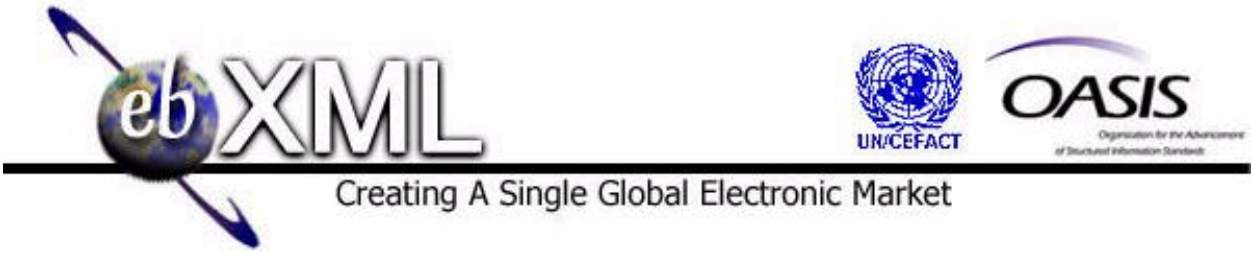


ebXML Transport, Routing & Packaging Strawman Message Header Specification

Working Draft 26-May-2000

This version:
ebXML Message Header Specification v0-5.doc
Download from http://www.ebxml.org/specindex.htm
Latest version:
http://www.ebxml.org/project_teams/requirements.htm
Previous version:
None
Editor:
David Burdett <david.burdett@commerceone.com>
Authors:
David Burdett <david.burdett@commerceone.com>
John Ibbotson <john_ibbotson@uk.ibm.com>
Contributors:
The members of the Transport Routing and Packaging Project Team

---

Abstract

This specification contains the definitions of the Message Header elements required in an ebXML Message. The purpose of the header elements is to contain the additional information required to enable electronic documents to be transported between two services. The key consideration when developing this specification was to provide data elements that enable software that is aware of this specification to:

- route messages to service handlers (these can be applications) that can process the message payload
- easily identify the content of the Message header and payload(s) within the message
- report errors found and understand errors received in the Message Header
- provide acknowledgements that messages have been received and placed in persistent storage
- provide reliable, at most once, delivery of a message.

Note that the reliable delivery of messages is described in a separate specification (under development).

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

Status of this Document

This specification is the first publicly available version of the ebXML Message Header. The header elements that have been identified represent those that are considered by the members of the ebXML Transport, Routing and Packaging Working group, to be the minimum required that provide the functionality outlined in the abstract. They meet subset of the requirements described in the ebXML Overview and Requirements document [1].

One of the objectives of ebXML Transport, Routing and Packaging is to develop specifications that "enable existing 'messaging' solutions to 'bridge' to the ebXML solution".

Therefore later versions of this specification will contain additional or modified elements required to meet this need.

This version of the specification is not a standard of any kind. Developers and implementers must not rely on this specification and it should be used for prototype use only.

Table of Contents

1 Introduction

This specification provides definitions of the content of Message Header elements for use within ebXML message handling.

Message Headers are contained within an outer Message Envelope. The structure and format of the Message Envelope is defined separately [2].

Definitions of words and terms that are used with ebXML Transport Routing and Packaging are contained in section 5. When these terms are used within the specification they are usually highlighted in *italics*. *<EdNote>Iin later versions of this specification this section will be removed and replaced by a reference to an ebXML wide glossary of terms.</EdNote>*

1.1 Message Headers

A Message Header is an XML construct that contains the additional data that needs to be associated with the documents in a message so that they can be delivered to and successfully processed by a *Party*.

1.2 Relationship to other ebXML Transport, Routing, and Packaging specifications

This specification will be one of a set of related specifications that will be delivered in phases. These specifications will cover:

- **ebXML Messaging Overview** - describes the relationship between the various ebXML specifications described below and how they are used in a compliant way with each other (under development)
- **ebXML Message Header Specification** - this specification
- **ebXML Message Envelope Specification** - how to wrap header's and bodies in a MIME wrapper [2]. This will be extended in the future to describe how to wrap headers and bodies in an XML envelope, although the XML Envelope may be developed as a separate specification. The specification also covers:
  - *ebXML Messaging over HTTP* - how to send ebXML Messages using HTTP or HTTPS
  - *ebXML Messaging over SMTP* - how to send ebXML Messages using SMTP
  - supplements for other protocols will be contained in separate documents
- **ebXML Reliable Messaging Specification** - how to achieve robust reliable once-only delivery of message in a vendor neutral, transport independent way (under development)
- **ebXML Messaging Security and Signature Specification** - this will cover (under development):
  - how to use S/MIME with ebXML Messages
  - how to use PGP with ebXML Messages

- how to use IETF/W3C XML Digital Signatures with ebXML Messages

- **ebXML Common Processes** contains specifications of commonly used processes that are applicable to many situations (under development):
  - *ebXML Message Set Status Inquiry Specification* - how to inquire on the current status of a message set
  - *ebXML Service Availability Specification* - how to determine if a Service is up and running
  - *ebXML Messaging Discovery Specification* - how to determine the parameters associated with a Service from the ebXML Messaging perspective.
  - *ebXML Publish & Subscribe Specification* - how to do Publish and Subscribe securely using any of the above
- **ebXML Messaging Audit Trail Specification** - how to use ebXML to create audit trails of the messages exchanged during a message set (under development).

