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

Committee Specification 02, 12 July 2007

Document Identifier:

ebms_core-3.0-spec-cs-02

This Version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cs02/ebms_core-3.0-spec-cs-02.pdf

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cs02/ebms_core-3.0-spec-cs-02.html

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cs02/ebms_core-3.0-spec-cs-02.odt

Previous Version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cd06/ebms_core-3.0-spec-cd-06.pdf

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cd06/ebms_core-3.0-spec-cd-06.html

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/cd06/ebms_core-3.0-spec-cd-06.odt

Latest Version:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.pdf

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.html

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms_core-3.0-spec.odt

Technical Committee:

OASIS ebXML Messaging Services TC

Chair:

Ian Jones, British Telecommunications plc <ian.c.jones@bt.com>

Editor:

Pete Wenzel, Sun Microsystems <pete.wenzel@sun.com>

Related Work:

This specification is related to:

- ebXML Message Services 2.0
- SOAP 1.1, 1.2
- Web Services Security: SOAP Message Security 1,0, 1.1
- WS-Reliability 1.1
- WS-ReliableMessaging 1.1

Declared XML Namespace:

http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/

Abstract:

This specification defines a communications-protocol neutral method for exchanging electronic business messages. It defines specific Web Services-based 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).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/ebxml-msg/.

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

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

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

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

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' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", "ebMXL", "OASIS ebXML Messaging Services", "ebMS" are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

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.

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. Multiple payloads, possibly of different MIME types, can be transported in a single ebMS message. 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, may 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. Web Services and Their Role in an eBusiness Messaging Framework

A major design choice in ebMS 3, is the specification of the MSH and its associated processing rules using Web Services standards. The intent is to make use of other relevant Web Services specifications that fulfill certain messaging requirements, and build upon that base by adding what is necessary for a complete and coherent eBusiness messaging service. ebMS 3 brings this all together into a single, coherent framework.

In order to achieve this, message security and reliability requirements are met through the use of other Web Services standards and their implementations. The message SOAP body has been freed for business payload. The ebMS header is just a SOAP extension among others. As a result, ebMS 3 is significantly more compliant than ebMS 2 with the SOAP processing model, and apt at composing Web services standards that are defined as SOAP extensions. Compliance of ebMS 3 implementations with the latest version of WS-I profiles - once approved as final material by the organization - will be addressed in the definition of conformance profiles that are adjunct to this specification (see Appendix G).

Compliance with Web services standards does not remove the rationale behind an Internet-based messaging middleware. Often, document-centric eBusiness and eGovernment exchanges need to clearly dissociate messaging functions from the way these messages are consumed on the back-end. Such consumption may take place according to various models, as mentioned in 1.1. The use of [SOAP] message header elements that represent standard business metadata (user or company ID, business conversation, business service and action, etc.), is a key feature for supporting a decoupled binding with back-end business processes. At the same time, experience has demonstrated that the messaging layer must be more supportive of business transactions: messages are parts of basic choreographies that map to higher-level business exchanges between partners. To this end, ebMS 3 supports a notion of message exchange pattern (MEP) the properties of which (reliability, security, binding to underlying transport, error handling, and other quality of service aspects such as timing, etc.) are controlled in a contract-based manner by the message producer and consumer layers.

1.4. 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.

Implementers 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 Web Services specifications that 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. Conformance to compatible profiles is a prerequisite to interoperability. See Appendix G for more details on conformance profiles.

1.5. 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 in the section that specifies that module.
2. When the keywords MUST, SHALL, or REQUIRED are used to qualify a feature, support for this feature--either message content or implementation behavior--is mandatory in an implementation with a conformance profile that requires this feature.
3. It complies with the following interpretation of the keywords OPTIONAL and MAY: When these keywords apply to the behavior of the implementation, the implementation is free to support these behaviors or not, as meant in [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.
4. If it has implemented optional syntax, features and/or behavior defined in this specification, it MUST be capable of interoperating with another implementation that has not implemented the optional syntax, features and/or behavior. It MUST be capable of processing the prescribed failure mechanism for those optional features it has chosen to implement.
5. It is capable of interoperating with another implementation that has chosen to implement optional syntax, features and/or behavior, defined in this specification, it has chosen not to implement. Handling of unsupported features SHALL be implemented in accordance with the prescribed failure mechanism defined for the feature.

