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.

Status:
This is a Working Draft, meaning that the TC has not necessarily reached consensus on any or all content, and all contents are subject to change.

This document was last revised or approved by the TC on the above date. The level of approval is also listed above. Check the current location noted above for possible
later revisions of this document. This document is updated periodically on no
particular schedule.

Technical Committee members should send comments on this specification to the
ebxml-msg@lists.oasis-open.org list. Others should use the comment form at

For information on whether any patents have been disclosed that may be essential to
implementing this specification, and any offers of patent licensing terms, please refer to the
Intellectual Property Rights section of the OASIS ebXML Messaging Services TC web page
<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
</tr>
<tr>
<td>52</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>55</td>
</tr>
<tr>
<td>56</td>
</tr>
<tr>
<td>57</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>59</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>61</td>
</tr>
<tr>
<td>62</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>65</td>
</tr>
<tr>
<td>66</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>69</td>
</tr>
<tr>
<td>70</td>
</tr>
<tr>
<td>71</td>
</tr>
<tr>
<td>72</td>
</tr>
<tr>
<td>73</td>
</tr>
<tr>
<td>74</td>
</tr>
<tr>
<td>75</td>
</tr>
<tr>
<td>76</td>
</tr>
<tr>
<td>77</td>
</tr>
<tr>
<td>78</td>
</tr>
<tr>
<td>79</td>
</tr>
<tr>
<td>80</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>83</td>
</tr>
</tbody>
</table>
5.2.2 Element eb:Messaging/eb:SignalMessage .............................................. 46
5.2.2.1 Element eb:Messaging/eb:SignalMessage/eb:PullRequest .................. 46
5.2.2.2 Error Message Packaging ................................................................. 46

6 Security Module ..................................................................................... 47
6.1 Security Element ................................................................................. 47
6.2 Securing the PullRequest Signal ........................................................ 47
6.2.1 Authentication .................................................................................. 47
6.2.1.1 Username / Password Option ...................................................... 48
6.2.1.2 BinaryToken Option ..................................................................... 48
6.2.2 Authorization .................................................................................... 49
6.2.3 Preventing Repeat Attacks ............................................................... 49

6.3 Countermeasure Technologies ............................................................ 49
6.3.1 Persistent Digital Signature ............................................................... 49
6.3.2 Persistent Signed Receipt .................................................................. 50
6.3.3 Non-Persistent Authentication ........................................................ 50
6.3.4 Non-persistent Integrity .................................................................... 50
6.3.5 Persistent Confidentiality ................................................................... 50
6.3.6 Non-persistent Confidentiality .......................................................... 50
6.3.7 Persistent Authorization ..................................................................... 50
6.3.8 Non-persistent Authorization ............................................................ 50

6.4 Security Considerations ....................................................................... 50

7 Error Handling Module .......................................................................... 52
7.1 Terminology ........................................................................................ 52
7.2 Packaging of ebMS Errors ................................................................... 52
7.2.1 eb:Error Element .............................................................................. 52
7.2.2 Attribute: eb:Error/@origin .............................................................. 53
7.2.3 Attribute: eb:Error/@category .......................................................... 53
7.2.4 Attribute: eb:Error/@errorCode ....................................................... 53
7.2.5 Attribute: eb:Error/@severity ........................................................... 53
7.2.6 Attribute: eb:Error/shortDescription .............................................. 53
7.2.7 Element: eb:Error/Description ........................................................ 53
7.2.8 Element: eb:Error/ErrorDetail ....................................................... 54
7.3 ebMS Error Message ......................................................................... 54
7.4 Extensibility of the Error Module ....................................................... 54
7.4.1 Adding new ebMS Errors ................................................................. 54
7.5 Generating ebMS Errors .................................................................... 55
194
195

Deleted: OASIS ebXML Messaging Services. 1
Version 3.0: Part 1, Core Features 1
1 Introduction 10
1.1 Background and Objectives 10
1.2 Scope 10
1.3 Caveats and Assumptions 11
1.4 General Rules for Normative Interpretation 11
1.5 Notation 12
2 Messaging Model 13
2.1 Terminology and Concepts 13
2.1.1 Components of the Model 13
2.1.2 Message Types 14
2.1.3 Messaging Roles 14
2.1.4 Abstract Messaging Operations 15
2.2 Message Exchange Patterns 15
2.2.1 Definition 15
2.2.2 Assumed SOAP Message Exchange Patterns 16
2.2.2.1 SOAP One-Way MEP 16
2.2.2.2 SOAP Request-Response MEP 16
2.2.3 Simple ebMS Message Exchange Patterns 16
2.2.3.1 The One-Way Push Message Exchange Pattern 17
2.2.3.2 The One-Way Pull Message Exchange Pattern 17
2.2.3.3 The Request-Reply Message Exchange Pattern 18
2.3 Message Pipes 19
2.3.1 Concept and Purpose 19
2.3.2 Definition and Usage Requirements 20
2.4 Transfer Modes 21
2.4.1 Definitions 21
2.4.2. Transfer Modes and MEPs 21
2.4.3 The Pulled Transfer Mode 21
3 Concept of Operation 23
3.1 Operation Mode and Contexts 23
3.2 Default Operation Mode 24
3.3 Message Service Processing Model 25
3.4 Examples of Supported Topologies 26
4 Message Pulling Module 27
4.1 Objectives 27
4.2 Supporting the Pull Mode 28
4.3 Combining Pulling with Security and Reliability 30
5 Message Packaging 32
5.1 Message Envelope and Message Parts 32
5.1.1 MIME Structure and...
## Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Entities of the Messaging Model and their Interaction</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>One-Way Push MEP</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>One-Way Pull MEP</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Request-Reply MEP</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Component Relationships</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>One-Way Pull with Message Pipes</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>User Message Structure</td>
<td>29</td>
</tr>
<tr>
<td>8</td>
<td>Signal Message Structure</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>Reliable Request</td>
<td>68</td>
</tr>
<tr>
<td>10</td>
<td>Reliable Response</td>
<td>69</td>
</tr>
<tr>
<td>11</td>
<td>Reliable One-Way Push MEP</td>
<td>72</td>
</tr>
<tr>
<td>12</td>
<td>Reliable One-Way Pull MEP</td>
<td>73</td>
</tr>
<tr>
<td>13</td>
<td>Reliable Request-Reply MEP</td>
<td>74</td>
</tr>
<tr>
<td>14</td>
<td>SMTP Model</td>
<td>87</td>
</tr>
</tbody>
</table>
1 Introduction

This specification describes a communication-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 that 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.

1.1 Background and Objectives

The prime objective of ebMS is to facilitate the exchange of electronic business messages within an XML framework that leverages common Internet standards, without making any assumption on the integration and consumption model these messages will follow on the back-end. These messages may be consumed in different ways that are out of scope of this specification: they may bind to a legacy application, to a service, be queued, enter a message workflow process, be expected by an already-running business process, be batched for delayed processing, be routed over an Enterprise Service Bus before reaching their consumer application, or be dispatched based on header data or payload data, etc.

It is becoming critical for broad adoption among all partners – large or small - of a supply-chain, to handle differences in message flow capacity, intermittent connectivity, and lack of static IP addresses or firewall restrictions. Such new capabilities played an important role in the motivation that led to ebMS 3.0, along with the need to integrate and profile the emerging SOAP-based QoS-supporting standards. The message header profiling that provided, in ebMS 2.0, a standard business-level header, has also been extended to better address the diversity of back-end binding models, as well as the emerging trend in business activity monitoring, the eBusiness side of which a message handler should be able to support.

Finally, ebMS 3.0 gives enhanced support for various roles required in diverse networking topologies such as acting as intermediary - either at SOAP or ebMS level - and experiencing restrictions in initiating a message transfer.

The ebXML messaging framework is not a restrictive one: business messages, identified as the ‘payloads’ of ebXML messages, are not limited to XML documents. Traditional EDI formats may also be transported by ebMS. These payloads 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 transfer protocol. This version of the specification provides bindings to HTTP and SMTP in Part 1, Core Features, and FTP in a forthcoming Part 2, Advanced Messaging Features, but other protocols to which SOAP may bind can be used. The choice of an XML framework rather reflects confidence in a growing XML-based Web infrastructure and development tools infrastructure, the components of which can be leveraged and reused by developers.

1.2 Scope

The ebXML infrastructure is composed of several independent, but related, components. Specifications for the individual components are fashioned as stand-alone documents. Each
specification is self-contained, meaning a conforming implementation may ignore other
ebXML specifications. Some references and bindings across ebXML specifications should be
interpreted as integration help, not requirement to integrate. This applies to ebMS also,
which may refer in particular to CPPA specification, though does not require its use: ebMS
relies on a concept of "Agreement" the concrete representation of which (e.g. CPA or other
configuration information) is left for implementers to decide.

The ebXML Message Service (ebMS) defines messaging functions, protocol and envelope
intended to operate over SOAP ([SOAP11] and [SOAPATTACH]). Binding to lower transport
layers such as HTTP and SMTP relies on standard SOAP bindings when these exist, and
ebMS only specifies some complement to these, as required.

This version of ebMS leverages established SOAP-based specifications that handle quality of
service in the domains of reliability and security. The ebMS specification defines how these
are composed in the ebMS context. The design of this composition takes into account the
reuse of existing implementations of these standards, not just the reuse of these standards
themselves.

The concept for an ebMS implementation is of an ebXML Message Service Handler (MSH),
that is abstractly defined as implementing the specified messaging functions. Any interface
to the MSH is out of scope of this specification. Although it is clearly helpful in many cases
to define a standard API, such an interface should not exclude other ways applications may
want to interact with an MSH. Such an interface definition will rather belong to an
implementation guideline companion document. An implementation of this specification
could be delivered as a wholly independent software component or as an embedded
component of a larger system.

Features defined in Part 1 (this document) only support the point-to-point MSH topology,
where no MSH or SOAP intermediary is assumed. Part 2 takes into account topologies with
intermediaries, hub or multi-hop, as well as topologies where the ultimate MSH acts as a
SOAP intermediary.

1.3 Caveats and Assumptions

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

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.

Implementers are strongly advised to read and understand the Collaboration Protocol Profile
& Agreement [ebCPPA] specification and its implications prior to implementation.

1.4 General Rules for Normative Interpretation

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
[ RFC2119].

For any given module described in this specification, an implementation MUST satisfy ALL of
the following conditions to be considered a conforming implementation of that module:
It supports all the mandatory syntax, features and behavior (as identified by the [RFC2119] key words MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT) defined in the section that specifies that module.

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 [RFC2119]. 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.

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.

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.

1.5 Notation

EdNote: Explain use of XPATH to refer to specific XML elements and attributes. (See WSS lines 173-177.)

Also, table of namespaces used in examples would be useful.
2 Messaging Model

2.1 Terminology and Concepts

This section defines the messaging model and its main concepts, along with the related terminology in use throughout the specification.

2.1.1 Components of the Model

The ebMS messaging model assumes the following components:

ebMS MSH (Message Service Handler): An entity able to generate or process messages that conform to this specification and to act in at least one of two ebMS roles defined below: Sending and Receiving. In terms of SOAP processing, an MSH is either a SOAP processor [SOAP11] or a chain of SOAP processors. In either case, an MSH must be able to understand headers intended for the "ebms" actor.

Producer (or Message Producer): An entity that interacts with a Sending MSH (i.e. an MSH in Sending role) to initiate the sending of a user message. Some examples are: an application, a queuing system, another SOAP processor (though not another MSH).

Consumer (or Message Consumer): An entity that interacts with a Receiving MSH (i.e. an MSH in Receiving role) to consume data from a received message. Some examples are: an application, a queuing system, another SOAP processor.

Figure 1 shows the entities and operations involved in a message exchange.

Note:

In all figures, the arrows do not represent a control flow, i.e. do not represent a component invoking an operation on another component. They only represent data transfer under the control of an operation which may be implemented in either component.
2.1.2 Message Types

An **ebMS Message** is a message that contains SOAP header(s) qualified with the ebMS namespace, and that conforms to this specification. An ebMS MSH component must be able to exchange the following types of ebMS Messages:

**ebMS Signal Message**: An ebMS message, the role of which is to activate a specific function in the Receiving MSH, and not to carry data to be delivered to a Consumer entity – i.e. Not subject to Deliver operation. An example is the ebMS PullRequest signal. A signal message is characterized by the presence of an `eb:SignalMessage` element in the ebMS header (see Section 5.2.2).

**ebMS User Message**: An ebMS message that is initiated by a Producer entity (via Submit operation), and intended for a Consumer (via Deliver operation). A user message is characterized by the presence of an `eb:UserMessage` element in the ebMS header (see Section 5.2.1).

An **ebMS Message Unit** is a logical unit of data that is a subset of an ebMS Message. There are two kinds of Message Units:

- an **ebMS User Message Unit**, which is represented by the XML infoset `eb:Messaging/eb:UserMessage`, together with referenced payload items, and
- an **ebMS Signal Message Unit**, represented by the XML infoset `eb:Messaging/eb:SignalMessage`.

2.1.3 Messaging Roles

The Messaging Model assumes the following roles for an MSH:

**Sending**: An MSH acts in Sending role when performing functions associated with generating an ebMS user message and sending this message to another MSH. The abstract
operations Submit, Send and Notify are supported by this role. (Note that in a Sending role, an MSH could also have to receive and process error messages associated with previously sent messages.)

**Receiving**: An MSH acts in Receiving role when performing functions associated with the receiving and processing of an ebMS user message. The abstract operations Receive, Deliver and Notify are supported by this role. (Note that in a Receiving role, an MSH could also have to send ebMS signal messages related to the reception of messages, such as error messages or.PullRequest signals.)

The transmission of an ebMS user message requires a pair of Sending and Receiving MSHs. Note that these roles are defined as only relevant to ebMS user messages, as are the abstract operations below.

### 2.1.4 Abstract Messaging Operations

An ebMS MSH supports the following abstract operations, depending on which role it is operating in:

- **Submit**: This operation transfers enough message data from the producer to the Sending MSH to generate an ebMS User Message Unit.

- **Deliver**: This operation makes data of a previously received (via Receive operation) ebMS User Message Unit available to the Consumer.

- **Notify**: This operation notifies either a Producer or a Consumer about the status of a previously submitted or received ebMS User Message Unit, or about general MSH status.

- **Send**: This operation initiates the transfer of an ebMS user message from the Sending MSH to the Receiving MSH, after all headers intended for the ebMS SOAP actor have been added (including security and/or reliability, as required).

- **Receive**: This operation completes the transfer of an ebMS user message from the Sending MSH to the Receiving MSH. A successful reception means that a contained User Message Unit is now available for further processing by the Receiving MSH.

### 2.2 Message Exchange Patterns

This section introduces the notion of an ebMS Message Exchange Pattern (MEP), and how it relates to SOAP MEPs. Such ebMS MEPs represent atomic units of choreography, i.e. different styles of exchange as required by connectivity constraints or application requirements.

### 2.2.1 Definition

An ebMS Message Exchange Pattern (MEP) is an abstract description of a typical sequence of ebMS message exchanges that may occur between two or more MSH instances. An MEP instance is a message exchange that conforms to the pattern described by the MEP. To be more precise, it is defined as an exchange of ebMS Message Units.

When more than one ebMS message unit is exchanged in the same MEP instance, there must be some explicit referencing between them in the ebMS header. In other words, for every message unit exchanged in the same MEP instance, either one of these statements is true:
the ebMS message unit is the first one to occur in the MEP instance.
the ebMS message unit is referring to the ID of a previously sent message unit (and only
one), in the same MEP instance.
This referencing is done using the eb:RefToMessageId element. The MSH sending the first
ebMS message of an MEP instance is called the initiator MSH. The other MSH is called
responding MSH.

2.2.2 Assumed SOAP Message Exchange Patterns

Each ebMS MEP is also defined in terms of what SOAP MEP(s) it is using, and how.

Note that the concept of SOAP MEP has only been introduced with SOAP 1.2 ([SOAP12]),
although this concept can apply as well to SOAP 1.1. This section is only concerned with
some assumed properties of these SOAP MEPs, not with their precise definitions.
Specific bindings of these MEPs to underlying protocols are addressed in a later section. It is
only assumed that underlying protocols are of two kinds: either one-way, or two-way, with
an implicit back-channel for returning messages in the latter case.
The SOAP MEPs that support the ebMS MEPs described in this specification are described in
the next two subsections.

2.2.2.1 SOAP One-Way MEP

No formal definition is available for this MEP at the time this specification is written. This
specification is only concerned here with identifying a set of properties for such an MEP, that
many SOAP implementations exhibit when sending a message that is not expecting a
response message. From an MSH perspective, support for this MEP assumes the following:
The Initiator MSH is able to initiate the sending of a SOAP envelope (an ebMS message)
over the underlying protocol (i.e., not as a result of a previous protocol action such as an HTTP GET or POST).

The response message, in case a two-way underlying protocol is used:
• (a) may contain a SOAP envelope with a SOAP body that is always empty, in
which case the One-way MEP is qualified as "composable".
• (b) may contain a SOAP envelope with a SOAP body that is either empty or
contains a SOAP Fault. In that case the MEP is qualified as "fault supporting" in
addition to being composable.
• (c) may never contain a SOAP envelope, in which case the SOAP One-way MEP is
qualified as "strict".

2.2.2.2 SOAP Request-Response MEP

The full definition of this MEP can be found in [SOAP12] Part 1, Adjunct. From an MSH
perspective, support for this MEP assumes the following:
The Sending MSH 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 MSH 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.)
2.2.3 Simple ebMS Message Exchange Patterns

This section describes the three basic ebMS MEPs involving user messages.

An ebMS MEP is of two types:
- Simple ebMS MEP: if it executes over a single SOAP MEP instance.
- Aggregate ebMS MEP: if it executes over more than one SOAP MEP instance.

The following sub-sections define three simple ebMS MEPs: One-Way Push, One-Way Pull, and Request-Reply.

2.2.3.1 The One-Way Push Message Exchange Pattern

This MEP involves a single ebMS user message. The message sending is initiated by a Sending MSH either:
- as a SOAP One-way MEP instance,
- as a SOAP Request in a SOAP Request-Response MEP instance, or
- as a SOAP Response in a SOAP Request-response MEP instance.

To conform to this MEP, the user message MUST NOT relate to any other user message (no eb:RefToMessageId element), and MUST NOT be referred to by any subsequent user message. Figure 2 illustrates the exchange pattern and operations involved for this MEP.

One-Way Push MEP

Initiator MSH

| User message |
| Send |

Responding MSH

| SOAP One-way or Request-response MEP |
| Receive |

Role: sending

Role: receiving

Figure 2: One-Way Push MEP

Deleted: The eb:soapresp attribute MUST either be set to "false", or not be used.
2.2.3.2 The One-Way Pull Message Exchange Pattern

This MEP involves a single ebMS user message. In this MEP the message sending is initiated by the Receiving MSH, over a SOAP Request-response MEP instance. The first leg of the SOAP MEP (Request) sends a PullRequest signal message unit. The second leg of the MEP returns the pulled user message unit as a SOAP Response. To conform to this MEP the user message MUST relate to the PullRequest message unit with eb:RefToMessageId, and MUST NOT be referred to by any subsequent user message. Also, The MessageInfo element in the PullRequest signal MUST NOT include eb:RefToMessageId element. In case no message is available for pulling, a SOAP Response with an ebMS error signal of severity level "warning" and error code "EmptyMBox", as listed in Section 7.7.1, SHALL be returned. Figure 3 illustrates the exchange pattern and operations involved for this MEP.

One-Way Pull MEP

![Figure 3: One-Way Pull MEP](image)

2.2.3.3 The Request-Reply Message Exchange Pattern

This MEP involves two ebMS user messages over a single SOAP Request-Response MEP instance. It is initiated by the Sending MSH. In the first leg of the MEP, an ebMS user message unit called the "ebMS request" is sent over the SOAP Request message. In the second leg of the MEP, a related user message unit called the "ebMS reply" is sent as the SOAP Response. To conform to this MEP, the ebMS request MUST NOT relate to any other user message (no eb:RefToMessageId element), and the ebMS reply MUST refer to the ebMS request via the eb:ReferenceId header element, as described in Section 5.2.1.1). Figure 4 illustrates the exchange pattern and operations involved for this MEP.

