OASIS ebXML Messaging Services

Version 3.0

Edited by

Matthew MacKenzie

Jeff Turpin

Ian Jones

British Telecom

<ian.c.jones@bt.com>

Doug Bunting

Sun Microsystems

<doug.bunting@sun.com>

Dale Moberg

Cyclone Commerce

<dmoberg@cyclonecommerce.com>

Jacques Durand

Fujitsu Software

<jdurand@us.fujitsu.com>

Pete Wenzel

SeeBeyond

<pete@seebeyond.com >

HTML

http://www.oasis-open.org/committees/ebxml-msg

Status

This is a *Working Draft*.

Current SVN Info id is: 35

Committee members should submit comments to the ebxml-msg@lists.oasis-open.org list. Others should submit comments by filling out the form at http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=ebxml-msg

5 April 2005

Abstract

This specification focuses on defining a communications-protocol neutral method for exchanging electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type. This versatility ensures legacy electronic business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or HL7) can leverage the advantages of the ebXML infrastructure along with users of emerging technologies.

Table of Contents

Introduction

This specification focuses on defining a communications-protocol neutral method for exchanging electronic business messages. It defines specific enveloping constructs supporting reliable, secure delivery of business information. Furthermore, the specification defines a flexible enveloping technique, permitting messages to contain payloads of any format type. This versatility ensures legacy electronic business systems employing traditional syntaxes (i.e. UN/EDIFACT, ASC X12, or HL7) can leverage the advantages of the ebXML infrastructure along with users of emerging technologies.

Terminology

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].

This specification uses capitalization to help make these key words stand out.

Audience

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

Caveats & Assumptions

It is assumed the reader has an understanding of communications protocols, MIME, XML, SOAP, SOAP Messages with Attachments and security technologies.

All examples are to be considered non-normative. If inconsistencies exist between the specification and the examples, the specification supersedes the examples.

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

Concept of Operation

Scope

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

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

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

Background and Objectives

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

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

An objective of ebXML Messaging protocol is to be capable of being carried over any available communications protocol. Therefore, this document does not mandate use of a specific communications protocol. This version of the specification provides bindings to HTTP, SMTP and FTP, but other protocols can, and reasonably will, be used.

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

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

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

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

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

Operational Policies and Constraints

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

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

MSH Operational Parameters

ebXML MSHs rely on external configuration information to drive message exchanges. Throughout this document, we refer to these abstract operational parameters which are defined below.

In a production environment, an MSH may obtain these operational parameters from a CPA or some other source of configuration.

OpParam_ToPartyValue

Identifier(s) of the receiving party in a message exchange.

OpParam_FromPartyValue

Identifier(s) of the sending party in a message exchange.

OpParam_ConversationID

A message's conversation ID.

OpParam_ServiceValue

A message's service identifier.

OpParam_AgreementRef

A message's AgreementRef.

OpParam_ActionValue

A message's action identifier.

OpParam_SecurityProfile

A message's security profile, which contains the following child parameters: *TBD*

OpParam_ReliabilityProfile

A message's reliability profile, which contains the following child parameters: *TBD*

OpParam_MEPMode

A message's MEP Mode.

Modes of Operation

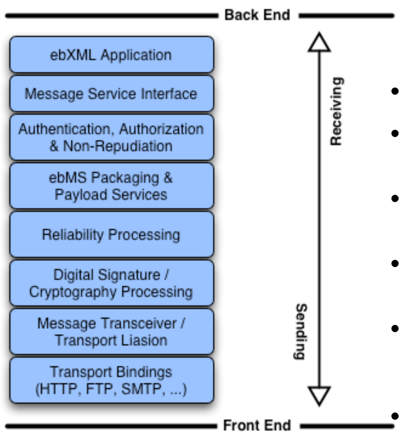
Change Title of 5.4

The ebXML Message Service may be conceptually broken down into the following three parts:

1. an abstract Service Interface,
2. functions provided by the MSH and
3. the mapping to underlying transport service(s).

Figure 1 depicts a logical arrangement of the functional modules existing within one possible implementation of the ebXML Message Services architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies.

Figure 1. Component Relationships


Back End
ebXML Application
Message Service Interface
Authentication, Authorization & Non-Repudiation
ebMS Packaging & Payload Services
Reliability Processing
Digital Signature / Cryptography Processing
Message Transceiver / Transport Liasion
Transport Bindings (HTTP, FTP, SMTP, ...)
Receiving
Sending
Front End

Following is a description of each module illustrated above. It should be noted that the stack diagram above is abstract, and this specification does not mandate that implementations adopt the architecture suggested by it.

- **ebXML Application** - This is where the business logic for a message exchange / business process exists.
- **Message Service Interface** - This is the interface through which messages are channelled between the MSH core and the the ebXML Application.
- **Authentication, Authorization & Non-Repudiation** - This module performs any authentication and authorization checks, and verifies the sender or receiver's identity.
- **ebMS Packaging & Payload Services** - Handling, (de)enveloping and execution of Payload Services are performed by this module.
- **ebXML Message** - A SOAP message that contains headers qualified with the ebMS namespace. Although reliability is an integral functional part of the MSH, a SOAP message with only WS-Reliability namespace qualified headers is NOT an ebXML message.
- **Reliability Processing** - This module fulfills the WS-Reliability requirements for a message.
- **Digital Signature / Cryptography Processing** - Verification of any digital signatures occurs in this module.
- **Message Transceiver / Transport Liasion** - This module manages interaction with the available transport bindings, acting as a normalizing layer between the MSH and its various transports.
- **Transport Bindings** - These are the actual transport bindings. This specification defines bindings for HTTP (the section called "HTTP"), FTP (the section called "FTP") and SMTP (the section called "SMTP") and supports the addition of other protocols.

Definition of MSH abstract operations and sending mode

An MSH is composed of the following abstract operations: Submit, Deliver, Notify, Respond, Pull. These operations have a semantics that are independent of reliability. When reliability is used, these operations respectively include their homonymous RMP operations, which we rename here RM-Submit, RM-Deliver, RM-Notify, and RM-Respond to avoid confusion.

Submit has a more precise semantics, depending on the sending mode of an MSH. The Pull operation is invoked for each "Pull" signal message received. The semantics of these two operations is as follows:

- When the MSH is in a Push sending mode, the Submit invocation for a reliable message on an MSH is associated with an RM-Submit invocation on the message. The sequence of RMSubmit invocations is in tset of reliable messages.
- When the MSH is in a Pull sending mode, the messages submitted to the MSH are handled according to a queuing semantics. If we model this behavior using an abstract queue, called the submit-queue, then Submit will put the message in the submit-queue. The messages are pulled out of the submit-queue only when the Pull operation is invoked. Messages pulled are subject to RM-Submit invocation in the same order as they have been pulled.

Abstract Message Exchange Patterns

An MEP is a typical sequence of message exchange that may occur between two or more MSH instances. An MEP instance is an actual message exchange that conforms to the pattern described in the MEP. This section defines the ebMS Message Exchange Patterns (MEP). An ebMS MEP is only defined in terms of ebMS messages, defined as SOAP messages carrying ebMS-qualified headers. An MEP is a sequence of ebMS message exchanges, that can be described along two characteristics:

- For each message that has business semantics (i.e. produced and consumed by the application layer or MSH-using layer), the direction of this message between two partners.
- For each message that has business semantics , the mode of transfer: Push, Pull.

Each MEP defined below is a combination of these two characteristics, although not all possible combinations have been specified.

When several messages with business semantics are exchanged, there must be some explicit reference relationship between them to belong to the same MEP. In other words, for every message in the same MEP instance, either one of these statements is true:

- the message is the first one to occur in the MEP
- the message is referring to the ebMS ID of another message (and only one), in the same MEP instance.

Assumed SOAP Message Exchange Patterns

SOAP One-way MEP:

From an MSH perspective, support for this MEP assumes the following:

- The Sending RMP (as a SOAP node) is able to initiate the sending of a SOAP envelope over the underlying protocol (i.e., not as a result of a previous protocol action such as an HTTP GET or POST).
- No response containing a SOAP envelope is sent back – although a non-SOAP response (e.g., an HTTP error code) may be returned.

SOAP Request-response MEP:

From an MSH perspective, support for this MEP assumes the following:

- The Sending RMP is able to initiate the sending of a SOAP envelope over the underlying protocol (i.e., not as a result of a previous protocol action such as an HTTP GET or POST).
- The Receiving RMP can send back a message with a SOAP envelope (called a response) after somehow associating the response with the request. (For example, this association can be realized by the use of a request-response underlying protocol such as HTTP.)

The full definition of this MEP can be found in [SOAP] part 1, Adjunct.

The concept of SOAP MEP has only been introduced with SOAP 1.2, although there is no essential issue when using this concept with SOAP 1.1. As far as an MSH is concerned, it is sufficient to assume the above properties that are associated with the One-way and Request-response SOAP MEPs.

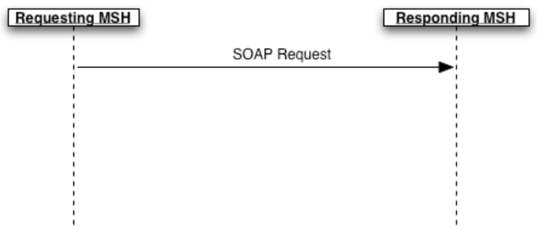
Simple ebMS Message Exchange Patterns

A simple ebMS MEP maps to a single SOAP MEP instance. This specification identifies three simple MEPs.

The Simple Push Message Exchange Pattern

This MEP involves a single ebMS message with business semantics. Sending a message in an ebMS Push MEP means that the message is sent either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance. Requesting MSH transmits a SOAP Request message. Responding MSH may return a SOAP response. If the responding MSH returns a SOAP response it MUST not contain an ebXML Message.

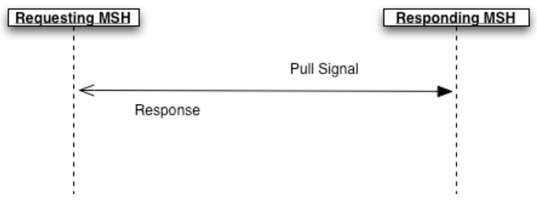
Figure 2. Push Sequence



The Simple Pull Message Exchange Pattern

This MEP involves a single ebMS message with business semantics, and an ebMS signal message. Sending a message in Pull mode means that the message is sent as a SOAP Reponse over a SOAP Request-response MEP instance, where the SOAP Request contains the ebMS PullRequest signal. The first leg of the MEP sends the PullRequest signal. The second leg of the MEP returns the pulled business message. In case no message is available for pulling, a SOAP Response is returned with an “empty queue” signal.

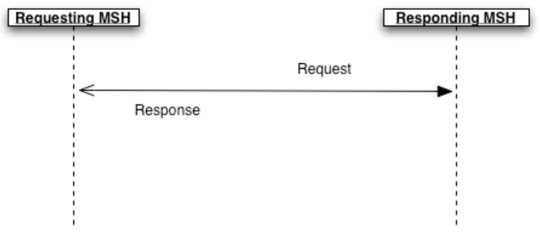
Figure 3. Pull Sequence



The Simple Request-Response Message Exchange Pattern

This MEP involves two ebMS messages with business semantics, over a single SOAP Request-response MEP. In the first leg of the MEP, a message called the “request” is sent as the SOAP Request message. In the second leg of the MEP, a message called the “response” is sent as the SOAP Response. The response message refers to the request message (via the ebMS message ID). The ebXML Message returned in the SOAP response must use the ebXML the section called “RefToMessageId Element” element to correlate the exchange of messages.

Figure 4. Synchronous Sequence



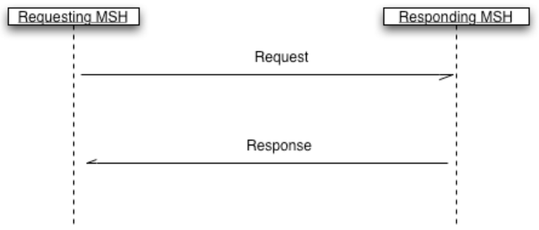
Aggregate ebMS Message Exchange Patterns

An aggregate ebMS MEP maps to two or more SOAP MEP instances. This specification identifies three different aggregate MEPs, each of them using two SOAP MEPs. This selection is an arbitrary subset of all possible aggregate MEPs. From a choreography point of view, an aggregate MEP can be seen as a composition of simple MEPs, which reference each other via ebMS message ID.

The Push/Push Message Exchange Pattern

This MEP involves two ebMS messages with business semantics between two partners. One message (the “response”) will refer to the other (the “request”), and will be sent the opposite direction. In the first leg of the MEP, the request message is sent either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance. In the second leg of the MEP, the response message is sent in the opposite direction either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance. The response refers to the message ID of the request.

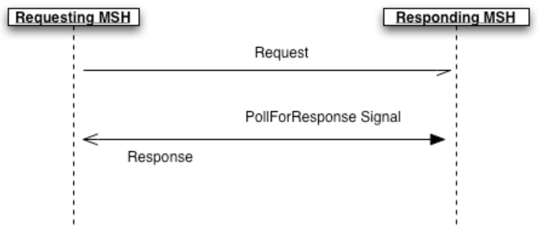
Figure 5. Push/Push Sequence



The Push/Pull Message Exchange Pattern

This MEP involves two ebMS messages with business semantics between two partners. One message (the “response”) will refer to the other (the “request”), and will be sent the opposite direction. In the first leg of the MEP, the request message is sent either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance. In the second leg of the MEP, a PullRequest signal message is sent in the same direction over a SOAP Request in a SOAP Request-response MEP instance. In the third leg of the MEP, the response message is returned over the SOAP Response matching the SOAP Request that carried the PullRequest. The response refers to the message ID of the request (first leg). This MEP is used for message transactions where the Requesting MSH does not allow incoming connections, for example due to firewall restrictions.