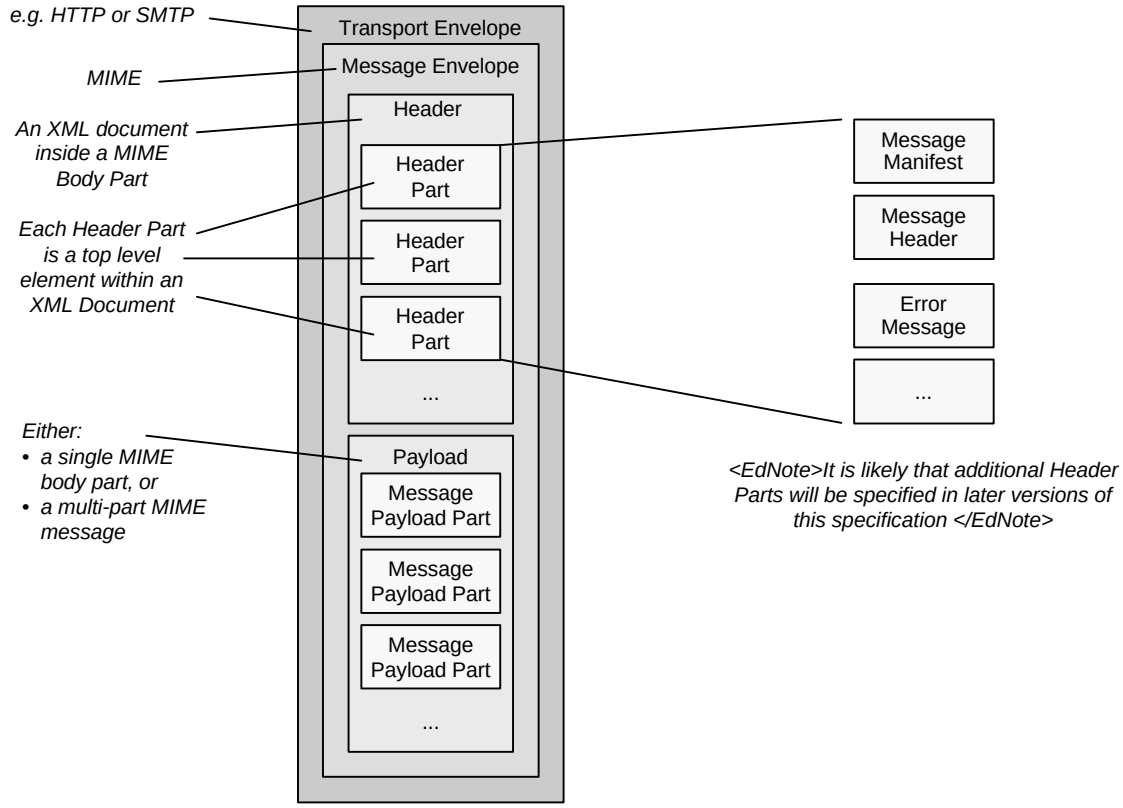
1.3 Specification Structure

The remainder of this specification contains:

- **Message Structure** - A description of the structure of the message header and the data items contained within them
- **Header Levels** - An introduction to the concept of using the data items in the message header in different ways and combinations depending on the nature of the problem which this specification is being used to solve
- **Template Document Exchanges** - An introduction to the different sequences in which messages of different *Message Types* are exchanged
- **Definitions** - Descriptions of the terms used within this and other ebXML Transport Routing and Packaging Specifications.
- **Schemas, DTD Definitions and Examples** - XML Schema and DTD definitions for the message headers described within this specification. *<EdNote>This does not yet exist.</EdNote>*
- **References** - References to other documents and specifications that are related to this specification
- **Acknowledgements** - References to the other people and organizations that have contributed to this specification
- **Authors' Address** - How to contact the main authors of this specification

2 Message Structure

The structure of a message is illustrated by the diagram below. Change picture to add Message Manifest



A Message Consists of:

- an outer **Transport Envelope**, such as HTTP or SMTP, that wraps,
- a transport independent **Message Envelope** [2] that contains the two main parts of the Message itself by wrapping:
  - a **Header** with one or more header parts inside, (the subject of this specification), and
  - a **Payload** that contains the document(s) associated with the message.

The Header is a single XML document with a number of **Header Parts** within it where each Header Part is a separate XML element. In general, separate header parts are used where:

- different software is likely to be used to generate that header part,
- the structure of the header part might vary independently of the other header parts, or
- the data contained in the header part may need to be digitally signed separately from the other header parts.

2.1 Header Parts

Using this principle, the following main header parts have been identified:

- a **Message Manifest** contains a list of references to he other parts of the Message. This includes references to:

- the Header Parts inside the Header,
- the Message Payload Parts.

