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 http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=ebxml-msg.

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 (http://www.oasis-open.org/committees/ebxml-msg/ipr.php).
# Table of Contents

- **OASIS ebXML Messaging Services** ............................................................... 1
- **Version 3.0: Part 1, Core Features** ......................................................... 1

## 1 Introduction .............................................................................................. 6
  - 1.1 Background and Objectives .................................................................. 6
  - 1.2 Scope .................................................................................................... 6
  - 1.3 Caveats and Assumptions .................................................................... 7
  - 1.4 General Rules for Normative Interpretation ......................................... 7
  - 1.5 Notation ............................................................................................... 8

## 2 Messaging Model ....................................................................................... 9
  - 2.1 Terminology and Concepts .................................................................. 9
  - 2.2 Message Exchange Patterns ............................................................... 11
  - 2.3 Message Boxes .................................................................................... 14
  - 2.4 Processing Model ................................................................................ 18

## 3 Processing Modes ..................................................................................... 20
  - 3.1 Processing Mode Features ................................................................... 20
  - 3.2 Default Features .................................................................................. 21

## 4 Message Pulling Module ........................................................................... 22
  - 4.1 Objectives ........................................................................................... 22
  - 4.2 Supporting the Pull Mode ..................................................................... 23
  - 4.3 Combining Pulling with Security and Reliability ................................. 24

## 5 Message Packaging ................................................................................... 27
  - 5.1 Message Envelope and Message Parts ................................................ 27
  - 5.2 The eb:Messaging Container Element ................................................ 36

## 6 Security Module ......................................................................................... 44
  - 6.1 Security Element ................................................................................ 44
  - 6.2 Securing the PullRequest Signal ........................................................ 44
  - 6.3 Countermeasure Technologies ............................................................ 46
  - 6.4 Security Considerations ...................................................................... 47

## 7 Error Handling Module ............................................................................ 49
  - 7.1 Terminology ........................................................................................ 49
  - 7.2 Packaging of ebMS Errors ................................................................... 49
  - 7.3 ebMS Error Message .......................................................................... 51
  - 7.4 Extensibility of the Error Module ......................................................... 51
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5</td>
<td>Generating ebMS Errors</td>
</tr>
<tr>
<td>7.6</td>
<td>Error Reporting</td>
</tr>
<tr>
<td>7.7</td>
<td>Standard ebMS Errors</td>
</tr>
<tr>
<td>8</td>
<td>Reliable Messaging Module</td>
</tr>
<tr>
<td>8.1</td>
<td>The Reliable Messaging Model</td>
</tr>
<tr>
<td>8.2</td>
<td>Reliability of ebMS Messages</td>
</tr>
<tr>
<td>8.3</td>
<td>Reliability of ebMS MEPs</td>
</tr>
<tr>
<td>9</td>
<td>The ebXML SOAP Extension Schema (Appendix)</td>
</tr>
<tr>
<td>10</td>
<td>Reliable Messaging Model</td>
</tr>
<tr>
<td>10.1</td>
<td>WS-Reliability Binding</td>
</tr>
<tr>
<td>10.2</td>
<td>WS-ReliableMessaging Binding</td>
</tr>
<tr>
<td>11</td>
<td>SOAP Format and Bindings (Appendix)</td>
</tr>
<tr>
<td>11.1</td>
<td>Using SwA with SOAP-1.1</td>
</tr>
<tr>
<td>11.2</td>
<td>Using SwA with SOAP-1.2</td>
</tr>
<tr>
<td>12</td>
<td>References (Appendix)</td>
</tr>
<tr>
<td>12.1</td>
<td>Normative References</td>
</tr>
<tr>
<td>13</td>
<td>Acknowledgments (Appendix)</td>
</tr>
<tr>
<td>14</td>
<td>Revision History (Appendix)</td>
</tr>
<tr>
<td>15</td>
<td>Notices (Appendix)</td>
</tr>
</tbody>
</table>