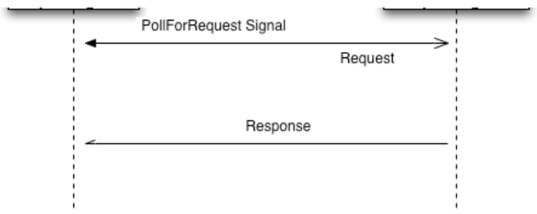
Figure 6. Push/Pull Sequence



The Pull/Push Message Exchange Pattern

This MEP involves two ebMS messages with business semantics between two partners. One message (the “response”) will refer to the other (the “request”), and will be sent the opposite direction. In the first leg of the MEP, a PullRequest signal message is sent over a SOAP Request in a SOAP Request-response MEP instance. In the second leg of the MEP, the request message is returned over the SOAP Response matching the SOAP Request that carried the PullRequest. In the third leg of the MEP, the response message is sent in the direction opposite to the request, either over a SOAP One-way MEP instance or as a SOAP Request in a SOAP Request-response MEP instance. The response refers to the message ID of the request (second leg). This MEP is used for message transactions where the Responding MSH does not allow incoming connections, for example due to firewall restrictions.

Figure 7. Pull/Push Sequence



Conformance

Minimum Requirements for Conformance

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

1. It supports all the mandatory syntax, features and behavior (as identified by the [RFC 2119] key words MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT) defined in the section called “Core Modules”.
2. It supports all the mandatory syntax, features and behavior defined for each of the additional module(s), defined in the section called “Additional (optional) Features”, the implementation has chosen to implement.
3. It complies with the following interpretation of the keywords OPTIONAL and MAY: When these keywords apply to the behavior of the implementation, the implementation is free to support these behaviors or not, as meant in [RFC 2119]. When these keywords apply to message contents relevant to a module of features, a conforming implementation of such a module MUST be capable of processing these optional message contents according to the described ebXML semantics.
4. If it has implemented optional syntax, features and/or behavior defined in this specification, it MUST be capable of interoperating with another implementation that has not implemented the optional syntax, features and/or behavior. It MUST be capable of processing the prescribed failure mechanism for those optional features it has chosen to implement.
5. It is capable of interoperating with another implementation that has chosen to implement optional syntax, features and/or behavior, defined in this specification, it has chosen not to implement. Handling of unsupported features SHALL be implemented in accordance with the prescribed failure mechanism defined for the feature.

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

Message Package Specification

The ebXML Message Service Specification defines a set of namespace-qualified SOAP Header element extensions within the SOAP Envelope. These can be packaged as a plain [SOAP] message, or within a MIME multipart to allow payloads or attachments to be included with the SOAP extension elements. Because either packaging option can be used, Implementations MUST support non-multipart messages. In general, separate ebXML SOAP extension elements are used where:

different software components may be used to generate ebXML SOAP extension elements,
an ebXML SOAP extension element is not always present or,
the data contained in the ebXML SOAP extension element MAY be digitally signed separately from the other ebXML SOAP extension elements.

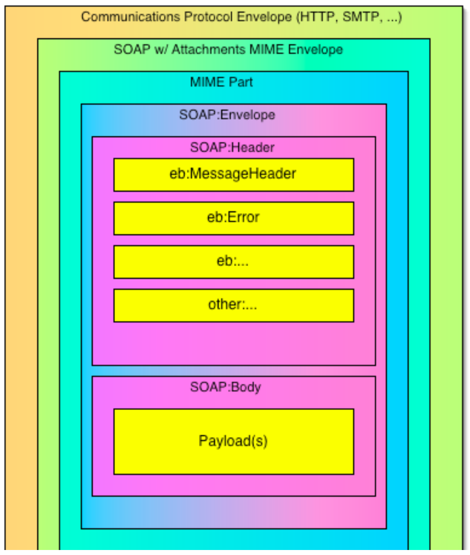
An ebXML Message is a communications protocol independent standard [SOAP] message, or MIME/Multipart message envelope. The MIME/Multipart message envelope MUST be structured in compliance with the SOAP Messages with Attachments [SOAP w/ Attachments] W3C Note, referred to as a Message Package.

There are two logical MIME parts within the Message Package:

The first MIME part, referred to as the Header Container, containing one SOAP 1.1 compliant message. This XML document is referred to as a SOAP Message for the remainder of this specification,
zero or more additional MIME parts, referred to as Payload Containers, containing application level payloads.

The general structure and composition of an ebXML Message is described in Figure 8.

Figure 8. Structure and Composition of an ebXML Message



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

One SOAP Header element. This is a generic mechanism for adding features to a SOAP Message, including ebXML specific header elements.
One SOAP Body element. This can be a container for the payload parts of the message.

SOAP Structural Conformance

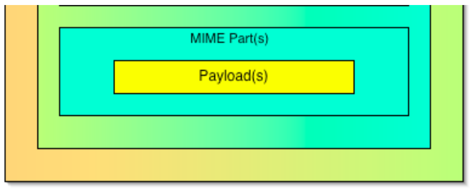
The ebXML Message packaging complies with the following specifications:

Simple Object Access Protocol (SOAP) 1.1 [SOAP]
SOAP Messages with Attachments [SOAP w/ Attachments]

Carrying ebXML headers in SOAP Messages does not mean ebXML overrides existing semantics of SOAP, but rather the semantics of ebXML over SOAP maps directly onto SOAP semantics.

Message Package

All MIME header elements of the Message Package are in conformance with the SOAP Messages with Attachments [SOAP w/ Attachments] W3C Note. In addition, the Content-Type MIME header in the Message Package contain a type attribute matching the MIME media type of the MIME body part containing the SOAP Message document. In



accordance with the [SOAP] specification, the MIME media type of the SOAP Message has the value "text/xml".

It is strongly RECOMMENDED the initial headers contain a Content-ID MIME header structured in accordance with MIME [RFC 2045], and in addition to the required parameters for the Multipart/Related media type, the start parameter (OPTIONAL in MIME Multipart/Related [RFC 2387]) always be present. This permits more robust error detection. The following fragment is an example of the MIME headers for the multipart/related Message Package:

Example 1. MIME Header fragment for the multipart/related Message Package

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

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

Because implementations MUST support non-multipart messages, an ebXML message with no payload may be sent either as a plain SOAP message or as a [SOAP w/ Attachments] multipart message with only one body part.

Header Container

The root body part of the Message Package is referred to in this specification as the Header Container. The Header Container is a MIME body part consisting of one SOAP Message as defined in the SOAP Messages with Attachments [SOAP w/ Attachments] W3C Note.

Content-Type

The MIME Content-Type header for the Header Container MUST have the value "text/xml" to match the MIME media type of the MIME body part containing the [SOAP] Message document. The Content-Type header MAY contain a "charset" attribute. For example:

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

charset Attribute

The MIME charset attribute identifies the character set used to create the SOAP Message. The semantics of this attribute are described in the "charset parameter / encoding considerations" of text/xml as specified in XML [XMLMedia]. The list of valid values can be found at http://www.iana.org/.

If both are present, the MIME charset attribute SHALL be equivalent to the encoding declaration of the SOAP Message. If provided, the MIME charset attribute MUST NOT contain a value conflicting with the encoding used when creating the SOAP Message.

For maximum interoperability it is RECOMMENDED UTF-8 [UTF-8] be used when encoding this document. Due to the processing rules defined for media types derived from text/xml [XMLMedia], this MIME attribute has no default.

Header Container Example

The following fragment represents an example of a Header Container:

```
Content-ID: <messagepackage-123@example.com>
Content-Type: text/xml;  charset="UTF-8"

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

Payload Container

Zero or more Payload Containers MAY be present within a Message Package in conformance with the SOAP Messages with Attachments [SOAP w/ Attachments] specification. Alternatively, payload(s) may be placed within the SOAP Body element, in conformance with [SOAP]

If the Message Package contains an application payload, it SHOULD be enclosed within a Payload Container.

If there is no application payload within the Message Package then a Payload Container MUST NOT be present.

The contents of each Payload Container MUST be identified in the ebXML Message PayloadInfo element within the SOAP Header (see ???).

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

Attachment Payload Example

The following fragment represents an application payload as an Attachment:

```
Content-ID: <domainname.example.com>
Content-Type: application/xml

<?xml version="1.0"?>
<Invoice>
 <Invoicedata>
 </Invoicedata>
</Invoice>
```

Embedded Payload Container

The following fragment represents an application payload nested within the SOAP Body element:

```
<SOAP-ENV:Body> <AppNS:Invoice
        xmlns:AppNS="http://my.app.com/ns"> <AppNS:Invoicedata/>
        </AppNS:Invoice> </SOAP-ENV:Body>
```

MIME Considerations

Additional MIME Parameters

Any MIME part described by this specification MAY contain additional MIME headers in conformance with the MIME [RFC 2045] specification. Implementations MAY ignore any MIME header not defined in this specification. Implementations MUST ignore any MIME header they do not recognize.

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

Reporting MIME Errors

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

XML Prolog

The SOAP Message's XML Prolog, if present, MAY contain an XML declaration. This specification has defined no additional comments or processing instructions appearing in the XML prolog. For example:

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

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

XML Declaration

The XML declaration MAY be present in a SOAP Message. If present, it MUST contain the version specification required by the XML Recommendation [XML] and MAY contain an encoding declaration. The semantics described below MUST be implemented by a compliant ebXML Message Service.

Encoding Declaration

If both the encoding declaration and the Header Container MIME charset are present, the XML prolog for the SOAP Message SHALL contain the encoding declaration SHALL be equivalent to the charset attribute of the MIME Content-Type of the Header Container (see the section called "charset Attribute").

If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding used when creating the SOAP Message. It is RECOMMENDED UTF-8 be used when encoding the SOAP Message.

If the character encoding cannot be determined by an XML processor using the rules specified in section 4.3.3 of XML [XML], the XML declaration and its contained encoding declaration SHALL be provided in the ebXML SOAP Header Document.

> **Note**
>
> The encoding declaration is not required in an XML document according to XML v1.0 specification [XML].

ebXML SOAP Envelope extensions

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

Namespace declarations (xmlns psuedo attributes) for the ebXML SOAP extensions may be included in the SOAP Envelope or Header elements, or directly in each of the ebXML SOAP extension elements.

Namespace pseudo attribute

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

```
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
```

xsi:schemaLocation attribute

The SOAP namespace:

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

resolves to a W3C XML Schema specification. All ebXML MSH implementations are strongly RECOMMENDED to include the XMLSchema-instance namespace qualified schemaLocation attribute in the SOAP Envelope element to indicate to validating parsers a location of the schema document that should be used to validate the document. Failure to include the schemaLocation attribute could prevent XML schema validation of received messages.

For example:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                          http://schemas.xmlsoap.org/soap/envelope/">
```

In addition, ebXML SOAP Header extension element content may be similarly qualified so as to identify the location where validating parsers can find the schema document containing the ebXML namespace qualified SOAP extension element definitions. The ebXML SOAP extension element schema has been defined using the W3C Recommendation version of the XML Schema specification [XMLSchema] (see Appendix A). The XMLSchema-instance namespace qualified schemaLocation attribute should include a mapping of the ebXML SOAP Envelope extensions namespace to its schema document in the same element that declares the ebXML SOAP Envelope extensions namespace.