- a **Message Header** header part, that contains the additional data that needs to be associated with the *Documents* in a *Message* so that they can be sent to and successfully processed by a *Party*. This data is typically set by the application that determines that a message needs to be sent. The data remains the same no matter what method or protocol is used to transport the message
- an **Error Message** header part that contains information about errors contained in an earlier message. *<EdNote>It is not clear whether this should be: a) in the Message Header or in the Message Payload, or b) be used just to highlight errors in the Message Header, or errors in any XML document.</EdNote>*

<EdNote>Additional Header Parts will probably be included in later versions of this specification. They might include, for example:

- *a digital signature based on the W3C XML Digital Signature standard*
- *a secure timestamp*
- *message routing information </EdNote>*

The Message Header and Message Manifest header parts are the only header parts that are present in every Message. Any additional header parts are optional.

2.1.1 Why Separate Header Parts

The following provides explanations as to why these header parts have been defined separately.

Message Manifest is held separately since the software that assembles the message from its constituent parts is likely to be separate from the software that creates the other parts

Message Header is held separately since the information it contains will frequently be specified by the business (or other) application that is generating the message. It is also held separately header part so that it may be digitally signed and its signature preserved, no matter where the message is routed or sent..

Descriptions of each Header Part and their element content follow.

2.2 Message Header, Header Part

The following is an overview of each of the elements in a ebXML Message Header.

Message Header Element	Outline Description
1) Message Header Version.	Contains the version of the header. This will follow whatever ebXML standard for versioning is adopted
2) Message Type (enumerated list)	Contains the type of the *message*. See separate documentation for details. Valid values are: • Normal • Acknowledgement • Error The Message Type is used to let ebXML aware software provide reliable messaging on behalf of an application
3) Service Type	Identifies the *business service interface* that should act on the payload in the message. It is a string. Service Types shall be unique within the domain in which they are being used. URN's may be considered suitable for this purpose.

		<EdNote> Need to coordinate with the Business Process Group to align with naming and semantics of their business model. </EdNote>
4)	Intent	Intent specifies the reason or intention for sending a message. Intent shall be unique within a Service Type. It is a string. *<EdNote> Need to coordinate with the Business Process Group to align with naming and semantics of their business model. </EdNote>*
5)	Message Set Data	Message Set Data contains information that describes a set of related Messages. All the data in this element is identical in all the messages in a Message Set. If a digital signature on a message signs the Message Set Data and some other data on the message, it is possible to prove that messages are related. In this way an audit trail of the messages in a Message Set can be provided.
	a) Message Set Id	This is a globally unique identifier for the Message Set. It is a string. *<EdNote> Guidelines on how to specify globally unique identifiers will be provided.</EdNote>*
6)	Message Data	Message Data contains information that describes an individual Message.
	a) Message Identifier	This is a globally unique identifier for an individual Message. It is a string. *<EdNote> Guidelines on how to specify globally unique identifiers will be provided.</EdNote>*
	b) Message Creation Timestamp	This is the time that the *Message Header* was created. It shall conform to ISO 8601.
	c) Ref To Message Identifier	If a message is created as the result of processing a previous message, then Ref To Message Identifier contains the Message Identifier of the message that was processed. Otherwise, the value is set to “Not Applicable”.
7)	Transport Service Level Agreement Id	Identifies the transport level *Service Level Agreement* to be used when sending and receiving this message. Defaults to an ebXML standard *Service Level Agreement* for a Transport Protocol. It is a string.
8)	From	Identifies the *Party* sending the message.
	a) Party Id	The identifier of the sending *Party*.
	i) Address Context	The context used to resolve the sending *Party* Address. It is an optional string. It is omitted if Address, on it's own is globally unique.
	ii) Address	The address of the sending *Party*. It is a string. *<ED NOTE> Need to describe how address and address context are used in an explanatory note in the data dictionary. </ED NOTE>*
9)	To	Identifies the *Party* receiving the message.
	a) Party Id	The identifier of the receiving *Party*.
	i) Address Context	The context used to resolve the receiving *Party* Address (Also see From)
	ii) Address	The address of the receiving *Party*. (Also see From)
10)	Reliable Messaging Info	
	a) Guaranteed Delivery	Indicates the degree of reliability of delivery. Valid values: • Unspecified (default) • AtMostOnce

2.3 Message Manifest

This shall be the first header part in the ebXML Header. It identifies the various header parts and payloads contained in the ebXML message envelope. The purpose of the Message Manifest is to make it easier to directly extract a particular document (or part of a document) associated with the Message.

Message Header Element	Outline Description
1) Message Manifest	This contains pointers to: • the other header parts that are present in the Message Header • payload(s) contained within the Payload part of the message, or • URLs to data associated with the message that are held elsewhere
a) Document Reference	Contains a reference to a part of a message. This value shall occur one or more times.
i) Document Identifier	An identifier for document. It is a string. If the referenced document is within the Message then the Document Identifier shall be unique within the Message Id. It may also be globally unique.
ii) Document Label	A textual description of the header part or payload referenced by the Document Identifier. It is an optional string. The purpose of the Document Label is to assist in the locating a document that is used for a particular purpose.