1.6. XML Notation

When describing concrete XML schemas and information items, this specification uses a convention in which each XML element or attribute is identified using abbreviated [XPATH] notation (e.g., /x:MyHeader/x:SomeProperty/@attribute).

1.7. Namespace Prefixes

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

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
ds	http://www.w3.org/2000/09/xmldsig#
eb	http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
enc	http://www.w3.org/2001/04/xmlenc#
wsr	http://docs.oasis-open.org/wsrm/2004/06/ws-reliability-1.1.xsd
wsrx	http://docs.oasis-open.org/ws-rx/wsrm/200702
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
ebbpsig	http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0

1.8. Example Domains

Hostnames used in the examples are fictitious, and conform to [RFC2606]. The example.org domain is intended to refer generically to a relevant industry standards organization, while the example.com domain represents a participant in a message exchange (whether commercial, government, or other entity).

1.9. Normative References

[HTTP11] R. Fielding, et al, *Hypertext Transfer Protocol -- HTTP/1.1*, 1999. <http://www.ietf.org/rfc/rfc2616.txt>

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

[RFC2045] N Freed, et al, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, 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>

[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>

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

[SMTP] J. Klensin, ed., *Simple Mail Transfer Protocol*, 2001. <http://www.ietf.org/rfc/rfc2821.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>

[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>

[WSR11] 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>

[WSRM11] D. Davis, et al, eds, *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1*, 2007. <http://docs.oasis-open.org/ws-rx/wsrm/v1.1/wsrm.pdf>

[WSRMP11] D. Davis, et al, eds, *Web Services Reliable Messaging Policy (WS-RM Policy) Version 1.1*, 2007. <http://docs.oasis-open.org/ws-rx/wsrmp/v1.1/wsrmp.pdf>

[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/>

[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/>

[XMLENC] D. Eastlake, et al, *XML Encryption Syntax and Processing*, 2002. <http://www.w3.org/TR/xmlenc-core/>

[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>

1.10. Non-Normative References

[ebBP-SIG] OASIS ebXML Business Process TC, *ebXML Business Signals Schema*, 2006. <http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0>

[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>

[ebCPPA21] OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee, *Collaboration-Protocol Profile and Agreement Specification Version 2.1*, 2005. <http://www.oasis-open.org/committees/download.php/12208/ebcpp-2.1-april-5-2005-draft.doc>

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

[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/>

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

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

[RFC2606] D. Eastlake, et al, *Reserved Top Level DNS Names*, 1999. <http://www.ietf.org/rfc/rfc2606.txt>

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

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

[SOAPEMAIL] H. M. Mountain, et al, *SOAP Version 1.2 Email Binding*, 2002. <http://www.w3.org/TR/soap12-email>

[WSPOLICY] A. Vedamuthu, et al, eds, *Web Services Policy 1.5: Framework*, 2007. <http://www.w3.org/TR/ws-policy/>

[WSSECPOL] A. Nadalin, et al, eds, *WS-SecurityPolicy 1.2*, 2007. <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf>

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 that is able to generate or process messages that conform to this specification, and to act in at least one of two ebMS roles defined below in Section 2.1.3: Sending and Receiving. In terms of SOAP processing, an MSH is either a SOAP processor 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 the 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 the Receiving role) to consume data from a received user message. Some examples are: an application, a queuing system, another SOAP processor.

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

> Notes:
>
> In all figures, the arrows do not represent control flow, i.e. they 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.
>
> Producer and Consumer are always MSH endpoints, and Submit and Deliver operations occur at the endpoints only once per message lifetime. Any actions performed by an intermediary will be defined in different terms.