![Figure 4: Request-Reply MEP](image)
2.3 Message Boxes

2.3.1 Concept and Purpose

A message box (mbox for short) defines a logical concept for labeling messages between Sending MSH and Receiving MSH. Several message boxes MAY be supported by a pair of Sending and Receiving MSHs. One mbox, called the “default” mbox, is always supported between a pair of communicating MSHs. If no additional mbox is explicitly defined between a Sending MSH and a Receiving MSH, all messages use the default mbox.

Intuitively, an mbox is intended to group and transfer messages independently from another mbox, i.e. the order and manner in which a message is transferred over SOAP is only determined within the scope of an mbox. The order in which two messages put in two different mboxes are transferred, is under control of the MSH, based on its context of operation or based on application input. This allows for a better control of when and in which order messages are transferred between two MSHs.

Support for assigning messages to mboxes – e.g. by automatically mapping messages submitted to or retrieved from an mbox based on some configuration and filtering - is out of scope of this specification. This specification only describes the concept of mboxes, and how the use of mboxes affects the message protocol. It does not prescribe a particular way to implement mboxes or to use them.
2.3.2 Definition and Usage Requirements

An mbox may represent a real or virtual box. Mboxes are used mainly for partitioning messages. For example, a receiving party may wish to have a separate mbox for invoices and another mbox for purchase order messages. The way mboxes are profiles and used in the implementation is out of scope of this specification. Mboxes are also heavily used by PullRequest messages. A PullRequest signal specifies via its attribute eb:PullRequest/@forMbox the mbox from which to pull User Messages. Only User Message can be mbox-labeled (A User Message is mbox-labeled by adding the attribute eb:mbox to the eb:UserMessage element). In the absence of the eb:UserMessage/@eb:mbox attribute, the default mbox (with value http://www.oasis-open.org/committee/ebxml-msg/defaultMbox) is assumed.

Mboxes, although specified within P-Mode definitions, are orthogonal to P-Modes. In other words, two different P-Modes may share the same mbox. For example, MSH A may be pushing User Messages to MSH C that are intended for mbox b, while MSH C may be at the same time pulling messages from MSH B intended for mbox b (that is mbox b is being used here to group together the UserMessages pushed by MSH A and those UserMessages pulled from MSH B).

A message submitted to a Sending MSH (Submit) MUST be associated with an mbox before it is sent. Whether this association is done at Submit time or after Submit, is implementation specific. Therefore at the "Send" operation, the MSH knows about two things: (a) which P-Mode to apply to the message-to-be-sent, and (b) which mbox the message is intended to.

Notes:

• When using reliable messaging, if the message ordering reliability contract is requested, all messages intended to the same ordered sequence between two MSHs MUST use the same mbox. For message ordering to be supported, messages assigned to an mbox MUST be sent in the same order they have been submitted to the Sending MSH.

2.4 Transfer Modes

2.4.1 Definitions

There are two modes of transfer for ebMS messages. The transfer modes are:

Pushed transfer mode (Push Mode). In this mode, messages are transferred to the Receiving MSH on the initiative of the Sending MSH itself. This mode is the default operating mode.

Pulled transfer mode (Pull Mode). In this mode, messages are transferred to the Receiving MSH as responses to PullRequest signals from the Receiving MSH that are targeted to a given mbox. A PullRequest specifies the mbox to use via its attribute eb:PullRequest/@forMbox, and the UserMessage being pulled MUST be mbox-labeled by having the attribute eb:UserMessage/@mbox with a value equal to the value of...
Note that neither transfer mode requires that both endpoints be associated with a fixed protocol address. For example, Pull Mode over HTTP only requires that the Sending MSH address be known.

### 2.4.2 Transfer Modes and MEPs

**The** Push Mode supports the transfer of user messages within the following ebMS MEPs:

- One-way Push MEP.
- Request-response MEP, first leg (request).
- Request-response MEP, second leg (response).

**The** Pull Mode supports the transfer of user messages within the following ebMS MEP:

- One-way Pull MEP, second leg (response).

### 2.4.3 The Pulled Transfer Mode

The Pulled transfer mode (or Pull Mode) is used in the One-Way Pull MEP. It allows a Receiving MSH to initiate the transfer of messages that have been previously submitted to a Sending MSH, but not sent yet. This could typically be achieved by a queuing capability at the Sending MSH, although this is an implementation decision. The Pull Mode is intended to better handle the following situations:

- Connectivity restrictions for the Receiving MSH, including such cases as: unavailability over significant periods of time, limited accessibility due to firewall restrictions or lack of static IP address.
- The Receiving MSH may wish to control when to receive messages and how many at a time. It may also wish to control which types of messages it wants to receive first.

For a secure transfer using the Pull Mode (see Security section), the **P-Modes** that may be used in Pull Mode between a Sending and Receiving MSH MUST be known prior to exchanging messages. This is done by sharing the **P-Modes**. For example, the list of **mbox** names (URIs) may be specified in an agreement such as a CPA. A PullRequest signal MUST specify the **mbox** it is pulling from, even in case of a default **mbox**.

### 2.5 Processing Model

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 5 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 5: Component Relationships**

- **Application or SOAP Processor** - This is where the business logic for a message exchange / business process exists.
- **Message Service Interface** - This is the interface through which messages are channeled between the MSH core and the the ebXML Application.
- **ebMS Packaging** - Handling, (de)enveloping and execution of Payload Services are performed by this module.
- **Reliability Module**
- **WSS Module**
- **Transport binding or Other SOAP Processor**

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.

- **Application or SOAP Processor** - This is where the business logic for a message exchange / business process exists.
- **Message Service Interface** - This is the interface through which messages are channeled between the MSH core and the the ebXML Application.
- **ebMS Packaging** - Handling, (de)enveloping and execution of Payload Services are performed by this module.
- **Reliable Message Processing** - This module fulfills the Quality of Service requirements for a message.
- **Web Services Security Processing** - Encryption/decryption of any SOAP message content and generation/verification of any digital signatures occurs in this module.
- **Transport Bindings** - These are the actual transport bindings. This specification defines bindings for HTTP, SMTP (Appendix SOAP Formats and Bindings), and supports the addition of other protocols.
3 Processing Modes

An MSH is operating – either for sending or receiving message - in knowledge of some contextual information that controls the way messages are processed. This contextual information is called Processing Mode (or P-Mode).

A Processing Mode represents some MSH input data that typically is not provided on a per-message basis, but that is common to a set of messages exchanged between two parties or more. To this extent, the P-Mode may be interpreted as configuration data for a deployed MSH. On a Sending MSH, together with the information provided by the application layer for each submitted message, the P-Mode fully determines the content of the message header. For example, the security part of a P-Mode will specify different certificates and keys as well as which messages will be subject to these. This in turn will determine the content of the Security header.

A Processing Mode may distinguish different ways to treat messages based on differentiating factors such as the destination service, the conversation Id, or other message properties. Which messages sent by an MSH are subject to (which) security and/or to (which) reliability, as well as which MEP is being used for this message, is determined by the P-Mode of the MSH.

Although a standard representation of P-Mode data is out of scope of this specification, an abstract definition of it helps to capture the contractual aspect of messaging, which in turn helps to define associated errors and the notion of default behavior. Note that a non-normative XML representation of P-Modes will be provided in a separate document called "ebMS-3 Adjuncts".

3.1 Processing Mode Features

The P-Mode is partitioned into six functional groups called P-Mode features. Each PMode feature covers one of the five functional areas that are critical to achieving interoperability between two partners: quality of service (reliability and/or security), communication channel (transport protocol, address, SOAP version and MEP), business collaboration, error reporting, and message boxes.

The six main P-Mode features are here identified by names of the kind: P-Mode <featurename>:

- **P-Mode.channel**: includes all transport related information that are necessary to achieve transport-level interoperability. This feature determines the type of transport involved (e.g., HTTP, SMTP, FTP) between two MSHs, and related configuration parameters such as addresses (email address, HTTP URL,...). This feature also includes information about which SOAP version is to be used (SOAP-1.1 or SOAP-1.2) as well as which MEP to use (One-Way Push, PullRequest, Request/Response).

- **P-Mode.reliability**: this part of the quality of service includes all reliability contracts, or references to these, that will govern the reliability of messages exchanged. This feature determines the content of the reliability headers.

- **P-Mode.security**: this part of the quality of service includes all security contracts, or references to these, including the security context and related resources (certificates, SAML assertions, etc.) that govern the message exchange. This feature determines the content of the wsse:Security header.
P-Mode.businessCollaboration: includes all message-relevant data related to a collaboration between two parties. This feature will complement or validate message data that is provided by the application on a per-message basis for these header elements:

- eb:UserMessage/eb:PartyInfo
- eb:UserMessage/eb:CollaborationInfo
- eb:UserMessage/eb:MessageProperties

P-Mode.MessageBoxes: includes the identification of all message boxes hosted by the MSH. This feature determines the possible values for the header attribute:

- eb:UserMessage/@eb:mbox. It also controls the use of the PullRequest signal via the attribute eb:PullRequest/@forMbox.

P-Mode.ErrorHandling: defines how each ebMS Error type is to be reported by this MSH. E.g. if the reporting is done using ebMS signal messages, it defines the address of the destination MSH. Also may include the policy chosen for raising ebMS Errors from the errors generated by functional modules (Reliability, Security).

In this specification a P-Mode feature is abstractly considered as applying to both sending and receiving roles, although implementations may choose to represent only the subset relevant to the role they operate in.

Agreeing on a P-Mode is essential for two parties in order for their MSHs to interoperate. P-Modes are the MSH-level expression of a prior agreement between partners. A reference to such an agreement may be present in the message header (see eb:Message/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef element in the Message Packaging section.). Although the “eb:AgreementRef” element is not always supposed to reference such a P-Mode agreement (in other words, “eb:AgreementRef” could be used in other purposes not related to P-Modes).

### 3.2 Default Features

In order to facilitate interoperability testing, or during the early phase of a deployment, it is however useful to be able to drive message exchanges without relying on user-agreed P-Modes, and without interfacing with any application. To this end, default semantics for each P-Mode feature is defined as follows. Note that these semantics cannot achieve any business exchanges between two MSHs which require a prior agreement on P-Modes. The following semantics can therefore be used only for simple exchanges such as a ping signal:

- **Default P-Mode.channel**: HTTP 1.0 transport assumed, with default configuration (on standard port). The One-Way Push MEP is assumed. SOAP-1.1 is assumed.
- **Default P-Mode.reliability**: no reliable messaging assumed (no reliability header will be present.)
- **Default P-Mode.security**: no secure messaging assumed (no security header will be present.)
- **Default P-Mode.businessCollaboration**: in the absence of any application input at message level as well as for this P-Mode feature, the following default header element will be used. Any part of these can be overide by application input.

  - The eb:To element contains a PartyId with value: http://www.oasis-open.org/committees/ebxml-msg/defaultTo.
• eb:UserMessage/eb:CollaborationInfo: Contains no eb:AgreementRef. The eb:Service element has value: http://www.oasis-open.org/committee/ebxml-msg/defaultService. The eb:Action element has value: ping. (Section 5 details the semantics of these values.) The eb:ConversationId element has value: 1.

• eb:UserMessage/eb:MessageProperties: This element is absent.

• eb:UserMessage/eb:PayloadInfo: This element is absent.

• **Default P-Mode.messageBoxes**: Only the default mbox is in use, and its value is http://www.oasis-open.org/committee/ebxml-msg/defaultMbox. The absence of an mbox value is always interpreted as having a mbox value with this default.

• **Default P-Mode.errorHandling**: No reporting via ebMS message is required. The MSH may handle error reporting in a way that does not involve the partner MSH.
4 Message Pulling Module

4.1 Objectives

Business partners may experience differences in their ability to handle message flow, intermittent connectivity, lack of static IP addresses or firewall restrictions. In addition, when a message is transferred and successfully acknowledged, the responsibility for its management is shifting sides. For these reasons, a receiver may want (a) to retain control on the transfer procedure of the underlying protocol by initiating transfers, (b) to decide which messages it wants to receive first and when. Two features have been introduced in ebMS 3 that support this:

- Pull mode
- Message boxes

Pulled-transfer mode or Pull mode is defined in an abstract way by the One-way Pull ebMS MEP (see Simple MEPS Section 2.3 in "Messaging Model"). This MEP allows an MSH to initiate the transfer of a message as a receiver. When used in combination with One-way push ebMS MEP, it allows an MSH to fully control and initiate from its side asynchronous transfers both ways with another MSH, engaging in a client-server type of interaction with the remote MSH, without any need to open a port to incoming requests. This MEP also supports exchanges with a partner that is intermittently connected: instead of periodically polling for partner presence, a sending MSH will simply wait for the partner MSH to pull messages.

Example: A mobile, occasionally connected device without static IP address and with limited storage capability can only initiate requests and receive messages as synchronous responses to these. The One-way Pull MEP allows this device to enable and control the flow of received messages, and to adjust it to its own resources.

Message boxes (see definition in section 2.4) allow for partitioning the flow of messages from an MSH to another MSH into separate flows so that each one of these flows can be controlled independently by either MSH.

Example: A pair of business partners – a large buyer and a small supplier - have decided to create two message boxes for transferring messages sent by the buyer. One mbox is assigned to urgent messages that require immediate processing (high priority Purchase Orders, and updates to prior P.O.) and the other mbox is assigned to less urgent messages (payments, catalog requests, confirmations, acknowledgments of receipts, etc.). The buyer decides of the level of urgency of a posting, which may not be manifested inside the message. Per an agreement with the buyer, the supplier will pull and process first all messages from the urgent mbox, then only the messages from the less-urgent mbox. This way, the low-capacity Receiving MSH (supplier) is able to prioritize the messages it received, focusing its resources on the most urgent messages and avoiding the overhead and risk in managing (persistence, recovery, security) less urgent but important messages that it cannot process in the short term.

Any more complex filtering mechanism that requires checking a filter condition on header data, is out of scope of this specification. It can be implemented on Sending MSH and/or on Receiving MSH in complement to mbox. The notion of an mbox is a simple and robust solution with low interoperability risk: it allows for partitioning messages based on prior agreement between producer and consumer on which type of message will use which mbox.
without a need to transfer and process filter expressions.

Figure 6: One-Way Pull with Message Boxes

Figure 6 illustrates how mboxes and the One-way Pull MEP can be used by a Consumer party to control the order of the messages it wants to receive and process. Mbox 1 is "pulled" in priority by the Consumer side.

In a variant of the example illustrated in the figure, the Consumer side may not experience any connectivity restriction, and may only want to control the transfer of non-critical messages while receiving critical messages on the initiative of the Producer. Mbox 1 may be used for Push mode while Mbox 2 is used in Pull mode. The Consumer will initiate the transfer of non-critical messages over Mbox 2 only when ready to process these.

4.2 Supporting the Pull Mode

Using the Pulled-transfer mode requires the ability for an MSH to support the One-way Pull MEP. The PullRequest signal that initiates this MEP is described in Section 4.3 (Signal Packaging). Because there is always at least one message box open between a Sending MSH and a Receiving MSH – the default mbox – the Pull mode can be supported regardless of the ability to support several message mboxes.

When sending a PullRequest signal, the name of the mbox to pull from must be specified (attribute eb:PullRequest@forMbox).

The processing model for a pulled message is as follows, for a typical and successful
instance of One-way Pull MEP:

On Responding MSH side:
1. Submit: submission of message data to the MSH by the Producer party, intended to
   the Consumer on the Initiator side. The message is associated with an mbox. If no
   mbox name is provided by the submitter, or if the MSH implementation has not been
   provided with a way to do this association by itself, the default mbox is used.

2. On Initiator MSH side:
3. Sending of a PullRequest signal by the MSH. The PullRequest signal specifies the
   mbox to pull from.

On Responder MSH side:
4. Reception of the PullRequest signal. For every PullRequest signal received the
   Responder MSH (acting in Sending role) selects a previously submitted message. It
   is RECOMMENDED to select messages according to a FIFO policy w/r to the Submit
   operation. If there is no user message available in the specified mbox for sending, a
   warning signal with short description: "EmptyMbox" (see Error section) MUST be
   sent back instead.
5. Send: the selected message is sent over the SOAP Response to the PullRequest.

On Initiator MSH side:
8. Receive: the pulled message is available for processing by the MSH. The attribute
   eb:UserMessage/@mbox indicates which mbox it has been pulled from and is same
   as the eb:PullRequest/@forMbox value in the PullRequest signal.
9. Deliver: after processing of ebMS headers, delivery of the pulled message data to
   the Consumer of the MSH.

Example: An example of eb:Messaging header for the PullRequest signal:

```
<SOAP:Envelope>
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:TimeStamp>2005-10-01T10:01:00</eb:TimeStamp>
          <eb:MessageId>UUID-4@example.com</eb:MessageId>
          <eb:PullRequest forMbox="pipe://mymsh.com/pipe123"/>
        </eb:MessageInfo>
      </eb:SignalMessage>
    </eb:Messaging>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```

Example: An outline of eb:Messaging header for the response to the above PullRequest
signal example:

```
<SOAP:Envelope>
  <SOAP:Header>
  </SOAP:Header>
</SOAP:Body/>
</SOAP:Envelope>
```

Deleted: a pipe
Deleted: pipe
Deleted: pipe
Deleted: The pipe associated with this message must have been configured for Pull mode
Deleted: pipe
Deleted: EmptyPipe
Deleted: header element
Deleted: eb:MessageInfo/eb:Pipe Id
Deleted: pipe
Deleted: fromPipe
Deleted: fromPipe
Deleted: \<eb:PipeId>pipe://mymsh.com/pipe123</eb:PipeId>
4.3 Combining Pulling with Security and Reliability

Reliability of a pulled message is usually associated with the reliability of the corresponding PullRequest signal. This is why the reliability of the complete One-way Pull MEP instance is addressed in Section 8.3.

Security for the PullRequest signal is described in details in Section 5.6.

Example: An outline of secure and reliable eb:Messaging header for the PullRequest signal. The reliability header used in the example assumes the use of WS-Reliability, and is specifying At-Least-Once delivery, with an acknowledgment to be sent back on the MEP response message:

```
<SOAP:Envelope>
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:MessageInfo>
        <eb:TimeStamp>2005-10-01T10:01:00</eb:TimeStamp>
        <eb:MessageId>UUID-4@example.com</eb:MessageId>
        <eb:PullRequest forMbox="pipe://mymsh.com/pipe123"/>
      </eb:MessageInfo>
      <wss:Security>
        ...
      </wss:Security>
      <wsr:Request SOAP:mustUnderstand="1">
        ...
        <ReplyPattern>
          <Value>Response</Value>
          <AckRequested/>
        </ReplyPattern>
        ...
      </wsr:Request>
    </eb:Messaging>
    <ws:Security/>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```