2.4 Message Type Dependent Header Parts

Depending on the Message Type, different header parts may be present in the Header in addition to the header parts described in section 2.1.

In this version of the specification the only Message Type Dependent Header Parts are the *Error Message Header Part* that must be present if the Message Type is set to Error. It indicates one or more errors in a *Message*

2.4.1 Error Header Part

Contains descriptions of and references to one or more locations in a message where errors have been detected.

<EdNote> To be completed. The structure of the Error Header part and the error codes that apply is to be specified. There is also an open issue over whether the header part should be in the Header, or in the Payload and whether it describes errors just in the header or errors in the Payload as well. </EdNote>

3 Header Levels

<EdNote> This specification contains the minimum data elements required. It is likely that, in order to implement the full requirements [1], additional data elements will need to be specified. There has been an open issue within the group over whether to specify a number of specific levels to describe the particular typical problems that the specification is designed to handle or whether to just make all elements optional.</EdNote>

4 Template Document Exchanges

The part of the specification describes valid sequences for exchanging messages. It consists of two parts:

- a section that describes the various Message Types (see the description of the Message Header), and
- a section that describes how those message types may be used together in Template Document Exchanges to support different types of interactions.

4.1 Message Types

This section provides an overview of how Message Type can be used. The main benefit of having a Message Type in the header is that it allows the ebXML transport to check that the messages of an appropriate type have been returned within the expected timeframes. For example if a Normal Message is sent with Guaranteed Delivery in the header indicating that At Most Once delivery is required, then an Acknowledgement message should be sent in return. If an Acknowledgement Message is not returned within the required timescale then the ebXML transport can detect this and attempt recovery by, for example, resending the original message. *<EdNote> More details on how to do reliable message delivery will be specified in a separate specification </EdNote>*

4.1.1 What is a Message Type

A Message Type contains information that can be used by the recipient of a message to determine how it should be handled and therefore what types of messages should be sent in return.

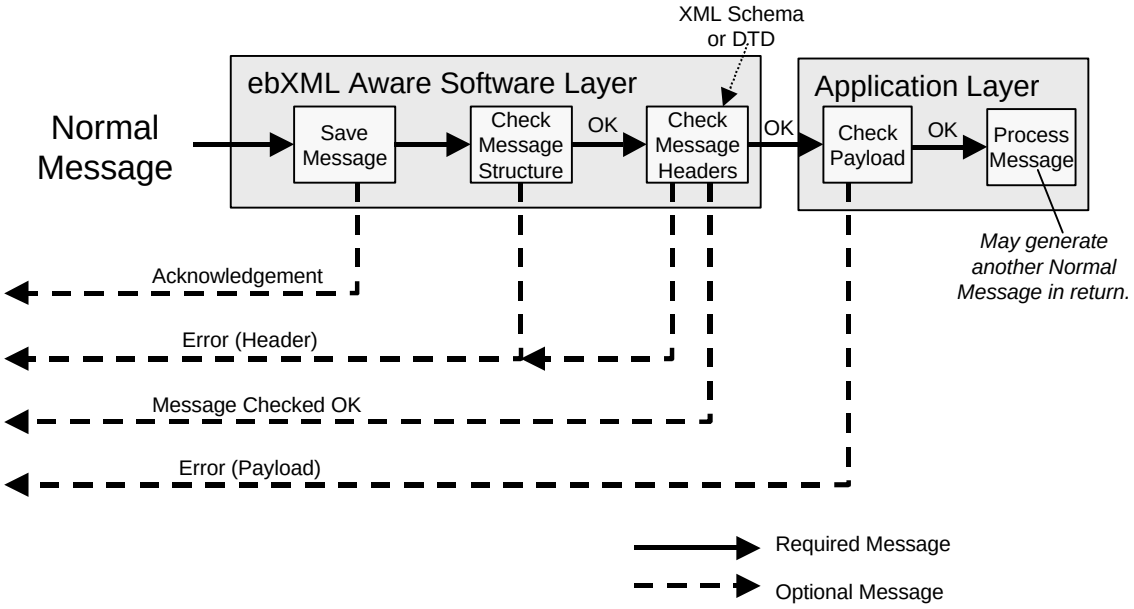
A message may be one of the following types:

- Normal Message
- Acknowledgement Message
- Error Message

Each of these is described below in more detail.

4.1.1.1 Normal Message

Normal Messages are sent to request that another party or server carry out a service or process of some kind. In more detail the messages that may result from processing a Normal Message are indicated in the diagram below.



Examples of Normal Messages include:

- requesting the processing of a new purchase order
- providing a purchase order response
- requesting that a previous purchase order is changed
- requesting a refund of a payment as a result of a problem
- requesting the review of a legal document
- the results of a review of a legal document
- requesting information on the services a business provides.