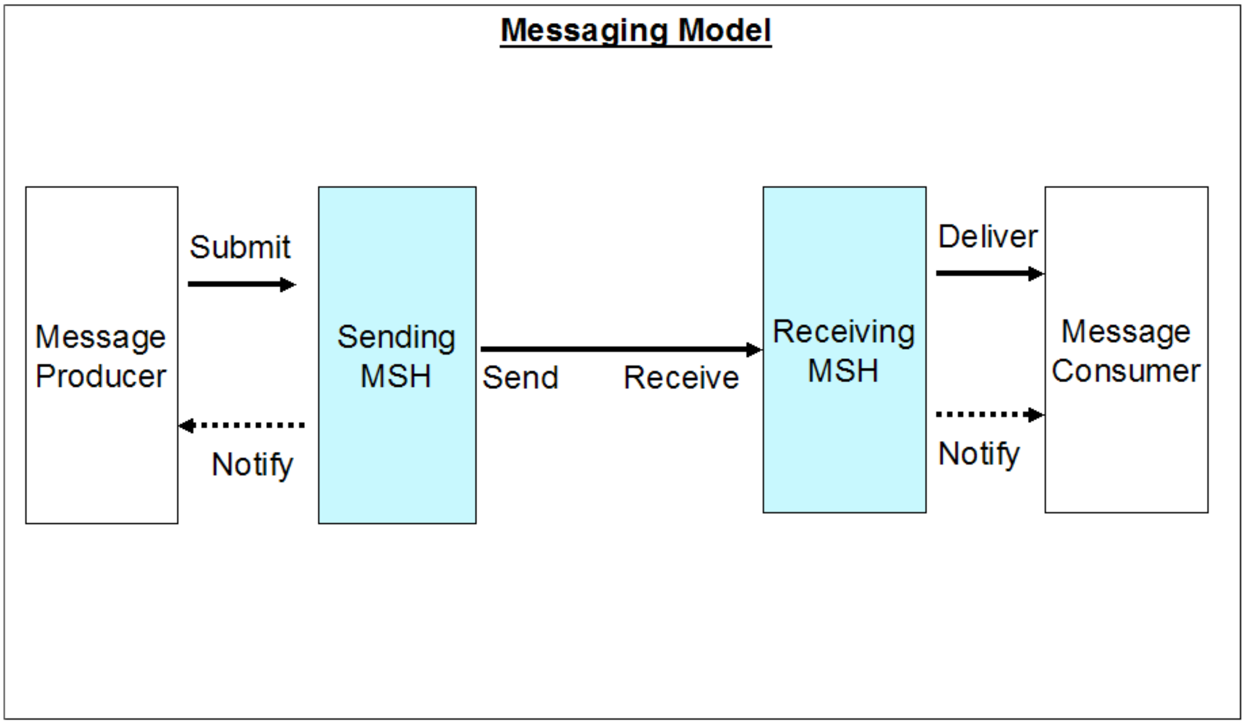

Figure 1: Entities of the Messaging Model and Their Interactions

2.1.2. Message Terminology

An **ebMS Message** is a SOAP message that contains SOAP header(s) qualified with the ebMS namespace, and that conforms to this specification.

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

- an **ebMS User Message Unit**, which is represented by the XML infoset eb:Messaging/eb:UserMessage, together with any referenced payload items. This is the part of the ebMS message that is submitted by a Producer (via Submit operation) and that is subject to delivery to a Consumer.
- an **ebMS Signal Message Unit**, represented by the XML infoset eb:Messaging/eb:SignalMessage. Its role is to activate a specific function in the Receiving MSH. It is not intended to be delivered to a message Consumer.

An **ebMS User Message** is an ebMS message that contains a User Message unit (in other words, it contains an eb:UserMessage element as a child of eb:Messaging).

An **ebMS Signal Message** is an ebMS message that contains a Signal Message unit. A Signal Message that contains an eb:PullRequest element is also called a Pull Signal Message.

