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

Committee Draft 02, 12 April 2006

Document identifier:
ebms_core-3.0-spec-cd-02

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

Technical Committee:
OASIS ebXML Messaging Services TC

Chair:
Ian Jones, British Telecom <ian.c.jones@bt.com>

Editors:
Matthew MacKenzie, Adobe Systems Incorporated <mattm@adobe.com>
Jeff Turpin, Cyclone Commerce <jturpin@cyclonecommerce.com>
Pete Wenzel, Sun Microsystems <pete.wenzel@sun.com>

Contributors:
Doug Bunting, Sun Microsystems <doug.bunting@sun.com>
Jacques Durand, Fujitsu Software <jdurand@us.fujitsu.com>
Ric Emery, Cyclone Commerce <remery@cyclonecommerce.com>
Kazunori Iwasa, Fujitsu Limited <kiwasa@jp.fujitsu.com>
Hamid Ben Malek, Fujitsu Software <hbenmalek@us.fujitsu.com>
Dale Moberg, Cyclone Commerce <dmoberg@cyclonecommerce.com>
Sacha Schlegel, Cyclone Commerce <sschlegel@cyclonecommerce.com>

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 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

Table of Figures

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. The ebMS V3 specification is divided into two parts described in two different documents.

1.1 Background and Objectives

The prime objective of the ebXML Messaging Service (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, 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.

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, but other protocols to which SOAP may bind can also 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. Some references and bindings to other ebXML specifications in this document should be interpreted as aids to integration, rather than as a requirement to integrate or to use in combination. For example, ebMS may refer to the [ebCPPA] specification, rather than require its use. The 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 ebMS defines messaging functions, protocol and envelope intended to operate over SOAP (SOAP 1.1 or SOAP 1.2, and SOAP with Attachments). 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 document, Part 1: Core Features, supports networking topologies in which there are limitations on initiating message transfer, but with only a point-to-point MSH topology, in which no intermediaries are present. A forthcoming Part 2, containing Advanced Features, will take into account topologies that contain intermediaries (e.g. hub, multi-hop), as well as those in which the ultimate MSH acts as a SOAP intermediary.

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 Messaging 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 should 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.

1.3 Caveats and Assumptions

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

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

This specification presents some alternatives regarding underlying specifications (e.g. SOAP 1.1/1.2, WSS1.0/1.1, and WS specifications that may support the reliability function). This does not imply that a conforming implementation must support them all, nor that it is free to support any option. The definition of conformance profiles - out of scope for this document, and to be described in an adjunct OASIS document - will complement this specification by asserting which option(s) must be supported in order to claim support for a particular conformance profile. Interoperability is conditioned by conformance to compatible profiles. See Section 12 for more details on conformance profiles.

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:

1. 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 inthe section that specifies that module.
2. 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.
3. 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.
4. It is capable of interoperating with another implementation that has chosen to implement optional syntax, features and/or behavior, defined in this specification, it has chosen not to implement. Handling of unsupported features SHALL be implemented in accordance with the prescribed failure mechanism defined for the feature.

1.5 XML Notation

When describing concrete XML schemas and information items, this specification uses a convention in

317 which each XML element or attribute is identified using abbreviated [XPATH] notation (e.g.,
318 /x:MyHeader/x:SomeProperty/@value1).

319 1.6 Namespace Prefixes

320 This table maps various prefixes that appear in XML examples to their intended corresponding
321 namespaces.

322

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
eb	http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
wsr	http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd
wsrx	http://docs.oasis-open.org/ws-rx/wsrm/200602
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd
wsswa	http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0.xsd

323

2 Messaging Model

2.1 Terminology and Concepts

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

2.1.1 Components of the Model

The ebMS messaging model assumes the following components:

- **ebMS MSH (Messaging 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 the eb:Messaging header (qualified with the ebMS namespace).
- **Producer (or Message Producer)**: An entity that interacts with a Sending MSH (i.e. an MSH in Sending role) to initiate the sending of a user message. Some examples are: an application, a queuing system, another SOAP processor (though not another MSH).
- **Consumer (or Message Consumer)**: An entity that interacts with a Receiving MSH (i.e. an MSH in Receiving role) to consume data from a received message. Some examples are: an application, a queuing system, another SOAP processor.

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

Note:

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

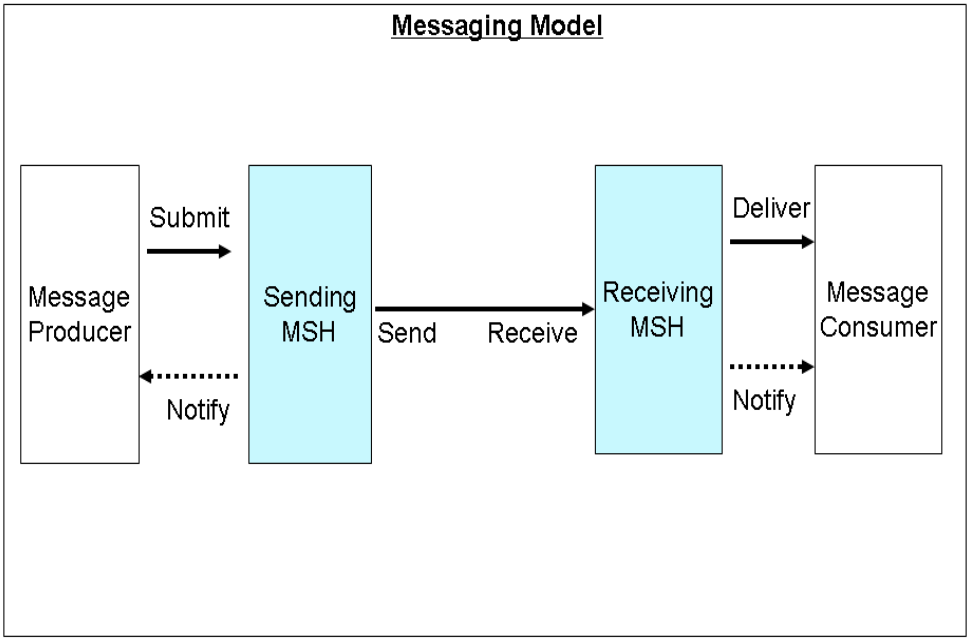


Figure 1: Entities of the Messaging Model and their Interaction

2.1.2 Message Types

An **ebMS Message** is a message that contains SOAP header(s) qualified with the ebMS namespace, and that conforms to this specification. There are two 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 application data to be delivered to a Consumer entity – i.e. not subject to the Deliver operation. An example is the ebMS PullRequest signal. A signal message is characterized by the presence of an eb:SignalMessage element in the ebMS header (see Section 5.2.2).
- **ebMS User Message**: An ebMS message that is initiated by a Producer entity (via Submit operation), and intended for a Consumer entity (via Deliver operation). A user message is characterized by the presence of an eb:UserMessage element in the ebMS header (see Section 5.2.1).

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

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

2.1.3 Messaging Roles

The Messaging Model assumes the following roles for an MSH:

- **Sending**: An MSH acts in Sending role when performing functions associated with generating an

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

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

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

2.1.4 Abstract Messaging Operations

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

- **Submit:** This operation transfers enough message data from the producer to the Sending MSH to generate an ebMS User Message Unit.
- **Deliver:** This operation makes data of a previously received (via Receive operation) ebMS User Message Unit available to the Consumer.
- **Notify:** This operation notifies either a Producer or a Consumer about the status of a previously submitted or received ebMS User Message Unit, or about general MSH status.
- **Send**: This operation initiates the transfer of an ebMS user message from the Sending MSH to the Receiving MSH, after all headers intended for the Receiving MSH have been added (including security and/or reliability, as required).
- **Receive**: This operation completes the transfer of an ebMS user message from the Sending MSH to the Receiving MSH. A successful reception means that a contained User Message Unit is now available for further processing by the Receiving MSH.

2.2 Message Exchange Patterns

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

2.2.1 Definition

An ebMS Message Exchange Pattern (MEP) is an abstract description of a typical sequence of ebMS message exchanges that may occur between two or more MSH instances.

An MEP instance is a message exchange that conforms to the pattern described by the MEP. More precisely, 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 is exchanged in the same MEP instance, all message units involved in this instance must be related. In other words, for every message unit exchanged in the same MEP instance, either one of these statements MUST be 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. The referencing of one message unit by another 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 the *Responding* MSH.

2.2.2 Assumed SOAP Message Exchange Patterns

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

Note that the concept of SOAP MEP has only been introduced with SOAP 1.2 ([SOAP12]), although this concept may be applied to SOAP 1.1 as well. This section is only concerned with some assumed properties of these SOAP MEPs, not with their precise definitions.

Specific bindings of these MEPs to underlying protocols are addressed in a later section. It is only assumed that underlying protocols are of two kinds: either one-way, or two-way, with an implicit back-channel for returning messages in the latter case.

The SOAP MEPs that support the ebMS MEPs described in this specification are described in the next two subsections.

2.2.2.1 SOAP One-Way MEP

No formal definition is available for this MEP at the time this specification is written. This specification is only concerned here with identifying a set of properties for such an MEP, that many SOAP implementations exhibit when sending a message that is not expecting a response message. From an MSH perspective, support for this MEP assumes the following:

- The Initiator MSH is able to initiate the sending of a SOAP envelope (an ebMS message) over the underlying protocol (over the first leg, if a two-way protocol).
- In case a two-way underlying protocol is used, the MEP MUST comply with one of the following rules:
    a. The response message contains a SOAP envelope with a SOAP body that is always empty, in which case the One-way MEP is qualified as "composable";
    b. The response message contains a SOAP envelope with a SOAP body that is either empty or contains a SOAP Fault, in which case the One-way MEP is qualified as "fault supporting", in addition to being composable; or
    c. The response message does not contain a SOAP envelope, in which case the SOAP One-way MEP is qualified as "strict".

2.2.2.2 SOAP Request-Response MEP

The full definition of this MEP can be found in [SOAP12] Part 1, Adjunct. From an MSH perspective, support for this MEP assumes the following:

- The Initator MSH is able to 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 (another ebMS Message) over the underlying protocol's response.

2.2.3 Simple ebMS Message Exchange Patterns

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

An ebMS MEP is of two types:

- Simple ebMS MEP: if it executes over a single SOAP MEP instance.
- Aggregate ebMS MEP: if it executes over more than one SOAP MEP instance.

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

2.2.3.1 The One-Way Push Message Exchange Pattern

This MEP involves a single ebMS user message. The message sending is initiated by a Sending MSH either:

- as a SOAP One-way MEP instance,
- as a SOAP Request in a SOAP Request-Response MEP instance, or
- as a SOAP Response in a SOAP Request-response MEP instance.

To conform to this MEP, the user message unit MUST NOT relate to any other user message unit (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.

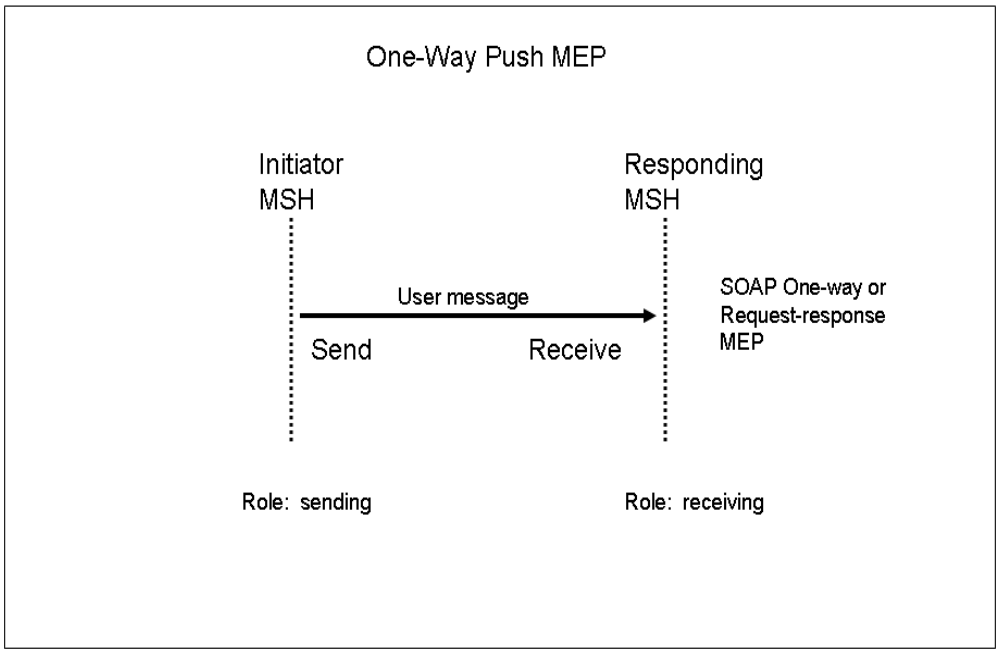
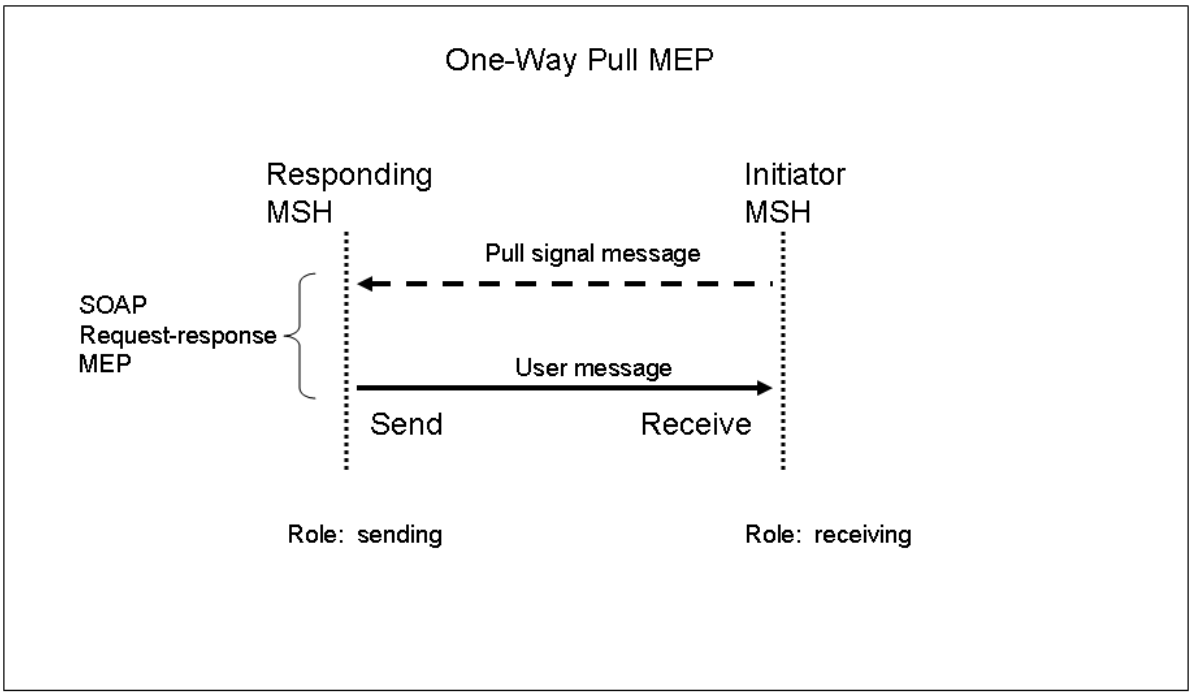


Figure 2: One-Way Push MEP

2.2.3.2 The One-Way Pull Message Exchange Pattern

This MEP involves a single ebMS user message unit. 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 Reponse. To conform to this MEP the user message unit MUST relate to the PullRequest message unit with eb:RefToMessageId, and MUST NOT be referred to by any subsequent user message unit. 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 short description of "EmptyMessagePartitionFlow", as listed in Section 7.7.1, MUST be returned. Figure 3 illustrates the exchange pattern and operations involved for this MEP.



