/form-data and the multipart/related Content-Type is the preferred Content-Type for ebXML Message Envelopes. Multipart/form-data Content-Type is typically associated with HTTP/HTML web forms, whereas multipart/related can be associated with any type of data.

Additionally, due to limitations in their handling of multipart ebXML payloads it was determined that existing web browsers are unable to support the full breadth of functions needed to package complex *ebXML Message*s containing multipart payloads. Therefore browser vendors are encouraged to add support for the ebXML enveloping standard as specified in this document.

Appendix E Communication Protocol Interfaces

The ebXML Messaging Service messages are carried by Transport Protocols as shown in the following sections.

E.1 HTTP [RFC 2068]

All ebXML Messaging Service messages are carried by an HTTP Request Message (POST method). The HTTP Response Message to an HTTP Request Message has no entity body. The following Figure E.1 shows how a Normal Message and its corresponding Acknowledgement Message (when Reliable Messaging is used) are carried using HTTP:

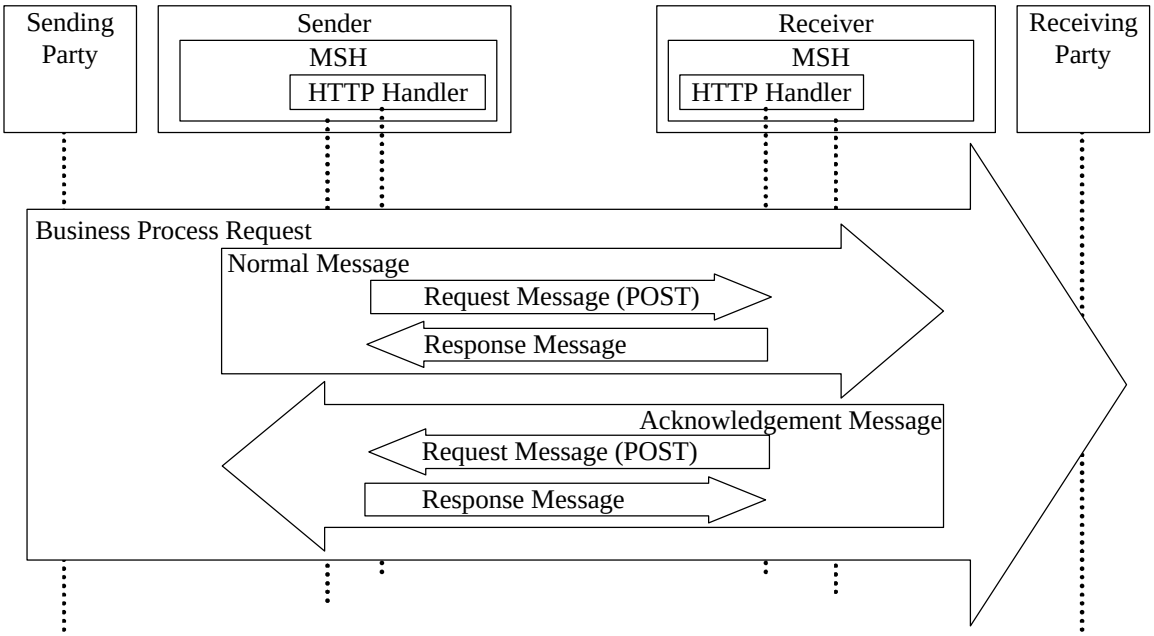


Figure E.1 HTTP Flow

Table E.1 Relationship with HTTP

ebXML Messaging Service message	*HTTP*
Normal Message	Request Message (POST method) from Sender to Receiver Response Message to the Request Message has no entity body
Acknowledgement Message	Request Message (POST method) from Receiver to Sender Response Message to the Request Message has no entity body

Error Message	Request Message (POST method) from Receiver to Sender Response Message to the Request Message has no entity body

E.2 SMTP [RFC 821]

All ebXML Messaging Service messages are carried as mail in an SMTP Mail Transaction as shown in the following Figures.