The schemaLocation for the namespace described above in the section called "Namespace pseudo attribute" is:

http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd

Separate schemaLocation attribute are RECOMMENDED so tools, which may not correctly use the schemaLocation attribute to resolve schema for more than one namespace, will still be capable of validating an ebXML SOAP message. For example:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
 http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Header
  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
  xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <eb:MessageHeader ...>
   <eb:PayloadInfo eb:version="3.0">...</eb:PayloadInfo>
   ...
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body
  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
  xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  ...
 </SOAP:Body>
</SOAP:Envelope>
```

SOAP Header Element

The SOAP Header element is the first child element of the SOAP Envelope element. It MUST have a namespace qualifier that matches the SOAP Envelope namespace declaration for the namespace "http://schemas.xmlsoap.org/soap/envelope/".

SOAP Body Element

The SOAP Body element is the second child element of the SOAP Envelope element. It MUST have a namespace qualifier that matches the SOAP Envelope namespace declaration for the namespace "http://schemas.xmlsoap.org/soap/envelope/".

ebXML SOAP Extensions

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

SOAP Header Extensions

MessageHeader – a REQUIRED element containing routing information for the message (To/From, etc.) as well as other context information about the message.

PayloadInfo – an element pointing to any data present either in the Payload Container(s) or elsewhere, e.g. on the web. This element MAY also contain optional payload services elements. This element MAY be omitted. see the section called “Payload Services Module”

SOAP Body Extensions

ebXML Messaging does not define any extension elements for SOAP Body.

Core ebXML Modules

- Error Handling Module

  ErrorList - a SOAP Header element containing a list of the errors being reported against a previous message. The ErrorList element is only used if reporting an error or warning on a previous message. This element MAY be omitted.

- Security Module

  Security – an element that contains a digital signature that conforms to [XMLDSIG] that signs data associated with the message. This element MAY be omitted.

#wildcard Element Content

Some ebXML SOAP extension elements, as indicated in the schema, allow for foreign namespace-qualified element content to be added for extensibility. The extension element content MUST be namespace-qualified in accordance with XMLNS [XMLNS] and MUST belong to a foreign namespace. A foreign namespace is one that is NOT `http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd`. The wildcard elements are provided wherever extensions might be required for private extensions or future expansions to the protocol.

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

id Attribute

Each of the ebXML SOAP extension elements defined in this specification has an id attribute which is an XML ID that MAY be added to provide for the ability to uniquely identify the element within the SOAP Message. This MAY be used when applying a digital signature to the ebXML SOAP Message as individual ebXML SOAP extension elements can be targeted for inclusion or exclusion by specifying a URI of "`#<idvalue>`" in the Reference element.

version Attribute

The REQUIRED version attribute indicates the version of the ebXML Message Service Header Specification to which the ebXML SOAP Header extensions conform. Its purpose is to provide future versioning capabilities. For conformance to this specification, all of the version attributes on any SOAP extension elements defined in this specification MUST have a value of "3.0". An ebXML message MAY contain SOAP header extension elements that have a value other than "3.0". An implementation conforming to this specification that receives a message with ebXML SOAP extensions qualified with a version other than "3.0" MAY process the message if it recognizes the version identified and is capable of processing it. It MUST respond with an error (details TBD) if it does not recognize the identified version. The version attribute MUST be namespace qualified for the ebXML SOAP Envelope extensions namespace defined above.

Use of multiple versions of ebXML SOAP extensions elements within the same ebXML SOAP document, while supported, should only be used in extreme cases where it becomes necessary to semantically change an element, which cannot wait for the next ebXML Message Service Specification version release.

SOAP mustUnderstand Attribute

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

ebXML "Next MSH" role URI

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

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

All ebXML MSH nodes MUST assume this role.

ebXML "To Party MSH" role URI

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

The MSH that is the ultimate destination of ebXML messages MUST assume the role of the To Party MSH actor URI in addition to assuming the standardized "next" role as defined by SOAP.

Core Extension Elements

MessageHeader Element

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

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

- an id attribute (see section 3.3.7 for details).
- a version attribute (see section 3.3.8 for details).
- a SOAP mustUnderstand attribute with a value of "1" (see section 3.3.9 for details).
- From element.
- To element.
- CollaborationInfo element.
- MessageInfo element.
- PayloadInfo element.
- Description element.

From and To Elements

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

The From and the To elements each contains:

- PartyId elements – occurs one or more times.
- Role element – occurs zero or one times.

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

> **Note**
>
> This mechanism is particularly useful when transport of a message between the parties may involve multiple intermediaries. More generally, the From Party should provide identification in all domains it knows in support of intermediaries and destinations that may give preference to particular identification systems.

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

PartyId Element

The PartyId element has a single attribute, type and the content is a string value. The type attribute indicates the domain of names to which the string in the content of the PartyId element belongs. The value of the type attribute MUST be mutually agreed and understood by each of the Parties. It is RECOMMENDED that the value of the type attribute be a URI. It is further recommended that these values be taken from the EDIRA (ISO 6523), EDIFACT ISO 9735 or ANSI ASC X12 I05 registries.

If the PartyId type attribute is not present, the content of the PartyId element MUST be a URI [RFC2396], otherwise the Receiving MSH SHOULD report an error (see section 5.1.5) with errorCode set to Inconsistent and severity set to Error. It is strongly RECOMMENDED that the content of the PartyId element be a URI.

Role Element

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

Note

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

The following fragment demonstrates usage of the From and To elements.

```
<eb:From>
 <eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
 <eb:PartyId eb:type="SCAC">RDWY</PartyId>
 <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
</eb:From>

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

CollaborationInfo Element

The required CollaborationInfo Element identifies the parameters governing the exchange of messages between the parties.

The CollaborationInfo element contains:

- AgreementRef element.
- Service element.
- Action element.

AgreementRef Element

The REQUIRED AgreementRef element is a string that identifies the entity or artifact governing the exchange of messages between the parties. The recipient of a message MUST be able to resolve the AgreementRef to an individual set of parameters, taking into account the sender of the message.

The value of a AgreementRef element MUST be unique within a namespace mutually agreed by the two parties. This could be a concatenation of the From and To PartyId values, a URI prefixed with the Internet domain name of one of the parties, or a namespace offered and managed by some other naming or registry service. It is RECOMMENDED that the AgreementRef be a URI.

The AgreementRef MAY reference an instance of a CPA as defined in the ebXML Collaboration Protocol Profile and Agreement Specification [ebCPPA]. An example of the CPAId element follows:

```
<eb:AgreementRef>http://example.com/cpas/ourcpawithyou.xml</eb:AgreementRef>
```

The messaging parameters are determined by the appropriate elements from the CPA, as identified by the `AgreementRef` element.

If a receiver determines that a message is in conflict with the CPA, the appropriate handling of this conflict is undefined by this specification. Therefore, senders SHOULD NOT generate such messages unless they have prior knowledge of the receiver's capability to deal with this conflict.

If a Receiving MSH detects an inconsistency, then it MUST report it with an `errorCode` of `Inconsistent` and a severity of Error. If the `AgreementRef` is not recognized, then it MUST report it with an errorCode of `NotRecognized` and a severity of Error.

Service Element

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

- a standards organization, or
- an individual or enterprise.

Note

In the context of an ebXML business process model, an action equates to the lowest possible role based activity in the Business Process (see [BPSS]) (requesting or responding role) and a service is a set of related actions for an authorized role within a party.

An example of the Service element follows:

```
<eb:Service>urn:services:SupplierOrderProcessing</eb:Service>
```

Note

URIs in the `Service` element that start with the namespace `urn:oasis:names:tc:ebxml-msg:service` are reserved for use by this specification.

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

9.1.2.2.1 type Attribute

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

If the `type` attribute is not present, the content of the `Service` element MUST be a URI (see [RFC 2396]). If it is not a URI then report an error with `errorCode` of `Inconsistent` and severity of `Error` (see the section called "Error Handling Module").

Action Element

The REQUIRED Action element identifies a process within a Service that processes the Message. Action SHALL be unique within the Service in which it is defined. The value of the Action element is specified by the designer of the service. An example of the Action element follows:

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

If the value of either the Service or Action element are unrecognized by the Receiving MSH, then it MUST report the error with an errorCode of NotRecognized and a severity of Error.

MessageInfo Element

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

- MessageId element.
- RefToMessageId element.
- ConversationId element.
- Timestamp element

The following fragment demonstrates the structure of the MessageInfo element:

```
<eb:MessageInfo>
 <eb:MessageId>20001209-133003-28572@example.com</eb:MessageId>
 <eb:RefToMessageId>20001209-133003-28571@example.com</eb:RefToMessageId>
 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
 <eb:Timestamp>2004-06-15T11:12:12</eb:Timestamp>
</eb:MessageInfo>
```

MessageId Element

The REQUIRED element MessageId is a globally unique identifier for each message conforming to MessageId [RFC2822].

Note

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

RefToMessageId Element

The RefToMessageId element has a cardinality of zero or one. When present, it MUST contain the MessageId value of an ebXML Message to which this message relates.

For Error messages, the RefToMessageId element is REQUIRED and its value MUST be the MessageId value of the message in error (as defined in the section called “Error Handling Module”).

ConversationId Element

The REQUIRED ConversationId element is a string identifying the set of related messages that make up a conversation between two Parties. It MUST be unique within the context of the specified CPAId. The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation.

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

The value used for a ConversationId is implementation dependent. An example of the ConversationId element follows:

```
<eb:ConversationId>20001209-133003-28572</eb:ConversationId>
```

Note

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

Timestamp Element

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

PayloadInfo Element

The PayloadInfo element MAY be present as a child of the SOAP Header element. Each PayloadInfo element identifies payload data associated with the message, whether included as part of the message as payload document(s) contained in a Payload Container, or remote resources accessible via a URL. The purpose of the PayloadInfo is:

- to make it easier to directly extract a particular payload associated with this ebXML Message,
- to allow an application to determine whether it can process the payload without having to parse it.
- to define pre and post processing payload services to be performed by the MSH.

The PayloadInfo element is comprised of the following:

- an id attribute (see the section called “id Attribute” for details)
- a version attribute (see the section called “version Attribute” for details)
- zero or more Schema elements – information about the schema(s) that define the instance document identified in the parent Reference element
- zero or more Description elements – a textual description of the payload object referenced by the parent Reference element
- zero or one Processing elements - a list of processing steps to be performed by the MSH

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

See http://www.w3.org/TR/xptr-framework/ for fragment identifier definition. Replaced the xlink: attributes with URI for now. JWT.*

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

- id – an XML ID for the Payload element,
- payloadRef – this REQUIRED attribute has a value that is the CID URI or fragment identifier of the payload object referenced. For example "cid:foo" or "#idref".
- Any other namespace-qualified attribute MAY be present. A Receiving MSH MAY choose to ignore any foreign namespace attributes other than those defined above.

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

Schema Element

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

- namespace - the REQUIRED target namespace of the schema
- location – the REQUIRED URI of the schema
- version – a version identifier of the schema

Description Element

See the section called “Description Element” for more information.

PayloadInfo Validation

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

If an eb:payloadRef attribute contains a hash mark ('#') followed by a string value then an XML element containing an xml:id attribute with its value matching the string value, excluding the hash mark MUST be present in the SOAP Body element. If it is not, then the error SHALL be reported to the From Party with an errorCode of MimeProblem and a severity of Error.

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

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

PayloadInfo Sample

```
<eb:PayloadInfo eb:id="…" eb:payloadRef="cid:foo | #idref">
  <eb:Schema eb:location="http://foo/bar.xsd" eb:version="1.0"/>
  <eb:Description xml:lang="en-US">Purchase Order for 100,000 foo widgets</eb:Description>
  <eb:ProcessingStep eb:id="urn:foo:ps:CompressionSvc">
    <eb:Parameter eb:name="command" eb:value="uncompress" />
    <eb:Parameter eb:name="algorithm" eb:value="gzip" />
  </eb:ProcessingStep>
</eb:PayloadInfo>
```

Description Element

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

MessageHeader Sample

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

```
<eb:MessageHeader eb:id="…" eb:version="3.0" SOAP:mustUnderstand="1">
 <eb:From>
  <eb:PartyId>uri:example.com</eb:PartyId>
  <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
 </eb:From>
 <eb:To>
  <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
  <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
 </eb:To>
 <eb:CollaborationInfo>
  <eb:AgreementRef>http://www.oasis-open.org/cpa/123456</eb:AgreementRef>
  <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
  <eb:Action>NewPurchaseOrder</eb:Action>
 </eb:CollaborationInfo>
 <eb:MessageInfo>
  <eb:MessageId>UUID-2@example.com</eb:MessageId>
  <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
  <eb:ConversationId>987654321</eb:ConversationId>
  <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp>
 </eb:MessageInfo>
</eb:MessageHeader>
```

Core Modules

Security Module

Currently reworking this section to reflect the use of WS-Security. JWT : MM->JT: remove ebTA refs, reeval risk statements.

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

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

Each security risk is described in detail in the ebXML Technical Architecture Risk Assessment Technical Report [secRISK].

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

Application of countermeasures SHOULD be balanced against an assessment of the inherent risks and the value of the asset(s) that might be placed at risk. For this specification, a Signed Message is any message containing a Signature element.

Security Element

An ebXML Message MAY be digitally signed to provide security countermeasures. Zero or more Signature elements, belonging to the XML Signature [XMLDSIG] defined namespace, MAY be present as a child of the SOAP Header. The Signature element MUST be namespace qualified in accordance with XML Signature [XMLDSIG]. The structure and content of the Signature element MUST conform to the XML Signature [XMLDSIG] specification. If there is more than one Signature element contained within the SOAP Header, the first MUST represent the digital signature of the ebXML Message as signed by the From Party MSH in conformance with section 5.1. Additional Signature elements MAY be present, but their purpose is undefined by this specification.

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

Security and Management

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

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

Collaboration Protocol Agreement

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

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

Signature Generation

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

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

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

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

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

described in [XMLC14N]. This algorithm excludes comments.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note

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

Each payload object requiring signing SHALL be represented by a [XMLDSIG] Reference element that SHALL have a URI attribute resolving to the payload object. This can be either the Content-Id URI of the MIME body part of the payload object, or a URI matching the Content-Location of the MIME body part of the payload object, or a URI that resolves to a payload object external to the Message Package. It is strongly RECOMMENDED that the URI attribute value match the xlink:href URI value of the corresponding Manifest/Reference element for the payload object.