473 *Figure 3: One-Way Pull MEP*
474

475 ### 2.2.3.3 The Request-Reply Message Exchange Pattern

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

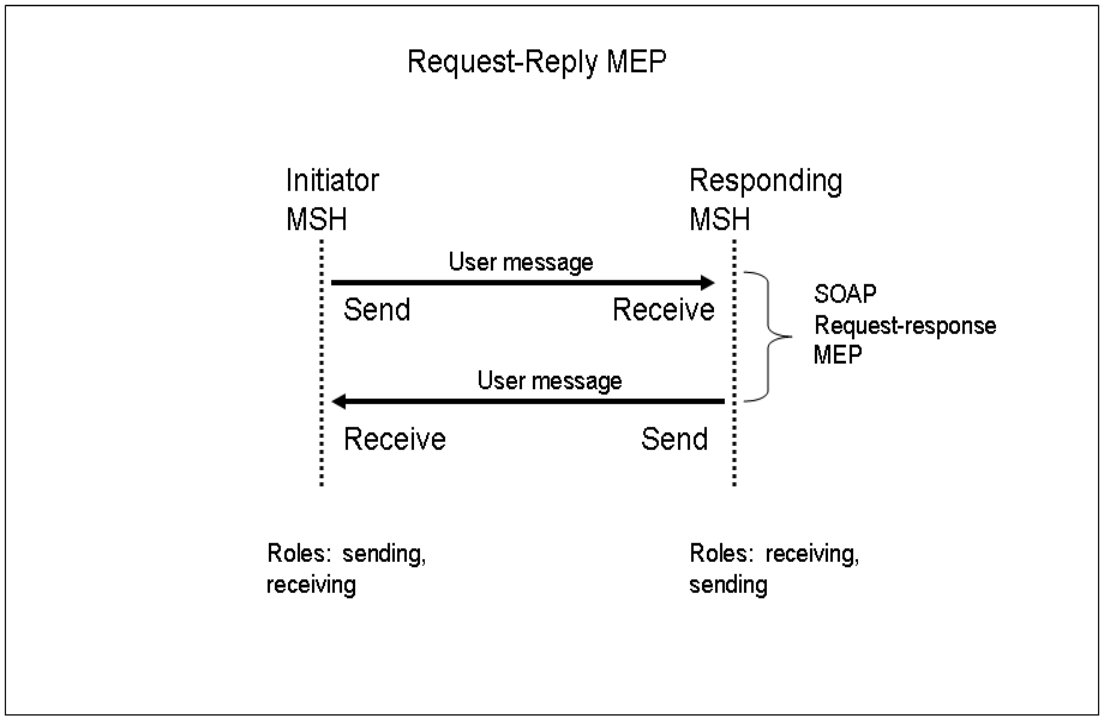


Figure 4: Request-Reply MEP

2.3 Message Partition Flows

2.3.1 Concept and Purpose

Message Partition Flows (MPFs) allow for partitioning the flow of messages from a Sending MSH to a Receiving MSH into several flows that can be controlled separately and consumed differently. They also allow for merging flows from several Sending MSHs, into a unique flow that will be treated as such by a Receiving MSH. In particular, MPFs allow for:

1. setting transfer priorities: some messages may be transferred with higher priority than others regardless in which order they all have been submitted. For example, when using pulling mode, a Receiving MSH may decide from which MPF to pull messages first, based on business needs and readiness to incur responsibility in managing these messages.
2. organizing the inflow of messages on receiving side, so that each flow can be consumed in a distinct way yet without having to filter messages based on various header elements or payload content. The agreement between two parties on when messages are to be transferred and how they are to be consumed may then be reduced to which MPF will be used.

The notion of MPF is abstract from any particular implementation device such as ports or queues: an implementation may choose to implement MPFs using queues and a FIFO policy, though it is not required to.

There is no requirement for ordering messages in an MPF, unless specified otherwise by the reliability requirements to which these messages are subject. The transfer of messages over an MPF is controlled by:

- The MEPs these messages participate in. Messages over an MPF can either be pulled or pushed, based on the different MEPs that govern the transfer of these messages.
- The regular addressing means used for sending messages (e.g. URL of Receiving MSH when pushing messages). MPFs do not have any routing or addressing capability.

Before it is transferred from a Sending MSH to a Receiving MSH, regardless whether it is pushed or pulled, a message is always assigned to an MPF. If no explicit assignment is requested (e.g. by the message Producer at Submit time or per configuration of the MSH) the default MPF name "http://www.oasis-open.org/committees/ebxml-msg/defaultMPF" is assigned.

2.3.2 Some Use Cases

Figure 5 illustrates various cases in using MPFs.

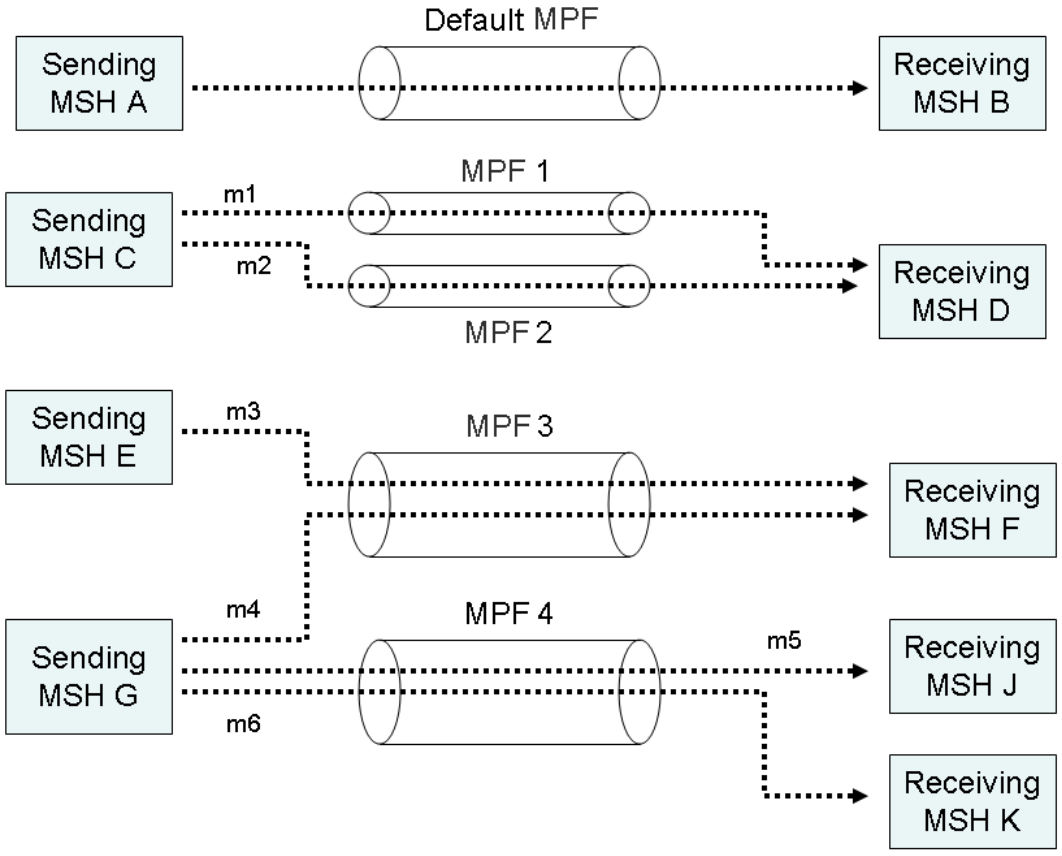


Figure 5: Message Partition Flow Use Cases

In the figure above, each arrow represents the transfer of a user message, which could be either pushed or pulled.

Between MSHs A and B, no MPF has been explicitly defined or assigned. All messages exchanged from A to B – whether pushed or pulled – will implicitly use the default MPF.

MSHs C and D have been configured to use MPFs 1 and 2 (in addition to the default MPF). Messages sent may be assigned to either one of these MPFs. In case these messages are pulled, MSH D may choose from which MPF to pull first.

MPF 3 is shared by two Sending MSHs, E and G. The effect of using this MPF is to define on the Receiving MSH F a merged inflow of messages from E and G, which may be presented to the Consumer as a single flow. If messages m3 and m4 are pulled, MSH F has control over which MSH to pull from first.

MPF 4 is used by MSH G to send either to MSH J or MSH K. When combined with message pulling, this use case allows for various scenarios. For example, the message flow might initially go exclusively from G to J. In case MSH J fails, another MSH (K) may immediately take over the message flow without any change on sender side (assuming K is authorized.) nor any knowledge by K of where the initial flow was intended for. Or, two Receiving MSHs (J and K) that are remote from each other but used by equivalent applications may split the processing of messages submitted to the same Sending MSH G. This may be the case of two agencies equally qualified to process trouble tickets, indiscriminately pulling messages from the same MPF at the pace allowed by their processing capacity. MPF 4 may also be used by concurrent, pushed message flows. Using the same MPF does not introduce any dependency between the processing of m5 and m6 in J and K, but may be associated with a particular business meaning (i.e. Is meaningful to Consumers of J and K).

2.3.3 Definition and Usage Requirements

An MPF is a flow of messages from a set of Sending MSHs to a set of Receiving MSHs, in the sense given in flow networks theory. It is identified by a name–a string of characters–that is assigned to every message of the flow. For every message it sends or receives, an MSH must be aware of which MPF this message is assigned to. MPF is a dynamic notion, the elements of which do not need to be fully defined prior to initiating this flow. For example, additional MSHs (either Sending or Receiving) may join the flow at any time, assuming they have knowledge of the MPF name, and assuming there is no other reason preventing them from transferring messages over this MPF (e.g. security).

The association between a user message and an MPF is apparent in the ebMS header of the message (see Section 5.2). Except for the default MPF, the MPF name must appear in the header of a user message transferred over this MPF.

> Note:
> As defined above, an MPF may involve more than a Sending MSH and a Receiving MSH. In particular, two unrelated pairs of Sending/Receiving MSHs (e.g. in the previous figure, C and D on the one hand, E and F on the other hand) could transfer messages using the same MPF name (e.g. MPF 3 in the figure could also be renamed MPF 2). Formally speaking, all these messages would be transferred over the same MPF. There might be some business significance in deciding whether two pairs of MSHs that have unconnected message flows should use the same MPF to transfer these messages, even though as far as the MSHs are concerned, they will process these two separate sub-flows of messages independently from each other.

Only user messages may be assigned to MPFs, not signal messages.

A PullRequest signal message always indicates in its header (see Section 5.2.2.1) the MPF on which the message must be pulled. If no MPF is explicitly referred to, the default MPF must be pulled from. The pulled message sent in response must be assigned to the indicated MPF.

The association of a message with an MPF must be done either at Submit time, e.g. requested by the message Producer; or at any time between Submit and Send, e.g. based on configuration or processing mode (see Section 3). This is left to the implementation.

Support for assigning messages to MPFs–e.g. by automatically mapping messages submitted by a Producer to a particular MPF 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 messages when pulling messages from an MPF. This specification only describes the properties of MPFs, and how their use affects the message protocol. It does not prescribe a particular way to implement MPFs or to use them.

A message associated with an MPF could fail to be transferred for various reasons (transport issue, security, intermediaries, etc.) and therefore could be removed from the MPF at any time. In other words, there is no additional delivery contract for messages over an MPF, other than what the reliability agreement specifies.

There is no specific quality of service associated with an MPF. Security and reliability remain associated

with parties or with MSHs, in a way that is orthogonal to MPFs, although an implementation is free to associate QoS with MPFs as long as this conforms to an agreement between parties.

2.4 Messaging Service Processing Model

The ebXML Messaging 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 6 depicts a logical arrangement of the functional modules existing within one possible implementation of the ebXML Messaging Services architecture. These modules are arranged in a manner to indicate their inter-relationships and dependencies.

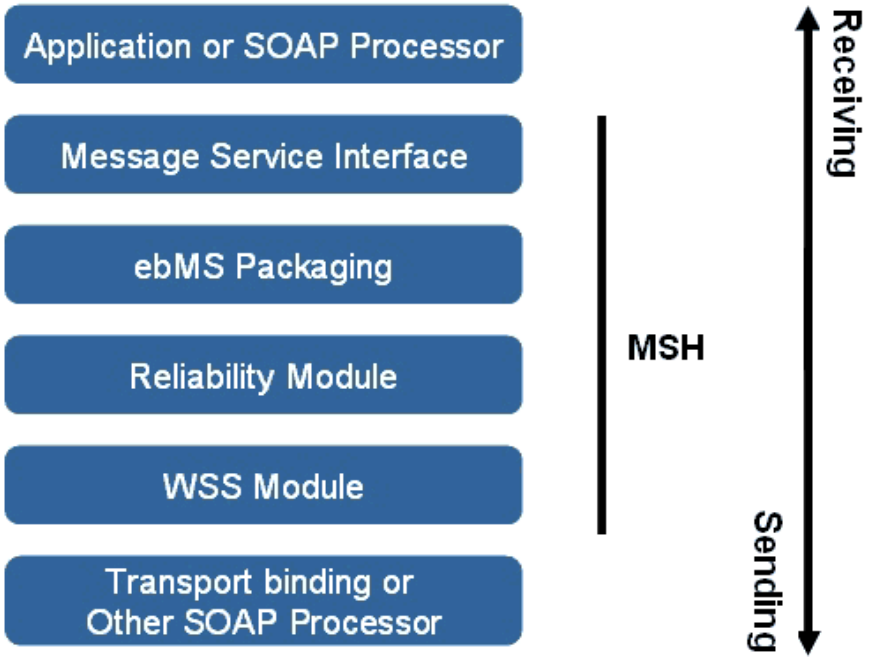


Figure 6: Component Relationships

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

- **Application or SOAP Processor** - This is where the business logic for a message exchange / business process exists.
- **Messaging Service Interface** - This is the interface through which messages are channelled between the MSH core and the the ebXML Application.
- **ebMS Packaging** - Handling, (de)enveloping and execution of Payload Services are performed by this module.
- **Reliable Message Processing** - This module fulfills the Quality of Service requirements for a message.
- **Web Services Security Processing** - Encryption/decryption of any SOAP message content and generation/verification of any digital signatures occurs in this module.
- **Transport Bindings** - These are the actual transport bindings. This specification defines bindings for HTTP (Section 2.2.3) and SMTP (Section), and supports the addition of other protocols.