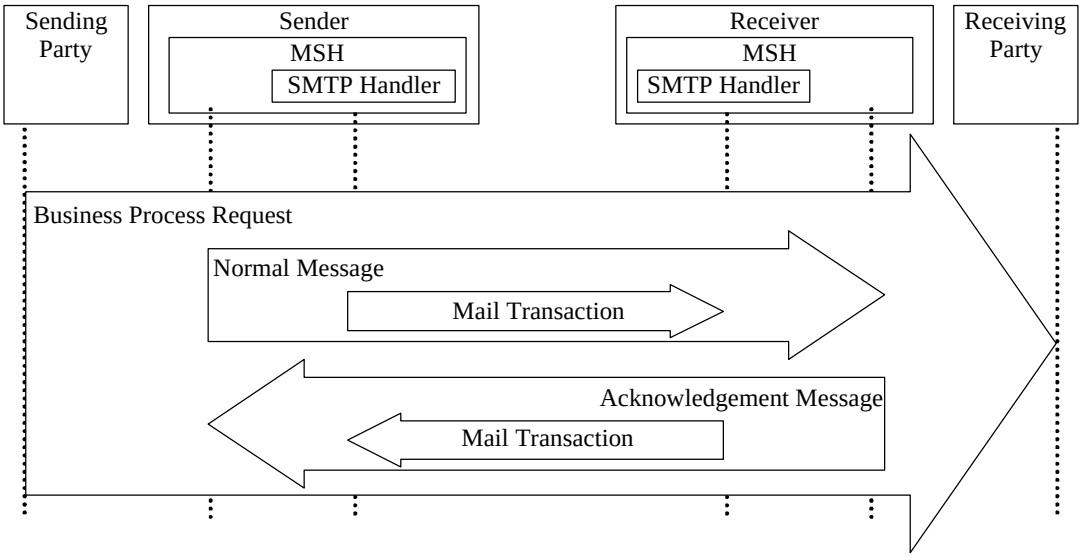


Figure E.2 SMTP Flow

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

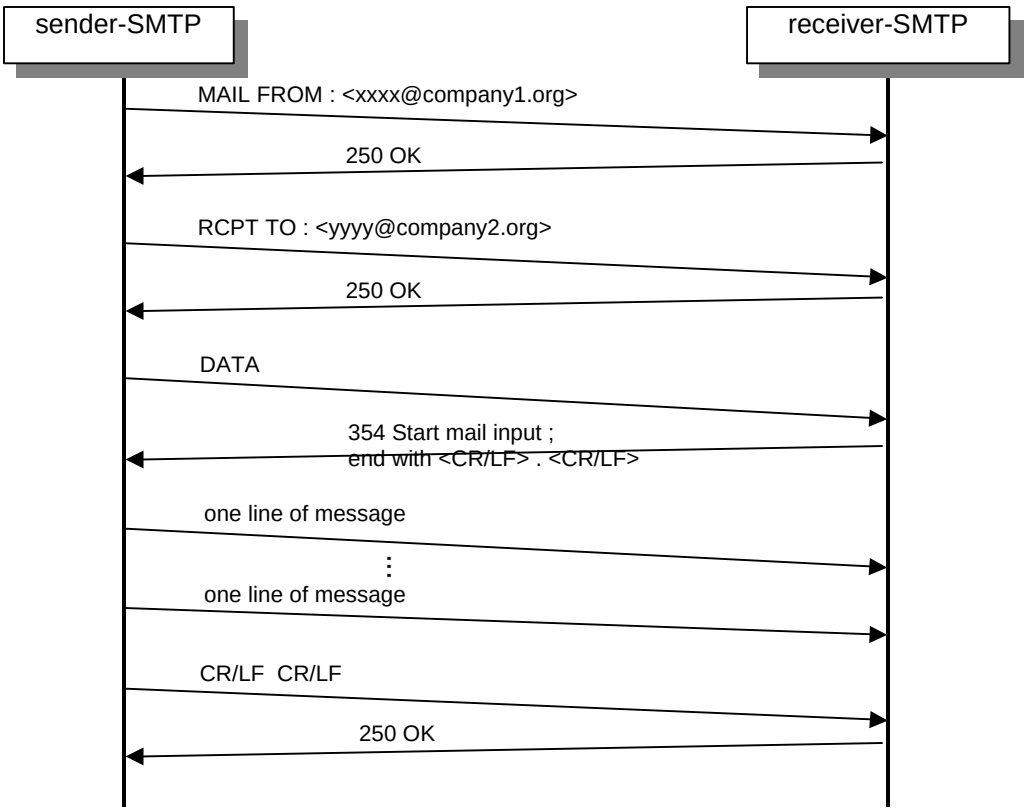


Figure E.3 SMTP Sequence

Table E.2 Relationship with SMTP

ebXML Messaging Service message	*SMTP*
Normal Message	Mail Transaction from Sender to Receiver
Acknowledgement Message	Mail Transaction from Receiver to Sender
Error M**e**ssage	Mail Transaction from Receiver to Sender

E.3 FTP [RFC 959]

This section to be added.

E.4 Communication Protocol Errors during Reliable Messaging

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

However, if the Sender detects a transport protocol level error that is unrecoverable at the transport protocol level, the appropriate recovery handler in the Sender will execute a Messaging