An ebMS Message may contain both a User Message Unit and a Signal Message Unit. In that case it is both a Signal Message and a User Message.

2.1.3. Messaging Roles

The Messaging Model assumes the following roles for an MSH:

- **Sending**: When an MSH acts in the Sending role, it performs the 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 even in a Sending role, an MSH MAY be required to receive and process some types of Signal Messages, depending on the conformance profile in use.)
- **Receiving**: An MSH acting in the Receiving role performs the 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 even in a Receiving role, an MSH MAY be required to generate and 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 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. Rationale

Two communicating partners may agree to conduct business transactions as message sequences that follow well defined patterns, or Message Exchange Patterns (MEP). Enforcing these patterns is usually done above the messaging layer. However it has proved useful to support some aspects of such MEPs in the messaging layer. In particular:

- The correlation between messages, when expressed directly via a referencing mechanism that appears in the message header, allows for efficient monitoring and enforcement of MEPs.
- As an MSH has to bind messages to the transport protocol, these binding requirements may be better expressed and controlled at MEP level. For example, different messages of the same MEP (such as a request and a response) may be required to bind differently to the transport.

An ebMS MEP represents the part of such exchange patterns that is controlled and implemented by an MSH, thus making an abstraction of the business semantics. Although the notion of MEP was not explicitly supported by ebMS 2.0, it can be noted that it provided some informal support for MEPs, such as message referencing (RefToMessageId) and the SyncReply element that controls the use of the back-channel of the underlying protocol. In the following, the acronym "MEP" implicitly means ebMS MEP, unless otherwise qualified.

The goal of this specification is to introduce a model for ebMS MEPs, rather than a formal representation of them. This model is the basis for partners agreeing to which MEPs their exchanges will conform. Such agreements are manifested in Processing Modes, or P-Modes, the representation of which is outside the scope of this specification. The P-Mode also defines which message profile is associated with which MEP, and the role it plays in this MEP. Processing Modes are described in detail in Section 4.

2.2.2. General Definition

An **ebMS MEP** defines a typical choreography of ebMS User Messages which are all related through the use of the referencing feature (RefToMessageId). Each message of an MEP instance refers to a previous message of the same instance, unless it is the first one to occur. Messages are associated with a label (e.g. "request", "reply") that precisely identifies their direction between the parties involved and their role in the choreography.

> Note: Because RefToMessageId more accurately defines a referencing between User Message Units than between User Messages (SOAP messages), MEPs are preferably defined here as exchanges of Message Units, rather than of ebMS Messages.

Two MEPs are defined in this specification, not exclusive of others:

- The **One-Way MEP** which governs the exchange of a single User Message Unit unrelated to other User Messages. Its label is "oneway" and is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay` .
- The **Two-Way MEP** which governs the exchange of two User Message Units in opposite directions, the first one to occur is labeled "request", the other one "reply". In an actual instance, the "reply" must reference the "request" using eb:RefToMessageId. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay`.

The MEP definitions are primarily concerned with the transfer of ebMS User Message Units. Instances of such MEPs may involve or cause the transfer of additional messages or the piggy-backing of additional elements (e.g. ebMS signal messages or units such as errors, receipts, pull requests, and low-level Acknowledgments when using reliability), but these are not taken into account in the MEP definition. Instead, the different ways these additions can be associated with the MEPs defined here, are considered as part of the execution mode of the MEP, which is controlled by some agreement/configuration external to the MEP definition (see P-Modes in Section 4). Some extra messages (Signal messages) may also be mandated by the binding of an ebMS MEP (see channel-binding), but are not relevant to the ebMS MEP definition itself.

MEP definitions in this document are restricted to exchanges between two MSHs.