3 Processing Modes

An MSH is operating–either for sending or receiving messages–in knowledge of some contextual information that controls the way messages are processed. The contextual information that governs the processing of a particular message is called Processing Mode (or P-Mode). Because different messages may be subject to different kinds of processing, an MSH is generally supporting several P-Modes. The set of all P-Modes that an MSH has been configured to support, is called the P-Mode set of the MSH.

A P-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 the 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.

The association of a P-Mode with a message may be based on various criteria that usually depend on header data (e.g. service / action, conversation Id, or other message properties). Which security and/or which reliability, as well as which MEP is being used when sending a message, is determined by the P-Mode associated with this message.

Although a standard representation of P-Mode data is out of scope of this specification, an abstract definition of it helps to capture some aspect of the messaging that are not directly tied to the protocol, such as the notion of default behavior and some errors of contractual nature.

3.1 Processing Mode Features

The P-Mode is partitioned into six functional groups called P-Mode features. Each P-Mode feature covers one of the six functional areas that is critical to achieving interoperability between two partners: security, reliability, transport, business-collaboration, error reporting, Message Exchange Patterns (MEPs) and Message Partition Flows (MPFs).

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

- **P-Mode.protocol**: includes all transport related information that is 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. This feature usually treats similarly all messages between two MSHs. It also includes information about which SOAP version is to be used (SOAP 1.1 or SOAP 1.2).
- **P-Mode.reliability**: 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**: 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

  It also indicates which MEPs and which MPFs are to be used by these parties.
- **P-Mode.messagePartitionFlows**: includes the identification of all MPFs hosted by the MSH, that will be used to transfer messages. This feature determines the possible values for the header attribute:
    - eb:UserMessage/@eb:mpf
- **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 set 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:AgreementRef element in Section 5.2.1.7.)

3.2 Default Features for Processing Mode

In order to facilitate interoperability testing, or during the early phase of a deployment, it is useful to be able to drive message exchanges without relying on user-agreed P-Modes, and without interfacing with any application. To this end, a default semantics for each P-Mode feature is defined as follows:

- **Default P-Mode.protocol**: HTTP 1.1 transport is assumed, with default configuration (on standard port), using SOAP 1.2.
- **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 with value: `http://www.oasis-open.org/committees/ebxml-msg/defaultFrom`. The eb:To element contains a PartyId with value: `http://www.oasis-open.org/committees/ebxml-msg/defaultTo`.
    - eb:UserMessage/eb:CollaborationInfo: Contains no eb:AgreementRef. The eb:Service element has value: `http://www.oasis-open.org/committees/ebxml-msg/service`. The eb:Action element has value: `http://www.oasis-open.org/committees/ebxml-msg/test`. (Section 5.2.1 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.
    - The default ebMS MEP is One-way Push.
- **Default P-Mode.messagePartitionFlows**: Only the default MPF is in use.
- **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.

In the absence of a user-agreed P-Mode feature, it is RECOMMENDED that an MSH operates based on the above default semantics for this feature except in the following cases:

1. The MSH is designed to conform to this specification along profiles (see Section 12) that are not compatible with the default P-Mode feature. For example, such an incompatibility would occur for the default P-Mode.businessCollaboration with a conformance profile that only requires One-way Pull MEP.
2. The MSH has been pre-configured to operate with a non-default P-Mode feature. This would be the case when an MSH is distributed along with a predefined P-Mode feature, e.g. built-in security. This amounts to using a user-defined P-Mode feature.

A Sending MSH and a Receiving MSH may use a mix of default and non-default P-Mode features.

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:

- Message pulling
- Message Partition Flows (MPFs)

Message pulling is defined in an abstract way by the One-way Pull ebMS MEP (see Section 2.2.3, Simple ebMS Message Exchange Patterns). 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 Partition Flows (see definition in Section 2.3) 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, in terms of transfer priorities.

***Example**: A pair of business partners – a large buyer and a small supplier - have decided to create two MPFs for transferring messages sent by the buyer. One MPF is assigned to urgent messages that require immediate processing (high priority Purchase Orders, and updates to prior P.O.) and the other MPF 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 MPF, then only the messages from the less urgent MPF. 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 MPFs. The notion of MPF 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 MPF, without a need to transfer and process filter expressions.

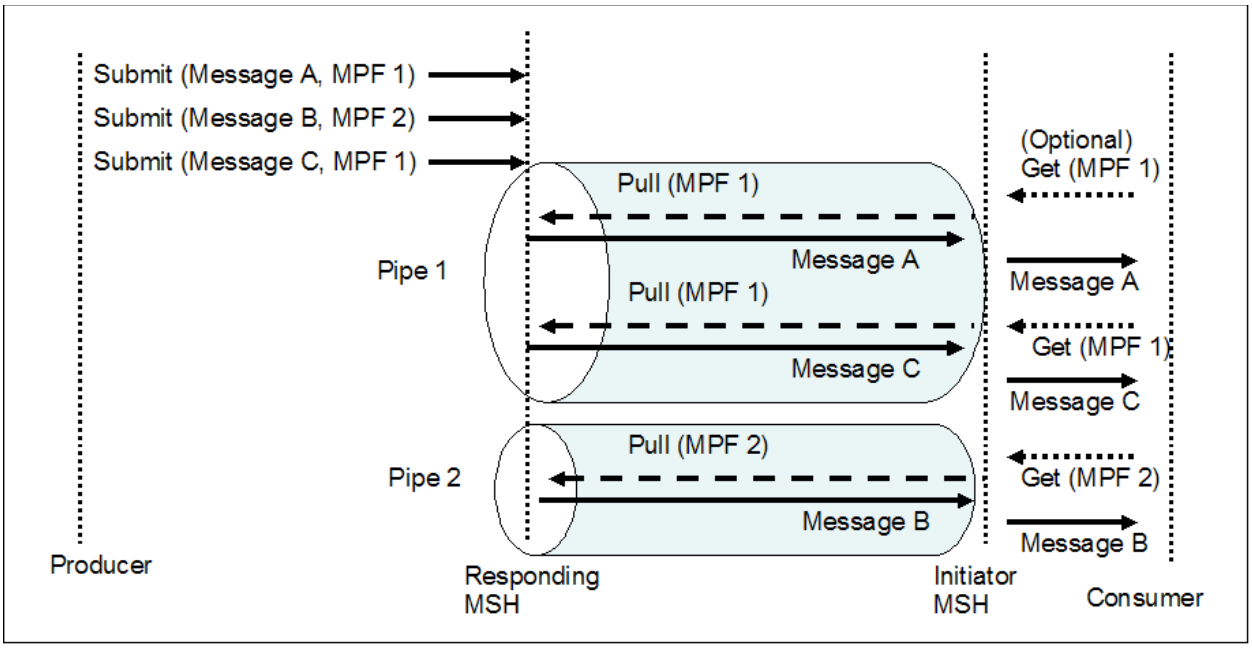


Figure 7: One-Way Pull with Message Partition Flows

Figure 7 illustrates how MPFs 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. MPF 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. MPF 1 may be configured for Push mode while MPF 2 is configured in Pull mode. The Consumer will initiate the transfer of non-critical messages over MPF 2 only when ready to process these.

4.2 Supporting Message Pulling

Using Message pulling 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). The Pull mode is always scoped by an MPF. Because there is always at least one MPF open between a Sending MSH and a Receiving MSH–the default MPF–the Pull mode can be supported regardless of the ability to support several MPFs.

When sending a PullRequest signal, the name of the MPF to pull messages from must be specified (@mpf).

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 MPF. If no MPF 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 MPF is used. The MPF associated with this message must have been configured for Pull mode (specified in a representation of P-Mode.messagePartitionFlow).

On Initiator MSH side:

2. Sending of a PullRequest signal by the MSH. The PullRequest signal specifies the MPF from which to pull messages.

On Responder MSH side:

3. 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 with respect to the Submit operation. If there is no user message available in the specified MPF for sending, a warning signal with short description: "EmptyMessagePartitionFlow" (see Section 7.7.1) MUST be sent back instead.
4. Send: the selected message is sent over the SOAP Response to the PullRequest. .

On Initiator MSH side:

5. Receive: the pulled message is available for processing by the MSH. The header @mpf attribute indicates which MPF it has been pulled from and is same as the @mpf value in the PullRequest signal.
6. Deliver: after processing of ebMS headers, delivery of the pulled message data to the Consumer of the MSH.

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

```
<SOAP:Envelope>
<SOAP:Header>
<eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1">
      <eb:SignalMessage>
            <eb:MessageInfo>
                  <eb:TimeStamp>2005-10-01T10:01:00</eb:TimeStamp>
                  <eb:MessageId>UUID-4@example.com</eb:MessageId>
            </eb:MessageInfo>
            <eb:PullRequest mpf="http://msh.example.com/mpf123"/>
      </eb:SignalMessage>
</eb:Messaging>
</SOAP:Header>
<SOAP:Body/>
</SOAP:Envelope>
```

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

```
<SOAP:Envelope>
<SOAP:Header>
<eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1" >
      <eb:UserMessage mpf="http://msh.example.com/mpf123">
            <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: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:

```
<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 mpf="http://msh.example.com/mpf123"/>
      </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:

```
<SOAP:Envelope>
<SOAP:Header>
<eb:Messaging eb:version="3.0" SOAP:mustUnderstand="1" >
      <eb:UserMessage mpf="http://msh.example.com/mpf123">
            <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>
<wsr:Response SOAP:mustUnderstand="1">
      ...
</wsr:Response>
<wss:Security>
      ...
</wss:Security>
</SOAP:Header>
<SOAP:Body>
```

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

Note: In the above example, the reliability header, which assumes the use of WS-Reliability, is a Response element. It contains the reliability acknowledgment for the PullRequest signal. In this example there is no wsr:Request reliability header, meaning that no acknowledgment is expected for this response. This does not prevent the At-Least-Once reliability contract to be supported for this ebMS user message: if the user message fails to be received by the Initiator MSH, the lack of acknowledgment will cause the resending of the PullRequest signal. This will cause in turn the resending of the same user message. In case of failure to deliver, a failure notification will be raised on the Initiator MSH side.

5 Message Packaging

5.1 Message Envelope and Message Parts

5.1.1 MIME Structure and SOAP Profile

In the ebMS SOAP header eb:Messaging, the prefix "eb" is an example prefix that corresponds to the ebMS 3.0 namespace, as defined in Section 1.5. The ebMS Message can be packaged as a plain [SOAP11] or [SOAP12]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.

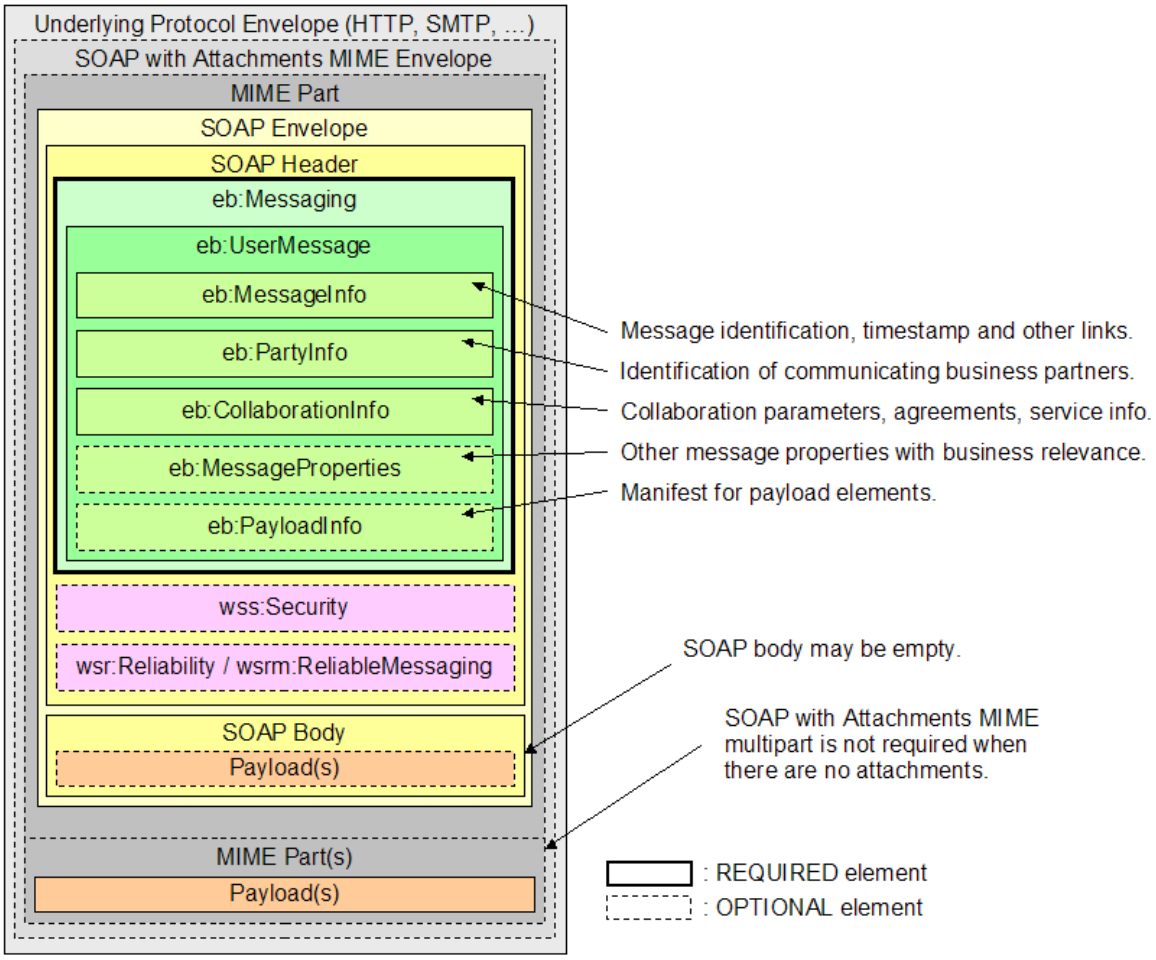
The ebMS Message MAY contain SOAP extension elements other than the eb:Messaging header block. For example, header blocks supporting message reliability and message security MAY be produced and consumed by an MSH in order to fulfill deployment requirements for those features.

An ebMS Message is packaged as a SOAP 1.1 or 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 8, and a Signal Message in Figure 9.