Service recovery sequence. This recovery sequence SHALL use a retry interval and SHALL re-send the last message to the Receiver. The format of the re-sent message is exactly the same as the original message. In the recovery sequence or after the recovery sequence:

- If the Sender detects a transport protocol level error again, which is unrecoverable at the transport protocol level, the recovery handler repeats the recovery sequence for an implementation-defined number of times.
- If the Sender detects or receives another error, the recovery handler executes an appropriate recovery sequence for the error.
- If the Sender receives an Acknowledgment Message, the message transmission is completed.

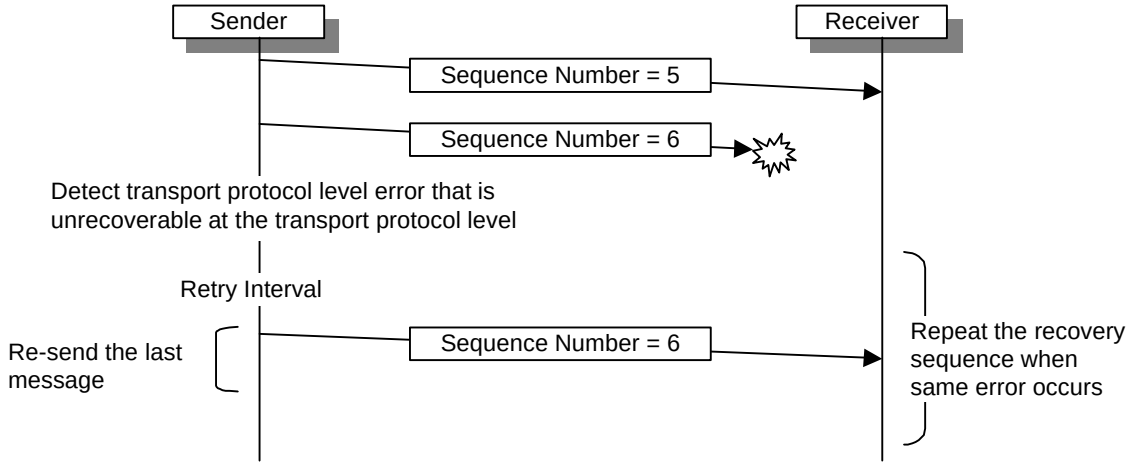


Figure E.4 Recovery Sequence for Communication Protocol Errors

Appendix F Detailed list of the Messaging Services Requirement Phases

(This section to be added.)

Copyright Statement