4.1.1.2 Acknowledgement Message

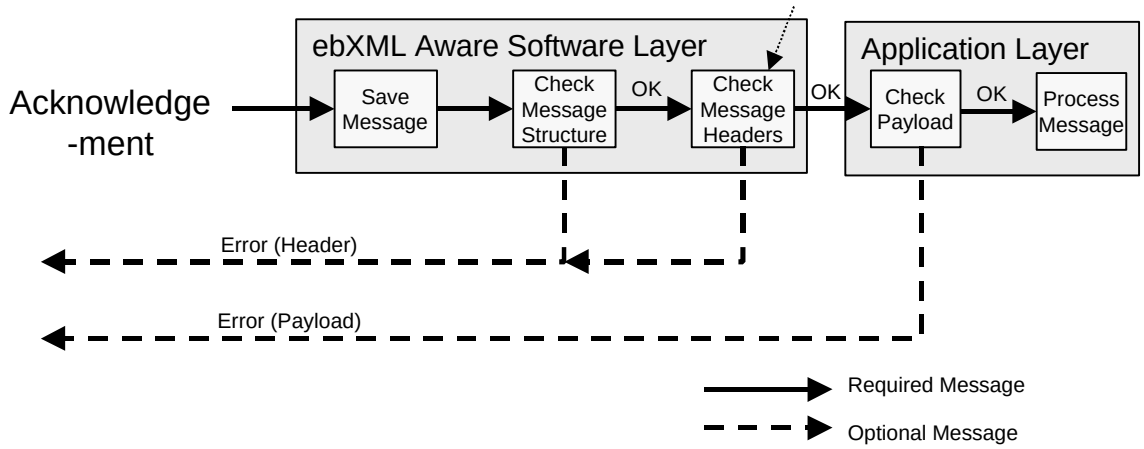
An Acknowledgement Message may be sent as a response to any Message (apart from an Acknowledgement Message) to indicate that a Message has been received.

It's recommended that Acknowledgement Messages are only sent after the message that is being acknowledged has been saved in some kind of persistent storage as Acknowledgement Messages can be used, for example, by the sender of a message as evidence that the original message was received.

On the other hand, if an Acknowledgement Message is not received within the expected time, then it indicates that something has probably gone wrong somewhere and the sender can choose to try and recover by, for example, re-sending the original message.

Acknowledgement Messages are also useful if a service that is being requested takes a long time to run. Acknowledgements Messages are never sent in response to another Acknowledgement Message as otherwise an infinite exchange of messages could occur.

More details on the message that may be generated as a result of processing an Acknowledgement Message are indicated by the diagram below.



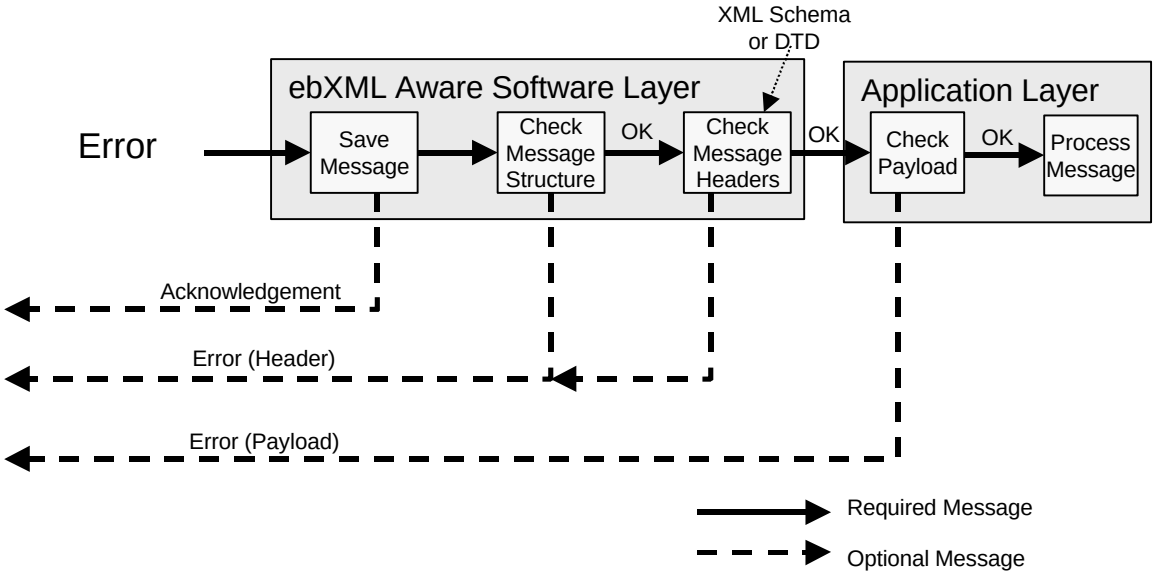
4.1.1.3 Error Message

An Error Message is a Message that reports on a problem in an earlier Message that prevents the earlier Message from being processed in a normal way. Examples of an Error Message include:

- an Error Message reporting that an XML document was invalid or did not conform to its XML schema
- an Error Message reporting a Transient Error that the Server processing a Message is busy and therefore the original Message should be resent at a later point in time
- an Error Message that reports on an error in the underlying transport protocol.