900

Figure 8: User Message Structure

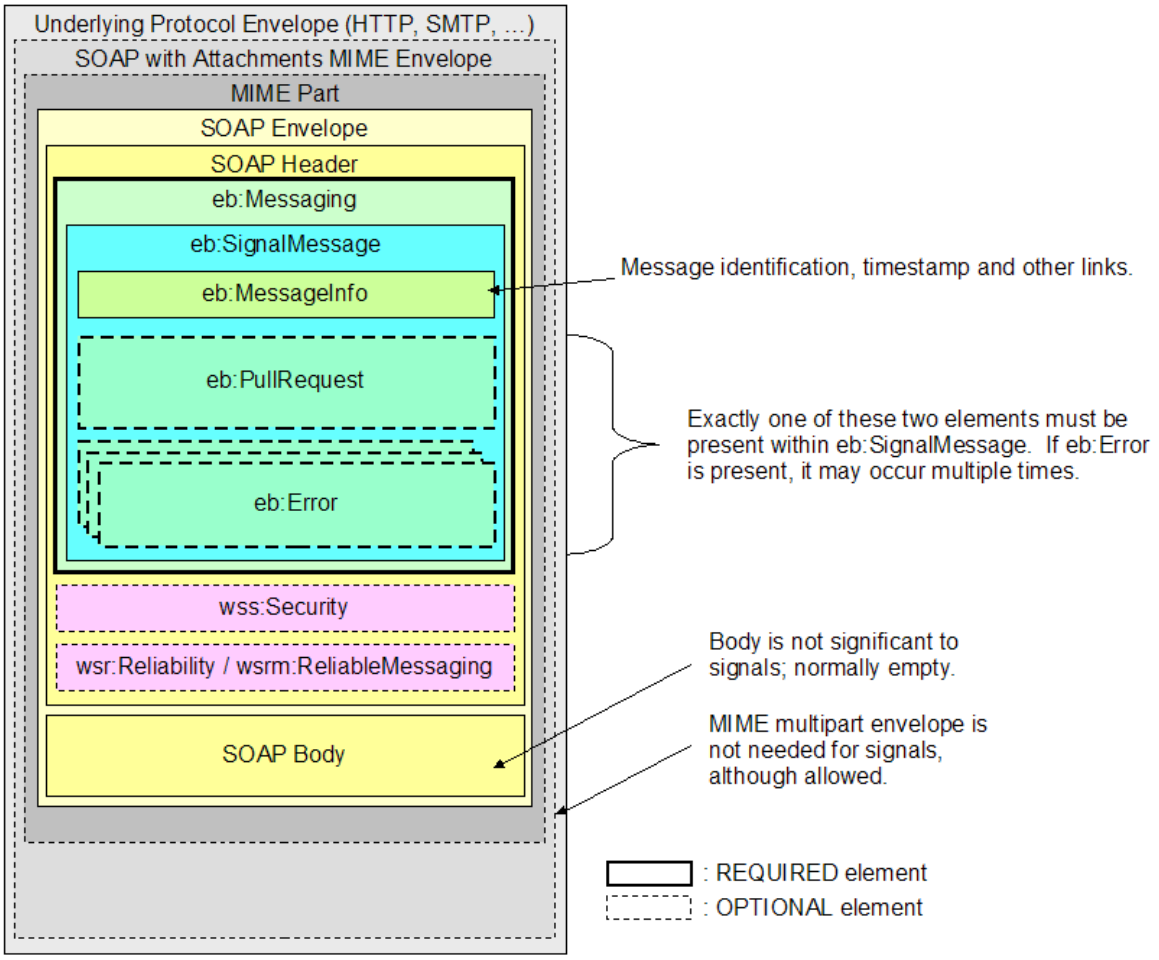


Figure 9: Signal Message Structure

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

All MIME header elements of the Message Package MUST conform with the SOAP Messages with Attachments [SOAPATTACH] W3C Note. In addition, the Content-Type MIME header in the Message Package contain a type attribute matching the MIME media type of the MIME body part containing the SOAP Message document. In accordance with the [SOAP11] specification, the MIME media type of the SOAP Message has the value "text/xml". It is strongly RECOMMENDED that the initial headers contain a Content-ID MIME header structured in accordance with MIME [RFC2045], and in addition to the required parameters for the Multipart/Related media type, the start parameter (OPTIONAL in MIME Multipart/Related [RFC2387]) always be present. This permits more robust error detection. The following fragment is an example of the MIME headers for the multipart/related Message Package:

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

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

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

5.1.2 MIME Considerations

5.1.2.1 Additional MIME Parameters

Any MIME part described by this specification MAY contain additional MIME headers in conformance with the MIME [RFC2045] specification. Implementations MAY ignore any MIME header not defined in this specification. Implementations MUST ignore any MIME header they do not recognize. For example, an implementation could include content-length in a message. However, a recipient of a message with content-length could ignore it.

5.1.2.2 Reporting MIME Errors

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

5.1.2.3 XML Prolog

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

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

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

5.1.2.4 XML Declaration

The XML declaration MAY be present in a SOAP Message. If present, it MUST contain the version specification required by the XML Recommendation [XML10] and MAY contain an encoding declaration. The semantics described below MUST be implemented by a compliant ebXML Message Service.

5.1.2.5 Encoding Declaration

If both the encoding declaration and the MIME root part charset are present, the XML prolog for the SOAP Message SHALL contain the encoding declaration SHALL be equivalent to the charset attribute of the MIME Content-Type of the root part (see Section 5.1.4). If provided, the encoding declaration MUST NOT contain a value conflicting with the encoding used when creating the SOAP Message. It is RECOMMENDED UTF-8 be used when encoding the SOAP Message. If the character encoding cannot be determined by an XML processor using the rules specified in section 4.3.3 of XML [XML10], the XML declaration and its contained encoding declaration SHALL be provided in the ebXML SOAP Header Document. **Note:** The encoding declaration is not required in an XML document according to XML v1.0 specification [XML10].

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 Section 9, 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/">
      <SOAP:Header>
            <Messaging xmlns:eb="http://www.oasis-
open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd"
                  xsi:schemaLocation="http://www.oasis-
open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd
                  http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-3_0.xsd" eb:version="3.0">
                  <eb:UserMessage>
                        <eb:MessageInfo >...</eb:MessageInfo>
                        ...
                        <eb:PayloadInfo >...</eb:PayloadInfo>
                        ...
                  </eb:UserMessage>
            </eb:Messaging>
      </SOAP:Header>
      <SOAP:Body>
            ...
      </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 wholely 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 conforming to [XMLID]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 Messaging 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, or "application/soap+xml" in the case of a SOAP 1.2 body. The Content-Type header MAY contain a "charset" attribute. For example:

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

The MIME charset attribute identifies the character set used to create the SOAP Message. The semantics of this attribute are described in the "charset parameter / encoding considerations" of text/xml as specified in [RFC3023]. The list of valid values can be found at [IANAMEDIA].

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

For maximum interoperability it is RECOMMENDED UTF-8 [UTF8] be used when encoding this document. Due to the processing rules defined for media types derived from text/xml [RFC3023], this MIME attribute has no default.

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

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

<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
```

```
        <SOAP:Header>
                <eb:Messaging>
                …
                </eb:Messaging>
        </SOAP:Header>
        <SOAP:Body>
        …
        </SOAP:Body>
</SOAP:Envelope>
```

5.1.5 Payload Containers

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

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

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

The ebXML Messaging 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 Messaging Service.

Example of SOAP Message containing an ebMS header:

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

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:

```
<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:

```
<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.
- `eb:Messaging/eb:` SignalMessage`/eb:[`*`signalname`*`]`: The OPTIONAL element is named after a Signal message. It contains all header information for the Signal message.

Example ebMS Message Header:

```
<SOAP:Header …>

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

    <eb:UserMessage>

      <eb:MessageInfo>
        <eb:timestamp>2000-07-25T12:19:05</eb:timestamp>
        <eb:MessageId>UUID-2@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
      </eb:MessageInfo>

      <eb:PartyInfo>
        <eb:From> (no change beside renaming)
          <eb:PartyId>uri:example.com</eb:PartyId>
            <eb:Role>http://rosettanet.org/roles/Buyer</eb:Role>
        </eb:From>

        <eb:To> (no change beside renaming)
          <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.example.com/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 xml:lang="en-US">Purchase Order for 100,000 foo
widgets</eb:Description>
        </eb:PartInfo>
        <eb:PartInfo href="#idref">
        </eb:PartInfo>
      </eb:PayloadInfo>

    </eb:UserMessage>
  </eb:Messaging>

</SOAP:Header>
```

5.2.1 eb:Messaging/eb:UserMessage

This element has the following attributes:

- `eb:Messaging/eb:UserMessage/@mpf`: This OPTIONAL attribute contains a URI that identifies the MPF to which the message is assigned. The absence of this element indicates the use of the default MPF. When the message is pulled, the value of this attribute MUST indicate the MPF requested in the PullRequest message.

This element has the following children elements:

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

5.2.1.1 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 Section 2.2.3).

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

This element has the following children 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 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From

This element has the following children 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.

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

5.2.1.4 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId

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

It has a single attribute, @eb:type. The type attribute indicates the domain of names to which the string in the content of the PartyId element belongs. It is RECOMMENDED that the value of the type attribute be a URI. It is further RECOMMENDED that these values be taken from the EDIRA [ISO6523], EDIFACT [ISO9735] or ANSI ASC X12 [ASCI05] registries.

An example of PartyId element is:

```
<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 "Inconsistent" error with severity "error". It is strongly RECOMMENDED that the content of the eb:PartyId element be a URI.

5.2.1.5 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.

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

```
</eb:To>
```

5.2.1.6 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 eb:Messaging/ eb:UserMessage/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.

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 "`Inconsistent`" error of severity "error". If the `AgreementRef` is not recognized, then it MUST report it as a "`NotRecognized`" error of severity "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 eb:Messaging/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 "Inconsistent" error of severity "error".

When the value of the element is `http://www.oasis-open.org/committees/ebxml-msg/service`, then the receiving MSH MUST NOT deliver this message to the Consumer. With the

exception of this delivery behavior, and unless indicated otherwise by the eb:Action element, the processing of the message is not different from any other user message.

5.2.1.9 eb:Messaging/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:

```
<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 a "NotRecognized" error of severity "error".

When the value of this element is `http://www.oasis-open.org/committees/ebxml-msg/test`, then the eb:Service element MUST have the value `http://www.oasis-open.org/committees/ebxml-msg/service`. Such a value for the eb:Action element only indicates that the user message is sent for testing purposes and does not require any specific handling by the MSH.

5.2.1.10 eb:Messaging/eb:UserMessage/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:

```
<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 eb:Messaging/eb:UserMessage/eb:MessageProperties

This element has zero or more `eb:Property` child 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:

```
<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 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 Section 5.1.3.6, id Attribute for details)
- a version attribute (see Section 5.1.3.7, version Attribute for details)

The PayloadInfo element has the following child element:

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo`
  This element occurs zero or more times. The PartInfo element is used to reference either a MIME attachment or an XML element within the SOAP Body, according to the value of the href attribute, described below.

5.2.1.13 eb:Messaging/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 [RFC2392] Content-ID URI of the payload object referenced, or an xml:id fragment identifier; for example, "cid:foo" or "#idref". The absence of the attribute href in the element eb:PartInfo indicates that the payload part being referenced is the SOAP Body element itself. For example, a declaration of the following form simply indicates that the entire SOAP Body is to be considered a payload part in this ebMS message:

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

Any other namespace-qualified attribute MAY be present. A Receiving MSH MAY choose to ignore any foreign namespace attributes other than those defined above.

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

This element has the following child elements:

- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema`
  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:

```
<eb:PayloadInfo>
  <eb:PartInfo href="cid:foo@example.com">
    <eb:Schema eb:location="http://foo/bar.xsd" eb:version="2.0"/>
    <eb:Description xml:lang="en-US">Purchase Order for 100,000
foowidgets</eb:Description>
```

```
  </eb:PartInfo>
  <eb:PartInfo href="#bar_payload_id">
    <eb:Schema eb:location="http://example.com/bar.xsd" eb:version="2.0"/>
    <eb:Description xml:lang="en-US">Purchase Order for 50,000 goo
widgets</eb:Description>
  </eb:PartInfo>
</eb:PayloadInfo>
```

5.2.2 eb:Messaging/eb:SignalMessage

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

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

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 eb:Messaging/eb:SignalMessage/eb:PullRequest

This element has the following attribute:

- `eb:Messaging/eb:SignalMessage/eb:PullRequest/@eb:mpf`
  This OPTIONAL attribute identifies which MPF the message is to be pulled from. The absence of this attribute indicates the default MPF.

5.2.2.2 eb:Messaging/eb:SignalMessage/eb:Error

The eb:Error element MAY occur zero or more times. For its complete specification, refer to Section 7, Error Handling Module.

5.3 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).

5.3.1 UserMessage Example

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

```
…
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
<S11:Header>

   <eb:Messaging eb:version="3.0" S11:mustUnderstand="1">
      <eb:UserMessage>
         <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://example.org/roles/Buyer</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
```

```
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://www.example.com/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://example.com/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:Body>
  …
</S11:Body>
</S11:Envelope>
…
```

The following is an example of a Response User Message:

```
…
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
<S11:Header>

   <eb:Messaging eb:version="3.0" S11:mustUnderstand="1">
      <eb:UserMessage>
         <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:CollaborationInfo>
               <eb:AgreementRef>http://www.example.com/cpa/123456</eb:AgreementRef>
               <eb:Service>QuoteToCollect</eb:Service>
               <eb:Action>PurchaseOrderResponse</eb:Action>
               <eb:ConversationID>4321</eb:ConversationID>
            </eb:CollaborationInfo>
            <eb:To>
               <eb:PartyId>uri:example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:To>
            <eb:From>
               <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:From>
         </eb:PartyInfo>
         <eb:PayloadInfo>
            <eb:PartInfo href="cid:foo">
               <eb:Schema eb:location="http://example.com/bar.xsd" eb:version="2.0"/>
               <eb:Description>Response to Purchase Order Request</eb:Description>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>
   </eb:Messaging>

</S11:Header>
<S11:Body>
  …
</S11:Body>
</S11:Envelope>
…
```

5.3.2 PullRequest Message Example

The following is an example of a PullRequest Signal Message:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
<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:mpf="http://msh.example.com/mpf123" />
      </eb:SignalMessage>
   </eb:Messaging>

</S11:Header>

<S11:Body/>
</S11:Envelope>
```

5.3.3 Error Message Example

The following is an example an Error Signal Message:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-3_0.xsd">
<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:Error eb:origin="ebMS" category="Content"
                  eb:errorCode="EBMS:0001" eb:severity="failure"
                  eb:RefToMessageInErrror="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>