Note

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

Example of digitally signed ebXML SOAP Message:

```
<?xml version="1.0" encoding="utf-8"?>
<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
                                   http://www.oasis-open.org/committees/ebxml-msg/schema/soap12.xsd
                                   http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
                                   http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
 <SOAP:Header>
  <eb:MessageHeader eb:id="..." eb:version="2.0" SOAP:mustUnderstand="1">...</eb:MessageHeader>
  <eb:Manifest eb:id="Mani01" eb:version="2.0">
  <eb:Reference xlink:href="cid://blahblahblah/" xlink:role="http://ebxml.org/gci/invoice">
   <eb:Schema eb:version="2.0" eb:location="http://ebxml.org/gci/busdocs/invoice.dtd"/>
  </eb:Reference>
 </eb:Manifest>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
   <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="">
     <Transforms>
      <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
      <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
       <XPath> not(ancestor-or-self::()[@SOAP:actor=&quot;urn:oasis:names:tc:ebxml-msg:actor:nextMSH&quot;] |
               ancestor-or-self::()[@SOAP:actor=&quot;http://schemas.xmlsoap.org/soap/actor/next&quot;])</XPath>
      </Transform>
      <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
    </Transforms>
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>...</DigestValue>
   </Reference>
   <Reference URI="cid://blahblahblah/">
    <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>...</DigestValue>
   </Reference>
  </SignedInfo>
  <SignatureValue>...</SignatureValue>
  <KeyInfo>...</KeyInfo>
 </Signature>
</SOAP:Header>
<SOAP:Body>
...
</SOAP:Body>
</SOAP:Envelope>
```

Countermeasure Technologies

Persistent Digital Signature

The only available technology that can be applied to the purpose of digitally signing an ebXML Message (the ebXML SOAP Header and Body and its associated payload objects) is provided by technology that conforms to the W3C/IETF joint XML Signature specification [XMLDSIG]. An XML Signature conforming to this specification can selectively sign portions of an XML document(s), permitting the documents to be augmented (new element content added) while preserving the validity of the signature(s).

If signatures are being used to digitally sign an ebXML Message then XML Signature [XMLDSIG] MUST be used to bind the ebXML SOAP Header and Body to the ebXML Payload Container(s) or data elsewhere on the web that relate to the message.

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

Persistent Signed Receipt

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

Non-persistent Authentication

Non-persistent authentication is provided by the communications channel used to transport the ebXML Message. This authentication MAY be either in one direction or bi-directional. The specific method will be determined by the communications protocol used. For instance, the use of a secure network protocol, such as TLS [RFC 2246] or IPSec [RFC 2402] provides the sender of an ebXML Message with a way to authenticate the destination for the TCP/IP environment.

Non-persistent Integrity

A secure network protocol such as TLS [RFC 2246] or IPSec [RFC 2402] MAY be configured to provide for digests and comparisons of the packets transmitted via the network connection.

Persistent Confidentiality

XML Encryption is a W3C/IETF joint activity actively engaged in the drafting of a specification for the selective encryption of an XML document(s). It is anticipated that this specification will be completed within the next year. The ebXML Transport, Routing and Packaging team for v1.0 of this specification has identified this technology as the only viable means of providing persistent, selective confidentiality of elements within an ebXML Message including the SOAP Header.

Confidentiality for ebXML Payload Containers MAY be provided by functionality possessed by a MSH. Payload confidentiality MAY be provided by using XML Encryption (when available) or some other cryptographic process (such as S/MIME [SMIME], [SMIMEV3], or PGP MIME [PGPMIME]) bilaterally agreed upon by the parties involved. The XML Encryption standard shall be the default encryption method when XML Encryption has achieved W3C Recommendation status.

Note

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

Non-persistent Confidentiality

A secure network protocol, such as TLS [RFC 2246] or IPSEC [RFC 2402], provides transient confidentiality of a message as it is transferred between two ebXML adjacent MSH nodes.

Persistent Authorization

The OASIS Security Services Technical Committee (TC) is actively engaged in the definition of a specification that provides for the exchange of security credentials, including Name Assertion and Entitlements, based on Security Assertion Markup Language [SAML]. Use of technology based on this anticipated specification may provide persistent authorization for an ebXML Message once it becomes available.

[[[DALE TO PROVIDE NEW DETAILS]]]

Non-persistent Authorization

A secure network protocol such as TLS [RFC 2246] or IPSEC [RFC 2402] MAY be configured to provide for bilateral authentication of certificates prior to establishing a session. This provides for the ability for an ebXML MSH to authenticate the source of a connection and to recognize the source as an authorized source of ebXML Messages.

Trusted Timestamp

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

[[[INVESTIGATE]]]

Security Considerations

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

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

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

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

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

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

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

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

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

Error Handling Module

This section describes how one ebXML Message Service Handler (MSH) reports errors it detects in an ebXML Message to another MSH. The ebXML Message Service error reporting and handling module is to be considered as a layer of processing above the SOAP processor layer. This means the ebXML MSH is essentially an application-level handler of a SOAP Message from the perspective of the SOAP Processor. The SOAP processor MAY generate a SOAP Fault message if it is unable to process the message. A Sending MSH MUST be prepared to accept and process these SOAP Fault values.

It is possible for the ebXML MSH software to cause a SOAP Fault to be generated and returned to the sender of a SOAP Message. In this event, the returned message MUST conform to the [SOAP] specification processing guidelines for SOAP Fault values.

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

Definitions

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

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

Types of Errors

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

- ebXML namespace qualified content of the SOAP Message document (see the section called “Namespace pseudo attribute”)
- reliable messaging failures [[[(see section 7.5.7)]]]
- security (see the section called “Security Module”)

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

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

ErrorList Element

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

The ErrorList element consists of:

- id attribute (see the section called “id Attribute” for details)
- a version attribute (see the section called “version Attribute” for details
- a SOAP mustUnderstand attribute with a value of "1" (see the section called “SOAP mustUnderstand Attribute” for details)
- highestSeverity attribute
- one or more Error elements

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

highestSeverity attribute

The highestSeverity attribute contains the highest severity of any of the Error elements. Specifically, if any of the Error elements have a severity of Error, highestSeverity MUST be set to Error; otherwise, highestSeverity MUST be set to Warning.

Error Element

An Error element consists of:

- id attribute (see the section called “id Attribute” for details)
- codeContext attribute
- errorCode attribute
- severity attribute
- location attribute
- Description element

codeContext Attribute

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

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

errorCode attribute

The REQUIRED errorCode attribute indicates the nature of the error in the message in error. Valid values for the errorCode and a description of the code’s meaning are given in the next section.

severity Attribute

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

- Warning – This indicates other messages in the conversation could be generated in the normal way in spite of this problem.
- Error – This indicates there is an unrecoverable error in the message and no further message processing should occur. Appropriate failure conditions should be communicated to the Application.

location Attribute

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

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

If the error is associated with an ebXML Payload Container, then location contains the content-id of the MIME part in error, using URI scheme "cid".

If the error is associated with Payload Services, the location should contain the value of the sequence attribute of the Processing Step that caused the error.

id Attribute

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

Description Attribute

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

ErrorList Sample

An example of the ErrorList element is given below.

```
<eb:ErrorList eb:id="3490sdo", eb:highestSeverity="error"
 eb:version="3.0" SOAP:mustUnderstand="1">
 <eb:Error eb:errorCode="SecurityFailure"
           eb:severity="Error" eb:location="URI_of_ds:Signature">
  <eb:Description xml:lang="en-US">Validation of signature failed<eb:Description>
 </eb:Error>
 <eb:Error ...> ... </eb:Error>
</eb:ErrorList>
```

errorCode Values

This section describes the values for the errorCode attribute used in a message reporting an error. They are described in a table with three headings:

- the first column contains the value to be used as an errorCode, e.g. SecurityFailure.
- the second column contains a "Short Description" of the errorCode. This narrative MUST NOT be used in the content of the Error element.
- the third column contains a "Long Description" that provides an explanation of the meaning of the error and provides guidance on when the particular errorCode should be used.

Errors in the ebXML Elements

[[[todo table]]]

Non-XML Document Errors

[[[todo table]]]

Implementing Error Reporting and Handling

When to Generate Error Messages

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

- the Error Reporting Location (see the section called “Identifying the Error Reporting Location”) to which the message reporting the error should be sent can be determined.
- the message in error does not have an ErrorList element with highestSeverity set to Error.
- If the Error Reporting Location cannot be found or the message in error has an ErrorList element with highestSeverity set to Error, it is RECOMMENDED:

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

Identifying the Error Reporting Location

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

The ErrorURI implied by the CPA, identified by the CPAId on the message, SHOULD be used. Otherwise, the recipient MAY resolve an ErrorURI using the From element of the message in error. If neither is possible, no error will be reported to the sending Party.

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

Service and Action Element Values

An ErrorList element can be included in a SOAP Header that is part of a message being sent as a result of processing of an earlier message. In this case, the values for the Service and Action elements are set by the designer of the Service. This method MUST NOT be used if the highestSeverity is Error.

An ErrorList element can also be included in an independent message. In this case the values of the Service and Action elements MUST be set as follows:

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

Payload Services Module

Introduction

Payload services refers to functionality implemented by the messaging server to automatically perform some manipulation on payload content either before envelope digital signing (if non repudiation is being used) and message transmission, or after digital signature verification (if non repudiation is being used) and before passing the payload(s) in question to the application.

Payload services are not meant to be used as application level message handlers, rather, they should be treated as "filters".

Example Use Cases for Payload Services

Transparently converting XML content to ASN.1 and vice-versa

There are cases where it is desireable to use a more compact format, such as Abstract Syntax Notation to transmit data between partners, while still maintaining the easy-to-process and display qualities of XML. By using ASN.1's XML Encoding Rules (X ER), it is possible to convert between ASN.1 and XML.

Payload services could be used to automatically create ASN.1 representations of XML payloads prior to the message being transmitted. On the receiver's side, the same payload service could be used to convert back to XML. The net result being that th e application developers only see XML on both sides.

Compression

Sometimes, using a specialized compression algorithm can yield impressive reductions in a server's network utilization.

Payload services could be used to automatically compress and decompress payload content.

Encryption

Users often apply encryption at the payload level to ensure confidentiality of their payloads. Often, all that is needed to ensure confidentiality is to encrypt a single payload, as oposed to heavy weight approaches such as using S/MIME to encrypt an ent ire message.

Payload services could be used to encrypt and decrypt payloads.

XSL Transforms

As XML vocabularies evolve, business processes making use of XML messages will need to evolve with changes to their XML vocabularies. In cases where it would be too costly to modify the application(s) producing and consuming the XML, XSL can be used to p erform structural transformations.

Payload services could be used to automatically execute an XSL stylesheet prior to transmission, or before being delivered to the application.

Payload Service Invocation

Payload service invocation can be requested using two methods: SOAP Header Extensions, or CPA entries.

In cases where invocation requests are specified in both the CPA and the SOAP header, the CPA takes precedence. If the CPA entry explicitly forbids the use of payload services, then the Payload Services SOAP Headers MUST be ignored.

Payload services MAY NOT be invoked on the 0th attachment -- the SOAP envelope.

Invocation by SOAP Header Extension

Payload services may be invoked upon payloads by inserting the Processing element into the Payload entry within the message's PayloadInfo block. See the section called “PayloadInfo Sample” and Appendix A for more information. The Processing element that is inserted into the PayloadInfo header block is meant to define the processing step, if any, required to be performed on the receiving end of a messaging exchange.

It is anticipated that implementations of Payload Services will export the concept of Pre and Post processing steps via their message service interfaces, although specifying in the Payload element what action was taken on the sending side is not required.

As the example above illustrates, it is possible to chain payload services together using the *sequence* attribute. When interpreting the *order* attribute, lower numbers have higher priorities, and must be executed first. The value of the *sequence* attribute does not have to start at 0, although that convention is recommended for the sake of simplicity.

Error Handling

If an error is encountered during the payload services processing phase, it must be reported back to the sender with an errorCode attribute value of PayloadServicesFailure and severity of Error. The location attribute of the Error element should refer to the value of the sequence attribute of the Processing Step causing the error.

Required Services

This section defines services that every ebMS 3.0 compliant message handler must implement.

Compression Service

This service, named *urn:oasis-open:committees:ebxml-msg:ps:compression:3_0* provides a way of compressing payloads using a variety of algorithms. This specification defines that the gzip compression method, which is a variation of Lempel-Ziv (LZ77), MUST be supported.

The compression service defines the following mandatory parameters:

- `algorithm` - specify which compression algorithm will be used by the compression service. The default is '`gzip`'.

In addition to the mandatory parameters above, the following parameters MAY be used to configure the default compression method (gzip):

- `compression-level` - a value between 1 and 9, where 1 is faster and less CPU intensive, while 9 provides the best compression at the cost of speed.

Example 2. Sample CPA entry

```
TODO
```

Message Service Handler Ping Service

The OPTIONAL Message Service Handler Ping Service enables one MSH to determine if another MSH is operating. It consists of one MSH sending a Message Service Handler Ping message to a MSH, and another MSH, receiving the Ping, responding with a Message Service Handler Pong message.

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

Message Service Handler Ping Message

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

- a MessageHeader element containing the following:
- a From element identifying the Party creating the MSH Ping message.
- a To element identifying the Party being sent the MSH Ping message.
- a CollaborationInfo element containing:
  - a AgreementRef element.
  - a Service element containing: urn:oasis:names:tc:ebxml-msg:service [[[WHAT @type???]]]
  - an Action element containing Ping.
- a MessageInfo element containing:
  - a MessageId element.
  - a ConversationId element.
  - a Timestamp element.
- an [XMLDSIG] Signature element (see the section called “Security Module” for details).

The message is then sent to the To Party.