Error Messages contain an Error Message Header Part that indicates the nature of the error that has been found.

More details on the message that may be generated as a result of processing an Error Message are indicated by the diagram below.



It is possible that an Error may be found in an Error Message which, in turn, also contains errors. This can result in an infinite loop. To avoid this, Error Messages are not sent, for errors found in Error Messages that are reporting errors in another earlier Error Message.

5 Definitions

<EdNote>This section contains mildly updated versions of the definitions contained in the TP&R Overview and Requirements document. Some items are defined here that are not used within the version of the Message Header Specification but are likely to be required in later versions of the specification. It is also likely that, in later versions of this specification, this section will be removed and replaced by reference to an ebXML wide Glossary of terms.</EdNote>

The following are a list of definitions of the terms associated with the transport of messages over the Internet.

It is split into two sections:

- Documents, Parties, Messages and Document Exchanges, and
- Services and Message Sets

Words or phrases that are defined elsewhere are highlighted in *italics*.

5.1 Documents, Parties, Messages and Document Exchanges

5.1.1 Overview

This section describes how *Parties*, such as buyers and suppliers, customers and merchants, can transmit *Documents* contained in *Messages* in order to request execution of *Services*.

All the *Documents* and other data in a *Message* are contained within an outermost *Message Envelope*.

A *Message* can optionally include *Digital Signatures* so that:

- the identity of the *Party* sending the *Message* can be authenticated
- any changes to the *message* and the *documents* they contain can be detected.

Services are requested by sending one or more *Documents* in a *Request Message* to a *Party* who then:

- processes the *Request Message* by carrying out a *Service* and
- optionally generates a *Response Message* to indicate the result.

At a minimum a *Document Exchange* consists of a *Request Message* and an optional *Response Message* although there might be additional *Exchange Messages* between the *Request Message* and the *Response Message*.

Error Messages are used to report permanent or transient problems or errors in a *Message*.

More detail is provided below.

5.1.2 A Document

A *Document* is any data that can be represented in a digital form.

Examples of *Documents* include:

- a set of XML Elements
- an XML Document
- an HTML Document
- a word processing file
- an Adobe Acrobat PDF file
- a binary file
- part of larger document.

5.1.3 Party

A *Party* is a company, organization or individual or other entity that can generate, receive or relay *Documents*. Parties are identifies by a *Party Address*.

Examples of a *Party* include:

- a Merchant
- a Customer
- a Lawyer
- a Bank
- a government department or agency
- an intermediary or agent
- a software agent

A *Party* is also used to refer to systems or servers that are carrying out *Services* or processes on behalf of a *Party*.

5.1.4 Party Address

The address of a party is in two parts:

- an optional Address Context, and
- the Address itself.

In combination, Address Context and Address must be globally unique.

The Address Context specifies the domain of the address, or the authority that allocated the Address to the Party. It is optional since the Address, on its own, may be globally unique. Examples of Address Context include:

- Duns - indicating the Address is a Dun & Bradsteet (DUNS) allocated number
- Telephone - indicate that the Address is an international Telephone number

Address identifies the individual Party within the Address Context. Examples of Address include:

- the Actual Duns number of the party
- a telephone number
- a URN - since URNs are globally unique, no Address Context is needed.

5.1.5 Message

A *Message* is data that is sent from one *Party* to another.

All the data in a *Message* is contained within a *Message Envelope*.

A Message consists of a *Message Header* and a *Message Body*

Examples of a *Message* include:

- a Purchase Order that is sent by a buyer to a supplier
- an Invoice that is sent by the supplier back to the buyer
- a request to make a payment of $50 sent to a Credit Card acquirer
- the authorization received from a Credit Card acquirer as a result of making a payment
- Status indicating the success or failure of a Service

5.1.6 Message Header

A *Message Header* is an XML construct that contains the additional data that needs to be associated with the *Documents* in a *message* so that they can be sent to and successfully processed by a *Party*.

5.1.7 Message Manifest

The Message Manifest contains references to the other documents, apart from the Message Routing Information document, that are contained within the same Message Envelope.

The purpose of the Message Manifest is to facilitate locating and validating that all required Documents contained within the Message Envelope are present.

Examples of the types of documents that might be referenced by a Message Manifest include:

- a Purchase Order
- a Purchase Order and a picture of the requested goods
- a Purchase Order and a digital signature

5.1.8 Message Routing Information

Message Routing Information contains data that indicates the path that should be or was taken by a *Message* in reaching its ultimate destination.

5.1.9 Digital Signature

A *Digital Signature* is a cryptographic signature over[1] data contained in a *Message*, or elsewhere that are addressable via [URI]s, that permits the authenticity of the signer of the data to be determined, and helps detect if the data in the *Message* has changed.

---

[1] A digital signature represents a string of binary digits of arbitrary length created by using a cryptographic key known only to the party sending a message. The string is composed of an encrypted digest of some or all of the data in the message or in another location addressable by a URI. It is accompanied by some method (such as a digital certificate)of identifying to the party receiving the message, what key can be used to validate the digest against the original data.