</SOAP:Header>

<S11:Body/>
</S11:Envelope>
```

6 Security Module

The ebXML Messaging Service, by its very nature, presents certain security risks. A Messaging 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

Web Services Security 1.0 [WSS10] or 1.1 [WSS11] can be utilized to secure an ebMS message. Web Services Security provides three mechanisms to secure messages: ability to send security tokens as part of a message, message integrity and message confidentiality.

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 [SOAPATTACH].

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].

> Note
> 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 examples in the WSS 1.1 specification depict only the SOAP 1.1 variant when securing attachments. In other words, 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. Refer to Section 11 for complete details of the ebMS SOAP binding.

This specification outlines the use of Web Services Security x.509 Certificate Token Profile [WSS10-X509] or [WSS11-X509] and the Web Services Security Username Token Profile [WSS10-USER] or [WSS11-USER]. An MSH implementation MAY choose to support other Web Services Security Profiles.

6.2 Signing Messages

Signing of ebMS Messages is defined in Web Services Security [WSS10] and [WSS11]. Support for WSS X.509 Certificate Token Profile is REQUIRED to sign a message.

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

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

To ensure the integrity of the user-specified payload data and ebMS message headers it is

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

6.3 Signing SOAP with Attachments Messages

Application payloads that are are built in conformance with the [SOAPATTACH] specification may be signed. To sign a SOAP with Attachment message the Security element must be built in accordance with WSS 1.1.

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

To ensure the integrity of the user-specified payload data and ebMS headers it is RECOMMENDED that the entire eb:Messaging Container Element, and all MIME Body parts of included payloads are included in the signature.

6.4 Encrypting Messages

Encryption of ebMS Messages is defined in Web Services Security [WSS10] and [WSS11]. Support for Web Services Security x.509 Certificate Token Profile is REQUIRED to encrypt message.

An MSH Implementation may encrypt the eb:Messaging Container Element. The eb:PartyInfo section may be used to aid in message routing before decryption has occurred. It is RECOMMENDED that the eb:PartyInfo section not be encrypted. To ensure the confidentiality of the user-specified payload data it is RECOMMENDED that the SOAP Body is encrypted.

6.5 Encrypting SOAP with Attachments Messages

Application payloads that are are built in conformance with the [SOAPATTACH] specification may be encrypted. To encrypt a SOAP with Attachment message the Security element must be built in accordance to WSS 1.1. To ensure the confidentiality of the user-specified payload data it is RECOMMENDED that the MIME Body parts of included payloads are encrypted,

6.6 Signing and Encrypting Messages

When both signature and encryption are required of the MSH, the message MUST be signed prior to being encrypted.

6.7 UsernameToken Authentication

In constrained environments where management of XML digital signatures is not possible, an authentication alternative that is based on Web Services Security Username Token Profile MUST be supported.

Support for wsse:PasswordText type passwords is REQUIRED. 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.

6.8 Security Policy Errors

A responding MSH MAY respond with an error if a received ebMS message does not meet the security policy of the responding MSH. For example, a security policy might indicate that messages with unsigned parts of the SOAP Body or eb:Messaging Container element are unauthorized for further processing. If a responding MSH receives a message with unsigned data within the SOAP Body and error MAY be returned to the initiating MSH.

6.9 Secured Message Examples

Example of a digitally signed and encrypted ebXML Message:

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

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-3_0.xsd">
        <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2005-10-31T17:36:20.656Z</eb:Timestamp>
                    <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
                    <eb:RefToMessageId>UUID-1@msh-
server.example.com</eb:RefToMessageId>
                </eb:MessageInfo>
                <eb:PartyInfo>
                    <eb:From>
                        <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
                        <eb:Role>http://example.org/roles/Buyer</eb:Role>
                    </eb:From>
                    <eb:To>
                        <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
                        <eb:Role>http://example.org/roles/Seller</eb:Role>
                    </eb:To>
                </eb:PartyInfo>
                <eb:CollaborationInfo>
                    <eb:AgreementRef>http://msh-
server.example.com/cpa/123456</eb:AgreementRef>
                    <eb:Service eb:type="someType">QuoteToCollect</eb:Service>
                    <eb:Action>NewPurchaseOrder</eb:Action>
                    <eb:ConversationId>2a81ffbd-0d3d-4cbd-8601-
d916e0ed2fe2</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="#enc">
                        <eb:Description xml:lang="en-US">PO Image</eb:Description>
                    </eb:PartInfo>
                </eb:PayloadInfo>
            </eb:UserMessage>
        </eb:Messaging>
        <wsse:Security soap:mustUnderstand="1"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-x509-token-profile-1.0#X509v3"
                wsu:Id="signingCert">...</wsse:BinarySecurityToken>
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-x509-token-profile-1.0#X509v3"
                wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
            <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
                <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                    xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd"/>
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#encryptionCert"/>
                    </wsse:SecurityTokenReference>
```

```
                </KeyInfo>
                <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <CipherValue>F3HmZ2Ldyn0umLCx/8Q9B9e8OoslJx9i9hOWQjh6JJwYqDLbd
g0QVFiVT1LVjazlThS9m9rkRtpkhCUIY1xjFKtDsuIIAW8cLZv7IHkVoDtQ7ihJc8hYIlEESX9qZN65Jgy
Aa3BYgW9ipjGHtNgZ9RzUdzKdeY74DFm27R6m8b0=</CipherValue>
                </CipherData>
                <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <DataReference URI="#enc"/>
                </ReferenceList>
            </enc:EncryptedKey>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#ebMessage">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestVal
ue>
                    </ds:Reference>
                    <ds:Reference URI="#body">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestVal
ue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    T24okA0MUh5iBNMG6tk8QAKZ+lFMmY1rcPnkOr9j3fHRGM2qqUnoBydOTnClcE
MzPZbnlhdN
                    YZYmab1lqa4N5ynLjwlM4kp0uMip9hapijwL67aBnUeHiFmUau0x9DBOdKZTVa
1QQ92106ge
                    j2YPDt3VKIlLLT2c8O4TfayGvuY= </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
                        <wsse:Reference URI="#signingCert"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body wsu:Id="body"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
        <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#Content"
            xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
            <CipherData>
                <CipherValue>tjOgUPMmQwd6hXiHuvl42swqv4dTYiBfmg8u1SuFVRC3yfNlokshv
oxs1/qQoqN1prDiSOxsxsFvg1la7dehjMWb0owuvU2de1eKr5KPcSApnG+kTvNrtg==</CipherValue>
            </CipherData>
        </EncryptedData>
    </soap:Body>
</soap:Envelope>
```

Example of a digitally signed and encrypted ebXML SOAP with Attachments Message:

```
Mime-Version: 1.0
Content-Type: multipart/related; type="text/xml";
    boundary="----=_Part_2_6825397.1130520599536"
```

```
SOAPAction: ""
Content-Length: 7860

------=_Part_2_6825397.1130520599536
Content-Type: text/xml
Content-Transfer-Encoding: binary

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Header xmlns:eb="http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-3_0.xsd">
        <eb:Messaging eb:version="3.0" id="ebMessage" soap:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2005-10-28T17:29:59.119Z</eb:Timestamp>
                    <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
                    <eb:RefToMessageId>UUID-1@msh-
server.example.com</eb:RefToMessageId>
                </eb:MessageInfo>
                <eb:PartyInfo>
                    <eb:From>
                        <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
                        <eb:Role>http://example.org/roles/Buyer</eb:Role>
                    </eb:From>
                    <eb:To>
                        <eb:PartyId eb:type="someType">QRS543</eb:PartyId>
                        <eb:Role>http://example.org/roles/Seller</eb:Role>
                    </eb:To>
                </eb:PartyInfo>
                <eb:CollaborationInfo>
                    <eb:AgreementRef>http://msh-
server.example.com/cpa/123456</eb:AgreementRef>
                    <eb:Service eb:type="someType">QuoteToCollect</eb:Service>
                    <eb:Action>NewPurchaseOrder</eb:Action>
                    <eb:ConversationId>782a5c5a-9dad-4cd9-9bbe-
94c0d737f22b</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:PO_Image@example.com">
                        <eb:Description xml:lang="en-US">PO Image</eb:Description>
                    </eb:PartInfo>
                </eb:PayloadInfo>
            </eb:UserMessage>
        </eb:Messaging>
        <wsse:Security soap:mustUnderstand="1"
            xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd"
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-x509-token-profile-1.0#X509v3"
                wsu:Id="signingCert">...</wsse:BinarySecurityToken>
            <wsse:BinarySecurityToken
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-soap-message-security-1.0#Base64Binary"
                ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-x509-token-profile-1.0#X509v3"
                wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
            <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
                <enc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
                    xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd"/>
                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#encryptionCert"/>
```

```
                    </wsse:SecurityTokenReference>
                </KeyInfo>
                <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <CipherValue>jJRbQBjzYpfdCkPk5F7jUoFjw6Ls6DQ8D9sdI62fwjW9Um/g9
QfivLeVzvSndgnthfEBC1Z6loKiuEF5/Ztw/tFrRgkboR7EBG5XaJUnt0rt8iCChy4PfxCEhH1KjFgTJhU
bXxNW3FxSLkouCn2qIBDrJqwZXAIistt29JrANCc=</CipherValue>
                </CipherData>
                <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
                    <DataReference URI="#encrypted-attachment"/>
                </ReferenceList>
            </enc:EncryptedKey>
            <EncryptedData Id="encrypted-attachment" MimeType="image/jpeg"
                Type="http://docs.oasis-open.org/wss/2004/XX/oasis-2004XX-wss-swa-
profile-1.0#Attachment-Content-Only"
                xmlns="http://www.w3.org/2001/04/xmlenc#">
                <EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
                <CipherData>
                    <CipherReference URI="cid:PO_Image@example.com">
                        <Transforms>
                            <Transform
                                Algorithm="http://docs.oasis-
open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-
Transform"
                                xmlns="http://www.w3.org/2000/09/xmldsig#"/>
                        </Transforms>
                    </CipherReference>
                </CipherData>
            </EncryptedData>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                    <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                    <ds:Reference URI="#ebMessage">
                        <ds:Transforms>
                            <ds:Transform
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>xUISuIg5eVxy3FL/4yCrZoEZrTM=</ds:DigestVal
ue>
                    </ds:Reference>
                    <ds:Reference URI="cid:PO_Image@example.com">
                        <ds:Transforms>
                            <ds:Transform
                                Algorithm="http://docs.oasis-
open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-Only-
Transform"
                                />
                        </ds:Transforms>
                        <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                        <ds:DigestValue>R4hCV4K4I5QZdSsrP4KrLu46hFo=</ds:DigestVal
ue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    BGnJV/b7EUbAEsn7GmNhZ8yYN6ZoO6uz29E5r9GHxDW+MUH4wksgA654w+sB0r
Wl8xNranag
                    3dhKoHbaRERzYHDGq1VfIRqgEwOrHwhz4h7uoLX4yxOU6G9T/gily67Q3pENGp
mVowzoppHm
                    /yd/A2T0+v4vso20aJiSieEIzSQ= </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference
                        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd">
                        <wsse:Reference URI="#signingCert"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </soap:Header>
    <soap:Body/>
```

```
</soap:Envelope>

------=_Part_2_6825397.1130520599536
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64
Content-Id: <PO_Image@example.com>
Content-Description: WSS XML Encryption message; type="image/jpeg"

VEhmwb4FHFhqQH8m5PKqVu8H0/bq2yUF

------=_Part_2_6825397.1130520599536--
```

6.10 Securing the PullRequest Signal

6.10.1 Authentication

A Sending MSH MUST be able to authenticate a Receiving MSH that sends a PullRequest. When authentication is required for a particular Receiving MSH, it is RECOMMENDED that the Sending MSH use security at the SOAP protocol level (WSS). In case a Receiving MSH is not able to use SOAP level security, other authentication mechanisms MAY be used, e.g. the HTTP Basic or Digest Access Authentication schemes [RFC2617].

6.10.2 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 MPFs on which these are allowed to initiate message transfer.

6.10.3 Preventing Repeat Attacks

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

6.11 Countermeasure Technologies

6.11.1 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.

6.11.2 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.

6.11.3 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.

6.11.4 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.

6.11.5 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.

6.11.6 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.

6.11.7 Persistent Authorization

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

6.11.8 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.

6.12 Security Considerations

Implementers should take note, there is a vulnerability present even when an Web Services Security is used to protect to protect the integrity and origin of ebMS Messages. The significance of the vulnerability necessarily depends on the deployed environment and the transport used to exchange ebMS Messages.

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

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

The MIME headers are not protected, even when a Web Services Security based digital signature and/or Web Services Security based encryption is applied. Thus, an ebMS Message may be at risk depending on

how the information in the MIME headers is processed as compared to the information in the SOAP Envelope.

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

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

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

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

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

7 Error Handling Module

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

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

7.1 Terminology

- **Fault**: A Fault always means a SOAP Fault. It must be generated and processed according to the [SOAP11] or [SOAP12] specification.
- **Error**: An error that is not a SOAP Fault, and occurs in one of the defined modules (ebMS Module, Reliability Module, Security Module).
- **ebMS Error**: This is a particular case of Error, which is generated by the ebMS Module in conformity with this specification.
- **Reliability Error**: This is a particular case of Error, generated by the Reliability Module.
- **Security Error**: This is a particular case of Error, generated by the Security Module.
- **Escalated ebMS Error**: This is an ebMS Error that originates in a module other than the ebMS Module (i.e. Security module, or Reliability module).
- **ebMS Error Generation**: The operation of creating an ebMS Error object based on some failure or warning condition.
- **ebMS Error Reporting**: The operation of communicating an ebMS Error object to some other entity.
- **Message-in-error**: A flawed message causing an error of some kind.