Example: An outline of secure and reliable eb:Messaging header for the response to the above PullRequest signal:

```
<SOAP:Envelope>
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:MessageInfo>
        <eb:TimeStamp>2005-10-01T10:02:00</eb:TimeStamp>
        <eb:MessageId>UUID-5@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-4@example.com</eb:RefToMessageId>
        <eb:PartyInfo/>
      </eb:MessageInfo>
      <eb:UserMessage eb:mbox="pipe://mymsh.com/pipe123">
        ...
      </eb:UserMessage>
    </eb:Messaging>
    <ws:Security/>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```
Note: In the above example, the reliability header, which assumes the use of WS-Reliability, is a Response element. It contains the reliability acknowledgment for the PullRequest signal. In this example there is no wsr:Request reliability header, meaning that no acknowledgment is expected for this response. This does not prevent the At-Least-Once reliability contract to be supported for this ebMS user message: if the user message fails to be received by the Initiator MSH, the lack of acknowledgment will cause the resending of the PullRequest signal. This will cause in turn the resending of the same user message. In case of failure to deliver, a failure notification will be raised on the Initiator MSH side.
5  Message Packaging

5.1  Message Envelope and Message Parts

5.1.1  MIME Structure and SOAP Profile

In the ebMS SOAP header eb:Messaging, the prefix “eb” is an example prefix that corresponds to the ebMS 3.0 namespace, as defined in Section 5.1.3.1. The ebMS Message can be packaged as a plain SOAP-1.1/SOAP-1.2 message, or within a MIME multipart to allow payloads or attachments to be included. Because either packaging option can be used, implementations MUST support non-multipart messages.

The ebMS Message MAY contain SOAP extension elements other than the eb:Messaging header block. For example, header blocks supporting message reliability and message security MAY be produced and consumed by an MSH in order to fulfill deployment requirements for those features.

An ebMS Message is packaged as a SOAP-1.1/SOAP-1.2 message independent from communications protocols. When represented as a MIME/Multipart message envelope, this envelope MUST be structured in compliance with the SOAP Messages with Attachments [SOAPATTACH] W3C Note, referred to as a Message Package.

There are two logical sections within the Message Package:

The first section is the ebMS Header (i.e. The eb:Messaging SOAP header block), itself contained in the SOAP Header.

The second section is the ebMS Payload, which is itself made of two sections: (a) the SOAP Body element within the SOAP Envelope, and in case of a MIME packaging, (b) zero or more additional MIME parts. containing additional application level payloads. SOAP Body and MIME parts are also referred to as ebMS Payload Containers. The SOAP Body is the only payload container that requires an XML content.

The general structure and composition of an ebMS User Message is described in Figure 7, and a Signal Message in Figure 8.
Figure 7: User Message Structure
Figure 8: Signal Message Structure

The processing of the SOAP eb:Messaging header block is done according to the SOAP processing semantics: an MSH behaves as a SOAP processor or SOAP node that MUST understand this header block. Other header blocks (except for those relevant to reliability and security of an ebMS Message) are not affected by the ebXML processing. Consequently, some Sending MSH implementation may generate an ebMS message from a well-formed SOAP message as input by just adding an eb:Messaging header, and some Receiving MSH implementation may deliver a well-formed SOAP message as output by just removing the eb:Messaging header.

All MIME header elements of the Message Package MUST conform with the SOAP Messages with Attachments [SOAPATTACH] 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 [SOAP11] specification, the MIME media type of the SOAP Message has the value "text/xml". It is strongly RECOMMENDED that the initial headers contain a Content-ID MIME header structured in accordance with MIME [RFC2045], and in addition to the required parameters for the Multipart/Related media type, the start parameter (OPTIONAL in MIME Multipart/Related [RFC2387]) 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="&lt;messagepackage-123@example.com"--boundaryValue
Content-ID: messagepackage-123@example.com
```

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

5.1.2 Examples of ebMS Messages

The following listings provide examples for various kind of ebMS messages: UserMessage, PullRequest Signal, and an Error Signal. The examples are using SOAP-1.1. However, ebMS message can be used with SOAP-1.2 as well. If SOAP-1.2 was being used instead, the ebMS headers inside eb:Messaging element are not affected, with the exception of the attribute eb:Messaging/@S11:mustUnderstand which becomes eb:Messaging/@S12:mustUnderstand having a boolean value (instead of the integer 1 when SOAP-1.1 is used).

5.1.2.1 UserMessage Example

The following is an example of an ebMS User Message packaged in a SOAP-1.1 message:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <eb:Messaging eb:version="3.0" S11:mustUnderstand="1">
      <eb:UserMessage eb:mbox="mbox-847984.345.345345">
        <eb:MessageInfo>
          <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
          <eb:MessageId>UUID-2@example.com</eb:MessageId>
          <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
        </eb:MessageInfo>
        <eb:PartyInfo>
          <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:PartyInfo>
        <eb:CollaborationInfo>
          <eb:AgreementRef>http://www.oasis-open.org/cpa/123456</eb:AgreementRef>
          <eb:Service>QuoteToCollect</eb:Service>
          <eb:Action>NewPurchaseOrder</eb:Action>
          <eb:ConversationID>4321</eb:ConversationID>
        </eb:CollaborationInfo>
        <eb:MessageProperties>
          <eb:property name="ProcessInst">PurchaseOrder:123456</eb:property>
          <eb:property name="ContextID"> 987654321</eb:property>
        </eb:MessageProperties>
        <eb:PayloadInfo>
          <eb:PartInfo href="cid:foo">
            <eb:Schema eb:location="http://foo/bar.xsd" eb:version="2.0"/>
            <eb:Description>Purchase Order for 100,000 foo widgets</eb:Description>
          </eb:PartInfo>
        </eb:PayloadInfo>
      </eb:UserMessage>
    </eb:Messaging>
  </soap:Header>
</soap:Envelope>
```
5.1.2.2 PullRequest Message Example

The following is an example of a PullRequest Signal Message without Authentication Token:

```
<Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <Header>
    <Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <SignalMessage>
        <MessageInfo>
          <timestamp>2000-07-25T12:19:05</timestamp>
          <MessageId>UUID-2@example.com</MessageId>
        </MessageInfo>
        <PullRequest eb:forMbox="mbox-32598.2345.345"/>
      </SignalMessage>
    </Messaging>
  </Header>
  <Body/>
</Envelope>
```

The following is an example of a PullRequest Signal Message with a Username Authentication Token:

```
<Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <Header>
    <Security xmlns:wsse="...">
      <UsernameToken wsu:Id="TokenId">
        <Username>hamid</Username>
        <Password>SomePassword</Password>
      </UsernameToken>
    </Security>
    <Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <SignalMessage>
        <SecurityTokenReference URI="#TokenId"/>
      </SignalMessage>
    </Messaging>
  </Header>
  <Body/>
</Envelope>
```
### 5.1.2.3 Error Message Example

The following is an example an Error Signal Message:

```xml
<SI1:Envelope xmlns:SI1="http://schemas.xmlsoap.org/soap/envelope/"
  <SI1:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
          <eb:MessageId>UUID-2@example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:Error eb:origin="ebMS" category="Content"
          eb:errorCode="EBMS:0001" eb:severity="failure"
          eb:RefToMessageInError="UUID-1@example.com">
          <eb:Description>Value not recognized</eb:Description>
        </eb:Error>
        <eb:Error eb:origin="Security" category="Processing"
          eb:errorCode="0101" eb:severity="failure"
          eb:RefToMessageInError="UUID-23@example.com">
          <eb:Description>Failed Authentication</eb:Description>
        </eb:Error>
      </eb:SignalMessage>
    </eb:Messaging>
  </SI1:Header>
  <SI1:Body/>
</SI1:Envelope>
```

### 5.1.3 ebXML SOAP Envelope Extension

In conformance with the [XML10] specification, all extension element content is namespace qualified. Namespace declaration (xmlns pseudo attribute) for the ebXML SOAP extension may be included in the SOAP Envelope or Header element, or directly in the ebXML SOAP extension element.

#### 5.1.3.1 Namespace pseudo attribute

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

```xml
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3.0.xsd
```

#### 5.1.3.2 xsi:schemaLocation attribute

The SOAP namespace:

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

resolves to a W3C XML Schema specification. It is strongly RECOMMENDED that ebXML MSH implementations 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, the 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 definition. The ebXML SOAP extension element schema, found in Appendix A, has been defined using the W3C Recommendation version of the XML Schema specification [XMLSCHEMA]. The XMLSchema-instance namespace qualified schemaLocation attribute should include a mapping of the ebXML SOAP Envelope extension namespace to its schema document in the same element that declares the ebXML SOAP Envelope extensions namespace.

The schemaLocation for the namespace described in Section 5.1.3.1 is:

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

Separate schemaLocation attributes are RECOMMENDED. 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/
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
```

```
<SOAP:Header>
  <eb:Messaging
    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:version="3.0">
    <eb:UserMessage>
      <eb:MessageInfo>...
      <eb:PayloadInfo>...
      </eb:UserMessage>
    </eb:Messaging>
  </SOAP:Header>
```

5.1.3.3 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/".

5.1.3.4 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/".

Note:

Unlike ebMS v2, ebXML Messaging 3.0 does not define or make use of any elements within the SOAP Body, which is wholly reserved for user-specified payload data.
5.1.3.5 ebXML SOAP Extensions

An ebMS Message extends the SOAP Message with the extension element eb:Messaging, where "eb" is the namespace for ebMS 3.0.

Other headers that support some aspects of ebMS messaging, such as the Security header (wsse:Security) and Reliability headers, may be present. They are not under the ebMS namespace.

5.1.3.6 id Attribute

The ebXML SOAP extension elements defined in this specification have 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.

5.1.3.7 Version Attribute

The REQUIRED version attribute indicates the version of the ebXML Message Service Header Specification to which the ebXML SOAP Header extension conforms. Its purpose is to provide future versioning capabilities. For conformance to this specification, the version attribute on the SOAP extension element defined in this specification MUST have a value of "3.0". An ebMS Message MAY contain a SOAP header extension element that has a value other than "3.0". An implementation conforming to this specification shall discard a message with an ebXML SOAP extension qualified with a version other than "3.0". If the implementation processes 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 ebMS namespace defined above.

5.1.4 ebMS Header

In case of MIME packaging, the root body part of the Message Package is the SOAP message, as defined in the SOAP Messages with Attachments [SOAPATTACH] W3C Note. This root part always contains the ebMS header.

The MIME Content-Type header for the root part MUST have the value "text/xml" to match the MIME media type of the MIME body part containing the [SOAP11] Message document. The Content-Type header MAY contain a "charset" attribute. For example:

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

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 [RFC3023]. The list of valid values can be found at [IANAMEDIA].

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 [UTF8] be used when encoding this document. Due to the processing rules defined for media types derived from text/xml
[RFC3023], this MIME attribute has no default.

The following fragment represents an example of a root part, for a MIME packaging of ebMS:

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

<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
SOAP:mustUnderstand="1">
  <SOAP:Header>
    <eb:Messaging>
      <eb:UserMessage>
        ...
      </eb:UserMessage>
    </eb:Messaging>
  </SOAP:Header>
  <SOAP:Body>
    ...
  </SOAP:Body>
</SOAP:Envelope>
```

5.1.5 Payload Containers

In addition to the SOAP Body, other Payload Containers MAY be present within a Message Package in conformance with the SOAP Messages with Attachments [SOAPATTACH] specification.

If there is no application payload within the Message Package then the SOAP Body MUST be empty, and there MUST NOT be additional Payload Containers.

The contents of each Payload Container (including the SOAP Body) MUST be identified in the /eb:Messaging/eb:UserMessage/eb:PayloadInfo element.

The ebXML Message Service Specification makes no provision, nor limits in any way, the structure or content of application payloads, except for the SOAP Body which must be an XML document. 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.

Example of SOAP Message containing an ebMS header:

```
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
SOAP:mustUnderstand="1">
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
      <eb:PayloadInfo>
        ...
      </eb:PayloadInfo>
    </eb:Messaging>
  </SOAP:Header>
  <SOAP:Body>
    ...
  </SOAP:Body>
</SOAP:Envelope>
```
5.2 The eb:Messaging Container Element

The REQUIRED eb:Messaging element is a child of the SOAP Header. It is a container for either a User message or a Signal message. It also contains an eb:timestamp element.

In case of a User message, the ebXML header block will be of the form:

```xml
<eb:Messaging>
  <eb:UserMessage>
    ...(header elements of the ebMS user message)
  </eb:UserMessage>
</eb:Messaging>
```

In case of a Signal message, the ebXML header block will be of the form:

```xml
<eb:Messaging>
  <eb:SignalMessage>
    <eb:[signalname]>
      ...(header elements of this ebMS signal message)
    </eb:[signalname]>
  </eb:SignalMessage>
</eb:Messaging>
```

For example, `signalname` can be "PullRequest".

The eb:Messaging element has the following attributes:

- `eb:Messaging/@eb:version`: its value MUST be set to "3.0". This attribute is REQUIRED.
- `eb:Messaging/@SOAP:mustUnderstand`: indicates whether the contents of the element MUST be understood by the MSH. This attribute is REQUIRED, with namespace qualified to the SOAP namespace (http://schemas.xmlsoap.org/soap/envelope/). It MUST have value of '1' (true) indicating the element MUST be understood or rejected.

The eb:Messaging element has the following children elements:

- `eb:Messaging/eb:UserMessage`: The OPTIONAL UserMessage element contains all header information for a User message. If this element is not present, an element describing a Signal message MUST be present.

Example ebMS Message Header:

```xml
<SOAP:Header …>
  <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1" >
    <eb:UserMessage eb:syncresp="false">
      <eb:MessageInfo>
        <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
        <eb:MessageId>UUID-2@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:PartyInfo>
        <eb:From> (no change beside renaming) </eb:From>
        <eb:To> (no change beside renaming) </eb:To>
      </eb:PartyInfo>
    </eb:UserMessage>
  </eb:Messaging>
</SOAP:Header>
```
5.2.1 Element eb:Messaging/eb:UserMessage

This element has the following attributes:

- eb:Messaging/eb:UserMessage/@syncresp: This OPTIONAL Boolean attribute has a value determined by the type of MEP in use, and the role of the message within this MEP. If this message is sent as the request of a SOAP Request-response MEP, and if a synchronous ebMS user message response is expected (as the Response leg of the same MEP instance), the value of this attribute MUST be "true". If not, either its value MUST be "false" (default value) or the attribute MUST NOT be present.

This element has the following children elements:

- eb:Messaging/eb:UserMessage/eb:MessageInfo: This REQUIRED element occurs once, and contains data that identifies the message, and relates to other messages' identifiers.
- eb:Messaging/eb:UserMessage/eb:PartyInfo: This REQUIRED element occurs once, and contains data about originating party and destination party.
- eb:Messaging/eb:UserMessage/eb:CollaborationInfo: This REQUIRED element occurs once, and contains elements that facilitate collaboration between parties.
- eb:Messaging/eb:UserMessage/eb:MessageProperties: This OPTIONAL element occurs at most once, and contains message properties that are user-specific. As parts of the header such properties allow for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) which would otherwise require payload access.
- eb:Messaging/eb:UserMessage/eb:PayloadInfo: This OPTIONAL element occurs at most once, and 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 (a) to make it easier to directly extract a particular payload
associated with this User message, (b) to allow an application to determine whether it can process the payload without having to parse it.

5.2.1.1 Element eb:Messaging/eb:UserMessage/eb:MessageInfo

This element has the following child elements:

- eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:timestamp: The REQUIRED Timestamp element has a value representing the date at which the message header was created, and is conforming to a dateTime (see [XMLSCHEMA]). It MUST be expressed as UTC. Indicating UTC in the Timestamp element by including the 'Z' identifier is optional.
- eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:MessageId: This REQUIRED element has a value representing – for each message – a globally unique identifier 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.
- eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:RefToMessageId: This OPTIONAL element occurs at most once. When present, it MUST contain the MessageId value of an ebMS Message to which this message relates, in a way that conforms to the MEP in use (see the MEP section).
- eb:Messaging/eb:UserMessage/@eb:mbox: This OPTIONAL element occurs at most once. It contains a URI that identifies the message box to which the message is assigned. When the MSH is in Pulled transfer mode, and this is a user message sent as a response to a PullRequest signal message, this element MUST be present and have same value as the eb:forMbox attribute in the PullRequest. When the MSH is in Pushed transfer mode, the absence of this element indicates the use of the default mbox.

5.2.1.2 Element eb:Messaging/eb:UserMessage/eb:PartyInfo

This element has the following child elements:

- eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To: The REQUIRED element occurs once, and contains information describing the destination party.

5.2.1.3 Element eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From

This element has the following child elements:

- eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId: The REQUIRED PartyId element occurs one or more times.
- eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role: The OPTIONAL eb:Role element occurs zero or once. The Role element identifies the authorized role (fromAuthorizedRole or toAuthorizedRole) of the Party sending (when present as a child of the From element) or receiving (when present as a child of the To element) the message. The value of the Role element is a non-empty string. Its possible values are specified in the CPA if such a document is used.
Example: The following fragment demonstrates usage of the From element.

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

5.2.1.4 Element eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To

This element has the same children elements as eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From.

Example: The following fragment demonstrates usage of the To element.

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

5.2.1.5 Element eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId

This element has a string value content that identifies a party, or that is one of the identifiers of this party.

It has a single attribute, @eb:type. The type attribute indicates the domain of names to which the string in the content of the PartyId element belongs. 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 [ISO6523], EDIFACT [ISO9735] or ANSI ASC X12 [ASCI05] registries.

An example of PartyId element is:

```xml
<eb:PartyId eb:type="urn:duns">123456789</eb:PartyId>
```

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

5.2.1.6 Element eb:Messaging/eb:UserMessage/eb:CollaborationInfo

This element has the following children elements:

- OPTIONAL element occurs zero or once. The AgreementRef element is a string that identifies the entity or artifact governing the exchange of messages between the parties.
- REQUIRED element occurs once. It is a string identifying the service that acts on the message and it is specified by the designer of the service.
- REQUIRED element occurs once. The element is a string identifying an operation or an activity within a Service that may support several of these.
- REQUIRED element occurs once. The element is a string identifying the set of related messages that make up a conversation between Parties.
5.2.1.7 Element eb:eb:CollaborationInfo/eb:AgreementRef

AgreementRef is a string value that identifies the agreement that governs the exchange. The P-Mode under which the MSH operates for this message MUST be aligned with this agreement (when eb:AgreementRef is used to configure the P-Modes). However, the element eb:AgreementRef may be used in a way that is independent of the P-Mode being used (in other terms, it can be profiled in any way, much like the eb:Service and eb:Action could be profiled in any way possible).

The value of an 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 [ebCPPA].

An example of the AgreementRef element follows:

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

If a CPA is referred to and a receiver determines that a message is in conflict with the referred 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.

The AgreementRef element has a single @type attribute that indicates how the parties sending and receiving the message will interpret the value of the reference (e.g. the value could be “ebcppa2.1” for parties using a CPA-based agreement representation). There is no restriction on the value of the type attribute: it is left to a profiling exercise that is out of scope of this specification.

5.2.1.8 Element eb:UserMessage/eb:CollaborationInfo/eb:Service

This element identifies the service that acts on the message. Its actual semantics is beyond the scope of this specification. The designer of the service may be a standards organization, an individual or enterprise.

An example of the Service element follows:

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

The Service element has a single @type attribute, that indicates how the parties sending and receiving the message will interpret the value of the element. There is no restriction on the value of the type attribute. If the type attribute is not present, the content of the Service element MUST be a URI (see [RFC2396]). If it is not a URI then the MSH MUST report an error with errorCode of Inconsistent and severity of Error (see the section called "Error Handling Module").

5.2.1.9 Element eb:UserMessage/eb:CollaborationInfo/eb:Action

This element is a string identifying an operation or an activity within a Service. Its actual semantics is beyond the scope of this specification. 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:

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

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

### 5.2.1.10 Element eb:CollaborationInfo/eb:ConversationId

This element is a string identifying the set of related messages that make up a conversation between Parties.

(EdNote: MOVE THIS AND OTHER REFERENCES TO CPA TO AN APPENDIX.)

If a CPA is referred to by eb:AgreementRef, the number of conversations related to this CPA MUST comply with CPA requirements. The value of eb:ConversationID MUST uniquely identify a conversation within the context of this CPA.

An example of the ConversationId element follows:

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

The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond the scope of this specification. Implementations SHOULD provide a facility for mapping between their identification scheme and a ConversationId generated by another implementation.

### 5.2.1.11 Element eb:UserMessage/eb:MessageProperties

This element has zero or more eb:Property children elements.

An eb:Property element is of xs:anySimpleType (e.g. string, URI) and has a @name attribute, the value of which must be agreed between partners.

Its actual semantics is beyond the scope of this specification. The element is intended to be consumed outside the ebMS specified functions. It may contain some information that qualifies or abstracts payload data, or that allows for binding the message to some business process. A representation in the header of such properties allows for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) that do not require payload access.

Example:

```xml
<eb:MessageProperties>
  <eb:property name="ContextId">C1234</eb:property>
  <eb:property name="processinstanceID">3A4-1234</eb:property>
  <eb:property name="transactionID">45764321</eb:property>
</eb:MessageProperties>
```

### 5.2.1.12 Element eb:Messaging/ eb:UserMessage/eb:PayloadInfo

Each PayloadInfo element identifies payload data associated with the message. The purpose of the PayloadInfo is:

- To make it easier to directly extract particular payload parts associated with this ebMS Message
- To allow an application to determine whether it can process these payload parts, without having to parse them.

---

1548  An example of the Action element follows:
1549  <eb:Action>NewOrder</eb:Action>
1550  If the value of either the Service or Action element is unrecognized by the Receiving MSH, then it MUST report an error with an errorCode of NotRecognized and a severity of Error.
1552  **5.2.1.10 Element eb:CollaborationInfo/eb:ConversationId**
1553  This element is a string identifying the set of related messages that make up a conversation between Parties.
1555  (EdNote: MOVE THIS AND OTHER REFERENCES TO CPA TO AN APPENDIX.)
1556  If a CPA is referred to by eb:AgreementRef, the number of conversations related to this CPA MUST comply with CPA requirements. The value of eb:ConversationID MUST uniquely identify a conversation within the context of this CPA.
1559  An example of the ConversationId element follows:
1560  <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
1561  The Party initiating a conversation determines the value of the ConversationId element that SHALL be reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond the scope of this specification. Implementations SHOULD provide a facility for mapping between their identification scheme and a ConversationId generated by another implementation.
1566  **5.2.1.11 Element eb:UserMessage/eb:MessageProperties**
1567  This element has zero or more eb:Property children elements.
1568  An eb:Property element is of xs:anySimpleType (e.g. string, URI) and has a @name attribute, the value of which must be agreed between partners.
1570  Its actual semantics is beyond the scope of this specification. The element is intended to be consumed outside the ebMS specified functions. It may contain some information that qualifies or abstracts payload data, or that allows for binding the message to some business process. A representation in the header of such properties allows for more efficient monitoring, correlating, dispatching and validating functions (even if these are out of scope of ebMS specification) that do not require payload access.
1576  Example:
1577  <eb:MessageProperties>
1578    <eb:property name="ContextId">C1234</eb:property>
1579    <eb:property name="processinstanceID">3A4-1234</eb:property>
1580    <eb:property name="transactionID">45764321</eb:property>
1581  </eb:MessageProperties>
1582  **5.2.1.12 Element eb:Messaging/ eb:UserMessage/eb:PayloadInfo**
1583  Each PayloadInfo element identifies payload data associated with the message. The purpose of the PayloadInfo is:
1585  - To make it easier to directly extract particular payload parts associated with this ebMS Message
1587  - To allow an application to determine whether it can process these payload parts, without having to parse them.
The PayloadInfo element has the following attributes:
- an id attribute (see the section called "id Attribute" for details)
- a version attribute (see the section called "version Attribute" for details)

The PayloadInfo element has the following child elements:
- eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo: This element occurs zero or more times. The PartInfo element can reference either an attachment or an XML element within the SOAP body. When PartInfo is referencing an attachment, the value of its href attribute starts with "cid:" followed by the Content-ID mime header of the attachment being referenced. When PartInfo is referencing an XML element within the SOAP body, the value of its href element starts with the symbol "#".

### 5.2.1.13 Element eb:UserMessage/eb:PayloadInfo/eb:PartInfo

This element has the following attribute:
- eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/@href
  this OPTIONAL attribute has a value that is either the CID URI of the payload object referenced or a reference to an XML ID. For example "cid:foo" or "#idref". The absence of the attribute eb:idref in the element eb:PartInfo indicates that the payload part being referenced is the SOAP Body element itself. For example, a declaration like the following simply states that the SOAP Body is the unique payload part in this ebMS message:

```xml
<eb:PayloadInfo> <eb:PartInfo/> </eb:PayloadInfo>
```

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.

This element has the following child elements:
  This element occurs zero or more times. It refers to schema(s) that define the instance document identified in the parent PartInfo 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 PartInfo element. It provides a means of identifying the schema and its version defining the payload object identified by the parent PartInfo element. The Schema element contains the following attributes: (a) namespace - the REQUIRED target namespace of the schema, (b) location - the REQUIRED URI of the schema, (c) version – a version identifier of the schema.

- eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Description
  This element occurs zero or more times. Its purpose is to provide a human readable description of the purpose or intent of the payload part. 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 [XML10]. Each occurrence SHOULD have a different value for xml:lang.
5.2.2 Element eb:Messaging/eb:SignalMessage

This element is an alternative to the eb:UserMessage element. It has two child elements:

- eb:Messaging/eb:SignalMessage/eb:MessageInfo: This REQUIRED element is similar to eb:MessageInfo as defined for user messages.
- eb:Messaging/eb:SignalMessage/eb:[SignalName]: This REQUIRED element contains an ebMS signal message.

Additionally, a wsse:SecurityTokenReference element MAY be added as a child element, when authentication of the signal is required.

An ebMS signal does not require any SOAP body: if the SOAP body is not empty, it MUST be ignored by the MSH as far as the interpretation of the signal is concerned.

5.2.2.1 Element eb:Messaging/eb:SignalMessage/eb:PullRequest

This element has the following attribute:

- eb:Messaging/eb:SignalMessage/eb:PullRequest/@eb:forMbox

This REQUIRED attribute contains an mbox (URI) that indicates which mbox the message must be pulled from. This in turn indicates which Receiving MSH is initiating the transfer. It is RECOMMENDED that mbox values be Globally Unique (meaning that if two different MSH are sending PullRequests to a third MSH for the same mbox value, then it is really the case that both MSH are pulling from the same mbox).

5.2.2.2 Error Message Packaging

EdNote: Determine whether or not this section is needed.
6 Security Module

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 [ebRISK]. 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.

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.

6.1 Security Element

An ebMS Message can be digitally signed and/or encrypted to provide security countermeasures. Message signature and encryption support is specified in the Web Services Security [WSS-10, WSS-1.1]. Zero or one Security elements per target, belonging to the Web Services Security-defined namespace, MAY be present as a child of the SOAP Header. The Security element MUST be namespace qualified in accordance with Web Services Security. The structure and content of the Security element MUST conform to the Web Services Security SOAP Messages with Attachments Profile [WSS-SWA].

To promote interoperability the security element MUST conform to the WS-I Basic Security Profile Version 1.0 [WSIBSP10], and WS-I Attachments Profile Version 1.0 [WSIAP10].

An MSH implementation may elect to leverage WSS-1.0 and/or WSS-1.1. Note that the security of attachment defined in WSS-1.1 is not only applicable to SOAP-1.1 (security of attachment is orthogonal to the SOAP version). In other terms, an MSH may leverage secure a SOAP-1.2 with Attachments message in the same way a SOAP-1.1 with attachment can be secured in WSS-1.1.

6.2 Securing the PullRequest Signal

6.2.1 Authentication

A Sending MSH MUST be able to authenticate a Receiving MSH that sends a PullRequest. MSH support for this feature is a requirement, though its use is optional. It is RECOMMENDED to achieve authentication in a way that does not depend on the parties that may be using this MSH. In other words, authentication of a Receiving MSH SHOULD be possible regardless of the security requirements that are specific to the parties using this...
MSH. For example, even if these parties decide to not use any security for exchanging messages (this could be indicated by a CPA or some other form of agreement), the Receiving MSH may still need to be authenticated when doing pulling, or when sending any other MSH-level signal.

When authentication is required for a particular Receiving MSH, it is RECOMMENDED that the Sending MSH uses security at SOAP protocol level. In case a Receiving MSH is not able to use SOAP level security, a password-based authentication MAY be used, e.g. as in the HTTP Basic Access Authentication scheme [RFC2617]. In that case a secure communication protocol SHOULD be used (example: TLS, SSL).

### 6.2.1.1 Username / Password Option

In constrained environments where full support for WSS and management of XML digital signatures are not possible, an authentication alternative that is based on the simplest mechanism possible - namely a combination of a username/password – MUST be supported. Username and password for the PullRequest signal MUST use the WSS username/password token.

An example PullRequest signal that is authenticated as above is:

```xml
<wsse:Security xmlns:wsse="..." xmlns:wsu="...">
  <wsse:UsernameToken wsu:Id="#TokenID">
    <wsse:Username>hamid</wsse:Username>
    <wsse:Password>SomePassword</wsse:Password>
  </wsse:UsernameToken>
</wsse:Security>
<eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
  <eb:SignalMessage>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#TokenID" />
    </wsse:SecurityTokenReference>
    <eb:MessageInfo> ... </eb:MessageInfo>
    <eb:PullRequest eb:forMbox="..." />
  </eb:SignalMessage>
</eb:Messaging>
```

**Notes:**

- The attribute `/wsse:Security/wsse:UsernameToken/wsse:Password/@Type` is absent as it defaults to a password of type text. Users should not be required to handle a password of digest type.
- The value of the `wsse:UserName` element is an implementation issue. The "user" may represent the MSH itself, or may represent a party using the MSH. In the latter case, there is no requirement that this user name be identical to some `eb:From/PartyId` value.

### 6.2.1.2 BinaryToken Option

This option requires a more complete implementation of WSS. The syntax for the PullRequest signal would be the same: a `<wsse:SecurityTokenReference>` element is introduced inside the `<eb:SignalMessage>` element. The following is a sample listing using an X509 certificate:

```xml
<wsse:Security soap:mustUnderstand="1"
  xmlns:wsse="..."
  xmlns:wsu="...">
  <wsse:BinarySecurityToken wsu:Id="signingCert"
```

Notes:
6.2.2 Authorization

The processing of a PullRequest signal received by a Sending MSH is authorized based on internal information that the Sending MSH maintains, that associates a list of endpoint information about pre-authorized Receiving MSHs, with the mboxes on which these are allowed to initiate message transfer.

6.2.3 Preventing Repeat Attacks

Malignant duplication and reuse of a PullRequest signals could lead to transfer of user messages to an unauthorized destination in spite of valid claims in the signal message. In order to prevent this attack, it is RECOMMENDED to (1) use At-Most-Once reliability so that duplicate elimination would eliminate PullRequest duplicates, (2) enforce the integrity of reliability headers by proper compliance with WSS.

6.3 Countermeasure Technologies

6.3.1 Persistent Digital Signature

The only available technology that can be applied to the purpose of digitally signing an ebMS Message (the ebXML SOAP Header and Body and its associated payload objects) is provided by technology that conforms to the Web Services Security and Web Services Security SOAP Messages with Attachments Profile. An XML Signature conforming to these specifications 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 ebMS Message then Web Services Security and Web Services Security SOAP Messages with Attachments Profile 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 ebMS 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 Web Services Security and Web Services Security SOAP Messages with Attachments Profile.
6.3.2 Persistent Signed Receipt

An ebMS 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 Web Services Security
Reference element list consistent with those contained in the Web Services Security and
Web Services Security Signature element of the original message.

6.3.3 Non-Persistent Authentication

Non-persistent authentication is provided by the communications channel used to transport
the ebMS 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 [RFC2246] or IPSec [RFC2402] provides the
sender of an ebMS Message with a way to authenticate the destination for the TCP/IP
environment.

6.3.4 Non-persistent Integrity

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

6.3.5 Persistent Confidentiality

Persistent confidentiality is provided by technology that conforms to Web Services Security
and Web Services Security SOAP Messages with Attachments Profile. Encryption conforming
to these specifications can provide persistent, selective confidentiality of elements within an
ebMS Message including the SOAP Header.

6.3.6 Non-persistent Confidentiality

A secure network protocol, such as TLS or IPSec, provides transient confidentiality of a
message as it is transferred between two ebXML adjacent MSH nodes.

6.3.7 Persistent Authorization

Persistent authorization MAY be provided using Web Services Security: SAML Token Profile.

6.3.8 Non-persistent Authorization

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

6.4 Security Considerations

Implementers should take note, there is a vulnerability present even when an Web Services
Security is used to protect the integrity and origin of ebMS Messages. The
significance of the vulnerability necessarily depends on the deployed environment and the transport used to exchange ebMS 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 a Web Services Security based digital signature and/or Web Services Security based encryption is applied. Thus, an ebMS 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 ebMS 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.
7 Error Handling Module

Error handling must take into account the composed nature of an MSH, which includes relatively independent (SOAP) modules such as those handling reliability and security. Error reporting is also subject to the same connectivity constraints as the exchange of regular messages. This calls for a more comprehensive error model. With regard to different ways to report errors, this model must allow for a clear distinction between what is relevant to an agreement, and what is relevant to immutable interoperability requirements.

Error generation and error reporting are treated here as orthogonal concepts. While the generation of errors is a matter of conformance, the reporting of errors may be subject to an agreement. Consequently, the way errors are to be reported is specified in the P-Mode (P-Mode.errorHandling feature) that results from such an agreement.

7.1 Terminology

**Fault:** A Fault always means a SOAP Fault. It must be generated and processed according to the [SOAP11] or [SOAP12] specification.

**Error:** An error that is not a SOAP Fault, and occurs in one of the defined modules (ebMS Module, Reliability Module, Security Module)

**ebMS Error:** This is a particular case of Error, which is generated by the ebMS Module in conformity with this specification.

**Reliability Error:** This is a particular case of Error, generated by the Reliability Module.

**Security Error:** This is a particular case of Error, generated by the Security Module.

**Escalated ebMS Error:** This is an ebMS Error that originates in a module other than the ebMS Module (i.e. Security module, or Reliability module).

**ebMS Error Generation:** The operation of creating an ebMS Error object based on some failure or warning condition.

**ebMS Error Reporting:** The operation of communicating an ebMS Error object to some other entity.

**Message-in-error:** A flawed message causing an error of some kind.

7.2 Packaging of ebMS Errors

7.2.1 eb:Error Element

An ebMS Error is represented by an eb:Error XML infoset, regardless of the way it is reported. Each error raised by an MSH has the following properties:

- origin (required attribute)
- category (optional attribute)
- errorCode (optional attribute)
- severity (optional attribute)
- shortDescription (optional attribute)
• Description (optional element)
• errorDetail (optional element)

Example:

```xml
<eb:Error eb:origin="ebMS" eb:category="Unpackaging"
  eb:errorCode="eb_InvalidHeader" eb:severity="fatal">
  <eb:Description> ... </eb:Description>
</eb:Error>
```

### 7.2.2 Attribute: `eb:Error/@origin`

This required attribute identifies the functional module within which the error occurred. This module could be the Reliability Module, the Security Module, the ebMS Module, or the Addressing Module. The possible values for this attribute are: ebMS, security, reliability, addressing.

### 7.2.3 Attribute: `eb:Error/@category`

This optional attribute identifies the type of error related to a particular origin. For example: Content, Packaging, UnPackaging, Communication, InternalProcess.

### 7.2.4 Attribute: `eb:Error/@errorCode`

This required attribute is a unique identifier for the type of error.

### 7.2.5 Attribute: `eb:Error/@severity`

This optional attribute indicates the severity of the error. Valid values are: warning, failure.

The **warning** value indicates that a potentially disabling condition has been detected, but no message processing and/or exchange has failed so far. In particular, if the message was supposed to be delivered to a consumer, it would be delivered even though a warning was issued. Other related messages in the conversation or MEP can be generated and exchanged in spite of this problem.

The **failure** value indicates that the processing of a message did not proceed as expected, and cannot be considered successful. If, in spite of this, the message payload is in a state of being delivered, the default behavior is not to deliver it, unless an agreement states otherwise (see OpCtx-ErrorHandling). This error does not presume the ability of the MSH to process other messages, although the conversation or the MEP instance this message was involved in is at risk of being invalid.

### 7.2.6 Attribute: `eb:Error/shortDescription`

This optional element provides a short description of the error that can be reported in a log in order to facilitate readability.

### 7.2.7 Element: `eb:Error/Description`

This optional element provides a narrative description of the error in the language defined by the xml:lang attribute. The content of this element is left to implementation-specific
decisions.

7.2.8 Element: eb:Error/ErrorDetail

This optional element provides additional details about the context in which the error occurred. For example, it may be an exception trace.

7.3 ebMS Error Message

When reported as messages, ebMS Errors are packaged as ebMS Signal Messages. Several eb:Error elements may be present under eb:SignalMessage. If this is the case, and if eb:RefToMessageId is present as a child of eb:SignalMessage/eb:MessageInfo, then every eb:Error element must be related to the ebMS message (message-in-error) identified by eb:RefToMessageId.

If the element eb:SignalMessage/eb:MessageInfo does not contain eb:RefToMessageId, then the eb:Error element(s) MUST NOT be related to a particular ebMS message.

Example of ebXML Error Message:

```xml
<SOAP:Header …>
  <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
    <eb:SignalMessage>
      <eb:MessageInfo>
        <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
        <eb:MessageId>UUID-2@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:Error eb:origin="Security" category="Processing"
        eb:errorCode="FailedAuthentication"
        eb:severity="failure">
        <eb:Description>Validation of signature failed</eb:Description>
      </eb:Error>
      <eb:Error eb:origin="ebMS" category="Communication"
        eb:errorCode="EmptyMbox"
        eb:severity="warning">
        <eb:Description>PullRequest done on an empty mbox</eb:Description>
      </eb:Error>
    </eb:SignalMessage>
  </eb:Messaging>
</SOAP:Header>
```

7.4 Extensibility of the Error Module

7.4.1 Adding new ebMS Errors

The errorCode attribute (eb:Messaging/eb:SignalMessage/eb:Error/@errorCode) must be an identifier that is unique within the scope of an MSH. New ebMS Errors beside those specified here may be added by creating new errorCode values. A value for the errorCode attribute must start with the five characters “EBMS:”.

Deleted: EmptyPipe
Deleted: pipe
7.5 Generating ebMS Errors

This specification identifies key ebMS Errors, as well as the conditions under which they must be generated. Some of these error-raising conditions include the escalation as ebMS Errors of either Faults or Errors generated by Reliability and Security modules. These modules could be those contained in the MSH raising the Error, or those contained in a remote MSH communicating with the MSH raising the Error. Except for some cases defined in this specification, Error escalation policies are left to an agreement between users, represented in the processing mode of an MSH (P-Mode_ErrorHandling).

7.6 Error Reporting

There are three primary means of Error Reporting:

- Reporting with Fault Sending: An MSH may generate a SOAP Fault when a certain type of ebMS Error is raised. It is not recommended to use the Fault reporting action if other reporting actions are sufficient.
- Reporting with Notification: An out-of-band transfer of error information from MSH to some entity (message producer, consumer, or any other entity, be it local or remote). In case of notification to the message Producer or Consumer, such reporting action is abstracted by the "Notify" operation in the messaging model.
- Error message: an ebMS signal message sent from one MSH to another, which contains at least one eb:Error element. Such a reporting action is modeled by Send and Receive abstract operations over such a message.

Example of different options in reporting errors raised on a Sending MSH: Some error detected on a submitted message and before it is even packaged, would normally be locally notified to the message Producer, and not even reported to the destination MSH. However, in case this message was part of a larger exchange that is holding its state waiting for completion on the receiving side, the preferred policy could state that the message-in-error be also reported (using an error message) to the Receiving MSH. If a Pull-mode of transfer is established so that the Receiving MSH is getting its messages as responses to PullRequest signals, such ebMS errors could be transferred with the pulled UserMessages. If a Push-mode is active from sender to receiver, it could be decided that errors generated on sender side will be pushed like any regular message.

Example of different options in reporting errors raised on a Receiving MSH: If a Receiving MSH detects an error in a received message, the reporting policy may vary depending on the context and the ability of parties to process such errors. For example, the error-raising Receiving MSH may just notify its own Consumer party, or send back an error message to the Sending MSH, or both. The usual common requirement in all these cases, is that the error be reported somehow, and complies with the eb:Error element structure.

7.7 Standard ebMS Errors

7.7.1 ebMS Processing Errors

The table below describes the Errors that may occur within the ebMS Module itself (ebMS Errors that are not Escalated Errors), i.e. with @origin="ebms". These errors MUST be supported by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.
<table>
<thead>
<tr>
<th>Error Code</th>
<th>Short Description</th>
<th>Recommended Severity</th>
<th>Category Value</th>
<th>Description or Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBMS:0001</td>
<td>ValueNotRecognized</td>
<td>failure</td>
<td>Content</td>
<td>although the message document is well formed and schema valid, some element/attribute contains a value that could not be recognized and therefore could not be used by the MSH.</td>
</tr>
<tr>
<td>EBMS:0002</td>
<td>FeatureNotSupported</td>
<td>warning</td>
<td>Content</td>
<td>although the message document is well formed and schema valid, some element/attribute value cannot be processed as expected because the related feature is not supported by the MSH.</td>
</tr>
<tr>
<td>EBMS:0003</td>
<td>ValueInconsistent</td>
<td>failure</td>
<td>Content</td>
<td>although the message document is well formed and schema valid, some element/attribute value is inconsistent either with the content of other element/attribute, or with the processing mode of the MSH, or with the normative requirements of the ebMS specification.</td>
</tr>
<tr>
<td>EBMS:0004</td>
<td>Other</td>
<td>failure</td>
<td>Content</td>
<td></td>
</tr>
<tr>
<td>EBMS:0005</td>
<td>ConnectionFailure</td>
<td>failure</td>
<td>Communication</td>
<td>the MSH is experiencing temporary or permanent failure in trying to open a transport connection with a remote MSH.</td>
</tr>
<tr>
<td>EBMS:0006</td>
<td>EmptyMbox</td>
<td>warning</td>
<td>Communication</td>
<td>There is no message available for pulling in this message box at this moment.</td>
</tr>
<tr>
<td>EBMS:0007</td>
<td>MimeInconsistency</td>
<td>failure</td>
<td>Unpackaging</td>
<td>The use of MIME is not consistent with the required usage in this specification.</td>
</tr>
</tbody>
</table>
### 7.7.2 Security Processing Errors

The table below describes the Errors that originate within the Security module, i.e. with @origin="security". These errors MUST be escalated by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Short Description</th>
<th>Recommended Severity</th>
<th>Category Value</th>
<th>Description or Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBMS:0101</td>
<td>FailedAuthentication</td>
<td>failure</td>
<td>Processing</td>
<td>The signature in the Security header intended to the ebms SOAP actor, could not be validated by the Security module.</td>
</tr>
</tbody>
</table>

### 7.7.3 Reliability Errors

The table below describes the Errors that originate within the Reliability module, i.e. with @origin="reliability". These errors MUST be escalated by an MSH, meaning generated appropriately, or understood by an MSH when reported to it.
<table>
<thead>
<tr>
<th>EBMS:0202</th>
<th>DeliveryFailure failure communication</th>
<th>Although the message was sent under Guaranteed delivery requirement, the Reliability module could not get assurance that the message was properly delivered, in spite of resending efforts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBMS:0201</td>
<td>DysfunctionalReliability failure Processing</td>
<td>Some reliability function as implemented by the Reliability module, is not operational, or the reliability state associated with this message sequence is not valid</td>
</tr>
</tbody>
</table>
8 Reliable Messaging Module

8.1 The Reliable Messaging Model

This section describes the reliable messaging model for ebMS Messages. The reliability model is expressed in the form of reliability contracts. These contracts are themselves expressed using abstract operations that represent the transfer of message data between the components involved in processing ebMS Messages. In itself, this model is not sufficient to provide guidance to implementers who need to develop interoperable MSHs. This model must be complemented by an instance of it which is based on an existing reliable messaging standard. Such an instance, called here a reliability module, is described in Appendix R.

A basic design principle in ebMS 3.0 is to modularize major messaging QoS features, meaning no interference – except of black-box style – with other aspects of message processing, so that (a) the MSH can rely on existing standards in the area of concern, but also (b) so that implementations of such standards can be reused with no or little modification.

The reliability function is processed separately from the ebms header. This processing will be abstractly defined as performed by a module possibly acting as a separate SOAP node, called a Reliable Messaging Processor (RMP). The reliability of ebMS Messages is supported by SOAP header extensions – called here “reliability header(s)” – that are distinct from ebms headers.

Processing an ebMS Message involves:

- The functions that process the ebms headers.
- The reliability functions as implemented by a Reliable Messaging Processor (RMP), that process reliability headers.
- Other functions that process other SOAP headers also required by this specification, such as security.

As illustrated in Figure 9 and Figure 10, the reliability model requires two instances of RMP playing different roles when executing a reliable MEP: the Initiator RMP (associated with the Initiator MSH) and the Responder RMP (associated with the Responder MSH). It must be noted that these roles do not change over the execution of a simple ebMS MEP instance, as opposed to the roles of Sending and Receiving, which may vary for each user message exchanged. This means, for example, that the Initiator will assume the necessary functions to send a request message reliably, and also receive its response, if any (assuming successively taking on a Sending and then Receiving role, as defined in the Messaging Model, Section 2.1.1).

Five abstract operations, RM-Submit, RM-Deliver, RM-SubmitResponse, RM-DeliverResponse, RM-Notify, represent the abstract interface of the RMP. They transfer either message data or notification data between an RMP and another component of the MSH – either the RM-Producer or the RM-Consumer. In this section, the expression “sent reliably” means that the sending is be subject to a reliability contract (see Section 8.2.1).

The abstract RM operations are defined as follows:

- **RM-Submit**
  An abstract operation that transfers a SOAP message from an RM-Producer to an Initiator RMP, so that this message can be sent reliably.
• **RM-Deliver**
  
  An abstract operation that transfers a SOAP message from a Responder RMP to its RM-Consumer, so that the payload from this message can later be delivered by the MSH.

• **RM-SubmitResponse**
  
  An abstract operation that transfers a SOAP message from an RM-Producer to a Responder RMP as a response to a previously received reliable message. This response is sent back reliably over the response leg of the same SOAP Request-response MEP instance that carried the previous reliable message.

• **RM-DeliverResponse**
  
  An abstract operation that transfers a received SOAP response message from an Initiator RMP to its RM-Consumer.

• **RM-Notify**
  
  An abstract operation that makes available to the RM-Producer or to the RM-Consumer a failure status of a previously sent message (e.g. a notification telling that a reliable message was not delivered).

---

**Figure 9: Reliable Request**

Figure 9 shows the operations involved when sending reliably a request – either a user message in the One-way Push MEP, or the first leg of a One-way Pull MEP or the first leg of a Request-reply MEP.
Figure 10: Reliable Response

Figure 10 shows the abstract operations and components involved when sending reliably a response – either a pulled user message in the One-way Pull MEP or the response user message in a Request-Response MEP (the second leg of a simple ebMS MEP). Note that depending on the reliability processing mode (P-Mode,Reliability), notification of delivery failure may be detected on either side.

8.2 Reliability of ebMS Messages

8.2.1 Reliability Contracts

The following reliability contracts – or quality of service profiles - MUST be supported by the RMP:

- **At-Least-Once Delivery**

  When sending a message with this reliability requirement (RM-Submit invocation), one of the two following outcomes shall occur: either (1) the Responder RMP successfully delivers (RM-Deliver operation) the message to the RM-Consumer or (2) either the Initiator RMP or the Responder RMP notifies (RM-Notify operation) respectively the RM Producer or the RM Consumer of a delivery failure.

- **At-Most-Once Delivery**

  Under this reliability requirement, a message submitted by an RM Producer (RM-Submit operation) to an Initiator RMP shall not be delivered more than once by the Responder RMP to its RM-Consumer. The notion of message duplicate is
based on a notion of message ID that must be supported by the reliability
specification being used.

- **In-Order Delivery**
  Under this reliability requirement, a sequence of messages submitted to an
  Initiator RMP (sequence of RM-Submit invocations) shall be delivered in the same
  order by the Responder RMP to its RM-Consumer.

These contracts MAY also apply to response messages, here called the MEP responses. In
such a case they are expressed in the above contracts with RM-SubmitResponse and RM-
DeliverResponse operations (instead of RM-Submit and RM-Deliver, respectively), and the
Responder and Initiator RMPs switch roles.

These contracts MAY be combined. In particular, Exactly-Once (resulting from the
combination of At-Least-Once and At-Most-Once) MUST be supported.

In order to support these reliability contracts, both Initiator and Responder RMPs MUST use
a reliability protocol independent from the transport protocol and that provide end-to-end
acknowledgement and message resending capabilities. The details and parameters
associated with these protocol functions are left to the reliability profile that instantiates this
reliability model [Appendix R].

8.2.2 Supporting Reliability Contracts in ebMS

Because reliability quality of service (QoS) must have significance for the user layer of the
MSH (Producer, Consumer), and not just for the internal components of the MSH (called
RM-Producer and RM-Consumer) that interact with the RMP component, it is necessary to
extend the above contracts and express them in terms of abstract MSH operations:

- **At-Least-Once ebMS Delivery**
  When sending a message with this reliability requirement (Submit invocation),
one of the two following outcomes shall occur: either (1) the Responder MSH
successfully delivers (Deliver operation) the message to the Consumer or (2)
either the Initiator MSH notifies (Notify operation) its Producer of a delivery
failure, or the Responder MSH notifies (Notify operation) its Consumer of a
delivery failure.

- **At-Most-Once ebMS Delivery**:
  Under this reliability requirement, a message transmitted as the result of a
Submit invocation on the Initiator MSH shall not be delivered more than once by
the Responder MSH to its Consumer.

- **In-Order ebMS Delivery**
  Under this reliability requirement, a sequence of messages submitted to the
Initiator MSH by its Producer shall be delivered by the Responder MSH in the
same order to its Consumer.

In order to fulfill the above QoS requirements, an MSH MUST do the following in addition to
interfacing with the reliability functions provided by the RMP:

- Ensure a proper mapping between MSH abstract operations and RMP abstract
  operations. This mapping, which depends on the ebMS MEP being used, is
described in a next section.
- Ensure the handling of additional failure cases that may happen outside the RMP
  processing and outside the transport layer. For example, in the case of At-Least-
  Once delivery, the MSH must ensure that if a message that has been submitted
  (Submit) fails before RM-Submit is invoked, then a delivery failure Error is
  generated, as would be the case if the message processing failed just after RM-
  Submit was invoked. Similarly, if a message fails to be delivered on receiver side
(Deliver) even after RM-Deliver has been successfully invoked, then a delivery failure Error must be generated. The reporting of these errors obeys the P-Mode.errorHandling.“.

Similar contracts apply to response messages (e.g. second leg of an ebMS Request-Reply MEP), by switching Initiator MSH and Responder MSH in the above definitions.

Messages that have eb:CollaborationInfo/eb:Service set to “http://www.oasis-open.org/committee/ebxml-msg/defaultService” are not intended to be delivered (Deliver) to an MSH Consumer, although they may be submitted by an MSH Producer. They are intended for internal MSH consumption. They may also be subject to reliability contracts. In this case, the delivery contract is fulfilled with a successful RM-delivery. In case of at-least-once delivery, a failure to deliver must cause the generation of a delivery failure Error. If this message was submitted by an MSH Producer (Submit) instead of the MSH itself, the Producer may be notified (Notify) of the failure depending on the reporting mode, as for regular user messages.

8.2.3 Message Headers Processing Rules

Some assumptions are made on the processing order of the headers of an ebMS message. The processing of an ebMS message by the MSH may include the processing of headers others than those that are ebms-qualified, such as WS-Security headers.

For the sake of composition and reusability of RMP implementations, it is desirable that the processing of SOAP headers that support reliability and the processing of the ebms header can be separated and strictly serialized.

The following serialization is REQUIRED, between reliability headers and ebms-qualified headers:

**On Sending side:**
1. processing of ebMS headers (the ebms-qualified headers are added to the message).
2. processing of reliability headers (the headers are added to the message).

**On Receiving side:**
1. processing of reliability headers (the headers are removed from the message).
2. processing of ebMS headers (the ebms-qualified headers are removed from the message).

Note
Other steps in the processing of ebXML headers, such as Security headers, are not mentioned here. The above workflows do not exclude the insertion of such additional steps as appropriate.

8.2.4 Reliability of Signal Messages

Some reliable messages do not result from submission of payloads (Submit) by a Producer to the MSH and are instead initiated by MSH functions (defined as ebMS signal messages). They can be made reliable too. For such messages, the reliability contract is expressed in terms of RMP abstract operations only. On the sending side, the contract starts with RM-Submit invocation (e.g. submission of message data to the RMP by the RM-Producer). The message has been reliably transmitted when RM-Deliver is successfully invoked, i.e. when delivered to a component of the receiving MSH (RM-Consumer).
8.3 Reliability of ebMS MEPs

8.3.1 Reliability of the One-Way Push MEP

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

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

On Responder 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 message data to the Consumer of the MSH.

Notes:
- In case of delivery failure, either step (4) (Deliver) fails and Notify is invoked on Responder side, or both (3) and (4) fail and RM-Notify (then Notify) is invoked on either one of each side. A step "fails" either when it is not invoked in the workflow, or when it is invoked but does not complete successfully.
- The semantics of RM-Deliver MAY be interpreted as including the delivery from MSH to its consumer (Deliver invocation). In other words, if the Deliver invocation fails for a received message, then the RM-Deliver invocation for the same message MAY also fail, triggering a failure notification either on the Responder MSH, or on the Initiator MSH (by virtue of the reliability protocol).

Figure 11 illustrates the message flow for this reliable MEP.
8.3.2 Reliability of the One-Way Pull MEP