2.2.3. MEP Bindings

The previous definition of ebMS MEP is quite abstract, and ignores any binding consideration to the transport protocol. This is intentional, so that application-level MEPs can be mapped to ebMS MEPs independently from the transport protocol to be used. In addition to agreeing on MEP usage, the following notions of MEP bindings should be subject to agreements between partners:

- An **ebMS MEP Transport Channel Binding** defines how the MEP maps to the channels allowed by the underlying transport protocol, while making an abstraction of this underlying transport. In case of a two-way transport, the transport channel binding defines whether each message of the MEP maps to the fore-channel (or first leg) or back-channel (second leg). It also tells if an ebMS Signal is needed to initiate the transfer - e.g. by pulling - and which one. Appendix E shows possible options for combining headers supporting reliable messaging as well as error reporting, when binding basic ebMS MEPs to a two-way protocol such as HTTP. The Appendix also shows how these combinations can be controlled with P-Mode parameters.
- An **ebMS MEP Transport Protocol Binding** defines further how an MEP transport channel binding is implemented over a specific underlying transport protocol such as HTTP or SMTP. For example, an HTTP transport protocol binding will define the usage of HTTP headers and methods for each message. A transport protocol binding usually relies on standard SOAP bindings when these exist.

A transport channel binding is a critical complement to an MEP, to be agreed on in order for partners to interoperate. The rationale in using different transport channel bindings for an ebMS MEP is to accommodate different connectivity constraints (e.g. firewall restrictions, intermittent availability, non-static IP address) by dictating how each message transfer is initiated over the underlying protocol. Because such connectivity constraints usually exist independently from the details of the transport protocol, the transport channel binding is the right level to address them. The transport channel bindings identified in this specification are:

- **Push**: maps an MEP User message to the 1st leg of an underlying 2-way transport protocol, or of a 1-way protocol. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push`.

- **Pull**: maps an MEP User message to the second leg of an underlying two-way transport protocol, as a result of an ebMS Pull Signal sent over the first leg. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull`.
- **Sync**: maps an exchange of two User messages respectively to the first and second legs of a two-way underlying transport protocol. This binding is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync`.

  Notes:

  - An underlying transport protocol qualifies as "two-way" if (a) it guarantees a transport channel for transferring the response of every message (request) initiated by an MSH, back to this MSH without need for explicit addressing information in SOAP headers, and regardless of connectivity restrictions such as inability to accept incoming new connections; and (b) it provides to the MSH initiator of the exchange, some means for correlating the response with the request, without relying on the SOAP header. For example, HTTP qualifies as two-way, but SMTP and FTP do not (although FTP has a notion of session, it does not inherently support the coupling of (b)). The channel offered in (a) is also called "back-channel" in this specification.
  - "Pull" and "Sync" above cannot be used with a one-way underlying protocol.
  - Communicating parties must agree on a transport channel binding: a sending MSH will treat a message submitted for pulling differently from a message submitted for pushing.

An MEP that is associated with a particular transport channel binding is also called a transport-channel-bound MEP. A transport-channel-bound MEP is identified by a pair <MEP name / transport-channel-binding name>. For example, a Two-Way ebMS MEP that executes over a single request-response exchange of the underlying transport (e.g. HTTP), is called a **Two-Way/Sync** MEP.

A channel-bound MEP has an **Initiating MSH**, or **Initiator**, which is the one that triggers the execution of the MEP. The other MSH is called the **Responding MSH**, or **Responder**. These MSH roles do not change for the duration of the MEP, regardless of the number of messages exchanged and of their direction. Due to endpoint addressing or availability restrictions, some MSHs may be required to act only as initiator, and never as responder.

On the wire, the only method by which messages from the same MEP instance are associated, is through a referencing link (RefToMessageId). This referencing is decided above the MSH layer (by the Producer entity). A receiving MSH relies on both this referencing and the interpretation of the P-Mode for associating a message with a specific MEP and for validating this association.

2.2.4. Relationship to SOAP MEPs

In theory, the transport-channel-bindings previously defined could be expressed in terms of SOAP MEPs instead of channels of the underlying transport protocol. However, the notion of SOAP MEP has only been introduced with SOAP 1.2, and would need to be extended to SOAP 1.1.