7.2 Packaging of ebMS Errors

7.2.1 eb:Error Element

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

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

Example:

```
<eb:Error eb:origin="ebMS" eb:category="Unpackaging"
```

```
            eb:shortDescription="InvalidHeader"
            eb:errorCode="EBMS:0009" eb:severity="fatal">
    <eb:Description> … </eb:Description>
</eb:Error>
```

7.2.2 eb:Error/@origin

This OPTIONAL 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 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 eb:Error/@errorCode

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

7.2.5 eb:Error/@severity

This REQUIRED 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 eb:Error/@refToMessageInError

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

7.2.7 eb:Error/shortDescription

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

7.2.8 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 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 :

```
<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="EBMS:0101" eb:severity="failure"
                  eb:shortDescription="FailedAuthentication">
                 <eb:Description>Validation of signature failed</eb:Description>
        </eb:Error>
        <eb:Error eb:origin="ebMS" category="Communication"
                  eb:errorCode="EBMS:0006" eb:severity="warning"
                  eb:shortDescription="EmptyMessagePartitionFlow">
        <eb:Description>PullRequest done on an empty MPF</eb:Description>

     </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. ebMS Errors in addition to those specified here may be added by creating new errorCode values. The value of the errorCode attribute must begin 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 MSH raising the Error. Except for some cases defined in this specification, Error escalation policies are left to an agreement between users, represented in the processing mode of an MSH (P-Mode.errorHandling).

7.6 Error Reporting

There are three primary means of Error Reporting:

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

one eb:Error element. Such a reporting action is modeled by Send and Receive abstract operations over such a message.

Example of different options in reporting errors raised on a Sending MSH*: Some error detected on a submitted message and before it is even packaged, would normally be locally notified to the message Producer, and not even reported to the destination MSH. However, in case this message was part of a larger exchange that is holding its state waiting for completion on the receiving side, the preferred policy could state that the message-in-error be also reported (using an error message) to the Receiving MSH. If the Receiving MSH is getting its messages as responses to PullRequest signals, such ebMS errors can be transmitted as responses to these signals. If user messages are pushed sender to receiver, it could be decided that errors generated on the 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.

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0001	ValueNotRecognized	failure	Content	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.*
EBMS:0002	FeatureNotSupported	warning	Content	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.*
EBMS:0003	ValueInconsistent	failure	Content	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.
EBMS:0004	Other	failure	Content	

EBMS:0005	ConnectionFailure	failure	Communication	the MSH is experiencing temporary or permanent failure in trying to open a transport connection with a remote MSH.
EBMS:0006	EmptyMessagePartitionFlow	warning	Communication	There is no message available for pulling from this MPF at this moment.
EBMS:0007	MimeInconsistency	failure	Unpackaging	The use of MIME is not consistent with the required usage in this specification.
EBMS:0008	FeatureNotSupported	failure	Unpackaging	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.
EBMS:0009	InvalidHeader	failure	Unpackaging	The ebMS header is either not well formed as an XML document, or does not conform to the ebMS packaging rules.
EBMS:0010	ProcessingModeMismatch	failure	Processing	The ebMS header is not compatible with expected content based on the associated P-Mode.

1861

1862 7.7.2 Security Processing Errors

1863 The table below describes the Errors that originate within the Security Module, i.e. with @origin="security".
1864 These errors MUST be escalated by an MSH, meaning generated appropriately, or understood by an
1865 MSH when reported to it.

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0101	FailedAuthentication	failure	Processing	The signature in the Security header intended to the ebms SOAP actor, could not be validated by the Security module.
EBMS:0102	FailedDecryption	failure	Processing	The encrypted data reference the SecurityHeader intended to the ebMS SOAP actor could not be decrypted by the Security Module.
EBMS:0103	PolicyNoncompliance	failure	Processing	The processor determined that the message's security methods, parameters, scope or other security policy-level requirements or agreements were not satisfied.

1864

7.7.3 Reliable Messaging Errors

The table below describes the Errors that originate within the Reliable Messaging 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.

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0201	"DysfunctionalReliability"	failure	Processing	Some reliability function as implemented by the Reliability module, is not operational, or the reliability state associated with this message sequence is not valid.
EBMS:0202	DeliveryFailure	failure	communication	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.

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 10 and Figure 11, 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 subject to a reliability contract (see Section 8.2.1).

The abstract RM operations are defined as follows:

- **RM-Submit**
  An abstract operation that transfers a SOAP message from an RM-Producer to an Initiator RMP, so that this message can be sent reliably.
- **RM-Deliver**
  An abstract operation that transfers a SOAP message from a Responder RMP to its RM-Consumer, so that the payload from this message can later be delivered by the MSH.
- **RM-SubmitResponse**
  An abstract operation that transfers a SOAP message from an RM-Producer to a Responder RMP as a response to a previously received reliable message. This response is sent back reliably over the response leg of the same SOAP Request-response MEP instance that carried the previous reliable message.
- **RM-DeliverResponse**
  An abstract operation that transfers a received SOAP response message from an Initiator RMP to

its RM-Consumer.

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

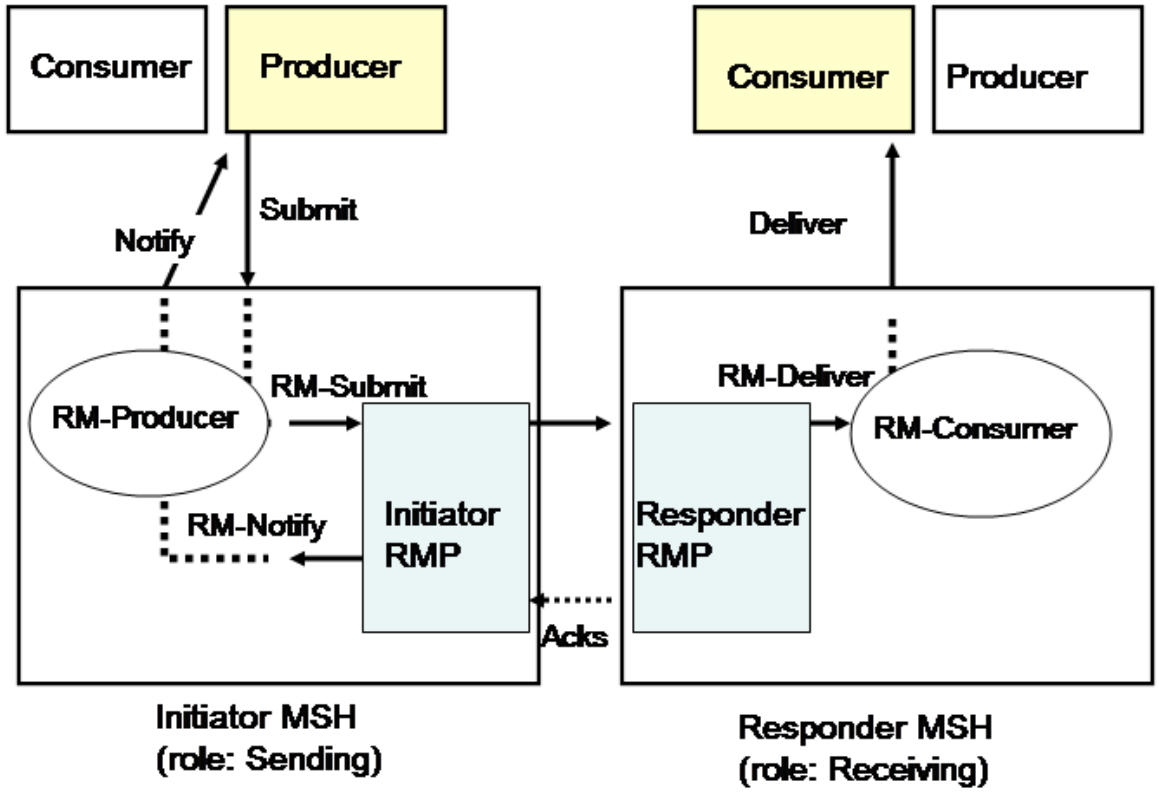


Figure 10: Reliable Request

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

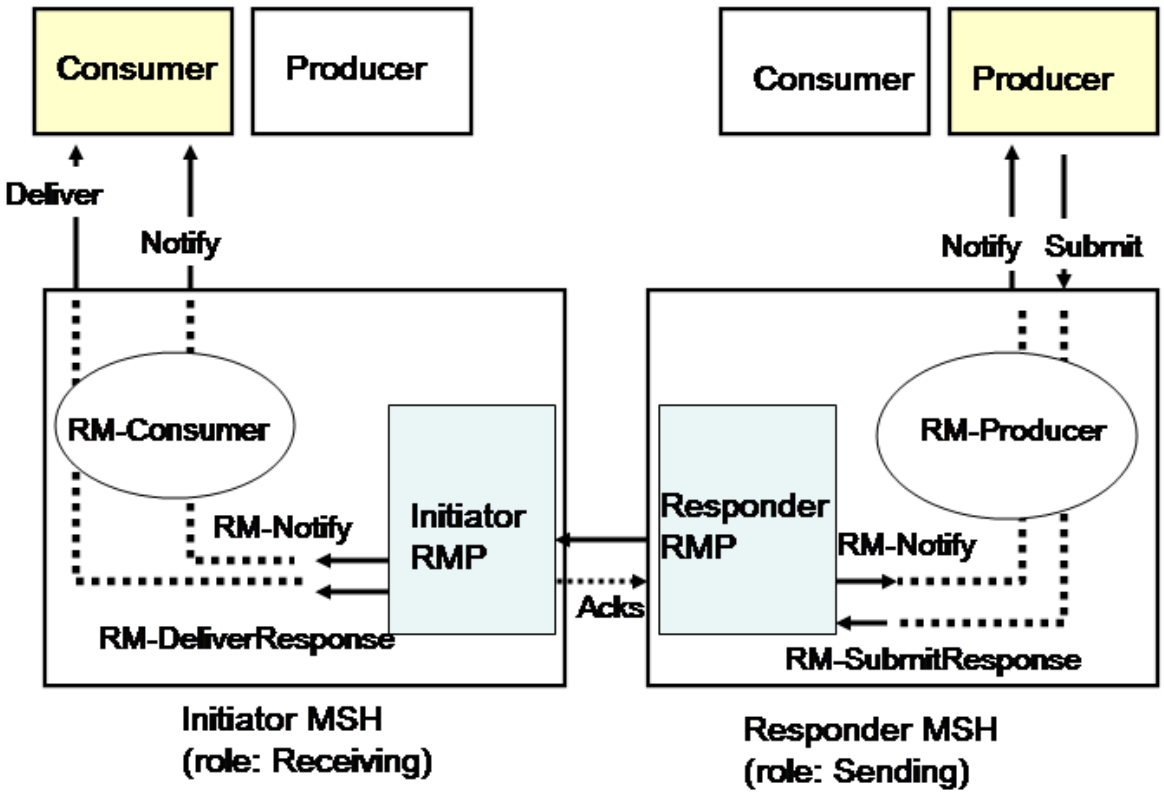


Figure 11: Reliable Response

Figure 11 shows the abstract operations and components involved when sending reliably a response – either a pulled user message in the One-way Pull MEP or the response user message in a Request-Response MEP (the second leg of a simple ebMS MEP). Note that depending on the reliability processing mode (P-Mode.reliability), awareness of delivery failure may occur on either side.

8.2 Reliability of ebMS Messages

8.2.1 Reliability Contracts

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

- **At-Least-Once Delivery**
  When sending a message with this reliability requirement (RM-Submit invocation), one of the two following outcomes shall occur: either (1) the Responder RMP successfully delivers (RM-Deliver operation) the message to the RM-Consumer or (2) either the Initiator RMP or the Responder RMP notifies (RM-Notify operation) respectively the RM Producer or the RM Consumer of a delivery failure.
- **At-Most-Once Delivery**
  Under this reliability requirement, a message submitted by an RM Producer (RM-Submit operation) to an Initiator RMP shall not be delivered more than once by the Responder RMP to its RM-Consumer. The notion of message duplicate is based on a notion of message ID that must be supported by the reliability specification being used.
- **In-Order Delivery**

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

These contracts MAY also apply to response messages, here called the MEP responses. In such a case they are expressed in the above contracts with RM-SubmitResponse and RM-DeliverResponse operations (instead ofRM-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/committees/ebxml-msg/service" 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 at-least-once contract is fulfilled with a successful RM-delivery. In case of at-least-once delivery, a failure do deliver must cause the generation of a delivery failure Error. If this message was submitted or initiated by an MSH Producer (Submit) instead of the MSH itself (e.g. a Ping initiated by the Producer), the Producer may be notified (Notify) of the failure depending on the reporting mode, as for regular user messages.

8.2.3 Message Header 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 12 illustrates the message flow for this reliable MEP.

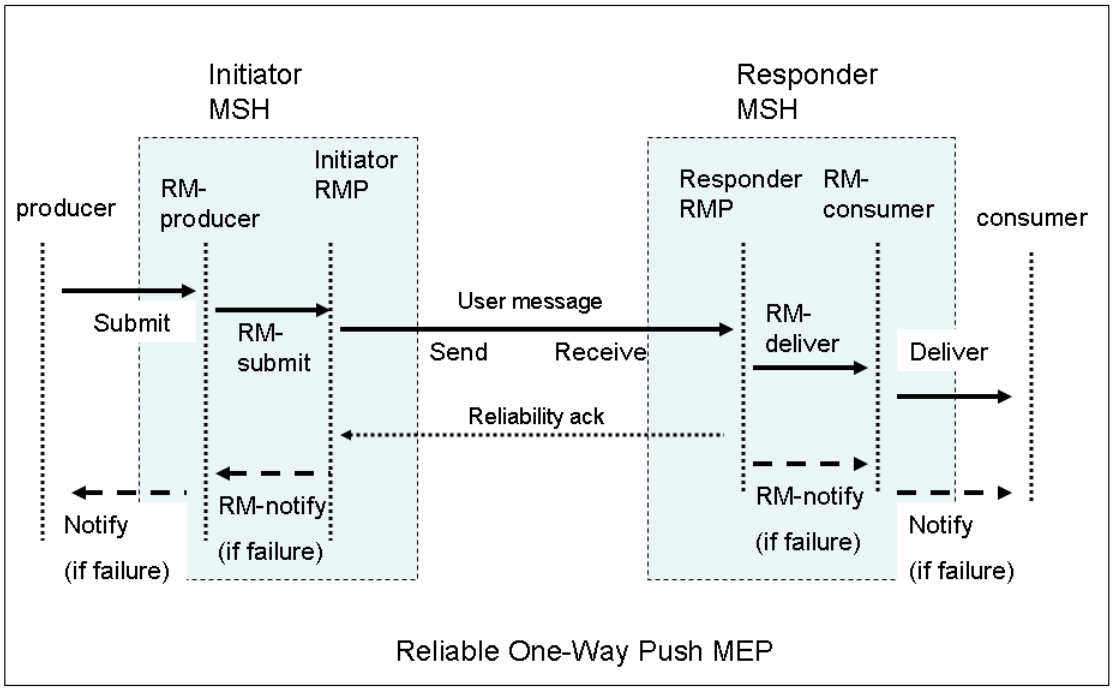


Figure 12: Reliable One-Way Push MEP

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 13 illustrates the message flow for this reliable MEP.

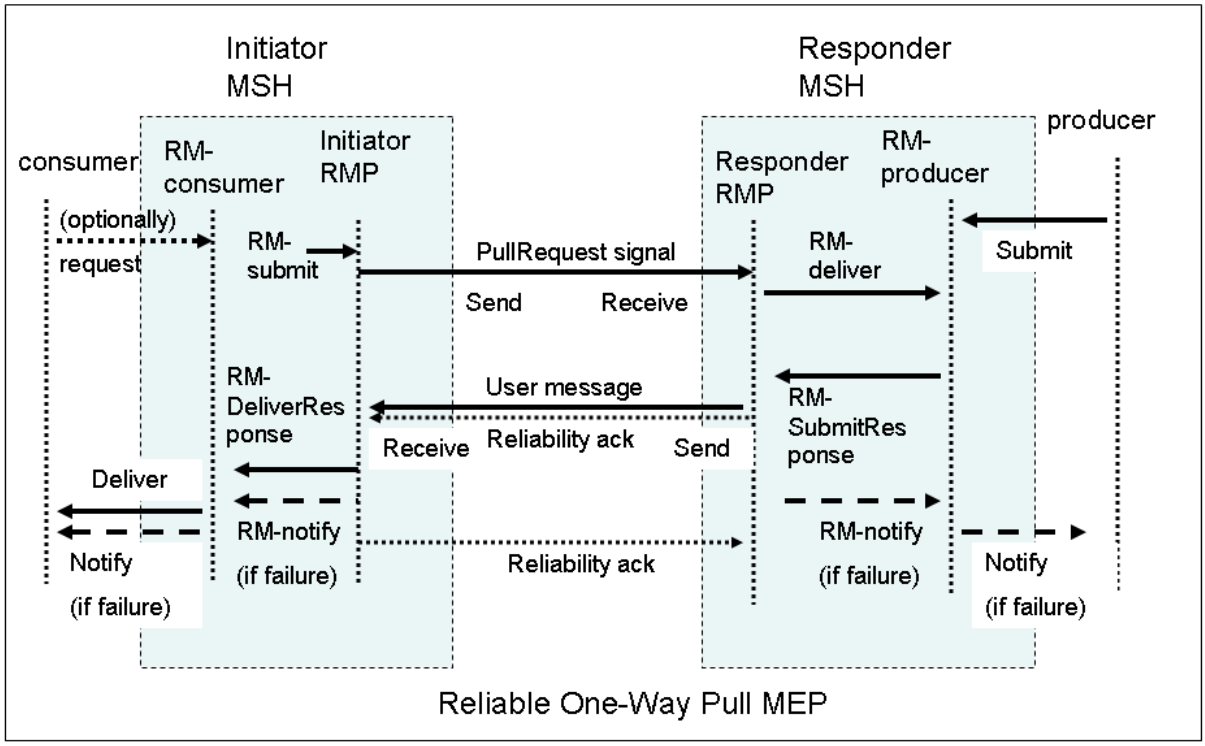


Figure 13: Reliable One-Way Pull MEP

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

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

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

8.3.3 Reliability of the Request-Reply MEP

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

On Initiator MSH side:

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

On Responder MSH side:

- Step (3): **RM-Deliver**: after processing of reliability headers, delivery of the request message to RM-Consumer.
- Step (4): **Deliver:** delivery of the request message data to the Consumer of the MSH.
- Step (5): **Submit**: submission of a response message data to the MSH by the Consumer of the request message, intended to the Producer on the Initiator side.
- Step (6): **RM-SubmitResponse:** submission by the RM-Producer of the response message to the Responder RMP.
- **On Initiator MSH side**:
- Step (7): **RM- DeliverResponse**: delivery of the response message to the RM-Consumer.
- Step (8): **Deliver**: delivery of the response message data to the Consumer of the Initiator MSH.

Figure 14 illustrates the message flow for this reliable MEP.

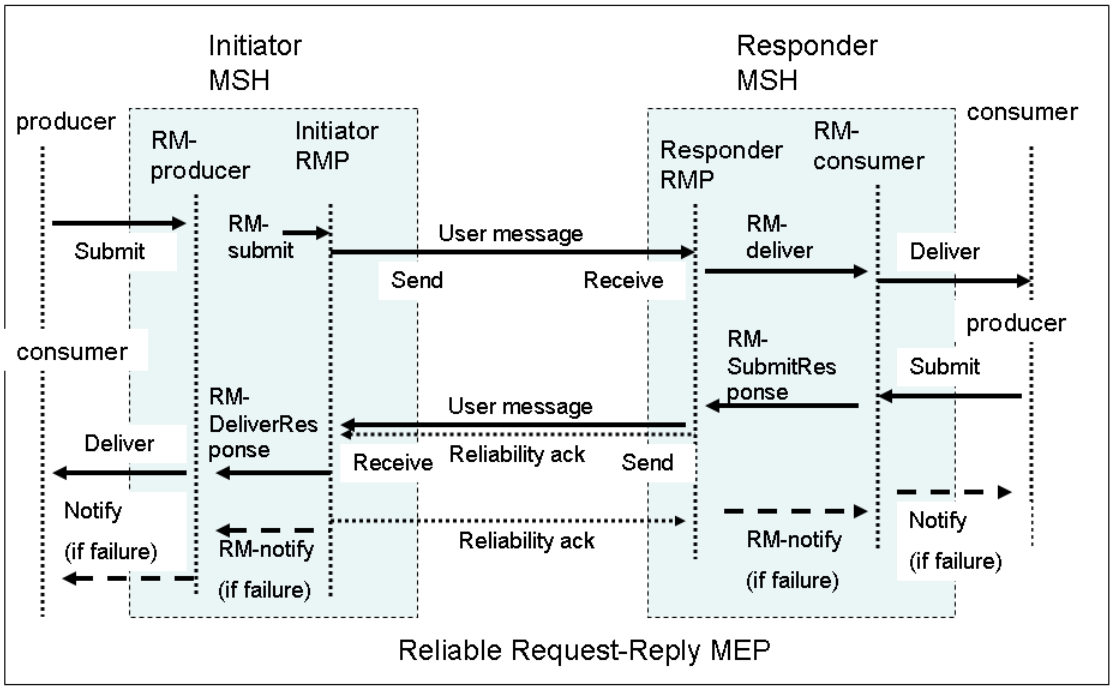


Figure 14: Reliable Request-Reply MEP

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.

9 The ebXML SOAP Extension Elements Schema (Appendix)

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

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

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

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

Following is the XML schema that describes the eb:Messaging header, as described in Section 5.2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
header-3_0.xsd"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-3_0.xsd">

  <xsd:element name="Messaging" type="Messaging"/>
  <xsd:complexType name="Messaging">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="SignalMessage"
type="SignalMessage"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="UserMessage"
type="UserMessage"/>
    </xsd:sequence>
    <xsd:attribute name="mustUnderstand" type="xsd:int"/>
    <xsd:attribute name="version" type="xsd:float"/>
  </xsd:complexType>

  <xsd:complexType name="SignalMessage">
    <xsd:sequence>
      <xsd:element name="MessageInfo" type="MessageInfo"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="PullRequest"
type="PullRequest"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="Error" type="Error"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="Error">
    <xsd:sequence>
      <xsd:element maxOccurs="1" minOccurs="0" name="Description"
type="xsd:token"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="ErrorDetail"
type="xsd:token"/>
    </xsd:sequence>
    <xsd:attribute minOccurs="0" name="category" type="xsd:token"/>
    <xsd:attribute minOccurs="0" name="RefToMessageInError" type="xsd:token"/>
    <xsd:attribute name="errorCode" type="xsd:token"/>
    <xsd:attribute minOccurs="0" name="origin" type="xsd:token"/>
    <xsd:attribute name="severity" type="xsd:token"/>
    <xsd:attribute minOccurs="0" name="shortDescription" type="xsd:token"/>
  </xsd:complexType>

  <xsd:complexType name="PullRequest">
    <xsd:sequence/>
    <xsd:attribute name="mpf" type="xsd:token"/>
  </xsd:complexType>

  <xsd:complexType name="UserMessage">
    <xsd:sequence>
      <xsd:element name="MessageInfo" type="MessageInfo"/>
      <xsd:element name="PartyInfo" type="PartyInfo"/>
```

```
      <xsd:element name="CollaborationInfo" type="CollaborationInfo"/>
      <xsd:element name="PayloadInfo" type="PayloadInfo"/>
    </xsd:sequence>
    <xsd:attribute minOccurs="0" name="mpf" type="xsd:token"/>
  </xsd:complexType>

  <xsd:complexType name="MessageInfo">
    <xsd:sequence>
      <xsd:element name="TimeStamp" type="xsd:dateTime"/>
      <xsd:element name="MessageId" type="xsd:token"/>
      <xsd:element minOccurs="0" name="RefToMessageId" type="xsd:token"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PartyInfo">
    <xsd:sequence>
      <xsd:element name="From" type="From"/>
      <xsd:element name="To" type="To"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="From">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" name="PartyId"
type="xsd:token"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="Role" type="xsd:token"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="To">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" name="PartyId"
type="xsd:token"/>
      <xsd:element maxOccurs="1" minOccurs="0" name="Role" type="xsd:token"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="CollaborationInfo">
    <xsd:sequence>
      <xsd:element name="Service" type="xsd:token"/>
      <xsd:element name="Action" type="xsd:token"/>
      <xsd:element name="ConversationID" type="xsd:token"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PayloadInfo">
    <xsd:sequence>
      <xsd:element maxOccurs="unbounded" minOccurs="1" name="PartInfo"
type="PartInfo"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="PartInfo">
    <xsd:sequence/>
    <xsd:attribute name="href" type="xsd:token"/>
  </xsd:complexType>
</xsd:schema>
```