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

On Responder MSH side:
- Step (1): Submit: submission of message data to the MSH by the Producer party, intended to the Consumer on the Initiator side.

On Initiator MSH side:
- Step (2): Generation of a reliable PullRequest signal by the MSH. RM-Submit is invoked on the Initiator RMP for this signal.

On Responder MSH side:
- Step (3): Reception of the PullRequest signal by MSH functions. RM-Deliver is invoked on the Responder RMP for this signal.
- Step (4): Submission of the pulled message to the RMP. This results in an RM-SubmitResponse invocation.

On Initiator MSH side:
- Step (6): Deliver: after processing of ebMS headers, delivery of the pulled message data to the Consumer of the MSH.
Figure 12 illustrates the message flow for this reliable MEP.

Figure 12: Reliable One-Way Pull MEP

In this MEP as well as in the Simple Request-reply MEP below, the same reliability contracts that apply to the MEP request (here the PullRequest signal) MAY apply to the MEP response handled by RM-SubmitResponse and RM-DeliverResponse operations.

In such cases, when an MEP response is under reliability contract, the following requirements apply:

- When the MEP response is under At-Least-Once reliability contract, then the MEP request MUST also be under At-Least-Once reliability contract. In addition, if the MEP request is also under At-Most-Once reliability contract, and if it has been delivered and responded to by the Responder RMP, then if a duplicate of the MEP request is received later, a duplicate of the same response that has been returned for the initial request MUST be returned for the duplicate request. Note: depending on where a response delivery failure needs be notified (either on Initiator or Responding side, based on OpCtx-Reliability content), an acknowledgment may or may not need be returned for the response message by the Initiator RMP.

- When the MEP response is under At-Most-Once delivery, then the MEP request MUST also be under At-Most-Once delivery.

8.3.3 Reliability of the Request-Reply MEP

Reliability of the Request-Reply MEP is handled similarly to the reliability of the One-Way Pull MEP, as far as the RMP is concerned. The processing model is as follows, for a typical and successful instance of this MEP:

On Initiator MSH side:

- Step (1): Submit: submission of the request message data to the MSH by the Producer party.
• Step (2): **RM-Submit**: submission of the request message to the Initiator RMP.

**On Responder MSH side:**
- Step (3): **RM-Deliver**: after processing of reliability headers, delivery of the request message to RM-Consumer.
- Step (4): **Deliver**: delivery of the request message data to the Consumer of the MSH.
- Step (5): **Submit**: submission of a response message data to the MSH by the Consumer of the request message, intended to the Producer on the Initiator side.
- Step (6): **RM-SubmitResponse**: submission by the RM-Producer of the response message to the Responder RMP.

**On Initiator MSH side:**
- Step (7): **RM-DeliverResponse**: delivery of the response message to the RM-Consumer.
- Step (8): **Deliver**: delivery of the response message data to the Consumer of the Initiator MSH.

Figure 13 illustrates the message flow for this reliable MEP.

![Figure 13: Reliable Request-Reply MEP](image)

When the MEP response is under reliability contract, the same dependencies with the reliability of the MEP request that are described for the One-way Pull MEP, also apply here.
9 The ebXML SOAP Extension Schema (Appendix)

The OASIS ebXML Messaging Technical Committee has provided a version of the [SOAP11] envelope schema specified using the schema vocabulary that conforms to the W3C XML Schema Recommendation specification [XMLSCHEMA].

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

It was necessary to craft a schema for the [XLINK] attribute vocabulary to conform to the W3C XML Schema Recommendation. This schema is referenced from the ebXML SOAP extension elements schema and is available from the following URL:

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


<?xml version="1.0" encoding="UTF-8"?>
...
</schema>
10 Reliable Messaging Bindings (Appendix)

The reliability contracts defined in Section 8 may be implemented by profiling different reliability specifications. Either one of two OASIS reliability specifications may be used by an MSH implementation: WS-Reliability 1.1 [WS-R11], or WS-ReliableMessaging 1.1 [WSRM11]. Although either one of the above OASIS reliability specifications is sufficient, each one has strong arguments in favor of its use. In the same way as two MSH implementations must support the same transfer protocol or cryptographic algorithms in order to interoperate, two MSHs must also implement the same reliability specification in order to have interoperable reliability features. The reliability specification being used in an implementation is a parameter of the conformance profiles for ebMS (see Section 0).

10.1 WS-Reliability Binding

10.1.1 Operations and Contracts Binding

The Reliable Messaging Processor (RMP) in ebMS is instantiated by the RMP as defined in WS-Reliability 1.1. To avoid confusion, we will call the RMP as defined in WS-Reliability the WSR-RMP.

The RMP abstract operations RM-Submit, RM-Deliver, RM-SubmitResponse, RM-DeliverResponse and RM-Notify, map respectively to Submit, Deliver, Respond, Notify and Notify in WS-Reliability 1.1. Note that a single operation in WS-Reliability (Notify) is used to carry both notification of failure, and response message. In order to avoid confusion with WS-Reliability operations, the MSH operations Submit, Deliver, Notify, are respectively renamed in this section: MSH-Submit, MSH-Deliver, MSH-Notify.

The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery respectively map to the RM agreement items: GuaranteedDelivery, NoDuplicateDelivery, OrderedDelivery in WS-Reliability.

Message processing faults such as FeatureNotSupported, PermanentProcessingFailure, or GroupAborted faults, when received by an RMP must be communicated to the MSH. The MSH must escalate such faults as DysfunctionalReliability ebMS errors (EBMS:0201). Message format faults, if they result in non-delivery, must be escalated as DeliveryFailure ebMS errors (EBMS:0202).

10.1.2 Complement to the Reliability of the One-Way Push MEP

When At-Least-Once delivery is required, it is RECOMMENDED that an Initiator MSH be made aware of a delivery failure from the Responder MSH to its Consumer. Such a failure is notified to the Producer party via MSH-Notify. In order to achieve this awareness, the RM-Deliver operation should be implemented so that it will fail if the MSH-Deliver invocation fails. In such a case the Responder WSR-RMP generates a MessageProcessingFailure.
fault, and will not acknowledge the reliable message that has not been successfully
delivered by the Responder MSH to its Consumer.

The RM-Agreement associated with the message, as defined in WS-Reliability, is restricted
as follows:

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

10.1.3 Complement to the Reliability of the One-Way Pull MEP

When At-Least-Once delivery is required, it is RECOMMENDED that a Responder MSH be
made aware of a delivery failure from the Initiator MSH to its Consumer. Such a failure is
notified to the Producer party (Responder side) via MSH-Notify. In order to achieve this
awareness, the RM-DeliverResponse operation should be implemented so that it will fail if
the MSH-Deliver invocation fails (Initiator side). In such a case the Initiator WSR-RMP
generates a **MessageProcessingFailure** fault, and will not acknowledge the reliable
message that has not been successfully delivered by the Initiator MSH to its Consumer.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Allowed Values</th>
<th>Additional Requirements</th>
</tr>
</thead>
</table>
| GuaranteedDelivery  | "enabled", "disabled" | When enabled, it is REQUIRED that the PullRequest signal message associated with this pulled message be also sent with this parameter enabled. When the PullRequest signal is sent with GuaranteedDelivery enabled, two additional requirements MUST be satisfied:
  • The ReplyPattern value associated with the PullRequest signal is "Response".
  • The NoDuplicateDelivery agreement item is also enabled for the PullRequest signal.

The Responder RMP sends back a copy of the original pulled message if the latter is
not expired, when a duplicate of the PullRequest signal is received, e.g. due to
resending (see Section 8.3.2). 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).
<table>
<thead>
<tr>
<th>Name</th>
<th>Allowed Values</th>
<th>Additional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoDuplicateDelivery</td>
<td>&quot;enabled&quot;, &quot;disabled&quot;</td>
<td>When enabled, the PullRequest signal message associated with this pulled message MUST also be sent with this parameter enabled.</td>
</tr>
<tr>
<td>OrderedDelivery</td>
<td>&quot;enabled&quot;, &quot;disabled&quot;</td>
<td>No restriction.</td>
</tr>
<tr>
<td>ReplyPattern</td>
<td>&quot;Callback&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Note

WS-Reliability 1.1 is silent about the reliability of messages submitted as responses to other messages, over the same SOAP MEP instance. Such messages would be submitted using the abstract operation RM-Respond, which requires a WSR-RMP to correlate the response message with the related request. This specification requires that the reliability of these responses, in the case of pulled messages, be also supported by the Responder MSH. This means that the implementation of WSR-RMP used in an MSH should also support RM agreements that cover such responses.

### 10.1.4 Complement to the Reliability of the Simple Request-Reply MEP

As already mentioned for the One-Way Push MEP and the One-Way Pull MEP when At-Least-Once delivery is required, it is RECOMMENDED that the Initiator MSH be made aware of a request delivery failure from the Responder MSH to its Consumer, and also that the Responder MSH be made aware of a response delivery failure from the Initiator MSH to its Consumer.

The RM-Agreement associated with the request message MUST comply with the same restrictions as for the One-Way Push MEP, and also with those entailed by the RM-Agreement options used for the response message (see below.)

The RM-Agreement associated with the Response message MUST comply with the following restrictions:
<table>
<thead>
<tr>
<th>Name</th>
<th>Allowed Values</th>
<th>Additional Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>GuaranteedDelivery</td>
<td>&quot;enabled&quot;, &quot;disabled&quot;</td>
<td>When enabled, it is REQUIRED 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:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The ReplyPattern value associated with the PullRequest signal is &quot;Response&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The NoDuplicateDelivery agreement item is also enabled for the Request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The Responder WSR-RMP sends back a copy of the original Response message if the latter is not expired, e.g. due to resending (see Section 8.3.2). 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).</td>
</tr>
<tr>
<td>NoDuplicateDelivery</td>
<td>&quot;enabled&quot;, &quot;disabled&quot;</td>
<td>When enabled, the Request message associated with this Response message MUST also be sent with this parameter enabled.</td>
</tr>
<tr>
<td>OrderedDelivery</td>
<td>&quot;enabled&quot;, &quot;disabled&quot;</td>
<td>No restriction.</td>
</tr>
<tr>
<td>ReplyPattern</td>
<td>&quot;Callback&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Note
The Request message and Response message do not have to share the same RM-Agreement.
10.2 WS-ReliableMessaging Binding

10.2.1 Operations and Contracts Binding

The Reliable Messaging Processor (RMP) in ebMS is mapping to the following notions in WS-RM [WS-ReliableMessaging]: the Sending RMP maps to RMS (Reliable Messaging Source), the Receiving RMP maps to RMD (Reliable Messaging Destination).

The RMP abstract operations RM-Submit, RM-Deliver, map respectively to Send, Deliver in WSRM. So do RM-SubmitResponse, RM-DeliverResponse, as there is no distinction in applying reliability features to a SOAP request and to a SOAP response in WS-RM. RM-Notify must be implemented so that failures detected by RMS are escalated to the MSH as follows: CreateSequenceRefused, SequenceTerminated, MessageNumberRollover or UnknownSequence faults, when received by an RMS and when the RMS cannot establish a substitute sequence that would support reliable transmission of messages in the same conditions as the failed sequence would have, must be communicated to the MSH. The MSH must escalate such faults as DysfunctionalReliability ebMS errors (EBMS:0201).

The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery do not have equivalent definitions in WS-RM, which only specifies protocol-level features. They are however supported by the specified protocol. The functions required by these reliability contracts as defined in section 8.2, must be implemented as appropriate in RMS and RMD or as extensions to these.

It is RECOMMENDED that all messages transmitted over a same sequence use the same message box. This become a requirement for the In-Order reliability contract.

Parameters of the WS-RM protocol such as acknowledgment interval, timeouts, resending frequency, etc. should be specified in the Processing Mode (see Section 3).

Sequence acknowledgements and sequence operations (such as CreateSequence, CreateSequenceResponse) MUST use MEPs of the underlying protocol in a way that is compatible with the conformance profile of the MSH which defines the ebMS MEPs that must be supported, along with the underlying protocol binding. For example, if the ebMS conformance profile for an MSH only requires ebMS messages to be reliably pulled by this MSH over HTTP, then their sequence must be created by a CreateSequence message carried over an HTTP response, the HTTP request being initiated by this MSH.

When At-Least-Once delivery is required for the ebMS User message carried by this MEP, the RMP on Initiator side is acting as an RMS, and the RMP on Responder side is acting as an RMD. It is RECOMMENDED that the Initiator MSH be made aware of a delivery failure from the Responder MSH to its Consumer. Such a failure is notified to the Producer party via Notify.

A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment for a
sent message, must be communicated to the Initiator MSH. The MSH must escalate such a fault as DeliveryFailure ebMS errors (EBMS:0202).

A failure to deliver that is detected by the RMD (Responder side), e.g. failure to deliver (operation Deliver) after the message has been received and acknowledged by the RMD, must be communicated to the Responder MSH. The MSH must escalate such a fault as DeliveryFailure ebMS errors (EBMS:0202). It is RECOMMENDED that this ebMS error be reported to the Initiator MSH.

10.2.3 Complement to the Reliability of the One-Way Pull MEP

When At-Least-Once delivery is required for the ebMS User message carried by this MEP, the RMP on Responder side is acting as an RMS, and the RMP on Initiator side (which sent the PullRequest) is acting as an RMD. It is RECOMMENDED that the Responder MSH be made aware of a delivery failure from the Initiator MSH to its Consumer. Such a failure is notified to the Producer party (Responder side) via Notify.

A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment on the Responder side for a sent message, must be communicated to the Responder MSH. The MSH must escalate such a fault as DeliveryFailure ebMS errors (EBMS:0202).

A failure to deliver that is detected by the RMD (Initiator side), e.g. failure to deliver (operations Deliver) after the message has been received and acknowledged by the RMD must be communicated to the Initiator MSH. The MSH must escalate such a fault as DeliveryFailure ebMS errors (EBMS:0202). It is RECOMMENDED that this ebMS error be reported to the Responder MSH.

It is RECOMMENDED that the sequence creation for sending reliably the PullSignal messages, offers to create a sequence (wsrm:CreateSequence/wsrm:Offer) in the reverse direction.

As mentioned in Section 8, At-Least-Once delivery is also required for the ebMS signal message "PullSignal". The corresponding acknowledgment should be sent over the second leg of the ebMS MEP, bundled with the pulled ebMS user message. However the frequency of acknowledgments may not need to be on a per-message basis.

10.2.4 Complement to the Reliability of the Simple Request-Reply MEP

Same handling as for the reliability of the One-way Pull MEP applies here.
11 SOAP Format and Bindings (Appendix)

This appendix specifies the SOAP format (SOAP versions, packaging of attachments and/or binary data) used in ebMS-3, as well as how this SOAP format is transported over HTTP and SMTP.

ebMS-3 does not require the usage of SOAP-1.1 and/or SwA (SOAP-1.1 With Attachments). We consider the attachments specification of SwA as being orthogonal to the SOAP version. In other words, attachments could well be used for SOAP 1.2 in the same way they are used for SOAP 1.1. Similarly, we also consider MTOM being orthogonal to the SOAP version (however, MTOM will not be addressed in this core specification).

A conformant implementation of ebMS-3 may well choose to use SOAP-1.2 instead of SOAP-1.1. Since SwA is orthogonal to the SOAP version, there are two possibilities:

- An implementation of ebMS-3 may choose SOAP-1.1 with Attachments
- An implementation of ebMS-3 may choose SOAP-1.2 with Attachments

By allowing the use of SOAP 1.2 with Attachments, it may seem that we are playing here a role that W3C should have done, but this is not the case. We are simply being conformant to standards (See the note below, a quotation from David Chappell):

Note: What are “standards” exactly?

When talking about the use and adoption of standards, it is hard to tell just exactly what the word means. We live in a world of multiple overlapping efforts from different standards bodies to define standard specifications. Vendor alliances are producing web services specifications outside the domain of any standards body or consortium.

[...] There are also “de facto standards” such as open source implementations, which either invented their own ways of doing things or conform to industry-accepted "standard“ specifications. The Apache SOAP and Apache Axis toolkits are examples of such standards.

[...] In short, the word "standard," in terms of standards-based integration, refers to either a specification or an implementation that has gained enough traction in the industry to have long-lasting staying power, and is open enough for multiple vendors to implement or repackage it.

We consider the word “standards” in a wide sense as outlined above by Chappell’s quotation. From the implementation point of view, all libraries that support SOAP 1.2 are able to add attachments to a SOAP message in the same way it is done for SOAP 1.1. In other words, adding attachments to SOAP 1.2 in the same way it is done to SOAP 1.1 is really a standard since all implementations (without exception) supporting SOAP 1.2 are doing it and have been doing it for a long time without waiting a W3C recommendation on attachments for SOAP 1.2.

SwA uses the multipart/related MIME encapsulation. This encapsulation is independent of the version of SOAP being used (in fact it can encapsulate any XML document, not just
SOAP), and also independent of the transport protocol (the encapsulation could be transported via HTTP, SMTP, etc...).

### 11.1 Using SwA with SOAP-1.1

The following example shows an ebMS-3 message using SOAP 1.1 with attachment. The ebMS-3 message in this example contains two payloads:

- The first payload is the picture of a car. This picture is in binary form as an attachment with a Content-ID equal to "car-photo".
- The second payload is an XML fragment within the SOAP body. This XML fragment has id attribute equal to "carData"

The XML fragment in the SOAP body contains a reference to another binary data, namely the picture of the car owner:

```xml
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="<car-data@toyoya.com>"
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <car-data@toyoya.com>

<?xml version='1.0' ?>
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">

<S11:Header>

<eb:Messaging eb:version="3.0" S11:mustUnderstand="1">

<eb:PayloadInfo>
<eb:PartInfo href="cid:car-photo" />
<eb:PartInfo href="#carData" />
</eb:PayloadInfo>

</eb:Messaging>
</S11:Header>

<S11:Body>

<t:Data id="carData" xmlns:t="http://toyota.com">

<t:Mileage>20000</t:Mileage>
<t:OwnerPicture href="cid:picture-of-owner"/>

</t:Data>
</S11:Body>
</S11:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <car-photo>

...binary TIFF image of the car...

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <picture-of-owner>

...binary TIFF image of the car's owner...

--MIME_boundary
```

Example 1: SOAP-1.1 with Attachment
The following (Example 2) shows the same message given in example 1, except that SOAP-1.2 is being used instead of SOAP-1.1:

```
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="<car-data@toyoya.com>

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <car-data@toyoya.com>

<?xml version='1.0' ?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <S12:Header>
    <eb:Messaging eb:version="3.0" S12:mustUnderstand="true">
      <eb:PayloadInfo>
        <eb:PartInfo href="cid:car-photo" />
        <eb:PartInfo href="#carData" />
      </eb:PayloadInfo>
    </eb:Messaging>
  </S12:Header>
  <S12:Body>
    <t:Data id="carData" xmlns:t="http://toyota.com">
      <t:Mileage>20000</t:Mileage>
      <t:OwnerPicture href="cid:picture-of-owner"/>
    </t:Data>
  </S12:Body>
</S12:Envelope>
```

Example 2: SOAP-1.2 with Attachments

What were the differences between Example 1 and Example 2 (SOAP 1.1/SOAP 1.2 with attachments)? The differences are the following:

In SOAP 1.1, the namespace of the SOAP elements (Envelope, Header, and Body) is http://schemas.xmlsoap.org/soap/envelope/ versus the namespace http://www.w3.org/2003/05/soap-envelope for SOAP 1.2.

In SOAP 1.1, the attribute mustUnderstand takes 0 or 1 as values, whereas in SOAP 1.2, the values for the attribute mustUnderstand are true and false.

That's it [We are not explaining here the differences in syntax between SOAP 1.1 and SOAP 1.2. See section 6 of the recommendation http://www.w3.org/TR/soap12-part0/ for more details on such differences. We are simply explaining here the differences between SOAP 1.1 and SOAP 1.2 in terms of Mime packaging headers only and in particular how the sample listing...
given in Example 1 is migrated to become a sample listing for SOAP 1.2 with attachments as given by Example 2]. Another difference between SOAP 1.1 and SOAP 1.2 would be in the SOAPAction header. When using HTTP as the transport protocol, there will be an HTTP header called SOAPAction if SOAP 1.1 is being used. If SOAP 1.2 is used, instead of the SOAPAction header there will be an action parameter, as illustrated in the following listings:

<table>
<thead>
<tr>
<th>HTTP headers when using SOAP 1.1 with attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAPAction: leasing</td>
</tr>
<tr>
<td>Content-Type: Multipart/Related; boundary=MIME boundary; type=text/xml; start=&quot;<a href="mailto:car-data@toyoya.com">car-data@toyoya.com</a>&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SOAPAction: leasing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Type: Multipart/Related; boundary=MIME boundary; type=text/xml; start=&quot;<a href="mailto:car-data@toyoya.com">car-data@toyoya.com</a>&quot;; action=leasing</td>
</tr>
</tbody>
</table>

When using SMTP transport, the only additional requirement is that the Mime-Version header must be present (among other SMTP related headers such as To, From, Date, etc...). The following listings show the headers for both SOAP 1.1 and SOAP 1.2 over SMTP:

<table>
<thead>
<tr>
<th>SMTP headers when using SOAP 1.1 with attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: <a href="mailto:hamid@us.fujitsu.com">hamid@us.fujitsu.com</a></td>
</tr>
<tr>
<td>To: <a href="mailto:leasing-office@toyota.com">leasing-office@toyota.com</a></td>
</tr>
<tr>
<td>Date: Mon, 23 Jan 2006 17:33:00 CST</td>
</tr>
<tr>
<td>Mime-Version: 1.0</td>
</tr>
<tr>
<td>SOAPAction: leasing</td>
</tr>
<tr>
<td>Content-Type: Multipart/Related; boundary=MIME boundary; type=text/xml; start=&quot;<a href="mailto:car-data@toyoya.com">car-data@toyoya.com</a>&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SMTP headers when using SOAP 1.2 with attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>From: <a href="mailto:hamid@us.fujitsu.com">hamid@us.fujitsu.com</a></td>
</tr>
<tr>
<td>To: <a href="mailto:leasing-office@toyota.com">leasing-office@toyota.com</a></td>
</tr>
<tr>
<td>Date: Mon, 23 Jan 2006 17:33:00 CST</td>
</tr>
<tr>
<td>Mime-Version: 1.0</td>
</tr>
<tr>
<td>Content-Type: Multipart/Related; boundary=MIME boundary; type=text/xml; start=&quot;<a href="mailto:car-data@toyoya.com">car-data@toyoya.com</a>&quot;; action=leasing</td>
</tr>
</tbody>
</table>
This section introduces the notion of conformance profiles for MSH implementations. It distinguishes a "main" conformance profile. It also outlines a common structure and syntax for defining additional "specialty" conformance profiles. The notion of conformance profile used here is defined in more details in [QAFW].

Note: conformance profile should not be confused with usage profile:

Conformance profile: defines a set of capabilities that an MSH implementation must have. This is determined at development time regardless of the way the MSH is being used later.

Usage profile: defines a way of using an MSH implementation, that a community of users has agreed upon. This may in turn require a particular conformance profile.

For example, a conformance profile may require that an MSH supports the optional MessageProperties header element, meaning it is able to extract it from a received message or to add it to a message to be sent. In contrast, a usage profile will in addition require that some specific property name be present in each message.

The interpretation of normative material follows the general rule below, as a complement to RFC2119:

A conforming implementation of this specification 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 [RFC2119]. When these keywords apply to message contents that relate to a more general feature, an implementation that conforms to a profile requiring support for this feature MUST be capable of processing these optional message contents according to the described ebXML semantics.

A conformance profile is...
12 References (Appendix)

12.1 Normative References

[ASCI05] Unknown, ASC X12 Registry, ????. \textless url-unknown\textgreater


[IANAMEDIA] Various, MIME Media Types, Various. \textless http://www.iana.org/assignments/media-types/\textgreater


2756 <http://www.ietf.org/rfc/rfc2821.txt>
2760 <http://www.ietf.org/rfc/rfc2821.txt>
2762 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
2764 <http://www.w3.org/TR/soap12-part1/>
2766 <http://www.w3.org/TR/SOAP-attachments>
2768 <http://wp.netscape.com/eng/ssl3/draft302.txt>
2770 <http://www.ietf.org/rfc/rfc2279.txt>
2778 <http://www.w3.org/TR/xlink/>
2780 <http://www.w3.org/TR/2004/REC-xml-20040204/>
2783 <http://www.w3.org/TR/xmldsig-core/>
13 Acknowledgments (Appendix)

EdNote: This section, if necessary, to be completed in a future draft.

The editors would like to acknowledge the contributions of the OASIS ebXML Messaging Services Technical Committee, whose voting members at the time of publication were:

In addition, the following people made contributions to this specification:
## 14 Revision History (Appendix)

This appendix is optional, but helpful. It should be removed for specifications that are at OASIS Standard level. Set the number format for the Rev and Date fields as you wish (select the desired string and choose Format>Number Format...); the examples below are user-defined formats.

<table>
<thead>
<tr>
<th>Rev</th>
<th>Date</th>
<th>By Whom</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>WD  1</td>
<td>5 May 2004</td>
<td>Matt MacKenzie</td>
<td>Moved content over from 2.0/2.1 document source.</td>
</tr>
<tr>
<td>WD  2</td>
<td>14 May 2004</td>
<td>Matt MacKenzie</td>
<td>A few updates to the explanations and more thorough usage of available styles.</td>
</tr>
<tr>
<td>WD  3</td>
<td>1 Oct 2004</td>
<td>Matt MacKenzie</td>
<td>Integrated Reliable messaging, many editorial changes also.</td>
</tr>
<tr>
<td>WD  4</td>
<td>28 Sept 2005</td>
<td>Pete Wenzel</td>
<td>Applied OpenOffice Template, formatting changes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Messaging Model section from Jacques (was Section 6 in draft 03; is now Section 3).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Message Packaging section from Jacques (was Section 8; is now Section 5).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Security Module section from Ric (was Section 10.1; is now Section 6.1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Reliable Messaging Module section from Jacques (was Section 10.6; is now Section 6.6).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New WS-Reliability Binding section from Jacques (Section 9.1).</td>
</tr>
<tr>
<td>WD  5</td>
<td>05 Oct 2005</td>
<td>Pete Wenzel</td>
<td>Changed title to indicate this is Part 1.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Moved several sections to a new Part 2 (Advanced Features) document, for future reference.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rewritten Introduction &amp; Operation sections from Jacques.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Messaging Model and Message Packaging section updates from Jacques.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Began Bibliography Database and insertion of references.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Section rearrangement and edits as discussed 10/05/2005.</td>
</tr>
<tr>
<td>Rev</td>
<td>Date</td>
<td>By Whom</td>
<td>What</td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Security Section updates from Ric.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added Iwasa and Hamid as contributors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Packaging Diagrams from Iwasa &amp; Jacques.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Removed sections (to be considered for later Advanced Features document): FTP Binding, Security Services Profiles, WSDL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Minor updates throughout.</td>
</tr>
<tr>
<td>WD 07</td>
<td>23 Nov 2005</td>
<td>Pete Wenzel</td>
<td>This revision is a candidate for Committee Draft status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Editorial corrections to Introduction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Overhaul of Messaging Model and Error Sections by Jacques &amp; Hamid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Editorial corrections to Operation Context.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Message Pulling Module from Fujitsu.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Minor updates to Message Packaging.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Additional Security Examples, New Sign+Encrypt Sections from Ric.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Additional minor corrections throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>References, formatting, reorganization, other editorial changes throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Designated several of the later Sections as (Appendix).</td>
</tr>
<tr>
<td>CD 01</td>
<td>30 Nov 2005</td>
<td>Pete Wenzel</td>
<td>This revision has been voted Committee Draft status.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated Status statement and other standard boilerplate text on title page.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Changed incorrect “RMP” references to &quot;MSH&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated Figure 5 and removed corresponding EdNote.</td>
</tr>
<tr>
<td>Rev</td>
<td>Date</td>
<td>By Whom</td>
<td>What</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>WD 08</td>
<td>13 Feb 2006</td>
<td>Pete Wenzel</td>
<td>Replaced eb:Message by eb:Messaging.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Update Figures 7 &amp; 8.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Renumbered Section 5.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New Conformance Appendix, from Jacques’ Draft 0.7 (for continued review)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>New SOAP Format and Bindings Appendix draft from Hamid (for review)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Editorial updates to Reliability Binding Section from Jacques</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WS-ReliableMessaging Binding from Jacques (for review)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Completed Bibliography; removed many redundant references.</td>
</tr>
</tbody>
</table>
15 Notices (Appendix)

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’s procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.

Copyright © OASIS Open 2005-2006. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself does not have to be modified in any way, such as by removing the copyright notice and this paragraph are included on all such copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to translate it into languages other than English.

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

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

Version 3.0: Part 1, Core Features

## 1 Introduction

1.1 Background and Objectives

1.2 Scope

1.3 Caveats and Assumptions

1.4 General Rules for Normative Interpretation

1.5 Notation

## 2 Messaging Model

2.1 Terminology and Concepts

2.1.1 Components of the Model

2.1.2 Message Types

2.1.3 Messaging Roles

2.1.4 Abstract Messaging Operations

2.2 Message Exchange Patterns

2.2.1 Definition

2.2.2 Assumed SOAP Message Exchange Patterns

2.2.2.1 SOAP One-Way MEP

2.2.2.2 SOAP Request-Response MEP

2.2.3 Simple ebMS Message Exchange Patterns

2.2.3.1 The One-Way Push Message Exchange Pattern

2.2.3.2 The One-Way Pull Message Exchange Pattern

2.2.3.3 The Request-Reply Message Exchange Pattern

2.3 Message Pipes

2.3.1 Concept and Purpose

2.3.2 Definition and Usage Requirements

2.4 Transfer Modes

2.4.1 Definitions

2.4.2 Transfer Modes and MEPs

2.4.3 The Pulled Transfer Mode

## 3 Concept of Operation

3.1 Operation Mode and Contexts

3.2 Default Operation Mode

3.3 Message Service Processing Model

3.4 Examples of Supported Topologies
4 Message Pulling Module .................................................................................................................. 27

4.1 Objectives ................................................................................................................................ 27

4.2 Supporting the Pull Mode ........................................................................................................... 28

4.3 Combining Pulling with Security and Reliability ......................................................................... 30

5 Message Packaging .......................................................................................................................... 32

5.1 Message Envelope and Message Parts ......................................................................................... 32

5.1.1 MIME Structure and SOAP Profile .......................................................................................... 32

5.1.2 MIME Considerations .............................................................................................................. 35

5.1.2.1 Additional MIME Parameters .......................................................................................... 35

5.1.2.2 Reporting MIME Errors .................................................................................................. 35

5.1.2.3 XML Prolog ...................................................................................................................... 35

5.1.2.4 XML Declaration .............................................................................................................. 35

5.1.2.5 Encoding Declaration ........................................................................................................ 35

5.1.3 ebXML SOAP Envelope Extension ......................................................................................... 36

5.1.3.1 Namespace pseudo attribute .......................................................................................... 36

5.1.3.2 xsi:schemaLocation attribute ........................................................................................ 36

5.1.3.3 SOAP Header Element ...................................................................................................... 37

5.1.3.4 SOAP Body Element ......................................................................................................... 37

5.1.3.5 ebXML SOAP Extensions ............................................................................................... 37

5.1.3.6 id Attribute ....................................................................................................................... 37

5.1.3.7 Version Attribute .............................................................................................................. 37

5.1.3.8 SOAP mustUnderstand Attribute .................................................................................. 38

5.1.3.9 ebXML "Next MSH" role URI ......................................................................................... 38

5.1.3.10 ebXML "To Party MSH" role URI ................................................................................... 38

5.1.4 ebMS Header ............................................................................................................................ 38

5.1.5 Payload Containers .................................................................................................................. 39

5.2 The eb:Messaging Container Element ....................................................................................... 40

5.2.1 Element eb:Messaging/eb:UserMessage .................................................................................. 41

5.2.1.1 Element eb:Messaging/eb:UserMessage/eb:MessageInfo .............................................. 42

5.2.1.2 Element eb:Messaging/eb:UserMessage/eb:PartyInfo ................................................... 42

5.2.1.3 Element eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From ................................... 42

5.2.1.4 Element eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To ........................................ 43

5.2.1.5 Element eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId ......................................... 43

5.2.1.6 Element eb:Messaging/eb:UserMessage/eb:CollaborationInfo .................................... 43

5.2.1.7 Element eb:eb:CollaborationInfo/eb:AgreementRef ...................................................... 44

5.2.1.8 Element eb:UserMessage/eb:CollaborationInfo/eb:Service ......................................... 44

5.2.1.9 Element eb:UserMessage/eb:CollaborationInfo/eb:Action ............................................ 44
A message pipe can only transfer messages in one direction: from its assigned Sending MSH to the Receiving MSH.

A message placed in a pipe could fail to be transferred for various reasons (transport issue, security, intermediaries) and therefore could be removed from the pipe at any time.

There is no specific quality of service requirement associated with a pipe. Security and reliability remain associated with parties or with MSHs, in a way that is orthogonal to pipes, although an implementation is free to associate QoS with pipes.

Pipes are used to carry both user messages and signal messages. The Sending and Receiving endpoints mentioned above as pipe attributes respectively match the roles of Sending and Receiving MSHs defined for user messages, and simply identify the source and destination in case of signal messages.

An MSH is operating in knowledge of some contextual information that drives the processing of messages and of the content of their headers, whether sent or received. Such information defines the operation mode of the MSH. It may be construed as an agreement between communicating parties, or a subset of such an agreement that is relevant to the messaging layer. A reference to such an agreement may be present in the message header (see the eb:AgreementRef element in Section 5.2.1.7.)

The Operation Mode contains MSH input data that typically is not provided on a per-message basis, but that is common to a set of messages exchanged between two or more parties. To this extent, it may be interpreted as configuration data for an MSH implementation.

Although the representation and usage of Operation Mode data is out of scope of this specification, it may be useful to abstractly refer to major elements of this mode, in order to facilitate later bindings with more concrete representations. The Operation Mode may be partitioned into six main entities, called Operation Contexts. These are identified by names of the form OpCtx-<entityname>:

**OpCtx-Transport:** includes all transport related information that is necessary to achieve transport-level interoperability. This context data determines the type of transport involved (e.g. HTTP, SMTP, FTP) between two MSHs, and related configuration parameters. The scope of application is usually between two MSHs.

**OpCtx-Reliability:** includes all reliability contracts, or references to these, that will govern the reliability of messages exchanged. This includes protocol and timing parameters, etc. The scope of application is usually between two parties. This context data determines the content of the reliability headers.

**OpCtx-Security:** includes all security contracts, or references to these, including the security context and related resources (certificates, SAML assertions, etc.) that will govern the message exchange. The scope of application is usually between two parties, although it may address MSH-level security (e.g. for signal messages). This context data determines the content of the wsse:Security header.
**OpCtx-BusinessCollaboration**: includes all data related to a collaboration between two parties, and the binding to their back-end systems. This drives the following header content:

- `eb:UserMessage/eb:PartyInfo`
- `eb:UserMessage/eb:CollaborationInfo`
- `eb:UserMessage/eb:MessageProperties`

In particular, which MEPs and which message pipes are to be used by these parties, may be specified here.

**OpCtx-MessagePipes**: includes the identification of all message pipes and related attributes, in which an MSH plays a role, i.e. on which messages will be exchanged with other MSHs. This context data determines the possible values for the attribute:

- `eb:UserMessage/eb:MessageInfo/eb:PipeId`

It also controls the use of the PullRequest signal.

**OpCtx-ErrorHandling**: includes the reporting actions that are chosen for each ebMS Error, and related parameters (e.g. if using ebMS signal message, define the address of the destination MSH. May also decide which message pipe is to be used for reporting errors.) Also may include the policy chosen for raising ebMS Errors from the Errors generated by contained functional modules (Reliability, Security).

---

Except for sharing OpCtx-Transport, two MSH should be able to communicate without a need to share (out of band) operation context information prior to the exchange.

If the following conditions are satisfied when deploying a pair of Sending and Receiving MSHs:

- There is no security configuration preventing the exchange of non-secure messages (i.e. messages without wsse:Security header).
- There is a prior agreement about the transport protocol to be used, along with configuration details.
- There is an understanding that these MSHs will not be used exclusively in the Pulled transfer mode (see definition in Messaging Model section).

Then this pair of MSHs MUST be able to exchange messages that do not refer to any agreement (i.e. the `eb:AgreementRef` element is absent from `eb:MessageInfo` in the header). When `eb:AgreementRef` is absent, the exchange MUST use the Default Operation mode (or DO Mode). The Default Operation mode between two MSHs is characterized by the following Operation Contexts:

- **OpCtx-Reliability**: empty: no reliable messaging assumed (no reliability header will be present.)
- **OpCtx-Security**: empty: no secure messaging assumed (no security header will be present.)
- **OpCtx-MessagePipes**: empty: by default, only the default pipe is used, and its default transfer mode is Push mode, meaning it allows for carrying an ebMS One-Way Push MEP, or the first leg of a Request-Reply MEP.

**Note**

DO Mode is about interoperability between MSHs, not between applications. In order for two applications respectively associated with a Sender and a
Receiver MSH, to be able to exchange messages in DO Mode, they would likely need to share an additional agreement about **OpCtx-BusinessCollaboration**, so that these messages bind properly to the application or business process.

**Message Service Processing Model**

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

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

Figure 5 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 5: Component Relationships*

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.

**Application or SOAP Processor** - This is where the business logic for a message exchange / business process exists.

**Message Service Interface** - This is the interface through which messages are channeled between the MSH core and the ebXML Application.
**ebMS Packaging** - Handling, (de)enveloping and execution of Payload Services are performed by this module.

**Reliable Message Processing** - This module fulfills the Quality of Service requirements for a message.

**Web Services Security Processing** - Encryption/decryption of any SOAP message content and generation/verification of any digital signatures occurs in this module.

**Transport Bindings** - These are the actual transport bindings. This specification defines bindings for HTTP (Section 8.7) and SMTP (Section 8.8), and supports the addition of other protocols.

---

### Examples of Supported Topologies

An ebMS MSH may be used in the following topologies of messaging. In every one of these topologies, the MSH can either be implemented as a SOAP node, or as a chain of SOAP nodes that understand different headers: (a) a node that understands ebMS headers, (b) a node that understands reliability headers, and (c) a node that understands security headers.

**MSH as a messaging endpoint.** In this context, a sending MSH is a SOAP message originator. A receiving MSH is an ultimate SOAP receiver. In such a case, the Deliver operation consists of delivering the message to an application-level consumer, such as a business application, or a queuing system. The Submit operation consists of passing to the MSH the data and payload that are necessary so that it can generate an ebMS message.

**MSH as a SOAP Intermediary.** In this context, a sending MSH is acting as SOAP intermediary and just adding headers to a SOAP message it received (via Submit operation). For an MSH to be used in this context, it is recommended that no security header be present in the input SOAP message. At least, no signature other than detached XML signature shall be present, so that the addition of headers does not break signatures. A receiving MSH acting as SOAP intermediary will process and remove some headers (including ebMS headers, reliability and some security headers) and forward the SOAP message – which is no longer an ebMS message - to the next SOAP node.

**MSH as intermediary in a MSH multi-hop chain.** In this context, a sending MSH acting as an MSH intermediary is forwarding the ebMS messages it received (via Receive operation) toward another MSH (via operation Send). See the multi-hop module for the behavior of an MSH intermediary.

---

### MIME Considerations

**Additional MIME Parameters**

Any MIME part described by this specification MAY contain additional MIME headers in conformance with the MIME [RFC2045] 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 [SOAPATTACH].

### 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:

```xml
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 [XML10] 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 MIME root part 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 root part (see Section 5.1.4). 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 [XML10], 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 [XML10].

### Signing Messages

An ebMS Message is signed using Web Services Security following these steps:

1. Create a Security element as prescribed by Web Services Security
2. Create a BinarySecurityToken element specifying

```xml
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.1#X509v3"
```

Add the BinarySecurityToken as a child element of the Security element.
Create a KeyInfo element containing a SecurityTokenReference as prescribed in Web Services Security. The SecurityTokenReference should reference the BinarySecurityToken element created above.

Create a SignedInfo element with SignatureMethod, CanonicalizationMethod and Reference elements for the SOAP Envelope as prescribed by Web Services Security.

Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo as prescribed by Web Services Security.

Construct the Signature element that includes the SignedInfo, KeyInfo and SignatureValue elements as prescribed in Web Services Security. Include the Signature Element within the Security element created above.

Recommended use of a BinarySecurityToken with ValueType of x509-token-profile, EncodingType – Base64Binary.

The RECOMMENDED canonicalization method applied to the data to be signed is described in [XMLC14N].

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.

It is REQUIRED that compliant MSH implementations support Detached Signatures as defined by the XML Signature Specification [XMLDSIG].

It is RECOMMENDED that the eb:Messaging Container Element and the SOAP Body be included in the signature.

An MSH implementation MAY support Enveloped Signatures as defined by the XML Signature Specification. Enveloped Signatures add an additional level of security in preventing the addition of XML elements to the SOAP Header. Enveloped Signatures may limit the ability of intermediaries to process messages.

Example of digitally signed ebXML SOAP Message:

```
Mime-Version: 1.0
Content-Type: text/xml
Content-Transfer-Encoding: binary
SOAPAction: ""
Content-Length: 4871
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3c.org/2001/XMLSchema-instance">
  <soap:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
    <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
      <eb:UserMessage eb:syncresp="false">
        <eb:MessageInfo>
          <eb:Timestamp>2005-10-28T17:29:54.606Z</eb:Timestamp>
          <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
          <eb:RefToMessageId>UUID-1@msh-server.example.com</eb:RefToMessageId>
        </eb:MessageInfo>
        <eb:PartyInfo>
          <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
          <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
```
An ebMS Message that contains attachments is signed using Web Services Security and Web Services Security SOAP Messages with Attachments Profile following these steps:

- Create a Security element as prescribed by Web Services Security
- Create a BinarySecurityToken element specifying
  - EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
  - ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
- Add the BinarySecurityToken as a child element of the Security element.
- Create a KeyInfo element containing a SecurityTokenReference as prescribed in Web Services Security. The SecurityTokenReference should reference the BinarySecurityToken element created above.
- Create a SignedInfo element with SignatureMethod, CanonicalizationMethod and Reference elements for the SOAP Envelope as prescribed by Web Services Security. Reference elements for each attachment should be created as prescribed by Web Services Security SOAP Messages with Attachments Profile.
- Canonicalize and then calculate the SignatureValue over SignedInfo based on algorithms specified in SignedInfo as prescribed by Web Services Security.
- Construct the Signature element that includes the SignedInfo, KeyInfo and SignatureValue elements as presecribed in Web Services Security.
- Include the Signature Element within the Security element created above.

It is REQUIRED that compliant MSH implementations support the Attachment-Content-Only transform. It is RECOMMENDED that compliant MSH implementations support the Attachment-Complete transform.

It is RECOMMENDED that the eb:Messaging Container Element, and all MIME Body parts of included payloads are included in the signature.

Example of digitally signed ebXML SOAP with Attachments Message:

```xml
Mime-Version: 1.0
Content-Type: multipart/related; type="text/xml"; boundary="----Part_0.10369852.1130520597081"
SOAPAction: ""
Content-Length: 5163

------Part_0.10369852.1130520597081
Content-Type: text/xml
Content-transfer-Encoding: binary

<!--XML version="1.0" encoding="UTF-8"-->
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
            <eb:UserMessage eb:syncresp="false">
                <eb:MessageInfo>
                    <eb:Timestamp>2005-10-28T17:29:56.963Z</eb:Timestamp>
                </eb:MessageInfo>
            </eb:UserMessage>
        </eb:Messaging>
    </soap:Header>
    <soap:Body>
    </soap:Body>
</soap:Envelope>
```
Encrypting Messages

An ebMS Message is encrypted using Web Services Security following these steps:

- Create a Security element as prescribed by Web Services Security.
- Create a BinarySecurityToken element specifying
  - EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
  - ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
- Add the BinarySecurityToken as a child element of the Security element.
- Create an EncryptedKey as a child element of the Security Element. The EncryptedKey element SHOULD contain a DataReference to each EncryptedData element that was encrypted using that key. The EncryptedKey element SHOULD contain a KeyInfo element that contains a SecurityToken reference sub-element that references the attached BinarySecurityToken.
- Locate data items to be encrypted.
- Encrypt the data items as follows: For each XML element or element content within the target SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification [XMLENC]. Each selected original element or element content MUST be removed and replaced by the resulting EncryptedData element.
- Create a DataReference element referencing the generated EncryptedData elements. Add the created DataReference element to the ReferenceList.

An MSH Implementation may encrypt the eb:MessagingContainer Element. The eb:PartyInfo section may be used to aid in message routing before decryption has occurred. It is RECOMMENDED that the eb:PartyInfo section not be encrypted.