Also, only the SOAP Request-Response MEP and Response MEP have been formally defined, as of the time this specification was written. A SOAP One-way MEP could also be defined, but how such an MEP may or may not bind to a two-way underlying protocol is yet to be determined.

Expressing the transport-channel-binding in terms of SOAP MEPs is only helpful if there is a published, non-ambiguous, standard way for these to map to the underlying protocol(s). This is currently only the case for some SOAP MEPs and some transport protocols. Consequently, this specification has chosen to express its transport-channel-bindings directly in terms of how to use the channels of the transport protocol, abstracting such a transport as either "One-Way" or "Two-Way".

2.2.5. The One-Way/Push MEP

This transport-channel-bound MEP involves the transfer of a single ebMS User Message unit (label: "oneway").

To conform to this MEP, the ebMS User Message unit that is exchanged MUST NOT relate to any other User Message unit (no eb:RefToMessageId element). Figure 2 illustrates the exchange pattern and MSH operations involved in this MEP.

> In case the One-Way/Push MEP is performed over a Two-way underlying transport protocol, the response message MAY carry an ebMS Signal Message, such as an error message, or other SOAP headers. Such an option is controlled by the P-Mode (see Section 4). However, the response message MUST NOT carry an ebMS User Message that refers to the request message. If the P-Mode allows Faults to be reported on the Two-way protocol's back-channel, the MEP can be qualified as a **robust** MEP, but is still an ebMS One-Way/Push MEP.