Copyright © OASIS Open 2005-2006. All Rights Reserved.
## 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>10</td>
</tr>
<tr>
<td>2</td>
<td>One-Way Push MEP</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>One-Way Pull MEP</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>Request-Reply MEP</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>Component Relationships</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>One-Way Pull with Message Boxes</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>User Message Structure</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>Signal Message Structure</td>
<td>29</td>
</tr>
<tr>
<td>9</td>
<td>Reliable Request</td>
<td>56</td>
</tr>
<tr>
<td>10</td>
<td>Reliable Response</td>
<td>57</td>
</tr>
<tr>
<td>11</td>
<td>Reliable One-Way Push MEP</td>
<td>60</td>
</tr>
<tr>
<td>12</td>
<td>Reliable One-Way Pull MEP</td>
<td>61</td>
</tr>
<tr>
<td>13</td>
<td>Reliable Request-Reply MEP</td>
<td>62</td>
</tr>
</tbody>
</table>

**Deleted:**
- Figure 1: Entities of the Messaging Model and their Interaction 10
- Figure 2: One-Way Push MEP 13
- Figure 3: One-Way Pull MEP 14
- Figure 4: Request-Reply MEP 15
- Figure 5: Component Relationships 18
- Figure 6: One-Way Pull with Message Boxes 23
- Figure 7: User Message Structure 28
- Figure 8: Signal Message Structure 29
- Figure 9: Reliable Request 56
- Figure 10: Reliable Response 57
- Figure 11: Reliable One-Way Push MEP 60
- Figure 12: Reliable One-Way Pull MEP 61
- Figure 13: Reliable Request-Reply MEP 62
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
Handling of unsupported features SHALL be implemented in accordance with the prescribed failure mechanism defined for the feature.

1.5 Notation

When describing concrete XML schemas, this specification uses a convention where each member of an element’s [children] or [attributes] property is described using an XPath-like notation (e.g., /x:MyHeader/x:SomeProperty/@value1). The use of @{any} indicates the presence of an attribute wildcard (<xs:anyAttribute/>). The use of @(any) indicates the presence of an element wildcard (<xs:any/>).

### Prefix Namespaces:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>S11</td>
<td><a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a></td>
</tr>
<tr>
<td>S12</td>
<td><a href="http://www.w3.org/2003/05/soap-envelope">http://www.w3.org/2003/05/soap-envelope</a></td>
</tr>
<tr>
<td>eb</td>
<td><a href="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</a></td>
</tr>
<tr>
<td>wsr</td>
<td><a href="http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd">http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd</a></td>
</tr>
<tr>
<td>wsrx</td>
<td><a href="http://docs.oasis-open.org/ws-rx/wsrm/200602">http://docs.oasis-open.org/ws-rx/wsrm/200602</a></td>
</tr>
<tr>
<td>wsse</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd</a></td>
</tr>
<tr>
<td>wsu</td>
<td><a href="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd</a></td>
</tr>
<tr>
<td>wsswa</td>
<td><a href="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0.xsd">http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0.xsd</a></td>
</tr>
</tbody>
</table>

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

Deleted: Also, table of namespaces used in examples would be useful.

Deleted: ¶

Formatted: Definition term

Formatted: Table Heading

Formatted: Table Contents

Formatted: Table

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Table Contents

Formatted: Font: Arial, Font color: Black, English U.S.

Formatted: Table Contents

Formatted: Font: (Default) ArialMT, Complex Script Font: ArialMT, English U.S.

Formatted: Table Contents

Formatted: Table Contents

Formatted: English U.S.
2 Messaging Model

2.1 Terminology and Concepts

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:** The arrows in all figures 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 on either component.
### 2.1.2 Message Types

An ebMS MSH component must be able to exchange the following types of messages:

- **ebMS Message**: 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), and intended to a Consumer entity (via Deliver). 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**, is represented by the XML infoset `eb:Messaging/eb:UserMessage`, together with referenced payload items, and
- **an ebMS Signal Message Unit**, is 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 with 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**: The operation transfers enough message data from the producer to the Sending MSH to generate an ebMS User Message Unit.
- **Deliver**: The operation makes data of a previously received (via Receive operation) ebMS User Message Unit available to the Consumer.
- **Notify**: The 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**: The 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).
• **Receive:** The 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, 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. The ebMS MEPs defined here involve at least one ebMS User Message Unit, and take place between a pair of MSHs.

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.