10 Reliable Messaging Bindings (Appendix)

The reliability contracts defined in Section 8 may be implemented by profiling different reliability specifications. Either one of two OASIS reliability specifications may be used by an MSH implementation: WS-Reliability 1.1 [WS-R11], or WS-ReliableMessaging 1.1 [WSRM11].

Although either one of the above OASIS reliability specifications is sufficient, each one has strong arguments in favor of its use. In the same way as two MSH implementations must support the same transfer protocol or cryptographic algorithms in order to interoperate, two MSHs must also implement the same reliability specification in order to have interoperable reliability features. The reliability specification being used in an implementation is a parameter of the conformance profiles for ebMS (see Section 12).

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:

Name	Allowed Values	Additional Requirements
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: 1. The ReplyPattern value associated with the PullRequest signal is "Response". 2. 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 an 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:

Name	Allowed Values	Additional Requirements
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: 1. The ReplyPattern value associated with the PullRequest signal is "Response". 2. 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).
NoDuplicateDelivery	"enabled", "disabled"	When enabled, the Request message associated with this Response message MUST also be sent with this parameter enabled.
OrderedDelivery	"enabled", "disabled"	No restriction.
ReplyPattern	"Callback"	

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

10.2 WS-ReliableMessaging Binding

Note
This section is based on Committee Draft 3 (February 06) of the WS-ReliableMessaging specification [WSRMCD3]. For this reason, it may not be accurate with regard to a final WS-ReliableMessaging OASIS Standard; yet should serve to provide an indication of how such a binding can be achieved.

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 MPF. 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.

Among the features that may require further specification or profiling in order to enable MSH interoperability based on WS-ReliableMessaging, are:

1. In case the reliability contract and parameters do not apply equally to all messages sent between two MSHs, the scope of application of a reliability contract SHOULD be the sequence. Because a reliability module is not required to associate reliability contracts with particular message profiles, the reliability QoS that applies to a sequence SHOULD be communicated via CreateSequence / CreateSequenceResponse extensibility points using a format that remains to be determined.
2. In the case of the HTTP binding, an agreement or profiling on how the operations CreateSequence, CloseSequence and TerminateSequence, as well as their responses, are expected to bind to HTTP MEPs. Also part of this agreement or profiling: how sequence acknowledgements may bind to HTTP, and how they can be bundled with other messages, if applicable.

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:

(1) An implementation of ebMS-3 may choose SOAP-1.1 with Attachments

(2) An implementation of ebMS-3 may choose SOAP-1.2 with Attachments

Although a SOAP 1.2 version of SwA has not been formally submitted to W3C, it appears that most SOAP products have anticipated that usage, and after investigation, it appears that they have done so in a consistent, interoperable way. This specification is acknowledging these *de facto* upgrades of SwA, which are summarized below.

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):

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

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

<?xml version='1.0' ?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
3_0.xsd">
  <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://cars.example.com">
        <t:Mileage>20000</t:Mileage>
        <t:OwnerPicture href="cid:picture-of-owner"/>
     </t:Data>
  </S11:Body>
</S11:Envelope>
```

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

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

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

...binary TIFF image of the car's owner...
--MIME_boundary–
```

Example 1: SOAP-1.1 with Attachment

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:

```
Content-Type: Multipart/Related; boundary=MIME_boundary;
              type=application/soap+xml;
              start="<car-data@cars.example.com>"

--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <car-data@cars.example.com>

<?xml version='1.0' ?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
    xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
3_0.xsd">
  <S12:Header>
     <eb:Messaging eb:version="3.0" S12:mustUnderstand="true">
         …
           <eb:PayloadInfo>
              <eb:PartInfo href="cid:car-photo" />
              <eb:PartInfo href="#carData" />
           </eb:PayloadInfo>
     </eb:Messaging>
  </S12:Header>

  <S12:Body>
    <t:Data id="carData" xmlns:t="http://car.example.com">
        <t:Mileage>20000</t:Mileage>
        <t:OwnerPicture href="cid:picture-of-owner"/>
     </t:Data>
  </S12:Body>
</S12:Envelope>

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

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

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

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

--MIME_boundary--
```

Example 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.

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:

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

HTTP headers when using SOAP 1.1 with attachments

```
Content-Type: Multipart/Related; boundary=MIME_boundary;
              type=application/soap+xml;
              start="<car-data@cars.example.com>"; action=leasing
```

HTTP headers when using SOAP 1.2 with attachments

11.3 SMTP Binding

When using SMTP transport, 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:

```
From: user@customer.example.com
To: leasing-office@cars.example.com
Date: Mon, 23 Jan 2006 17:33:00 CST
Mime-Version: 1.0
SOAPAction: leasing
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
              start="<car-data@cars.example.com>"
```

SMTP headers when using SOAP 1.1 with attachments

```
From: user@customer.example.com
To: leasing-office@cars.example.com
Date: Mon, 23 Jan 2006 17:33:00 CST
Mime-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
              type=application/soap+xml;
              start="<car-data@cars.example.com>"; action=leasing
```

SMTP headers when using SOAP 1.2 with attachments

The remaining portions of the messages in the two examples above are respectively the same as the first two HTTP binding examples of Section 11.

An SMTP binding to ebMS One-way Pull or ebMS Request-reply MEPs relies on the SMTP binding of the SOAP Request-response MEP. The first message (SOAP request) must contain a resolvable SMTP address of the Sending MSH in the "From" SMTP header (this is not required when binding a SOAP One-way). Each leg of the SOAP Request-response MEP binds to an SMTP message. The "From" and "To" SMTP header values must be swapped from the SOAP request to the SOAP response.

12 Conformance (Appendix)

12.1 Introduction

This section introduces the notion of conformance profiles for MSH implementations. The expression "conformance profile" is to be understood in the sense of [QAFW]. A conformance profile in ebMS will define a class of implementations that may implement only a subset of this specification, and/or a particular set of options (e.g. transport protocol binding, SOAP version). This specification does not define nor recommend any specific conformance profile. Such conformance profiles will be defined separately from the ebMS standard, in an adjunct document. A particular conformance profile will be distinguished as the baseline for achieving interoperability between most implementations dedicated to e-Business or e-Government.