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

```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
```

This RECOMMENDED value SHALL be supported by all compliant ebXML Message
Example of an encrypted ebXML SOAP Message

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Header>
    <wsse:Security soap:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wsse:BinarySecurityToken
      EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      soap-message-security-1.0#Base64Binary"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      x509-token-profile-1.0#X509v3"
      wsu:Id="encryptionCert">...
    </wsse:Security>
  </soap:Header>
  <soap:Body>
    <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
      <eb:UserMessage eb:syncresp="false">
        <eb:MessageInfo>
          <eb:Timestamp>2005-10-28T17:29:57.720Z</eb:Timestamp>
          <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
          <eb:RefToMessageId>UUID-1@msh-server.example.com</eb:RefToMessageId>
        </eb:MessageInfo>
        <eb:PartyInfo>
          <eb:From>
            <eb:PartyId>uri:msh-server.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:PartyInfo>
        <eb:CollaborationInfo>
          <eb:AgreementRef>http://msh-server.example.com/cpa/123456</eb:AgreementRef>
          <eb:Service eb:type="someType">QuoteToCollect</eb:Service>
          <eb:Action>NewPurchaseOrder</eb:Action>
          <eb:ConversationId>54e98d3b-ed1a-4a70-997a-d6245d098912</eb:ConversationId>
        </eb:CollaborationInfo>
        <eb:MessageProperties>
          <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
          <eb:Property name="ContextID">987654321</eb:Property>
        </eb:MessageProperties>
        <eb:PayloadInfo>
          <eb:Partref id="ID19367511130520597724" eb:idref="cid:PO_Image@example.com">
            <eb:Description xml:lang="en-US">PO Image</eb:Description>
          </eb:Partref>
        </eb:PayloadInfo>
      </eb:UserMessage>
    </eb:Messaging>
  </soap:Body>
</soap:Envelope>
Encrypting SOAP with Attachments Messages

An ebMS Message is encrypted using Web Services Security and Web Services Security SOAP Messages with Attachments Profile following these steps:

2. Create a BinarySecurityToken element specifying

   EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary"
   ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"

4. Create an EncryptedKey as a child element of the Security Element. The EncryptedKey element SHOULD contain a DataReference to each EncryptedData element that was encrypted using that key. The EncryptedKey element SHOULD contain a KeyInfo element that contains a SecurityToken reference sub-element that references the attached BinarySecurityToken.
5. Locate data items to be encrypted.
6. When encrypting both attachments and primary SOAP envelope content perform the attachment processing first.
7. Encrypt each attachment part using XML Encryption according to the rules of XML Encryption.
8. Set the EncryptedData Type attribute to a URI that specifies what was encrypted (MIME content or the entire MIME part including headers).
9. Set the EncryptedData MimeType attribute to match the attachment MIME part Content-Type header before encryption.
10. Set the EncryptedData Encoding attribute to reflect the attachment content encoding, as visible to the security layer at the time of encryption.
11. Set the EncryptedData CipherReference to the same reference URL for the attachment that was used before encryption. This MUST be a CID scheme URL referring to the attachment part Content-ID. Ensure this MIME header is in the part conveying the cipher data after encryption.
Include the MIME Part Reference Transform in the CipherReference Transforms list.
Prepend the EncryptedData element to the Security element.
Update the attachment MIME part replacing the original content with the cipher text generated by the XML encryption step.
Update the attachment MIME part header MIME content-type and Content-Length appropriate to the cipher data.
Repeat steps for each additional attachment.
Encrypt the SOAP Header data items as follows: For each XML element or element content within the target SOAP envelope, encrypt it according to the processing rules of the XML Encryption specification [XMLENC]. Each selected original element or element content MUST be removed and replaced by the resulting EncryptedData element.
Create a DataReference element referencing the generated EncryptedData elements. Add the created DataReference element to the ReferenceList.

Example of an encrypted ebXML SOAP with Attachments Message:

|Mime-Version: 1.0|
|Content-Type: multipart/related; type="text/xml"; boundary="----Part_1_1804119.1130520598736"
|SOAPAction: ""
|Content-Length: 5172

------Part_1_1804119.1130520598736
|Content-Type: text/xml
|Content-Transfer-Encoding: binary

<xml version="1.0" encoding="UTF-8">  
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soap:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
<eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
<eb:UserMessage eb:syncresp="false">
<eb:MessageInfo>
<eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
<eb:RefToMessageId>UUID-1@msh-server.example.com</eb:RefToMessageId>
</eb:MessageInfo>
<eb:PartyInfo>
<eb:From>
<eb:PartyId>uri:msh-server.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:PartyInfo>
<eb:CollaborationInfo>
<eb:AgreementRef>http://msh-server.example.com/cpa/123456</eb:AgreementRef>
<eb:Service eb:type="someType">QuoteToCollect</eb:Service>
<eb:Action>NewPurchaseOrder</eb:Action>
<eb:ConversationId>37074bd0-2363-4a7e-ac53-cf5ce39ceed2</eb:ConversationId>
</eb:CollaborationInfo>
<eb:MessageProperties>
<eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
<eb:Property name="ContextID">987654321</eb:Property>
</eb:MessageProperties>
<eb:PayloadInfo>
<eb:Partref eb:id="ID1348572311305205987670" eb:idref="cid:PO_Image@example.com>
Signing and Encrypting Messages

An ebMS Message is signed and encrypted using Web Services Security. When both
signature and encryption are required of the MSH, sign first and then encrypt. To sign and encrypt an ebMS Message follow these steps:

Follow the steps to sign a message described in Section 6.2, above.
Follow the steps to encrypt a message as described in Section 6.4, above.
The EncryptedKey sub-element should be prepended to the existing Security element that was created when signing the message.

Example of a digitally signed and encrypted ebXML Message:

```xml
Mime-Version: 1.0
Content-Type: text/xml
Content-Transfer-Encoding: binary
SOAPAction: ""
Content-Length: 7205

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
            <eb:UserMessage eb:syncresp="false">
                <eb:MessageInfo>
                    <eb:From>
                        <eb:PartyId uri="msh-server.example.com"/>
                        <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:MessageInfo>
            </eb:UserMessage>
            <eb:MessageProperties>
                <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
                <eb:Property name="ContextID">987654321</eb:Property>
            </eb:MessageProperties>
            <eb:PayloadInfo>
                <eb:Part ref id="ID701611130780180659" eb:idref="#enc">
                    <eb:Description xml:lang="en-US">PO Image</eb:Description>
                </eb:Part>
            </eb:PayloadInfo>
        </eb:Messaging>
    </eb:Messaging>
</soap:Envelope>
```
<wsse:BinarySecurityToken
    EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    soap-message-security-1.0#Base64Binary"
    ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    x509-token-profile-1.0#X509v3"
    wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
<enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
    <enc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
    1_5" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-secext-1.0.xsd" />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <wsse:SecurityTokenReference>
            <wsse:Reference URI="#encryptionCert" />
        </wsse:SecurityTokenReference>
    </KeyInfo>
    <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
        <CipherValue>F3HmZ2Ldyn0umLCx/8Q9B9e8OosIJxJy9i9hOWQjh6JjJWqDldgd0QVF1Vt1LVjazlThS9m9rKrt
        pkhCUIY1yjFKtDsui1Aw8clLZv1HkVoDgt17rhzce8hY1IvESX9qZN65JgyAa3BYgW1ipjGHtNgZ9RzUdzKdeY74DF
        Fm27R6m8b0=</CipherValue>
    </CipherData>
    <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
        <DataReference URI="#enc" />
    </ReferenceList>
</enc:EncryptedKey>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
        <ds:Reference URI="#ebMessage">
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>Ae0PLUKJUnUyAMXxLQD/WwKiFiI=</ds:DigestValue>
        </ds:Reference>
        <ds:Reference URI="#body">
            <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>kNY6X7LnRTwxXXBzSw07tcA0KSu=</ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>
        T24oka0Muh5iBNNMG6tk8QAKZ1vPMvY1rcPnkOr9j3fHnRGM2qQn0yBydOTnC1cEMz2pblhhdN
        YZYMab1qa4N5ynLjw1k8p0uMP9hapij3Wf67aBnUeHlFMuau0x9DBQdKXzTmVa1QQg2106g
        j2YPDc3VZ1llLT2c80TfayGvU=</ds:SignatureValue>
    </ds:Signature>
</soap:Body>
</soap:Header>
<EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#Content"
    xmlns="http://www.w3.org/2001/04/xmlenc#">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
    cbc" />
    <CipherData>
Signing and Encrypting SOAP with Attachments

An ebMS Message is signed and encrypted using Web Services Security and Web Services Security SOAP Messages with Attachments Profile following these steps:

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

Follow the steps to sign a message described in Section 6.3, above.
Follow the steps to encrypt a message as described in Section 6.5, above.

The EncryptedKey sub-element should be prepended to the existing Security element that was created when signing the message.

Example of a digitally signed and encrypted ebXML SOAP with Attachments Message:
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>xUISuIg5eVxy3FL/4yCrZoEZrTM=</ds:DigestValue>
</ds:Reference>
<ds:Reference URI="cid:PO_Image">
<ds:Transforms>
</ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>R4hCV4K4I5Q2dSrP4KrLu46hFo=</ds:DigestValue>
</ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>
BGnJV/b7EUbAESn7GmNhZ8yYN6zoO6uz29E5r9GHxDW+MUH4wksqA654w+sB0rWl8xNranag3dhKhbaRERzYHDGq1VFIRqgExorHwzh4h7uoLX4yxOu6G9T/gily67Q3pENGpmVozwopphm/yd/A2T0+v4vso20aJsiieEIzSQ=</ds:SignatureValue>
<ds:KeyInfo>
<wsse:Reference URI="#signingCert"/>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</soap:Header>
<soap:Body/>
</soap:Envelope>

------- Part 2 6825397.1130520599536
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Id: <PO_Image@example.com>
Content-Description: WSS XML Encryption message; type="image/jpeg"

VEhmwb4FHhqQH8m5PKqVu8H0/bq2yUF

------- Part 2 6825397.1130520599536--
Communications Protocol Bindings (Appendix)

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 ebMS Messages and provide the underlying services necessary to carry out a complete ebMS 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 MSH 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 [HTTP11], in both asynchronous and synchronous forms of transfer.
- Simple Mail Transfer Protocol [SMTP], in asynchronous form of transfer only.
- File Transfer Protocol [FTP], in asynchronous form of transfer only.

HTTP

Minimum Level of HTTP Protocol

Hypertext Transfer Protocol Version 1.1 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 ebMS 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 HTTP 1.1 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 ebMS 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 recommended that sending MSHs act liberally in allowing the presence 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 ebMS 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:

**HTTP POST Example**

```plaintext
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"
>
<SOAP:Header>
<eb:Messaging eb:id="..." eb:version="3.0" SOAP:mustUnderstand="1">
```

HTTP Response Codes

In general, semantics of communicating over HTTP as specified in HTTP 1.1 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 Section8.7.3, HTTP Response Codes, and with an empty HTTP body MUST be returned in response to the HTTP POST.

**Access Control**

EdNote: This section to be completed in a future draft.

**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" [RFC2617] 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 [RFC2617], including support of the Basic Authentication mechanism, as described in its 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**

EdNote: This section to be completed in a future draft.

**Confidentiality and Transport Protocol Level Security**

An ebXML Message Service Handler MAY use transport layer encryption to protect
the confidentiality of ebMS Messages and HTTP transport headers. The IETF Transport Layer Security specification TLS [RFC2246] 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 [RFC2246].

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

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

Both TLS and SSL 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) [RFC2821] 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) [RFC2045], [RFC2046], [RFC2387].

SMTP Service Extension for Authentication [RFC2554].

SMTP Service Extension for Secure SMTP over TLS [RFC2487].

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 ebMS Message exchange over SMTP from the standpoint of a MTA.

**Minimum Level of Supported Protocols**

Simple Mail Transfer Protocol

MIME [RFC2045] and [RFC2046]
Sending ebXML Service Messages over SMTP

Prior to sending messages over SMTP an ebMS 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, 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 ebMS Message.

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

If using SMTP restricted transport paths, apply transfer encoding to all 8-bit data that will be transported in an ebMS Message, according to the encoding rules defined in section 6 of [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 ebMS Message Envelope MUST appear as an email MIME header.

All other MIME headers that constitute the ebMS 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 [RFC2821]-compliant email address of the ebXML Message Service Handler.

The email header "From:" MUST contain the [RFC2821]-compliant email address of the senders ebXML Message Service Handler.

Construct a "Date:" email header in accordance with [RFC2821].

Other headers MAY occur within the email message header in accordance with [RFC2821] and [RFC2045], however ebXML Message Service Handlers MAY choose to ignore them.

The example below shows a minimal example of an email message containing an ebMS Message:

From: ebXMLhandler@example.com
To: ebXMLhandler@example2.com
This is an ebXML SMTP Example

<?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://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:foop:CompressionSvc">
              <eb:Parameter eb:name="command" eb:value="uncompress"/>
              <eb:Parameter eb:name="algorithm" eb:value="gzip"/>
            </eb:Step>
          </eb:Processing>
        </eb:Payload>
      </eb:PayloadInfo>
    </eb:MessageHeader>
  </SOAP:Header>
</SOAP:Envelope>
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 ebMS 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 ebMS 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 ebMS 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 14: 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.
Conformance (Appendix)

Introduction

This section introduces the notion of conformance profiles for MSH implementations. It distinguishes a “main” conformance profile. It also outlines a common structure and syntax for defining additional “specialty” conformance profiles. The notion of conformance profile used here is defined in more details in [QAFW].

Note: conformance profile should not be confused with usage profile:

Conformance profile: defines a set of capabilities that an MSH implementation must have. This is determined at development time regardless of the way the MSH is being used later.

Usage profile: defines a way of using an MSH implementation, that a community of users has agreed upon. This may in turn require a particular conformance profile.

For example, a conformance profile may require that an MSH supports the optional MessageProperties header element, meaning it is able to extract it from a received message or to add it to a message to be sent. In contrast, a usage profile will in addition require that some specific property name be present in each message.

The interpretation of normative material follows the general rule below, as a complement to RFC2119:

A conforming implementation of this specification 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 [RFC2119]. When these keywords apply to message contents that relate to a more general feature, an implementation that conforms to a profile requiring support for this feature MUST be capable of processing these optional message contents according to the described ebXML semantics.

Terminology

A conformance profile is primarily associated with a common type of deployment or usage of an MSH implementation. It identifies a set of features that must be implemented in order for an MSH to support this type of deployment.

A conformance profile for ebMS is expressed as a combination of the following properties:

role

deployment type

level

interoperability parameters

Role: This property refers to one of the two roles identified in the messaging model: Sending and Receiving.
**Deployment Type:** A deployment type characterizes a context in which the implementation operates and the expected functional use for this implementation. Two deployment types are expected to be among the most common, non exclusive from others:

"resource-constrained handler". This characterizes an implementation that generally is not always connected, may not be directly addressable - may have no static IP address-, has limited persistent capability, and is not subject to high volume traffic.

"enterprise gateway". This characterizes an implementation that generally is acting as the B2B gateway for an enterprise. It has a fixed address. It may have connectivity restrictions due to security. It must support various types of connectivity with diverse partners.

**Level:** This property represents a level of capability for this conformance profile, expressed as a positive integer (starting from 1). All other properties being equal, an implementation that is conforming to a profile at level N (with N>1) is also conforming to the same profile at level N-1.

**Interoperability parameters:** This property is a composed property. It is a vector of parameters that must (in general) be similar pairwise between two implementations in order for these to interoperate. Two parameters are identified here, not exclusive from others:

The transport protocol supported, for which a non-exhaustive list of values is: HTTP, SMTP, HTTPS.

The reliability specification supported, either WS-Reliability or WS-ReliableMessaging.

**Conformance Profile:** A conformance profile is then fully identified by a quadruple < Role / DeploymentType / Level / InteropParameters>, or <R / D / L / P> which is called the profile summary.

**Functional Aspect:** A conformance profile will impose specific requirements on different aspects of the specification, that are called here functional aspects. A set of (non exhaustive) functional aspects is: MEP, Error reporting, Reliability, Security, Message pipes, Message packaging, transport.

**Profile Feature Set:** The set of specification requirements associated with a conformance profile. This set is partitioned using the functional aspects listed for the specification: it can be expressed as a list of functional aspects, annotated with the required features for each aspect.

**Main Conformance Profile**

This specification defines a main conformance profile, named MCP, that is to be considered as the baseline for conducting electronic business. The MCP addresses the requirements of most enterprise e-Business gateways.

Additional profiles – either weaker or stronger in terms of supported features – may be defined in order to allow for MSH implementations that align more closely to specific messaging needs, e.g. as required by a particular market. It is also expected that user communities will generate variants of the MCP that differ by their interoperability parameters, e.g. a variant that uses a transport other than HTTP. Such additional conformance profiles are called "specialty conformance profiles". The
next section defines some of these.

The main conformance profile may evolve over time to reflect an evolution of the
enterprise gateway requirements among the user community.

<table>
<thead>
<tr>
<th>Main Conformance Profile: MCP</th>
<th>Profile summary: &lt;“Sending+Receiving” / “enterprise gateway” / Level 1 / HTTP1.0 + WS-Reliability 1.1 &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional Aspects</td>
<td>Profile Feature Set</td>
</tr>
<tr>
<td>ebMS MEP</td>
<td>Support for all ebMS simple MEPS: One-way Push, Request-reply, One-way Pull.</td>
</tr>
<tr>
<td>Reliability</td>
<td>Support for the following QoS features for pushed or pulled ebMS messages: at-least-once, at-most-once, exactly-once. Ability to acknowledge pulled messages.</td>
</tr>
<tr>
<td>Security</td>
<td>Support for username / password token, digital signatures and encryption. No support for security of attachments required (WSS1.0 only).</td>
</tr>
<tr>
<td>Error reporting</td>
<td>Capability to report errors from message processing on the receiving side, either as ebMS error messages or as Faults to the Sending MSH. Capability to report errors generated on Sending side, as notifications to the message producer (e.g. delivery failure). On receiver side, no requirement to generate error EBMS:0010 for discrepancies between message header and the processing mode features: P-Mode.reliability and P-Mode.security, but requirement to generate such error for other discrepancies.</td>
</tr>
<tr>
<td>Message pipes</td>
<td>Sending role: support for additional pipes to default pipe required, so that selective pulling by a partner MSH is possible.</td>
</tr>
<tr>
<td>Message packaging</td>
<td>Support for attachments required, support for MessageProperties required.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interop.</th>
<th>Transport</th>
<th>SOAP</th>
<th>Reliability specification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP 1.0</td>
<td>SOAP 1.1</td>
<td>WS-Reliability 1.1. Only &quot;Response&quot; or &quot;Callback&quot; ReplyPattern values are required to be supported.</td>
</tr>
</tbody>
</table>

### Some Specialty Conformance Profiles

An MSH implementation may be required to conform to some of the profiles below instead of the MCP. Other specialty conformance profiles may be defined by and for a community of users in case particular usage conditions are not well addressed by the MCP.

It is recommended to identify conformance profiles with short identifiers such as CS1, CR1 defined in this section. For example, **CS1** means [Resource-]Constrained handler, **Sending role, Level 1**. If an “Enterprise Gateway” profile more advanced than MCP needs to be defined, it could be called **EG2** (the MCP being Level 1).
### Conformance Profile: CS1
Profile summary: "Sending" / "constrained handler" / Level 1 / HTTP1.0 + WS-Reliability 1.1

<table>
<thead>
<tr>
<th>Functional Aspects</th>
<th>Profile Feature Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebMS MEP</td>
<td>Support for One-way Push (as initiator).</td>
</tr>
<tr>
<td>Reliability</td>
<td>Support for guaranteed delivery: must be able to receive reliability acks on the SOAP response to the Push, and to resend.</td>
</tr>
<tr>
<td>Security</td>
<td>Support for username / password token</td>
</tr>
<tr>
<td>Error reporting</td>
<td>Support for error notification to the message producer (e.g. delivery failure).</td>
</tr>
<tr>
<td>Message pipes</td>
<td>Sending on default pipe only (no support for additional pipes required.)</td>
</tr>
<tr>
<td>Message packaging</td>
<td>No support for attachments required, no support for MessageProperties required</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interop</th>
<th>Transport</th>
<th>SOAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP 1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOAP 1.1</td>
<td></td>
</tr>
<tr>
<td>Reliability specification</td>
<td>WS-Reliability 1.1</td>
<td></td>
</tr>
</tbody>
</table>

### Conformance Profile: CR1
Profile summary: "Receiving" / "constrained handler" / Level 1 / HTTP1.0

<table>
<thead>
<tr>
<th>Functional Aspects</th>
<th>Profile Feature Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebMS MEP</td>
<td>Support for One-way Pull (initiator)</td>
</tr>
<tr>
<td>Reliability</td>
<td>none.</td>
</tr>
<tr>
<td>Security</td>
<td>Support for username / password token (for Pull signal, and for received messages)</td>
</tr>
<tr>
<td>Error reporting</td>
<td>Support for generating errors associated with receiving user messages. Support for notifying its own party, and for reporting to the sender via ebMS message.</td>
</tr>
<tr>
<td>Message pipes</td>
<td>Ability to pull from different pipes, in addition to the default pipe.</td>
</tr>
<tr>
<td>Message packaging</td>
<td>No support for attachments required, no support for MessageProperties required.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interop</th>
<th>Transport</th>
<th>SOAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HTTP 1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOAP 1.1</td>
<td></td>
</tr>
<tr>
<td>Reliability specification</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>