The concept of SOAP MEP has only been introduced with SOAP 1.2, although this concept may also 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:

**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 (first leg if a two-way protocol).
- The response message, in case a two-way underlying protocol is used, either does not contain a SOAP envelope (in which case the SOAP One-Way MEP is qualified as "strict") or contains a SOAP envelope with an empty SOAP body, or contains a SOAP envelope with a SOAP Fault in the body (in which case the MEP is qualified as "robust").

**SOAP Request-Response MEP:**

The full definition of this MEP can be found in SOAP-1.2 Part 1, Adjunct. From an MSH perspective,
support for this MEP assumes the following:

- The Initiator MSH can initiate the sending of a SOAP envelope (an ebMS Message) over a two-way underlying protocol.
- The Responding MSH can send back a message with a SOAP envelope (an ebMS Message) over a response of the underlying protocol.

An ebMS MEP is of two kinds:

- 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 subsections define three simple ebMS MEPs: One-Way Push, One-Way Pull, and Request-Reply.

### 2.2.3 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 or
- 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**

![One-Way Push MEP Diagram](image)

**Figure 2: One-Way Push MEP**

### 2.2.4 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 “EBMS:0006”. Figure 3 illustrates the exchange pattern and operations involved for this MEP.

![One-Way Pull MEP](image)

**Figure 3: One-Way Pull MEP**

### 2.2.5 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:RefToMessageId header element, as described in Section 5.2.1.1). Figure 4 illustrates the exchange pattern and operations involved for this MEP.
2.3 Message Boxes

2.3.1 Concept and Purpose

A message box (mbox for short) defines a logical concept for labeling User Messages between Sending MSH and Receiving MSH. Several message boxes may be used 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.

Before it is transferred from a Sending MSH to a Receiving MSH, regardless whether pushed or pulled, a User Message is always assigned to an mbox. If no explicit assignment is requested, the default mbox is assigned.

An mbox is an abstract concept (much like eb:Service and eb:Action parameters are abstract parameters) and an MSH is free to give it its own special interpretation at implementation level. The following are two examples of such interpretation (although not the only ones possible):

- Setting transfer priorities: some User Messages may be transferred with higher priority than others regardless in which order they all have been submitted.
- Organizing the inflow of messages on the Receiving MSH. User Messages from different origin may be assigned to the same mbox, or messages from a Sending MSH to a Receiving MSH may be assigned to different mboxes. These mboxes may be associated with different business meaning and processing conditions on Consumer side.

When a message is assigned an mbox on sending side, it is still associated with this mbox on receiving side. In doing so the message box becomes an attribute of the message flow between two MSHs. Because mboxes effectively define flows of messages that can be controlled differently and independently from each other, they can be seen as defining virtual “pipes” that carry as many flows of messages between a Sending MSH and a Receiving MSH.

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

Support for assigning User Messages to mboxes – e.g. by automatically mapping User Messages submitted by a Producer to a particular mbox based on some rules, queries or filters – is out of scope of this specification. Similarly, there is no requirement on what criteria (e.g. query expression, FIFO policy) can be used to select User Messages when pulling from an mbox. This specification only describes the properties of message boxes, and how their use affects the message protocol. It does not prescribe a particular way to implement message boxes or to use them.

There is no requirement for ordering User Messages assigned to a given mbox. Neither the send/receive pair of operations nor the submit/deliver pair are required to be executed on a sequence of messages of the same mbox in the same order – unless specified otherwise by the reliability requirements. An implementation may choose to implement boxes using queues and FIFO policy though does not have to.

The transfer of User Messages assigned to an mbox is controlled by the MEP associated with these messages. On the sender side, User Messages assigned to the same mbox can be pulled and/or pushed, based on the MEP or P-Mode each one of these messages has been declared to participate in. Similarly, on the receiver side, an mbox could be used for both pulled and pushed messages.

A P-Mode (which includes quality of service) is orthogonal to mboxes. In other terms, a P-Mode may declare a set of mboxes and two different P-Modes may share some common mboxes. An implementation however, is free to choose that each P-Mode declare only one mbox (in this way, mboxes would be associated with a quality of service and would have a one-to-one relationship with P-Modes).

When using reliable messaging, if the message ordering reliability contract (In-Order) is requested, all user messages intended to the same ordered sequence between 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 Sending MSH.