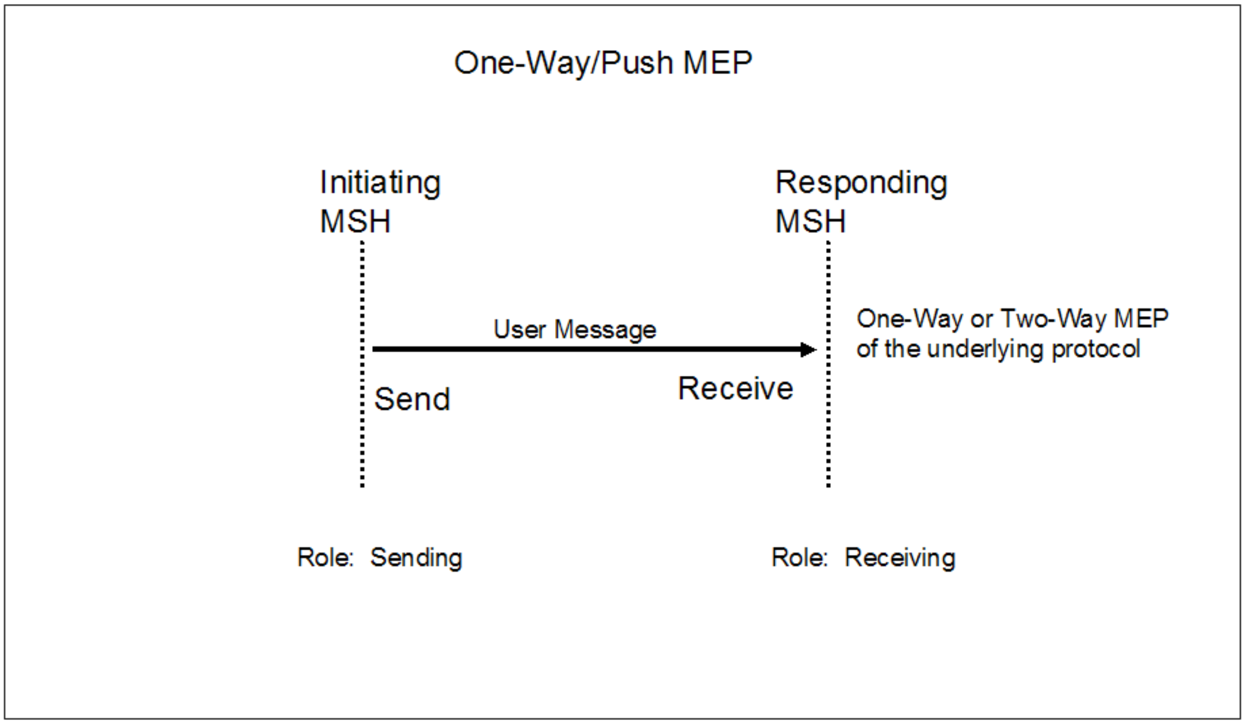

Figure 2: One-Way/Push MEP

2.2.6. The One-Way/Pull MEP

This transport-channel-bound MEP involves the transfer of a single ebMS User Message unit (label: "oneway"). This MEP is initiated by the Receiving MSH, over a two-way underlying transport protocol. The first leg of the protocol exchange carries a Pull Signal message. The second leg returns the pulled User Message unit. To conform to this MEP the pulled User Message unit MUST NOT include an eb:RefToMessageId element. In case no message is available for pulling, an ebMS error signal of severity level "warning" and short description of "EmptyMessagePartitionChannel", as listed in Section 6.7.1, MUST be returned over the response leg. Figure 3 illustrates this MEP.