An example Ping:

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

--ebXMLBoundary
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
 xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
 http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
  xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <eb:MessageHeader version="3.0" SOAP:mustUnderstand="1">
   <eb:From>
    <eb:PartyId>urn:duns:123456789</eb:PartyId>
   </eb:From>
   <eb:To>
    <eb:PartyId>urn:duns:912345678</eb:PartyId>
   </eb:To>
   <eb:CollaborationInfo>
    <eb:AgreementRef>20001209-133003-28572</eb:AgreementRef>
    <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
    <eb:Action>Ping</eb:Action>
   </eb:CollaborationInfo>
   <eb:MessageInfo>
    <eb:MessageId>20010215-111212-28572@example.com</eb:MessageId>
    <eb:ConversationId>20010215-111213-28572</eb:ConversationId>
    <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
   </eb:MessageInfo>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>

--ebXMLBoundary––
```

Note

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

Message Service Handler Pong Message

Once the To Party receives the MSH Ping message, they MAY generate a Message Service Handler Pong (MSH Pong) message consisting of an ebXML Message containing no ebXML Payload Container and the following:

A MessageHeader element containing the following:

- a From element identifying the creator of the MSH Pong message.
- a To element identifying a Party that generated the MSH Ping message.
- a CollaborationInfo element containing:
  - a AgreementRef element.
  - a Service element containing: urn:oasis:names:tc:ebxml-msg:service [[[WHAT @type???]]]
  - an Action element containing Pong.
- a MessageInfo element containing:
  - a MessageId element.
  - a RefToMessageId identifying the MSH Ping message.
  - a ConversationId element.
  - a Timestamp element.
- an [XMLDSIG] Signature element (see the section called “Security Module” for details).

An example Pong:

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

--ebXMLBoundary
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
 xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
 http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
  xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <eb:MessageHeader version="3.0" SOAP:mustUnderstand="1">
   <eb:From>
    <eb:PartyId>urn:duns:912345678</eb:PartyId>
   </eb:From>
   <eb:To>
    <eb:PartyId>urn:duns:123456789</eb:PartyId>
   </eb:To>
   <eb:CollaborationInfo>
    <eb:AgreementRef>20001209-133003-28572</eb:AgreementRef>
    <eb:Service>urn:oasis:names:tc:ebxml-msg:service</eb:Service>
    <eb:Action>Pong</eb:Action>
   </eb:CollaborationInfo>
   <eb:MessageInfo>
    <eb:MessageId>20010215-111315-28573@example.com</eb:MessageId>
    <eb:RefToMessageId>20010215-111212-28572@example.com</eb:RefToMessageId>
    <eb:ConversationId>20010215-111213-28572</eb:ConversationId>
    <eb:Timestamp>2001-02-15T11:12:12</eb:Timestamp>
   </eb:MessageInfo>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>

--ebXMLBoundary––
```

Note

This example shows a non-multipart MIME structure.

Security Considerations

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

Pull Module

TBD

Pull Message Structure

A Pull Message consists of the following:

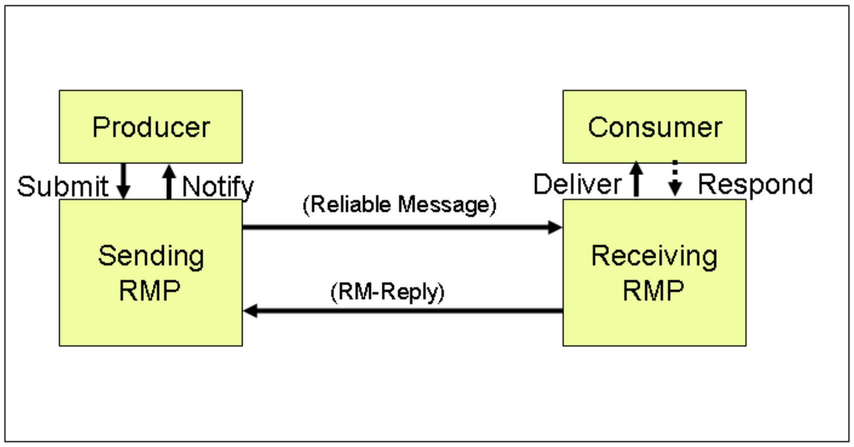
- The Service element MUST contain "urn:oasis:names:tc:msh:service".
- The Action element MUST contain "Pull".
- The PullRequest element, can occur zero or one time.
- The PullResponse element, can occur zero or one time. (The PullRequest and PullResponse elements are mutually exclusive)

Reliable Messaging Module

This ebXML Messaging Specification relies on the WS-Reliability 1.1 specification for its reliable messaging functionality. The ebXML MSH is then supporting the functions of a Reliable Messaging Processor (RMP) as defined in [WS-Reliability]. This module MUST be implemented by a conforming implementation. The Reliable Messaging Processor is a SOAP processor capable of performing Reliable Messaging functions.

As illustrated in Figure 9, the WS-Reliability model requires two instances of RMP playing different roles when transmitting a Reliable Message: the Sending RMP and the Receiving RMP.

Figure 9. Reliable Messaging Model



The reliability contract, which defines a particular quality of service about reliability for messages exchanged between two parties, has two aspects:

- The **protocol aspect** which defines how this contract affects the wire representation of the messages. This protocol defines what SOAP Reliability headers are used, and also which sequence of messages is expected.
- The **quality of service (QoS) aspect**, which focuses on what this contract means for the users of the messaging service (the parties that need to communicate reliably, called here Producer and Consumer). This aspect of the contract is expressed in terms of abstract operations: RM-Submit, RM-Deliver, RM-Respond, RM-Notify, between the users and the messaging layer considered as a black box.

Note: In this specification, we rename the RMP abstract operations – as defined in [WS-Reliability] - by prefixing their name with “RM-” in order to avoid confusion with the MSH operations. These operations are renamed RM-Submit, RM-Deliver, RM-Respond, RM-Notify. The abstract components involved in this messaging model (Producer, Consumer, RMP) as well as the abstract operations, are defined in [WS-Reliability].

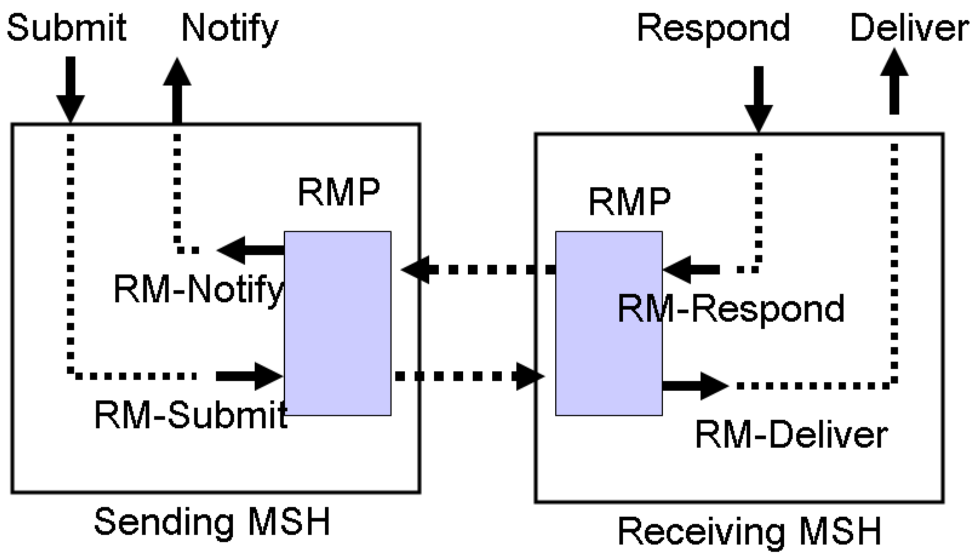
Reliability Contract for ebXML Messages

The reliability QoS is defined, in WS-Reliability, in terms of RMP abstract operations. For example, the guaranteed delivery contract for a message takes effect when the RM-Submit operation is invoked on the sending RMP. The contract is fulfilled when the RM-Deliver operation is invoked on the receiving RMP for this message, or else when the RM-Notify operation is invoked with “delivery failure” on the sending RMP.

The MSH abstract operations Submit, Deliver, Notify, Respond, which define an abstract interface between the MSH and its users, have additional ebMS semantics compared to RMP abstract operations. Concretely, this translates into additional header processing before and after the RMP operations are invoked. For example, when a message to be sent is passed to an MSH (Submit invocation), some processing occurs before the RM-Submit operation is invoked on this message for reliability processing.

Figure 10 illustrates the respective abstract interfaces of an MSH and its RMP component, in particular the subset of operations involved respectively in a sending and receiving roles.

Figure 10. Abstract MSH RMP Interfaces



Note: the arrows in the figure above represent the direction of transfer of message data, and do not have semantics related to which component is the target of the operation invocation.

Because the QoS reliability contract for ebMS is between the user layer and the MSH - and not between the MSH and its RMP component – it must be expressed in terms of MSH abstract operations. The three main reliability features – guaranteed delivery, duplicate elimination, and message ordering, as defined in WS-Reliability – must be interpreted in terms of MSH operations, not just RMP operations. As a consequence, it is not sufficient for an MSH to use an RMP component. In addition, an MSH must:

- Ensure a proper mapping between MSH abstract operations and RMP abstract operations. This mapping, which depends on the MEP being used, is described in a next section.
- Ensure the handling of additional failure cases that may happen outside the RMP processing. For example, in case of guaranteed delivery, the MSH must ensure that if a message that has been submitted fails before RM-Submit is invoked, then the user layer gets a notification of faiilure (Notify invocation), as would be the case if the message failed after RM-Submit was invoked.

SOAP Header Processing Rules

Some assumptions must be made on the processing of the headers of an ebMS message. The processing of an ebMS message by the MSH includes the processing of headers others than those that are ebMS-qualified, such as WS-Reliability, WS-Security and WS-Addressing headers. In particular, the functions of an MSH include RMP functions.

For the sake of composability and reusability of RMP implementations, it is desirable that the processing of SOAP headers that support WS-Reliability and the processing of other ebMS headers (as identified by the ebXML namespace) can be separated and strictly serialized.

The following serialization is REQUIRED, between Reliability headers and ebMS-qualified headers:

On the Sending Side:

- processing of ebMS headers (the ebms-qualified headers are added to the message).
- processing of Reliability headers (the WS-Reliability headers are added to the message).

On the Receiving Side:

- processing of Reliability headers (the WS-Reliability headers are removed from the message).
- processing of ebMS headers (the ebms-qualified headers are removed from the message).

Note: in the above workflows, the term “ebMS headers” must be understood as including WS-Addressing headers that are needed for the processing of ebMS-qualified headers.

WS-Reliability Implementation Requirements

This specification places the following constraints upon the use of WS-Reliability:

The following RM features MUST be supported:

- GuaranteedDelivery
- NoDuplicateDelivery
- OrderedDelivery
- GroupMaxIdleDuration
- GroupExpiryTime
- ExpiryTime
- ReplyPattern

The following Reply patterns MUST be supported:

- Response RM-Reply Pattern.
- Callback RM-Reply Pattern.
- Synchronous Poll RM-Reply.

Reliability of ebXML Messages

Some reliable messages do not involve the invocation of MSH abstract operations. These are ebMS messages that do not result from submission of payloads (Submit) by a Producer to the MSH and are instead initiated by MSH functions ("signal" messages) can be made reliable too. For such messages, the reliability contract is expressed in terms of RMP abstact operations. For example, it starts with RM-Submit invocation (submission by the MSH to the RMP). The message has been reliably transmitted when RM-Deliver is successfully invoked, i.e. when delivered to the receiving MSH functions.

Reliability of the Simple Push MEP

The pushed message, to be sent either as a SOAP One-way or a SOAP Request, is submitted to the RMP module via the "RM-Submit" operation. The sequence of abstract operation invocations steps for a successful instance of this MEP is as follows:

On Requesting MSH side:

- Step (1): **Submit**: submission of the message payload to the MSH by the Producer party.
- Step (2): **RM-Submit**: after processing of ebXML headers, submission to the RMP.

On Responding MSH side:

- Step (3): **RM-Deliver**: after processing of reliability headers, delivery to other MSH functions.
- Step (4): **Deliver**: after processing of ebXML headers, delivery of the message payload to the Consumer of the MSH.
- The semantics of RM-Deliver must be interpreted as including the delivery from MSH to its consumer (Deliver invocation). In other words, if the Deliver invocation fails, then the RM-Deliver invocation must also fail. In such case the RMP will generate a MessageProcessingFailure fault, and will never acknowledge a reliable message (with guaranteed delivery) that has not been successfully delivered by the MSH to its consumer.
- Other steps for the processing of other headers, such as Security headers, are not mentioned here. The above sequence does not exclude the insertion of such additional steps at an appropriate place.

The RM-Agreement associated with the message, as defined in WS-Reliability, can be used without restrictions, except for the following:

- In case ReplyPattern has value "Poll" in a reliable message, the PollRequest sent later by the sending RMP for this message must be synchronous (the ReplyTo element MUST NOT present).

Reliability of the Simple Pull MEP

The processing model is as follows, for a typical and successful instance of this MEP:

On Requesting MSH side:

- Step (1): Sending of a Pull signal by the MSH. If the Pull is sent reliably, as recommended later in this section, **RM-Submit** is invoked on the sending RMP for this signal.