Mboxes are also heavily used by PullRequest messages. A PullRequest signal specifies via its attribute eb:PullRequest/@eb: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.

2.3.3 Simple Use-Cases

There are 9 simple use-cases when using message boxes to transfer User Messages. These simple use-cases involve two or three MSHs only. Other use-cases could be formed by aggregating the simple use-cases and involving more than three MSHs. The following describes these 9 simple use-cases:

Use-Case 1:

In this case, one MSH is pushing a User Message intended for a given mbox
Use-Case 2:

In this case, one MSH is pulling a User Message from a given mbox.

Use-Case 3:

Two different MSHs pushing User Messages to a common MSH, intended for the same mbox.

Use-Case 4:

One MSH pulling User Messages from a second MSH, while at the same time receiving pushed User Messages from a third MSH, all intended for the same mbox.

Use-Case 5:

One MSH pushing User Messages from one mbox to two different MSHs.
Use-Case 6:

One MSH is pulling User Messages from a second MSH from a given mbox while this second MSH is pushing User Messages to a third MSH from the same mbox.

Use-Case 7:

One MSH is pulling User Messages from two different other MSHs, all intented for the same mbox.

Use-Case 8:

One MSH is pushing User Messages to a second MSH intended for a given mbox, while this second MSH is pulling User Messages from the first MSH intended for the same mbox.

Use-Case 9:

Two different MSHs are pulling User Messages from a third MSH from the same mbox.
This last use-case is a little special in the sense that it is not always feasible. This case is feasible only when the two MSHs that are pulling are serving the same application (meaning that they have the same eb:To Party, same eb:Service, and eb:Action). This is because on the sender side when an mbox is used to be pulled from, that mbox is associated uniquely with information about the parties behind the MSH who will be sending PullRequests. Since a pulled message is a User Message, this User Message needs to have eb:To Party, eb:Service, and eb:Action in its headers. This information is uniquely associated with an mbox prior to pulling. When a PullRequest is received, there is nothing in the protocol about this information, and that’s why the above use-case is feasible only when the two pulling MSHs are serving the same application (same eb:To Party, eb:Service, and eb:Action).

2.4 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.
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, 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), businesscollaboration, 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
• 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
override by application input.
  • eb:UserMessage/eb:PartyInfo: The eb:From element contains a PartyId
  The eb:To element contains a PartyId with value: http://www.oasis-
  open.org/committee/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:

- **Pulled-transfer 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 mboxes. 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.

**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:**
5. 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.

6. Send: the selected message is sent over the SOAP Response to the PullRequest.

7. **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:

```xml
<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:MessageInfo>
        <eb:PullRequest forMbox="pipe://mymsh.com/pipe123"/>
      </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:

```xml
<SOAP:Envelope>
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:UserMessage eb:mbox="pipe://mymsh.com/pipe123">
        <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:MessageInfo>
        <eb:PartyInfo>
          ...
        </eb:PartyInfo>
        <eb:CollaborationInfo>
          ...
        </eb:CollaborationInfo>
        <eb:PayloadInfo>
          ...
        </eb:PayloadInfo>
      </eb:UserMessage>
    </eb:Messaging>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```

### 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:

```xml
<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:MessageInfo>
        <eb:PullRequest forMbox="pipe://mymsh.com/pipe123"/>
      </eb:SignalMessage>
    </eb:Messaging>
    <wss:Security>
      ...
    </wss:Security>
    <wsr:Request SOAP:mustUnderstand="1">
      <ReplyPattern>
        <Value>Response</Value>
      </ReplyPattern>
      <AckRequested/>
    </wsr:Request>
  </SOAP:Header>
  <SOAP:Body/>
</SOAP:Envelope>
```

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

```xml
<SOAP:Envelope>
  <SOAP:Header>
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:UserMessage eb:mbox="pipe://mymsh.com/pipe123">
        <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:MessageInfo>
        <eb:PartyInfo/>
        <eb:CollaborationInfo/>
        <eb:PayloadInfo/>
      </eb:UserMessage>
    </eb:Messaging>
    <wss:Security>
      ...
    </wss:Security>
    <wsr:Response SOAP:mustUnderstand="1">
      ...
    </wsr:Response>
  </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 wrs: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
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 Multpart/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**

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

Figure 8: Signal Message Structure
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 Request User Message packaged in a SOAP-1.1 message:

```xml
  <SOAP-ENV: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: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-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Deleted:
```
<eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>[
```
The following is an example of a Response User Message:

```xml
 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:UserMessage eb:box="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: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>
  </S11:Header>
</S11:Envelope>
```

5.1.2.2 PullRequest Message Example

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

```xml
 xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <S11: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:PullRequest eb:forMbox="mbox-32598.2345.345" /> 
      </eb:SignalMessage>
    </eb:Messaging>
  </S11:Header>
</S11:Envelope>
```

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

```xml
 xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <S11:Header>
    <eb:Messaging eb:version="3.0">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
          <eb:MessageId>UUID-2@example.com</eb:MessageId>
          <eb:PullRequest eb:forMbox="mbox-32598.2345.345" />
        </eb:MessageInfo>
      </eb:SignalMessage>
    </eb:Messaging>
  </S11:Header>
</S11:Envelope>
```
5.1.2.3 Error Message Example

The following is an example an Error Signal Message:

```
<envelope xmlns:envelope="http://schemas.xmlsoap.org/soap/envelope/*"
    xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
  <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>
  </header>
</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:

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

### 5.1.3.2 xsi:schemaLocation attribute

The SOAP namespace:

```
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:MessageInfo>
                <eb:PayloadInfo>...</eb:PayloadInfo>
                </eb:UserMessage>
            </eb:Messaging>
        </SOAP:Header>
    </SOAP:Body>
</SOAP:Envelope>
```
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 that receives a message with an ebXML SOAP extension qualified with a version other than "3.0" MAY process the message if it recognizes the version identified and is capable of processing it. It MUST respond with an error (details TBD) if it does not recognize the identified version. The version attribute MUST be namespace qualified for the 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

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.
header MAY contain a "charset" attribute. For example:

```plaintext
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:

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

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
  http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header
    xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
    xsi:schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
    <eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      ...
      <eb:PayloadInfo>
        ...
      </eb:PayloadInfo>
      ...
    </eb:Messaging>
  </SOAP:Header>
</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: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 is a REQUIRED element for a Request User Message and optional for a Response User Message. It 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:From**: The **REQUIRED** element occurs once, and contains information describing the originating party.

- **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 [ASC105] 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:

- eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef: This 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.
- eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service: This 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.
• `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action` : This REQUIRED element occurs once. The element is a string identifying an operation or an activity within a Service that may support several of these.

• `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationID` : This 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: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:

```
<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, or an individual or enterprise.

An example of the Service element follows:

```
<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.

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)
• 
  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 href="cid:foo@example.com"/>
</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.

Example:

```xml
<eb:PayloadInfo>
  <eb:PartInfo href="cid:foo@example.com"/>
</eb:PayloadInfo>
```
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/[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).

Deleted: <#>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 specification and the Web Services Security SOAP Messages with Attachments Profile.

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 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, even though all the examples in WSS-1.1 are only about SOAP-1.1 when securing attachments). In other terms, an MSH may 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:

```
<wsse:Security xmlns:wsse="…" xmlns:wsu="…">
  <wsse:UsernameToken wsu:Id="#TokenID">
    <wsse:Username>hamid</wsse:Username>
    <wsse:Password>Password</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:

```
<wsse:Security soap:mustUnderstand="1" xmlns:wsse="…" xmlns:wsu="…">
  <wsse:BinarySecurityToken wsu:Id="signingCert"
    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">
    ...
  </wsse:BinarySecurityToken>
<eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
  <eb:SignalMessage>
    <wsse:Security>
      <wsse:Reference URI="#signingCert" />
    </wsse:Security>
  </eb:SignalMessage>
</wsse:SecurityTokenReference>
```
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 in 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 a 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


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)
- refToMessageInError (optional attribute)
- shortDescription (optional attribute)
- Description (optional element)
- errorDetail (optional element)
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/@refToMessageInError

This optional attribute indicates the messageId of the message in error for which this error is raised.

7.2.7 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.

7.2.8 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.
7.2.9 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:"

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
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>Code</td>
<td>Description</td>
<td>Type</td>
<td>Context</td>
<td>Content</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------</td>
<td>--------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------</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>
<tr>
<td>EBMS:0008</td>
<td>FeatureNotSupported</td>
<td>failure</td>
<td>Unpackaging</td>
<td>although the message document is well formed and schema valid, the presence or absence of some element/attribute is not consistent with the capability of the MSH, with respect to supported features.</td>
</tr>
<tr>
<td>EBMS:0009</td>
<td>InvalidHeader</td>
<td>failure</td>
<td>Unpackaging</td>
<td>The ebMS header is either not well formed as an XML document, or does not conform to the ebMS packaging rules.</td>
</tr>
<tr>
<td>EBMS:0010</td>
<td>ProcessingModeMismatch</td>
<td>failure</td>
<td>Processing</td>
<td>The ebMS header is not compatible with expected content based on the associated P-Mode.</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.
### Error Code  Short Description  Recommended Severity  Category Value  Description or Semantics

<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>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:0202</td>
<td>DeliveryFailure</td>
<td>failure</td>
<td>communication</td>
<td>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.</td>
</tr>
<tr>
<td>EBMS:0201</td>
<td>DysfunctionalReliability</td>
<td>failure</td>
<td>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).

![Diagram of Reliable Request](image)

**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.
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/committe/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 (5): **RM-DeliverResponse**: after processing of reliability headers of the pulled message, delivery to the RM-Consumer.
- 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.
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 (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

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.
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"?>

...
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 Error! Reference source not found.).

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 1.1 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). |
| NoDuplicateDelivery   | "enabled", "disabled" | When enabled, the PullRequest signal message associated with this pulled message MUST also be sent with this parameter enabled. |
| OrderedDelivery       | "enabled", "disabled" | No restriction.                                                                          |
| ReplyPattern          | "Callback"        |                                                                                          |

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>
</table>
| GuaranteedDelivery    | "enabled", "disabled" | 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:
  • The ReplyPattern value associated with the PullRequest signal is "Response".
  • The NoDuplicateDelivery agreement item is also enabled for the Request.

The Responder WSR-RMP sends back a copy of the original Response message if the latter is not expired, when a duplicate of the Request is received, e.g. due to resending (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 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, SequenceClosed, 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.

Note: the WS-RM protocol always assumes acknowledgment of messages. Although acknowledgments
are unnecessary for the At-Most-Once reliability contract, the use of sequence numbers allows for an
efficient duplicate detection. It is then RECOMMENDED to use the WS-RM protocol for At-Most-Once.

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.

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

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 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
<?xml version='1.0' ?>
  <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>
```

---

Example 1: SOAP-1.1 with Attachment

11.2 Using SwA with SOAP-1.2

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:

```xml
<?xml version='1.0' ?>
  <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>
```
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:

HTTP headers when using SOAP 1.1 with attachments

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

HTTP headers when using SOAP 1.2 with attachments

Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
start="<car-data@toyoya.com>; action=leasing"
HTTP headers when using SOAP 1.2 with attachments

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:

SMTP headers when using SOAP 1.1 with attachments

SMTP headers when using SOAP 1.2 with attachments
12 References (Appendix)

12.1 Normative References

[ASCI05] Unknown, ASC X12 Registry, ????. <url-unknown>
<http://www.ietf.org/rfc/rfc2821.txt>


<http://www.ietf.org/rfc/rfc2821.txt>

<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

<http://www.w3.org/TR/soap12-part1/>

<http://www.w3.org/TR/SOAP-attachments>

<http://wp.netscape.com/eng/ssl3/draft302.txt>

<http://www.ietf.org/rfc/rfc2279.txt>


<http://www.w3.org/TR/xlink/>
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 01</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 02</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 03</td>
<td>1 Oct 2004</td>
<td>Matt MacKenzie</td>
<td>Integrated Reliable messaging, many editorial changes also.</td>
</tr>
<tr>
<td>WD 04</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 05</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 &quot;RMP&quot; 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 be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed for the purpose of developing OASIS specifications, in which case the procedures for 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.
Transfer Modes

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 `eb:PullRequest/@forMbox` attribute.

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.
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).

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.