5.1.10 Message Envelope

A *Message Envelope* is the outermost container for a Message. It can be such things as:

- an XML Document, or
- a multi-part MIME message

5.1.11 Message Types

Messages may be of several different types. These are described below.

5.1.11.1 One-Way Message

A *One-Way Message* is a *Message* sent from one party to another. The receiving *Party* MAY only respond back to the From Party with either a *Message Acknowledgement* and, if there is an error, an *Error Message*.

5.1.11.2 Request Message

A *Request Message* is a *Message* sent from one *Party* to a another *Party's Service* with the intent that the other *Party* act upon the data in the *Request Message* by carrying out the *Service*.

The results of processing the *Request Message* MUST be included in a *Response Message* that is sent back to the sender of the previous *Message*.

5.1.11.3 Acknowledgement Message

An *Acknowledgement Message* may sent as a response to any *Message* (apart from an *Acknowledgement Message*) to indicate that the Message has been received[2].

5.1.11.4 Checked OK Message

A *Checked OK Message* may be sent in response to a *Request Message* to indicate that the content of the *Request Message* has been validated and no errors were found. A Checked OK Message MUST be sent after any *Acknowledgement Message* that was sent.

5.1.11.5 Response Message

A *Response Message* is a *Message* that is generated by the *Service* that received a *Request Message*. It is produced as a result of carrying out the requested *Service*. It is the last *Message* in a *Document Exchange* unless the *Response Message* contains errors.

Response Messages are sent back to the sender of the *Request Message*.

5.1.11.6 Exchange Message

An *Exchange Message* is a *Message* that is sent between one *Party* and another after the sending of the initial *Request Message* and before the sending of the final *Response Message*.

Examples of *Exchange Messages* include:

- intermediate messages that are part of a Payment Protocol
- a counter offer to an offer made as part of a negotiation.

---

[2] It is recommended that messages are saved in some type of persistent storage before they are acknowledged.

5.1.11.7 Error Message

An *Error Message* is a *Message* that reports on a problem in an earlier *Message* that prevents the earlier *Message* from being processed in a normal way.

Examples of an *Error Message* include:

- an *Error Message* reporting that an XML document was invalid or did not conform to its XML schema
- an *Error Message* reporting a Transient Error that the Server processing a *Message* is busy and therefore the original *Message* should be resent at a later point in time
- an *Error Message* that reports on an error in the underlying transport protocol.

5.1.12 Document Exchange

A Document Exchange is a generic term for either a:

- a *One-Way Document Exchange*,
- a *Simple Document Exchange*, or
- a *Multiple Round Trip Document Exchange*.

5.1.12.1 One-Way Document Exchange

A One-Way Document Exchange consist of:

- a One-Way Message sent from one Party to a second Party, followed by
- an optional *Acknowledgement Message* sent by the second party back to the first party, followed by
- an optional *Error Message* if an error was detected in the *One-Way Message*

Examples of a *One-Way Document Exchange* include:

- a supplier sending catalog updates to their buyers
- a supplier offering goods for sale by auction

5.1.12.2 Simple Document Exchange

A Simple Document Exchange consists of:

- a *Request Message* sent from one *Party* to a second *Party*, followed by
- an optional *Acknowledgement Message* sent by the second party back to the first party, followed by
  - an optional *Error Message* if an error was detected in the Request Message, or
  - an optional *Checked OK Message*, if no errors were detected that is sent by the second party back to the first party followed by
- a *Response Message* that is returned as a result of processing the *Request Message*.

Examples of instances of a *Simple Document Exchange* include:

- a Purchase Order sent by a buyer to a seller and the acknowledgement from the seller of its receipt
- a Purchase Order sent by a buyer to a seller and the Invoice that is sent back as a result of fulfilling the order

- sending a document for review by a lawyer followed by the legal opinion that is sent back as a result

5.1.12.3 Multiple Round Trip Document Exchange

A *Multiple Round Trip Document Exchange* consists of:

- a *Request Message* sent from one *Party* to a second *Party*, followed by
- a series of *Exchange Messages* that are exchanged between the two *Parties* until finally
- the either the first or the second *Party* generates and sends a *Response Message* back to the other *Party*.

Examples of *Multiple Round Trip Document Exchanges* include:

- the exchange of messages required to make a payment using payment method protocols such as [SET] or [Mondex]
- the exchange of messages required to negotiate an agreement on terms and conditions.

5.2 Services and Message Sets

5.2.1 Overview

A *Service* is a process that can be carried out by a *Party.* It is implemented by either a *Document Exchange* or a set of *Sub-Services*. Each *Sub-Service* is a *Service* in its own right. So, at the lowest level, all *Services* are implemented in terms of a *Document Exchange*.

The dependencies between the *Sub-Services* in a *Service* are described in a *Service Choreography*.

An instance of the execution of a *Service* is called a *Message Set*.

More detail is provided below.

5.2.2 Service

A *Service* is a process that can be carried out by a *Party* as a result of receiving a *Request Message* or *One-Way Message* that requests the execution of that *Service*.