The section defines a common structure and syntax for defining conformance profiles.

Note: "Conformance profile" should not be confused with "usage profile":

- *Conformance profile*: defines a set of capabilities that an MSH implementation must have. This is determined at development time regardless of the way the MSH is being used later.
- *Usage profile*: defines a way of using an MSH implementation, that a community of users has agreed upon. This may in turn require a particular conformance profile.

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

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

- When the keywords OPTIONAL, SHOULD and MAY apply to the behavior of the implementation, the implementation is free to support these behaviors or not, as meant in [RFC2119].
- When the keywords OPTIONAL, SHOULD and MAY apply to message contents that relate to a more general feature, an implementation that conforms to a profile requiring support for this feature MUST be capable of processing these optional message contents according to the described ebXML semantics.
- The keywords REQUIRED, SHALL or MUST indicate features that an MSH must support or implement, but only within the context of a conformance profile requiring support for this feature or module containing this feature.
- When an MSH receives a message that exhibits some content feature that is either recommended or required by the specification, and if this MSH implements a conformance profile that does not require support for that content feature, then it MUST generate a FeatureNotSupported error (see Section 7).

12.2 Terminology

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

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

- role
- deployment type
- level
- interoperability parameters

Role: This property refers to one of the two roles identified in the messaging model (Section 2): Sending and Receiving.

Deployment Type: A deployment type characterizes a context in which the implementation operates and

the expected functional use for this implementation. For example, the following deployment types are expected to be among the most common, nonexclusive from others:

1. "*resource-constrained handler*". This characterizes an implementation that generally is not always connected, may not be directly addressable, may have no static IP address, has limited persistent capability, and is not subject to high-volume traffic.
2. "B2B *gateway*". This characterizes an implementation that generally is acting as the B2B gateway for an enterprise. It has a fixed address; it may have connectivity restrictions due to security; and it must support various types of connectivity with diverse partners.

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

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

1. The transport protocol supported, for which a non-exhaustive list of values is: HTTP, SMTP, HTTPS.
2. SOAP version: either SOAP 1.1 or SOAP 1.2.
3. The reliability specification supported, either WS-Reliability or WS-ReliableMessaging.

Conformance Profile: A conformance profile is then fully identified by a quadruple

< Role / DeploymentType / Level / InteropParameters>, or <R / D / L / P>, which is called the *profile summary*.

Functional Aspect: A conformance profile will impose specific requirements on different aspects of the specification, that are called here functional aspects. A set of (non-exhaustive) functional aspects is: Message Exchage Patterns, Error Reporting, Reliability, Security, Message Partition Flows, Message Packaging, Transport.

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

12.3 Conformance Profile Definition Template

Conformance Profile: **<Name>**	Profile summary: **<Role / Deployment Type / Level / Interoperability Parameters>**
Functional Aspects	**Profile Feature Set**
ebMS MEP	
Reliability	
Security	
Error Reporting	
Message Partition Flows	
Message Packaging	

Conformance Profile: **<Name>**		Profile summary: **<Role / Deployment Type / Level / Interoperability Parameters>**
Interoperability Parameters	**Transport and version**	
	SOAP version	
	Reliability specification and version	
	Security specification and version	

2354

13 References (Appendix)

[ASCI05] Unknown, *ASC X12 Registry*, ????. <url-unknown>

[ebCPPA] OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, *Collaboration-Protocol Profile and Agreement Specification Version 2.0*, 2002. <http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf>

[ebRISK] ebXML Security Team, *ebXML Technical Architecture Risk Assessment v1.0*, 2001. <http://ebxml.org/specs/secRISK.pdf>

[IANAMEDIA] Various, *MIME Media Types*, Various. <http://www.iana.org/assignments/media-types/>

[ISO6523] Unknown, *Identification of organization identification schemes*, 1998. <http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=25773>

[ISO9735] Unknown, *EDIFACT*, 1988. <http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17592>

[QAFW] Karl Dubost, et al, eds, *QA Framework: Specification Guidelines*, 2005. <http://www.w3.org/TR/qaframe-spec/>

[RFC2045] N Freed, et al, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet M*, 1996. <http://www.ietf.org/rfc/rfc2045.txt>

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, 1997. <http://www.ietf.org/rfc/rfc2119.txt>

[RFC2246] T. Dierks, et al, *The TLS Protocol Version 1.0*, 1999. <http://www.ietf.org/rfc/rfc2246.txt>

[RFC2387] E. Levinson, *The MIME Multipart/Related Content-type*, 1998. <http://www.ietf.org/rfc/rfc2387.txt>

[RFC2392] E. Levinson, *Content-ID and Message-ID Uniform Resource Locators*, 1998. <http://www.ietf.org/rfc/rfc2392.txt>

[RFC2396] T. Berners-Lee, et al, *Uniform Resource Identifiers (URI): Generic Syntax*, 1998. <http://www.ietf.org/rfc/rfc2396.txt>

[RFC2402] S. Kent, et al, *IP Authentication Header*, 1998. <http://www.ietf.org/rfc/rfc2402.txt>

[RFC2617] J. Franks, et al, *HTTP Authentication: Basic and Digest Access Authentication*, 1999. <http://www.ietf.org/rfc/rfc2617.txt>

[RFC2822] P. Resnick, ed., *Internet Message Format*, 2001. <http://www.ietf.org/rfc/rfc2822.txt>

[RFC3023] M. Murata, et al, *XML Media Types*, 2001. <http://www.ietf.org/rfc/rfc3023.txt>

[SOAP11] D. Box, et al, *Simple Object Access Protocol (SOAP) 1.1*, 2000. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

[SOAP12] M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework*, 2003. <http://www.w3.org/TR/soap12-part1/>

[SOAPATTACH]..... J. Barton, et al, *SOAP Messages with Attachments*, 2000. <http://www.w3.org/TR/SOAP-attachments>

[UTF8] F. Yergeau, *UTF-8, a transformation format of ISO 10646*, 1998. <http://www.ietf.org/rfc/rfc2279.txt>

[WS-R11] Kazunori Iwasa, et al, eds, *WS-Reliability 1.1*, 2004. <http://docs.oasis-open.org/wsrm/ws-reliability/v1.1/wsrm-ws_reliability-1.1-spec-os.pdf>

[WSIAP10] Chris Ferris, et al, eds, *Attachments Profile Version 1.0*, 2004. <http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html>

[WSIBSP10] Abbie Barbir, et al, eds, *Basic Security Profile Version 1.0*, 2005. <http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html>

[WSRM11] Gilbert Pilz, et al, eds, *Web Services Reliable Messaging*, 2005. <http://www.oasis-open.org/apps/org/workgroup/ws-rx/download.php/15177/wsrm-1.1-spec-cd-01.pdf>

[WSRMCD3] Doug Davis, et al, eds., *Web Services Reliable Messaging*, 2006. <http://www.oasis-open.org/>

[WSS10] Anthony Nadalin, et al, eds., *Web Services Security: SOAP Message Security 1.0*, 2004.

<http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>

[WSS10-USER] P. Hallam-Baker, et al, eds., *Web Services Security UsernameToken Profile 1.0*, 2004. <http://docs.oasis-open.org/wss/2004/01/>

[WSS10-X509] P. Hallam-Baker, et al, eds., *Web Services Security X.509 Certificate Token Profile*, 2004. <http://docs.oasis-open.org/wss/2004/01/>

[WSS11] Anthony Nadalin, et al, eds., *Web Services Security: SOAP Message Security 1.1*, 2005. <http://docs.oasis-open.org/wss/v1.1/>

[WSS11-USER] A. Nadalin, et al, eds., *Web Services Security UsernameToken Profile 1.1*, 2006. <http://docs.oasis-open.org/wss/v1.1/>

[WSS11-X509] A. Nadalin, et al, eds., *Web Services Security X.509 Certificate Token Profile 1.1*, 2006. <http://docs.oasis-open.org/wss/v1.1/>

[XLINK] Steve DeRose, et al, eds., *XML Linking Language (XLink) Version 1.0*, 2001. <http://www.w3.org/TR/xlink/>

[XML10] Tim Bray, et al, eds., *Extensible Markup Language (XML) 1.0 (Third Edition)*, 2004. <http://www.w3.org/TR/2004/REC-xml-20040204/>

[XMLDSIG] Donald Eastlake, et al, eds, *XML-Signature Syntax and Processing*, 2002. <http://www.w3.org/TR/xmldsig-core/>

[XMLID] J. Marsh, et al, eds., *xml:id Version 1.0*, 2005. <http://www.w3.org/TR/xml-id/>

[XMLNS] Tim Bray, et al, eds, *Namespaces in XML*, 1999. <http://www.w3.org/TR/REC-xml-names/>

[XMLSCHEMA] Henry S. Thompson, et al, eds., *XML Schema Part 1: Structures Second Edition*, 2004. <http://www.w3.org/TR/xmlschema-1/>

[XPATH] James Clark, et al, eds., *XML Path Language (XPath) Version 1.0*, 1999. <http://www.w3.org/TR/xpath>

A 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:

•

B 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.]

Rev	Date	By Whom	What
WD 01	5 May 2004	Matt MacKenzie	Moved content over from 2.0/2.1 document source.
WD 02	14 May 2004	Matt MacKenzie	A few updates to the explanations and more thorough usage of available styles.
WD 03	1 Oct 2004	Matt MacKenzie	Integrated Reliable messaging, many editorial changes also.
WD 04	28 Sept 2005	Pete Wenzel	• Applied OpenOffice Template, formatting changes. • New Messaging Model section from Jacques (was Section 6 in draft 03; is now Section 3). • New Message Packaging section from Jacques (was Section 8; is now Section 5). • New Security Module section from Ric (was Section 10.1; is now Section 6.1). • New Reliable Messaging Module section from Jacques (was Section 10.6; is now Section 6.6). • New WS-Reliability Binding section from Jacques (Section 9.1).
WD 05	05 Oct 2005	Pete Wenzel	• Changed title to indicate this is Part 1. • Moved several sections to a new Part 2 (Advanced Features) document, for future reference. • Rewritten Introduction & Operation sections from Jacques. • Messaging Model and Message Packaging section updates from Jacques. • Began Bibliography Database and insertion of references. • Section rearrangement and edits as discussed 10/05/2005.
WD 06	21 Oct 2005	Pete Wenzel	• New Error Module section from Hamid & Jacques. • Security Section updates from Ric. • Added Iwasa and Hamid as contributors. • New Packaging Diagrams from Iwasa & Jacques. • Removed sections (to be considered for later Advanced Features document): FTP Binding, Security Services Profiles, WSDL. • Minor updates throughout.

Rev	Date	By Whom	What
WD 07	23 Nov 2005	Pete Wenzel	This revision is a candidate for Committee Draft status. • Editorial corrections to Introduction. • Overhaul of Messaging Model and Error Sections by Jacques & Hamid. • Editorial corrections to Operation Context. • New Message Pulling Module from Fujitsu. • Minor updates to Message Packaging. • Additional Security Examples, New Sign+Encrypt Sections from Ric. • Additional minor corrections throughout. • References, formatting, reorganization, other editorial changes throughout. • Designated several of the later Sections as (Appendix).
CD 01	30 Nov 2005	Pete Wenzel	This revision has been voted Committee Draft status. • Updated Status statement and other standard boilerplate text on title page. • Changed incorrect "RMP" references to "MSH". • Updated Figure 5 and removed corresponding EdNote.
WD 08	13 Feb 2006	Pete Wenzel	• Replaced eb:Message by eb:Messaging. • Update Figures 7 & 8. • Renumbered Section 5.2 • New Conformance Appendix, from Jacques' Draft 0.7 (for continued review) • New SOAP Format and Bindings Appendix draft from Hamid (for review) • Editorial updates to Reliability Binding Section from Jacques • WS-ReliableMessaging Binding from Jacques (for review) • Completed Bibliography; removed many redundant references.

Rev	Date	By Whom	What
WD 10	07 Mar 2006	Pete Wenzel	• Updated occurrences of Partref (now PartInfo) and @idref (now @href), and removed eb:id. • Removed sections related to SOAP actors. • Removed @mustUnderstand section (redundant). • Removed references to @soapresp; no longer used. • Corrections in sections 8.1 and 8.2.2 from Jacques. • Added ProcessingModeMismatch and DysfunctionalReliability errors; renumbered error codes by section. • Corrections to SOAP One-Way MEP (2.2.2.1). • Corrections to Message Pulling Objectives (4.1). • Replaced Concept of Operation section with Processing Mode (#91 from Jacques & Hamid); changed terminology from "operation context" to "P-Mode". • Added Message Packaging Examples section. • Corrections to Reliability Protocol Bindings. • Conformance Appendix: Removed specific conformance profiles; replaced with template (Conformance #10 from Jacques). • Removed StatusRequest/Response signals from Signal Message Packaging Figure. • Replaced Message Pipes section with latest (#12 from Jacques). • Removed Examples of Supported Topologies section. • Added Namespace Table from Hamid. • Note about WSS 1.0/1.1 in Section 6.1. • Minor edits, Sections 2, 4, 5 from Hamid. • Added @refToMessagInError. • Corrected references to errorCodes/shortDescriptions. • Removed/replaced justification text from SOAP Binding section. • Removed Section 11 (old Protocol Binding section). • Corrected SOAP 1.2 media type. • Removed 4 simplest Security packaging examples from Section 6; retained signed+encrypted examples, which depict all necessary elements. • Added proposed Security Requirements section from Ric. • Added WS-ReliableMessaging status statements to binding section. Did not yet change: • In 5.2.1, eb:CollaborationInfo OPTIONAL for Response User Message. ??

Rev	Date	By Whom	What
WD 11	20 Mar 2006	Pete Wenzel	• Removed SecurityTokenReference from 6.9.1.1 • Editorial corrections in Sections 2 & 3 from Jacques. • Updated Figure 9 to depict multiple eb:Error elements. • Removed @syncresp. • Updated @pipe usage. • Updated URI constants to be URLs instead of URNs. • Changed reference to "ping" to a new "test" action (5.2.1.9). • Section 1.2, CPP/A positioning, suggested by M. Martin. • Security Section rewrite from Ric. • Added new Security Error Codes: FailedDecryption and PolicyNoncompliance. • Removed unneeded material from PullRequest Security Section. • Added Section 1.3 caveat regarding specification alternatives as proposed by Jacques. • Additions to SMTP Binding from Jacques.
WD 12	10 Apr 2006	Pete Wenzel	• Message Service -> Messaging Service. • Pipe -> Message Partition Flow in text and figures. • Fixed URI in examples. • Editorial Corrections from Jacques. • XML Schema from Hamid.
CD 02	12 Apr 2006	Pete Wenzel	• Renamed @mpflow -> @mpf. • Adjusted cardinality of error attributes. • Inserted ConversationInfo in example.

2441

C 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 implementors 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.