On Responding MSH side:

- Step (2): **Submit**: submission of a message payload to the MSH by the Producer party, intended to the Consumer on the Requesting side. That submission may actually have occurred any time before Step (1)
- Step (3): Reception of the Pull signal by MSH functions. If the Pull was sent reliably, as recommended, this delivery is achieved by **RM-Deliver** invocation on the receiving RMP for this signal.
- Step (4): Pull: The **Pull** signal invokes the related MSH operation.
- Step (5): Submission of the pulled message to the RMP. This results in an **RM-Respond** invocation if the Pull signal was sent reliably, as recommended. Otherwise, RM-Submit is invoked.

On Requesting MSH side:

- Step (6): **RM-Notify**: after processing of reliability headers of the pulled message, delivery to other MSH functions.
- Step (7): **Notify**: after processing of ebXML headers, delivery of the pulled message payload to the Consumer of the MSH.

It is important to note that in step (6) the invocation of RM-Notify is considered successful once the invocation of Notify (step 7) has successfully completed. Otherwise, a

MessageProcessingFailure fault must be generated by the RMP, for a message with guaranteed delivery, and the pulled message must not be acknowledged. Other steps for the processing of other headers, such as Security headers, are not mentioned here. The above sequence does not exclude the insertion of such additional steps at an appropriate place.

The RM-Agreement associated with the pulled message MUST comply with the following restrictions:

Table 1.

Name	Allowed Values
GuaranteedDelivery	“enabled”, “disabled” When enabled, it is RECOMMENDED that the Pull signal message associated with this pulled message be also sent with this parameter enabled. When the Pull signal is sent with GuaranteedDelivery enabled, two additional requirements MUST be satisfied: • The NoDuplicateDelivery agreement item is also enabled for the Pull signal. • The RMP module MUST send back a copy of the original pulled message if the latter is not expired, when a duplicate of the Pull signal is received, e.g. due to resending. This is achieved by supporting the first option for responding to duplicates of messages sent with Response ReplyPattern (Section 3.2.2 of [WS-Reliability], second part of protocol requirements).
NoDuplicateDelivery	“enabled”, “disabled” When enabled, the Pull signal message associated with this pulled message MUST also be sent with this parameter enabled.
OrderedDelivery	“enabled”, “disabled” No restriction.
ReplyPattern	“Callback” In case the Pull signal message associated with this pulled message was made reliable with GuaranteedDelivery enabled, the ReplyPattern value associated with the Pull signal MUST be “Response”.

It is important to note that WS-Reliability 1.1 is silent about the reliability of messages submitted as responses to other messages. Such messages would be submitted using the abstract operation RM-Respond, which requires an RMP to correlate the response message with the related request. This specification requires that the reliablity of these responses, in the case of pulled messages, be also supported by the MSH. This means that the implementation of RMP used in an MSH must also support RM agreements that cover these responses.

Reliability of the Simple Request-Response MEP

The processing model is as follows, for a typical and successful instance of this MEP:

On Requesting MSH Side:

- Step (1): **Submit**: submission of the request message payload to the MSH by the Producer party.
- Step (2): **RM-Submit**: after processing of ebXML headers, submission of the request message to the RMP.

On Responding MSH Side:

- Step (3): **RM-Deliver**: after processing of reliability headers, delivery of the request message to other MSH functions.
- Step (4): **Deliver**: after processing of ebXML headers, delivery of the request message payload to the Consumer of the MSH.
- Step (4): **RM-Respond**: submission of the response message to the RMP.

On the Requesting MSH Side:

- Step (5): **RM-Notify**: after processing of reliability headers of the response message, delivery to other MSH functions.
- Step (6): **Notify**: after processing of ebXML headers, delivery of the response message payload to the Producer of the MSH.

It is important to note that in step (3), as already mentioned for the Simple Push MEP, the invocation of RM-Deliver is considered successful once the invocation of Deliver (step 4) has successfully completed. Otherwise, a Messageprocessing Failure will be generated by the RMP, for a message with guaranteed delivery. The same interpretation applies to steps (5) and (6): when a reliable response fails to be delivered by theMSH to the requesting party (Notify).

Other steps for the processing of other headers, such as Security headers, are not mentioned here. The above sequence does not exclude the insertion of such additional steps at an appropriate place.

The RM-Agreement associated with the Request message MUST comply with the same restrictions as for the Simple Push MEP, and also with those entailed by the RM-Agreement options used for the Response (see below.) Note that Request message and Response message do not have to share the same RM-Agreement.

The RM-Agreement associated with the Response message MUST comply with the following restrictions:

Table 2.

Name	Allowed Values
GuaranteedDelivery	“enabled”, “disabled” When enabled, it is RECOMMENDED that the Request message associated with this Response message be also sent with this parameter enabled. When the Request is sent with GuaranteedDelivery enabled, two additional requirements MUST be satisfied:(1) The NoDuplicateDelivery agreement item is also enabled for the Request.(2) The RMP module MUST send back a copy of the original Response message if the latter is not expired, when a duplicate of the Request is received, e.g. due to resending. This is achieved by supporting the first option for responding to duplicates of messages sent with Response ReplyPattern (Section 3.2.2 of [WS-Reliability], second part of protocol requirements).

Name	Allowed Values
NoDuplicateDelivery	“enabled”, “disabled” When enabled, the Request message associated with this Response message MUST also be sent with this parameter enabled.
OrderedDelivery	“enabled”, “disabled” No restriction.
ReplyPattern	“Callback” In case the Request message associated with this Response message was made reliable with GuaranteedDelivery enabled, the ReplyPattern value associated with the Request signal MUST be “Response”.

Reliability of Aggregate MEPs

Aggregate MEPs are combinations of simple MEPs that are related to each other via references to ebMS message IDs. Their reliability is based on the reliability of the simpler MEPs of which they are composed. All the messages involved in an aggregate MEP do not have to share the same RM-Agreement (in the same way as Request message and Response message of a simple Request-response MEP do not.)

The processing model for (some) messages of an aggregate MEP must respect the requirement of referencing to a previous message. This is abstractly specified by the use of the MSH Respond operation. A reliable Push-Push MEP (with both Request and Response reliable) will be processed as follows:

On Requesting MSH side:

- Step (1): Submit: submission of the Request message payload to the MSH by the Producer party.
- Step (2): RM-Submit: after processing of ebXML headers, submission to the RMP.

On Responding MSH side:

- Step (3): RM-Deliver: after processing of reliability headers, delivery to other MSH functions.
- Step (4): Deliver: after processing of ebXML headers, delivery of the Request message payload to the Consumer of the MSH.
- Step (5): Respond: submission of the Response message payload to the MSH by the Producer party.
- Step (6): RM-Submit: after processing of ebXML headers, submission to the RMP.

On Requesting MSH side:

- Step (7): RM-Deliver: after processing of reliability headers, delivery to other MSH functions.
- Step (8): Deliver: after processing of ebXML headers, delivery of the Response message payload to the Consumer of the MSH.

It is important to note that this sequence of steps amounts to a concatenation of twice the same sequence of steps defined for the simple Push MEP, with the following modifications in the second sequence: (a) it runs in the opposite direction to the first sequence, (b) Step (5) is a Respond MSH invocation (instead of a Submit), which will relate the Response to the Request message. The interpretation of RM-Deliver (steps 3 and 7) must only consider its invocations successful if the following invocations of Deliver (steps 4 and 8) have completed successfully, as alrady mentioned for previous MEPs.

The processing model of other reliable aggregate MEPs can similarly be derived from the processing models of the simple reliable MEPs they are composed with. The MSH Respond operation must be used every time a submitted message must refer to a previously received message.

Message Delivery Semantics

Message Delivery is an abstract operation that transfers a payload from the Receiving MSH to the Consumer. A message is only acknowledged after successful Message Delivery. The interpretation of the WS-Reliability 1.1 RMP "Deliver" operation MUST be as follows:

The RMP "Deliver" operation includes the MSH Message Delivery operation. Failure of the MSH Message Delivery operation results in the failure of the RMP "Deliver" operation.

Fault Handling

Any processing error that results in the received message not being delivered to the consumer MUST be reported as an RMP delivery failure as defined in WS-Reliability 1.1, generating a MessageProcessingFailure Fault to the sending RMP.

Combining ebXML SOAP Extension Elements

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

MessageHeader Element Interaction

The MessageHeader element MUST be present in every message.

PayloadInfo Element Interaction

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

Signature Element Interaction

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

Errorlist Element Interaction

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

If the highestSeverity attribute on the ErrorList is set to Error, then this element MUST NOT be present with the Manifest element.

PayloadServices Element Interaction

The PayloadServices element MAY be present on any message sent or received.

Additional (optional) Features

Message Status Service

The Message Status Request Service consists of the following:

- Message Status Request message containing details regarding a message previously sent is sent to a Message Service Handler (MSH).
- The Message Service Handler receiving the request responds with a Message Status Response message.

A Message Service Handler SHOULD respond to Message Status Requests for messages that have been sent reliably and the MessageId in the RefToMessageId is present in persistent storage (see the section called “Message Status Messages”).

A Message Service Handler MAY respond to Message Status Requests for messages that have not been sent reliably.

A Message Service SHOULD NOT use the Message Status Request Service to implement Reliable Messaging.

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

Message Status Messages

Message Status Request Message

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

A MessageHeader element containing:

- a From element identifying the Party that created the Message Status Request message.
- a To element identifying a Party who should receive the message.
- a CollaborationInfo element containing:
    - an AgreementRef element.
    - a Service element that contains: urn:oasis:names:tc:ebxml-msg:service
    - an Action element that contains StatusRequest.
- a MessageInfo element containing:
    - a MessageId element.
    - a ConversationId element.
    - a Timestamp element.
- A StatusRequest element (see the section called “StatusRequest Element”) containing:
    - a RefToMessageId element in StatusRequest element containing the MessageId of the message whose status is being queried.
- an [XMLDSIG] Signature element (see the section called “Security Module” for more details).

The message is then sent to the To Party.

Message Status Response Message

Once the To Party receives the Message Status Request message, they SHOULD generate a Message Status Response message with no ebXML Payload Container consisting of the following:

A MessageHeader element containing:

- a From element that identifies the sender of the Message Status Response message.
- a To element set to the value of the From element in the Message Status Request message
- a CollaborationInfo element containing:
    - an AgreementRef element.
    - a Service element that contains: urn:oasis:names:tc:ebxml-msg:service.
    - an Action element that contains StatusResponse.
- An Action element that contains StatusResponse
- A MessageInfo element containing a RefToMessageId that identifies the Message Status Request message.

  - a MessageId element.
  - a RefToMessageId element that identifies the Message Status Request message.
  - a ConversationId element.
  - a Timestamp element.
- StatusResponse element (see the section called “StatusResponse Element”)
- an [XMLDSIG] Signature element (see the section called “Security Module” for more details).

The message is then sent to the To Party.

Security Considerations

Parties who receive a Message Status Request message SHOULD always respond to the message. However, they MAY ignore the message instead of responding with messageStatus set to UnAuthorized if they consider the sender of the message to be unauthorized. The decision process resulting in this course of action is implementation dependent.

StatusRequest Element

The OPTIONAL StatusRequest element is an immediate child of a SOAP Body and is used to identify an earlier message whose status is being requested (see the section called “StatusResponse Element Interaction”).

The StatusRequest element consists of the following:

- an id attribute (see the section called “id Attribute” for details)
- a version attribute (see the section called “version Attribute” for details)
- a RefToMessageId element

RefToMessageId Element

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

StatusRequest Sample

An example of the StatusRequest element is given below:

```
<eb:StatusRequest eb:version="3.0" >
 <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
</eb:StatusRequest>
```

StatusRequest Element Interaction

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

- a PayloadInfo element
- a StatusResponse element
- an ErrorList element

StatusResponse Element

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

The StatusResponse element consists of the following elements and attributes:

- an id attribute (see the section called “id Attribute” for details).
- a version attribute (see the section called “version Attribute” for details).
- a RefToMessageId element.
- a Timestamp element.
- a messageStatus attribute.

RefToMessageId Element

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

Timestamp Element

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

messageStatus attribute

The REQUIRED messageStatus attribute identifies the status of the message identified by the RefToMessageId element. It SHALL be set to one of the following values:

- UnAuthorized – the Message Status Request is not authorized or accepted.

- NotRecognized – the message identified by the RefToMessageId element in the StatusResponse element is not recognized.
- Received – the message identified by the RefToMessageId element in the StatusResponse element has been received by the MSH.
- Processed – the message identified by the RefToMessageId element in the StatusResponse element has been processed by the MSH.
- Forwarded – the message identified by the RefToMessageId element in the StatusResponse element has been forwarded by the MSH to another MSH.

Note

If a Message Status Request is sent after the elapsed time indicated by PersistDuration has passed since the message being queried was sent, the Message Status Response may indicate the MessageId was NotRecognized – the MessageId is no longer in persistent storage.

StatusResponse Sample

An example of the StatusResponse element is given below:

```
<eb:StatusResponse eb:version="3.0" eb:messageStatus="Received">
 <eb:RefToMessageId>323210:e52151ec74:-7ffc@xtacy</eb:RefToMessageId>
 <eb:Timestamp>2001-03-09T12:22:30</eb:Timestamp>
</eb:StatusResponse>
```

StatusResponse Element Interaction

This element MUST NOT be present with the following elements:

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

Multi-Hop Module

A. The ebXML SOAP Extension Elements Schema

The OASIS ebXML Messaging Technical Committee has provided a version of the SOAP 1.1 envelope schema specified using the schema vocabulary that conforms to the W3C XML Schema Recommendation specification [XMLSchema].