A *Service* can consist of either:

- a Document Exchange, or
- a set of *Sub-Services*

Examples of a *Service* include:

- a Purchasing Service that enables a customer to purchase goods on-line
- an Order Processing Service that processes an Order and generates a response as a result
- a Payment Service that accepts a payment and provides a receipt
- a Fulfillment Service that fulfills an order at the request of a Merchant.

5.2.3 Sub-Service

A *Sub-Service* is a *Service* that is executed at the request of and as part of another *Service*.

Examples of *Sub-Services* include:

- a payment service that occurs as part of a purchase
- a tax calculation service that calculates the tax due as part of an order processing service.

5.2.4 Service Choreography

A *Service Choreography* is a description of the dependencies that control the sequence and choices that determine which *Sub-Services* are executed when carrying out a *Transaction*.

The *Sub-Services* in a *Service* will have dependencies between them. Dependencies can be:

- *Serial*. One *Sub-Service* must start only after the completion of another *Sub-Service*
- *Alternative*. One *Sub-Service* may be executed as an alternative to another
- *Iterative Loop*. A *Sub-Service* may be repeated a variable number of times
- *Conditional*. The execution of a *Sub-Service* is conditional on the state of another *Service*. This may be used in conjunction with *Serial*, *Alternative* and *Iterative Loop* dependencies.
- *Parallel*. A *Sub-Service* may execute in parallel with another *Service*
- *Concurrent*. A *Sub-Service* must execute at the same time as another *Sub-Service*.

An example of a simple *Sub-Service Choreography* is a Purchase Service that consists of three *Sub-Services*:

- an Offer Service that conveys an Offer for sale of goods. This *Sub-Service* has no dependencies and therefore starts first
- a Payment Service that carries out the Payment which has a Serial dependency on the Offer Service
- a Delivery Service that delivers the Digital Goods, that has a Serial Dependency on the Payment Service

5.2.5 Application

An *Application* is software that may implement a *Service* by processing one or more of the *Messages* in the *Document Exchanges* associated with the *Service.*

5.2.6 Transaction

A *Transaction* is an instance of the execution of a *Service*[3].

[3] There are several different meanings that have been associated with transactions:

- "ACID" transactions. A term that is used to describe the properties of a transaction in terms of:
  - **Atomicity**. A transaction's changes to the state are atomic: either all actions happen or none happen.
  - **Consistency**. A transaction is a correct transformation of the state. The actions taken as a whole do not violate any of the integrity constraints associated with the state. This requires that the transaction be a correct program.

Examples of a *Transaction* include:

- a Purchase Transaction that buys a Company Report for $20. It consists of three Sub-Service instances:
- an Offer Service instance to buy the Company Report for $20
- a Payment Service instance that accepts a Payment for $20 using a credit card, and finally
- a Delivery Service instance that delivers the Company Report as an HTML web page.
- a Buying Service that consists of the following Sub-Services:
  - three Price Negotiation Service instances that negotiate the price of a Photocopier
  - a Purchase Order Service instance that places the order for the Photocopier.

6 Schemas, DTD Definitions and Examples

<EdNote>Note we will only define this section once the structure of the header parts are finalized (or nearly finalized) <EdNote>

6.1 XML Header DTD

<EdNote> Will contain an XML DTD version of the Header Parts whose structure is defined in this document</EdNote>

6.2 XML Header Schema Definition

<EdNote> Will contain an XSDL Schema version of the Header Parts whose structure is defined in this document</EdNote>

  - **Isolation**. Even though transactions execute concurrently, it appears to each transaction T, that others executed either before or after T, but not both. In other words, each transaction is isolated from any others.
  - **Durability**. Once a transaction completes successfully (commits), its changes to the state survive failures.
- "EDI" transactions - "The information included in a transaction set is, for the most part, the same as the information in a conventionally printed document. A transaction set is the data that is exchanged in order to convey meaning between parties engaged in EDI
- "Conversational" transactions - A conversation is a sequence of related Messages between two parties separated in time. A complete "unit of business" for example, the negotiation of a purchase, placement, confirmation, payment and delivery of goods, may be represented as multiple transactions in a longer running conversation." From DISA publication titled "Introduction to EDI", (ASC X12S/94-190)
- "Read-only" transactions - a transaction that consists of a document exchange where the information is obtained from a service without changing the state of the service

7 References

[1] ebXML Transport, Routing and Packaging: Overview and Requirements document version 0.96. Published 25 May 2000

[2] ebXML Transport, Routing & Packaging Message Envelope Specification. Version 0.5. Published 25 May 2000

8 Acknowledgements

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

9 Authors' Address

Contact information for the Author's of this specification follow.

David Burdett
Commerce One Inc
4400 Rosewood Drive 3rd Fl, Bldg 4,
Pleasanton, CA 94588,
USA

Tel: +1 (925) 520 4422 or +1 (650) 623 2888
Email: david.burdett@commerceone.com

John Ibbotson
IBM UK Ltd
Hursley Park
Winchester
SO21 2JN
United Kingdom

Tel: +44 (1962) 815188
Email: john_ibbotson@uk.ibm.com