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

It was necessary to craft a schema for the XLINK [XLINK] attribute vocabulary to conform to the W3C XML Schema Recommendation [XMLSchema]. This schema is referenced from the ebXML SOAP extension elements schema and is available from the following URL: Xlink - http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
        version="1.0" xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:tns="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
        xmlns:ns="http://www.w3.org/2001/XMLSchema"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <import namespace="http://www.w3.org/XML/1998/namespace"
          schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>

  <import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
          schemaLocation="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"/>

  <import namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
          schemaLocation="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"/>

  <complexType name="HeaderBaseType">
    <sequence>
      <any maxOccurs="unbounded" minOccurs="0" namespace="##other"
           processContents="lax"/>
    </sequence>

    <attributeGroup ref="tns:headerExtension.grp"/>
  </complexType>

  <element name="MessageHeader">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element ref="tns:From"/>

            <element ref="tns:To"/>

            <element ref="tns:CollaborationInfo"/>

            <element ref="tns:MessageInfo"/>

            <element maxOccurs="unbounded" minOccurs="0" ref="tns:PayloadInfo"/>

            <element maxOccurs="unbounded" minOccurs="0" ref="tns:Description"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="To">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" ref="tns:PartyId"/>

        <element minOccurs="0" ref="tns:Role"/>
      </sequence>
    </complexType>
  </element>

  <element name="From">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" ref="tns:PartyId"/>

        <element minOccurs="0" ref="tns:Role"/>
      </sequence>
    </complexType>
  </element>

  <element name="PayloadInfo">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element maxOccurs="unbounded" minOccurs="0" ref="tns:Schema"/>

            <element maxOccurs="unbounded" minOccurs="0" ref="tns:Description"/>

            <element maxOccurs="unbounded" minOccurs="0" ref="tns:ProcessingStep"/>
          </sequence>

          <attribute ref="tns:payloadRef" use="required"/>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="Schema">
    <complexType>
      <attribute ref="tns:namespace" use="required"/>
      <attribute ref="tns:location" use="required"/>
      <attribute ref="tns:version"/>
    </complexType>
  </element>

  <element name="ProcessingStep">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element maxOccurs="unbounded" minOccurs="0" ref="tns:Parameter"/>
          </sequence>
          <attribute ref="tns:service" use="required"/>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="Parameter">
    <complexType>
      <attribute ref="tns:name" use="required"/>

      <attribute ref="tns:value" use="required"/>
    </complexType>
  </element>

  <element name="AgreementRef" type="tns:non-empty-string"/>

  <element name="ConversationId" type="tns:non-empty-string"/>

  <element name="Service">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute name="type" type="tns:non-empty-string"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

  <element name="Action" type="tns:non-empty-string"/>

  <element name="MessageInfo">
    <complexType>
      <sequence>
        <element ref="tns:ConversationId"/>

        <element ref="tns:Timestamp"/>

        <element ref="tns:TimeToLive" minOccurs="0"/>
      </sequence>
    </complexType>
  </element>

  <element name="CollaborationInfo">
    <complexType>
      <sequence>
        <element ref="tns:AgreementRef"/>

        <element ref="tns:Service"/>

        <element ref="tns:Action"/>
      </sequence>
    </complexType>
  </element>

  <element name="TimeToLive" type="dateTime"/>

  <!-- ERROR LIST, for use in soap:Header element -->

  <element name="ErrorList">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element maxOccurs="unbounded" ref="tns:Error"/>
          </sequence>
          <attribute name="highestSeverity" type="tns:severity.type"
                     use="required"/>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="Error">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element minOccurs="0" ref="tns:Description"/>
          </sequence>

          <attribute default="urn:oasis:names:tc:ebxml-msg:service:errors"
                     name="codeContext" type="anyURI"/>

          <attribute name="errorCode" type="tns:non-empty-string"
                     use="required"/>

          <attribute name="severity" type="tns:severity.type" use="required"/>

          <attribute name="location" type="tns:non-empty-string"/>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <element name="PullRequest">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <choice>
              <element ref="tns:To"/>

              <element ref="tns:RefToMessageId"/>
            </choice>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <!-- STATUS RESPONSE, for use in soap:Header element -->

  <element name="StatusResponse">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element ref="tns:RefToMessageId"/>

            <element minOccurs="0" ref="tns:Timestamp"/>
          </sequence>

          <attribute name="messageStatus" type="tns:messageStatus.type"
                     use="required"/>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <!-- STATUS REQUEST, for use in soap:Header element -->

  <element name="StatusRequest">
    <complexType>
      <complexContent>
        <extension base="tns:HeaderBaseType">
          <sequence>
            <element ref="tns:RefToMessageId"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>

  <!-- COMMON TYPES -->

  <simpleType name="status.type">
    <restriction base="NMTOKEN">
      <enumeration value="Reset"/>

      <enumeration value="Continue"/>
    </restriction>
  </simpleType>

  <simpleType name="messageStatus.type">
    <restriction base="NMTOKEN">
      <enumeration value="UnAuthorized"/>

      <enumeration value="NotRecognized"/>

      <enumeration value="Received"/>

      <enumeration value="Processed"/>

      <enumeration value="Forwarded"/>

      <enumeration value="MessageNotAvailable"/>
    </restriction>
  </simpleType>

  <simpleType name="non-empty-string">
    <restriction base="string">
      <minLength value="1"/>
    </restriction>
  </simpleType>

  <simpleType name="non-negative-integer">
    <restriction base="nonNegativeInteger"/>
  </simpleType>

  <simpleType name="severity.type">
    <restriction base="NMTOKEN">
      <enumeration value="Warning"/>

      <enumeration value="Error"/>
    </restriction>
  </simpleType>

  <!-- ATTRIBUTES and ATTRIBUTE GROUPS -->

  <attribute name="id" type="ID"/>

  <attribute name="version" type="tns:non-empty-string"/>

  <attributeGroup name="headerExtension.grp">
    <attribute ref="tns:id"/>

    <attribute ref="tns:version" use="required"/>

    <anyAttribute namespace="##other" processContents="lax"/>
  </attributeGroup>

  <attribute name="service" type="anyURI"/>

  <attribute name="name" type="tns:non-empty-string"/>

  <attribute name="value" type="tns:non-empty-string"/>

  <attribute name="location" type="anyURI"/>

  <attribute name="namespace" type="anyURI"/>

  <attribute name="payloadRef" type="tns:non-empty-string"/>

  <!-- COMMON ELEMENTS -->

  <element name="PartyId">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute name="type" type="tns:non-empty-string"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

  <element name="Description">
    <complexType>
      <simpleContent>
        <extension base="tns:non-empty-string">
          <attribute ref="xml:lang" use="required"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

  <element name="RefToMessageId" type="tns:non-empty-string"/>

  <element name="Role" type="tns:non-empty-string"/>

  <element name="Timestamp" type="dateTime"/>
</schema>
```

B. Communications Protocol Bindings

Introduction

One of the goals of this specification is to design a message handling service usable over a variety of network and application level transport protocols. These protocols serve as the "carrier" of ebXML Messages and provide the underlying services necessary to carry out a complete ebXML Message exchange between two parties.

HTTP, FTP, Java Message Service (JMS) and SMTP are examples of application level transport protocols. TCP and SNA/LU6.2 are examples of network transport protocols. Transport protocols vary in their support for data content, processing behavior and error handling and reporting. For example, it is customary to send binary data in raw form over HTTP. However, in the case of SMTP it is customary to "encode" binary data into a 7-bit representation. HTTP is equally capable of carrying out synchronous or asynchronous message exchanges whereas it is likely that message exchanges occurring over SMTP will be asynchronous.

This section describes the technical details needed to implement this abstract ebXML Message Handling Service over particular transport protocols.

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

- Hypertext Transfer Protocol [RFC 2616], in both asynchronous and synchronous forms of transfer.
- Simple Mail Transfer Protocol [RFC 2821], in asynchronous form of transfer only.
- File Transfer Protocol [RFC 949], in asynchronous form of transfer only.

HTTP

Minimum Level of HTTP Protocol

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

Sending ebXML Service Messages over HTTP

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

POST /ebxmlhandler HTTP/1.1

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

HTTP protocol natively supports 8-bit and Binary data. Hence, transfer encoding is OPTIONAL for such parts in an ebXML Service Message prior to sending over HTTP. However, content-transfer-encoding of such parts (e.g. using base64 encoding scheme) is not precluded by this specification.

The rules for forming an HTTP message containing an ebXML Service Message are as follows:

- The Content-Type MIME header with the associated parameters, from the ebXML Service Message Envelope MUST appear as an HTTP header.
- All other MIME headers that constitute the ebXML Message Envelope MUST also become part of the HTTP header.
- The mandatory SOAPAction HTTP header field must also be included in the HTTP header and MAY have a value of "ebXML"

  Further, it is recomended that sending MSHs act liberally in allowing the prescence and/or absence of the SOAPAction header in synchronous responses, and that implementers refer to the SOAP 1.1 specifications for guidance.
- Other headers with semantics defined by MIME specifications, such as Content-Transfer-Encoding, SHALL NOT appear as HTTP headers. Specifically, the "MIME-Version: 1.0" header MUST NOT appear as an HTTP header. However, HTTP-specific MIME-like headers defined by HTTP 1.1 MAY be used with the semantic defined in the HTTP specification.
- All ebXML Service Message parts that follow the ebXML Message Envelope, including the MIME boundary string, constitute the HTTP entity body. This encompasses the SOAP Envelope and the constituent ebXML parts and attachments including the trailing MIME boundary strings.

The example below shows an example instance of an HTTP POST ebXML Service Message:

Example B.1. HTTP POST Example

```
POST /servlet/ebXMLhandler HTTP/1.1
Host: www.example2.com
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
 start="<ebxhmheader111@example.com>"

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

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

<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
 <SOAP:Header>
  <eb:MessageHeader eb:id="…" eb:version="3.0" SOAP:mustUnderstand="1">
   <eb:From>
    <eb:PartyId>uri:example.com</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
   </eb:From>
   <eb:To>
    <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
   </eb:To>
   <eb:CollaborationInfo>
    <eb:AgreementRef>http://www.oasis-open.org/cpa/123456</eb:AgreementRef>
    <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
    <eb:Action>NewPurchaseOrder</eb:Action>
   </eb:CollaborationInfo>
   <eb:MessageInfo>
    <eb:MessageId>UUID-2@example.com</eb:MessageId>
    <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
    <eb:ConversationId>987654321</eb:ConversationId>
    <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp>
   </eb:MessageInfo>
   <eb:PayloadInfo>
    <eb:Payload eb:id="…" eb:payloadRef="cid:ebxmlpayload111@example.com">
     <eb:Schema eb:location="http://foo/bar.xsd" eb:version="1.0"/>
     <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
     <eb:Processing>
     <eb:Step eb:sequence="0" eb:id="urn:foo:ps:CompressionSvc">
      <eb:Parameter eb:name="command" eb:value="uncompress" />
      <eb:Parameter eb:name="algorithm" eb:value="gzip" />
     </eb:Step>
     </eb:PostProcessing>
    <eb:Payload>
   </eb:PayloadInfo>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>

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

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

--BoundarY––
```

HTTP Response Codes

In general, semantics of communicating over HTTP as specified in [RFC 2616] MUST be followed, for returning the HTTP level response codes. A 2xx code MUST be returned when the HTTP Posted message is successfully received by the receiving HTTP entity. However, see exception for SOAP error conditions below. Similarly, other HTTP codes in the 3xx, 4xx, 5xx range MAY be returned for conditions corresponding to them. However, error conditions encountered while processing an ebXML Service Message MUST be reported using the error mechanism defined by the ebXML Message Service Specification.

SOAP Error Conditions and Synchronous Exchanges

The SOAP 1.1 specification states:

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

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

Synchronous vs. Asynchronous

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

Access Control

Basic & Digest Authentication

Implementers MAY protect their ebXML Message Service Handlers from unauthorized access through the use of an access control mechanism. The HTTP access authentication process described in "HTTP Authentication: Basic and Digest Access Authentication" [RFC 2617] defines the access control mechanisms allowed to protect an ebXML Message Service Handler from unauthorized access.

Implementers MAY support all of the access control schemes defined in [RFC 2617] including support of the Basic Authentication mechanism, as described in [RFC 2617] section 2, when Access Control is used.

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

SSL Client (Digital Certificate) Authentication

Confidentiality and Transport Protocol Level Security

An ebXML Message Service Handler MAY use transport layer encryption to protect the confidentiality of ebXML Messages and HTTP transport headers. The IETF Transport Layer Security specification TLS [RFC 2246] provides the specific technical details and list of allowable options, which may be used by ebXML Message Service Handlers. ebXML Message Service Handlers MUST be capable of operating in backwards compatibility mode with SSL [SSL3], as defined in Appendix E of TLS [RFC 2246].

ebXML Message Service Handlers MAY use any of the allowable encryption algorithms and key sizes specified within TLS [RFC 2246]. At a minimum ebXML Message Service Handlers MUST support the key sizes and algorithms necessary for backward compatibility with [SSL3].

The use of 40-bit encryption keys/algorithms is permitted, however it is RECOMMENDED that stronger encryption keys/algorithms SHOULD be used.

Both TLS [RFC 2246] and SSL [SSL3] require the use of server side digital certificates. Client side certificate based authentication is also permitted. All ebXML Message Service handlers MUST support hierarchical and peer-to-peer or direct-trust trust models.

SMTP

The Simple Mail Transfer Protocol (SMTP) [RFC 2821] specification is commonly referred to as Internet Electronic Mail. This specifications has been augmented over the years by other specifications, which define additional functionality "layered on top" of this baseline specifications. These include:

- Multipurpose Internet Mail Extensions (MIME) [RFC 2045], [RFC 2046], [RFC 2387].
- SMTP Service Extension for Authentication [RFC 2554].
- SMTP Service Extension for Secure SMTP over TLS [RFC 2487].

Typically, Internet Electronic Mail Implementations consist of two "agent" types:

Message Transfer Agent (MTA): Programs that send and receive mail messages with other MTA's on behalf of MUA's. Microsoft Exchange Server, Postfix and Sendmail are all MTAs.

Mail User Agent (MUA): Electronic Mail programs are used to construct electronic mail messages and communicate with an MTA to send/retrieve mail messages. Microsoft Outlook, Eudora and Evolution are all MUAs.

MTA's often serve as "mail hubs" and can typically service hundreds or more MUA's.

MUA's are responsible for constructing electronic mail messages in accordance with the Internet Electronic Mail Specifications identified above. This section describes the "binding" of an ebXML compliant message for transport via eMail from the perspective of a MUA. No attempt is made to define the binding of an ebXML Message exchange over SMTP from the standpoint of a MTA.

Minimum Level of Supported Protocols

- Simple Mail Transfer Protocol [RFC 2821]
- MIME [RFC2045] and [RFC 2046]

- Multipart/Related MIME [RFC 2387]

Sending ebXML Service Messages over SMTP

Prior to sending messages over SMTP an ebXML Message MUST be formatted according to the ebXML Message Service Specification. Additionally the messages must also conform to the syntax, format and encoding rules specified by MIME [RFC2045], [RFC2046] and [RFC2387].

Many types of data that a party might desire to transport via email are represented as 8bit characters or binary data. Such data cannot be transmitted over SMTP [RFC2821], which restricts mail messages to 7bit US-ASCII data with lines no longer than 1000 characters including any trailing CRLF line separator. If a sending Message Service Handler knows that a receiving MTA, or ANY intermediary MTA's, are restricted to handling 7-bit data then any document part that uses 8 bit (or binary) representation must be "transformed" according to the encoding rules specified in section 6 of MIME [RFC2045]. In cases where a Message Service Handler knows that a receiving MTA and ALL intermediary MTA's are capable of handling 8-bit data then no transformation is needed on any part of the ebXML Message.

The rules for forming an ebXML Message for transport via SMTP are as follows:

- If using SMTP [RFC2821] restricted transport paths, apply transfer encoding to all 8-bit data that will be transported in an ebXML message, according to the encoding rules defined in section 6 of MIME [RFC2045]. The Content-Transfer-Encoding MIME header MUST be included in the MIME envelope portion of any body part that has been transformed (encoded).
- The Content-Type MIME header with the associated parameters, from the ebXML Message Envelope MUST appear as an eMail MIME header.
- All other MIME headers that constitute the ebXML Message Envelope MUST also become part of the eMail MIME header.
- The SOAPAction MIME header field must also be included in the eMail MIME header and MAY have the value of ebXML:

  SOAPAction: "ebXML"
- The "MIME-Version: 1.0" header must appear as an eMail MIME header.
- The eMail header "To:" MUST contain the SMTP [RFC2821] compliant eMail address of the ebXML Message Service Handler.
- The eMail header "From:" MUST contain the SMTP [RFC2821] compliant eMail address of the senders ebXML Message Service Handler.
- Construct a "Date:" eMail header in accordance with SMTP [RFC2821]
- Other headers MAY occur within the eMail message header in accordance with SMTP [RFC2821] and MIME [RFC2045], however ebXML Message Service Handlers MAY choose to ignore them.

The example below shows a minimal example of an eMail message containing an ebXML Message:

```
From: ebXMLhandler@example.com
To: ebXMLhandler@example2.com
Date: Thu, 08 Feb 2001 19:32:11 CST
MIME-Version: 1.0
SOAPAction: "ebXML"
Content-type: multipart/related; boundary="BoundarY"; type="text/xml";
        start="<ebxhmheader111@example.com>"

     This is an ebXML SMTP Example

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

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

<SOAP:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
  http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
 <SOAP:Header>
  <eb:MessageHeader eb:id="…" eb:version="3.0" SOAP:mustUnderstand="1">
   <eb:From>
    <eb:PartyId>uri:example.com</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
   </eb:From>
   <eb:To>
    <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
    <eb:Role>http://rosettanet.org/roles/Seller</eb:Role>
   </eb:To>
   <eb:CollaborationInfo>
    <eb:AgreementRef>http://www.oasis-open.org/cpa/123456</eb:AgreementRef>
    <eb:Service eb:type="myservicetypes">QuoteToCollect</eb:Service>
    <eb:Action>NewPurchaseOrder</eb:Action>
   </eb:CollaborationInfo>
   <eb:MessageInfo>
    <eb:MessageId>UUID-2@example.com</eb:MessageId>
    <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
    <eb:ConversationId>987654321</eb:ConversationId>
    <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp>
   </eb:MessageInfo>
   <eb:PayloadInfo>
    <eb:Payload eb:id="…" eb:payloadRef="cid:ebxmlpayload111@example.com">
     <eb:Schema eb:location="http://foo/bar.xsd" eb:version="1.0"/>
     <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
     <eb:Processing>
     <eb:Step eb:sequence="0" eb:id="urn:foo:ps:CompressionSvc">
      <eb:Parameter eb:name="command" eb:value="uncompress" />
      <eb:Parameter eb:name="algorithm" eb:value="gzip" />
     </eb:Step>
     </eb:PostProcessing>
    <eb:Payload>
   </eb:PayloadInfo>
  </eb:MessageHeader>
 </SOAP:Header>
 <SOAP:Body/>
</SOAP:Envelope>
```

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

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

--BoundarY--
```

Response Messages

All ebXML response messages, including errors and acknowledgments, are delivered asynchronously between ebXML Message Service Handlers.

All ebXML Message Service Handlers MUST be capable of receiving a delivery failure notification message sent by an MTA. A MSH that receives a delivery failure notification message SHOULD examine the message to determine which ebXML message, sent by the MSH, resulted in a message delivery failure. The MSH SHOULD attempt to identify the application responsible for sending the offending message causing the failure. The MSH SHOULD attempt to notify the application that a message delivery failure has occurred. If the MSH is unable to determine the source of the offending message the MSH administrator should be notified.

MSH's which cannot identify a received message as a valid ebXML message or a message delivery failure SHOULD retain the unidentified message in a "dead letter" folder.

A MSH SHOULD place an entry in an audit log indicating the disposition of each received message.

Access Control

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

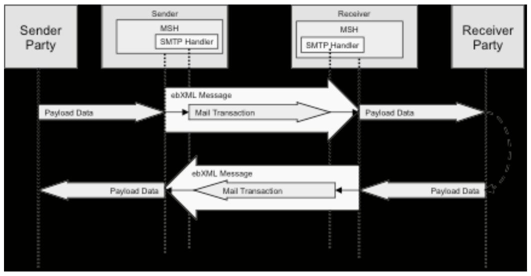
Confidentiality and Transport Protocol Level Security

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

SMTP Model

All ebXML Message Service messages carried as mail in an SMTP [RFC2821] Mail Transaction:

Figure B.1. SMTP Model



Communication Errors during Reliable Messaging

When the Sender or the Receiver detects a communications protocol level error (such as an HTTP, SMTP or FTP error) and Reliable Messaging is being used then the appropriate transport recovery handler will execute a recovery sequence. Only if the error is unrecoverable, does Reliable Messaging recovery take place. *Jeff: Please make sure this jives with new RM.*

FTP

This section defines the File Transfer Protocol binding for ebXML Messaging.

Minimum Level of Supported Protocols

Implementations of the ebXML Messaging FTP transport binding MUST conform with [RFC 949].

Sending ebXML Service Messages over FTP

Message transmission via FTP is accomplished by having the sender connect to the recipient's FTP server, and uploading the message contents into a file located in the FTP server's root directory, named thusly: `<message id>.ebms`.

Messages MAY be placed in a directory other than the root directory of the FTP filesystem depending on values defined in the CPA. *TODO: Dale, how's this work?*

The example below illustrates a probably command sequence for transferring an ebXML Message using FTP.

Example B.2. Sample FTP Session

```
Connected to ftp.partner.com.
220 ebMS-FTP Server (ftp.partner.com FTP) [ftp]
Name (ftp.partner.com:anonymous): anonymous
331 Password required for anonymous.
Password:
230 User anonymous logged in.
Remote system type is UNIX.
```

```
Using binary mode to transfer files.
ftp> cdup
250 CDUP command successful.
ftp> put message@id.ebms
local: message@id.ebms remote: message@id.ebms
502 Command not implemented.
227 Entering Passive Mode (10,36,3,5,67,144)
150 File status okay; about to open data connection.
100% |*************************************************************************************************************|    10       7.28 KB/s    00:00 ETA
226 Transfer complete, closing data connection.
10 bytes sent in 00:00 (0.13 KB/s)
ftp> bye
221 Service closing control connection.
```

Response Messages

Since FTP is designed for one way file transfer sessions, all response and error messages will be returned asynchronously via the transport that is configured in the CPA. Transmission and FTP protocol level errors will, however, be handled as specified in [RFC 959].

Access Control Considerations

If the CPA defines access control settings (username and password), that information MUST be used to perform the login operation at the start of the FTP session. Otherwise, username anonymous should be used with the password set as the same value as the From header field of the ebXML Service Message.

Confidentiality and Transport Protocol Level Security.

Security extensions to FTP such as those defined by RFC 2228 (Security Extensions for FTP) have not been widely adopted by vendors of FTP software. To achieve confidentiality during message transmission, it is recommended that security be enforced at a higher level, possibly via a VPN connection or SSL tunnel. These approaches to confidentiality can be setup so as to be completely transparent to the message service handler.

C. Supported Security Services

D. Relationship to WSDL

E. WS-I Compliance

F. Notices

G. Revision History

Revision History		
Revision 01	5 May 2004	mm
Create outline, document structure, added payload services.		
Revision 01-1	14 May 2004	mm
Moved content over from 2.0/2.1 document source.		
Revision 02	1 Oct 2004	mm
Integrated Reliable messaging, many editorial changes also.		

References

Normative

NOTE: Most of these references are not correct, and are just placeholders.

[ebCPPA] OASIS ebXML CPP/A TC.*DNS-SD: OASIS ebXML Collaboration-Protocol Profile and Agreement Specification* . OASIS Open. 2002.

[RFC 2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF (Internet Engineering Task Force). 1997.

[RFC 2045] N Freed, N Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. IETF (Internet Engineering Task Force). 1996.

[RFC 2046] N Freed, N Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. IETF (Internet Engineering Task Force). 1996.

[RFC 2387] E. Levinson. *The MIME Multipart/Related Content-type*. IETF (Internet Engineering Task Force). 1998.

[XMLMedia] A. Person. *XML Media Types*. SomeConsortia. 1998.

[SOAP w/ Attachments] John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen. *SOAP Messages with Attachments*. W3C. 2000.

[SOAP] W3C SOAP Work Group *SOAP [TODO]*. W3C. 2002.

[XML] W3C XML Work Group *XML [TODO]*. W3C. 2002.

[XMLNS] W3C XML Work Group *XMLNS [TODO]*. W3C. 2002.

[XMLDSIG] W3C DSIG Work Group *XMLDSIG [TODO]*. W3C. 2002.

[XMLSchema] W3C XML Schema Work Group *XMLSchema [TODO]*. W3C. 2002.

[BPSS] BPSS Work Group *BPSS [TODO]*. W3C. 2002.

[XPath] W3C XPath Work Group *XMLNS [TODO]*. W3C. 2002.

[RFC 2396] 2396 Work Group *XMLNS [TODO]*. W3C. 2002.

[RFC 2246] W3C XML Work Group *2246 [TODO]*. W3C. 2002.

[XMLC14N] W3C XML Canonicalization Work Group *XMLNS [TODO]*. W3C. 2002.

[RFC 2402] W3C XML Work Group *2402 [TODO]*. W3C. 2002.

[SMIME] SMIME Work Group *SMIME [TODO]*. W3C. 2002.

[SMIMEV3] SMIME Work Group *SMIMEV3 [TODO]*. W3C. 2002.

[PGPMIME] SMIME Work Group *PGPMIME [TODO]*. W3C. 2002.

[SSL3] SSL3 Work Group *SSL [TODO]*. W3C. 2002.

[RFC 2821] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 2554] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 2487] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 949] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 959] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 2616] rfc Work Group *rfc [TODO]*. W3C. 2002.

[RFC 2617] rfc Work Group *rfc [TODO]*. W3C. 2002.

[XPointer] xpointer Work Group *xpointer [TODO]*. W3C. 2002.