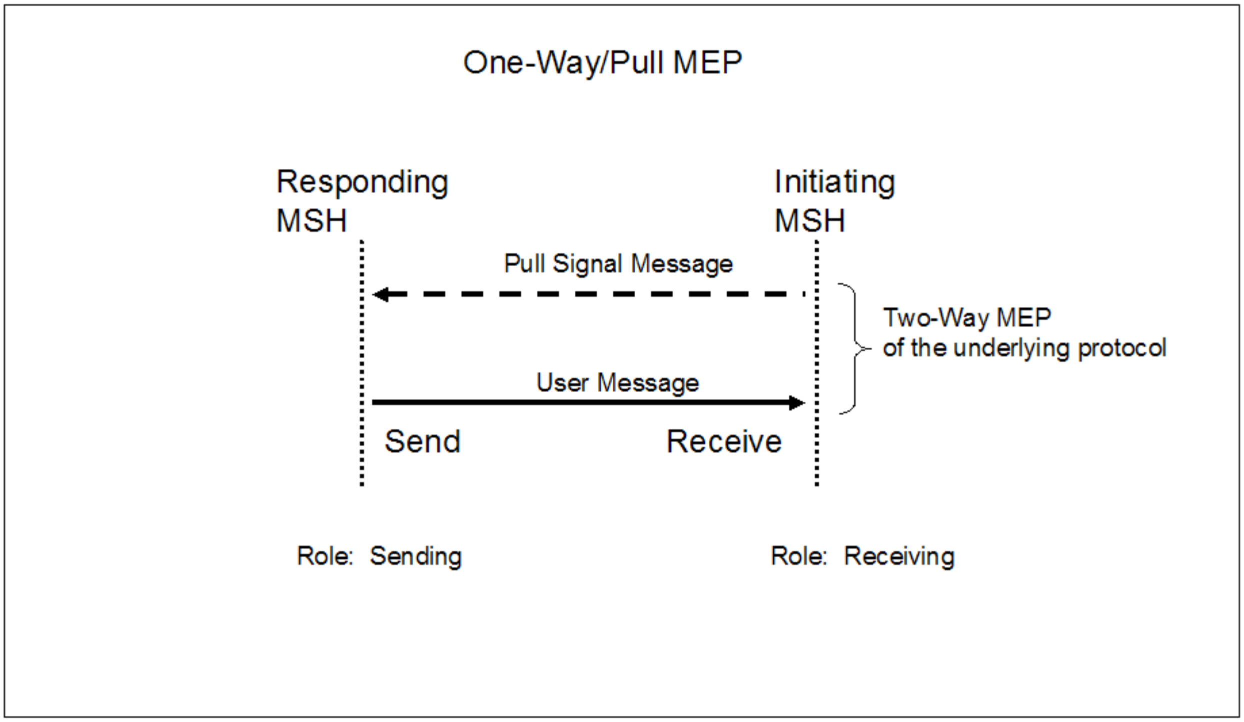

Figure 3: One-Way/Pull MEP

2.2.7. The Two-Way/Sync MEP

This transport-channel-bound MEP transfers two ebMS User Message units over a single Request-Response exchange of the underlying 2-way transport protocol. The Initiator MSH sends the first User Message called the "request" . In the second leg of the MEP, a related User Message unit called the "reply" is sent back. To conform to this MEP, the request MUST NOT relate to any other User Message unit (no eb:RefToMessageId element), and the reply MUST refer to the request via the eb:RefToMessageId header element, as described in Section 5.2.2.1. Figure 4 illustrates this MEP.

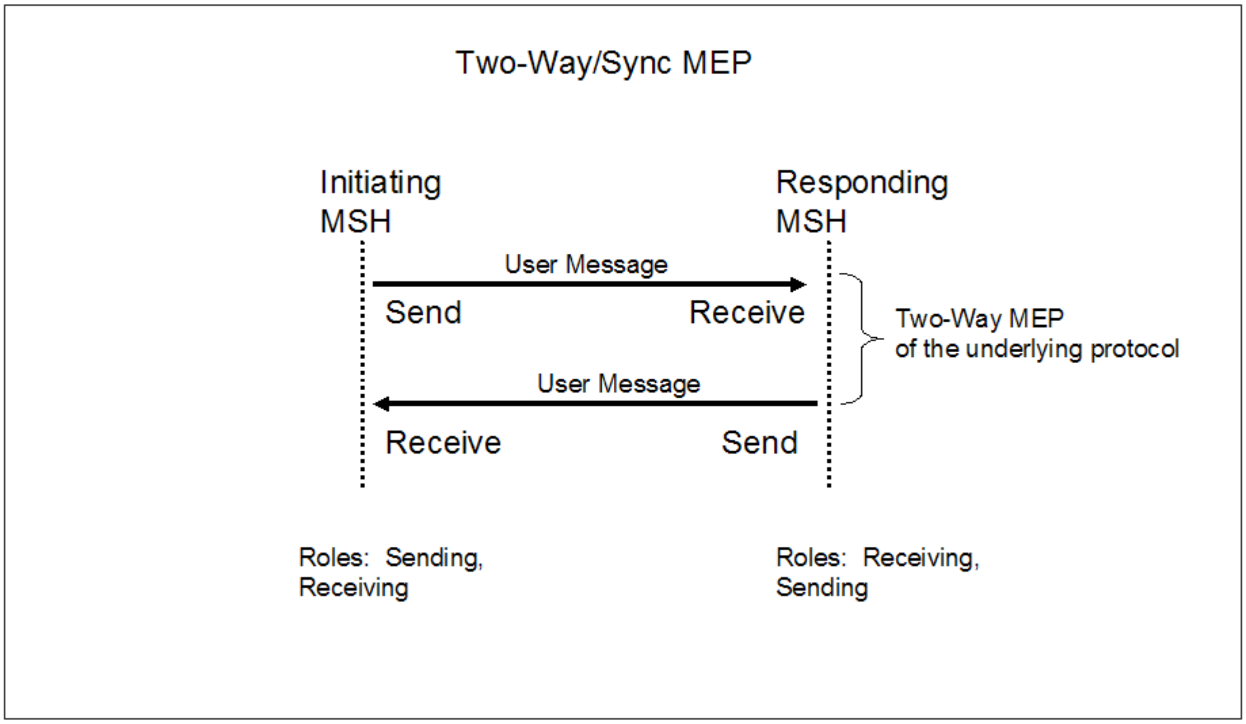

Figure 4: Two-Way/Sync MEP

2.2.8. Other Transport-Channel-Bound MEPs

So far, message exchanges conforming to either one of the three previous transport-channel-bound MEPs were only using one instance of an underlying transport protocol MEP for their binding. The following channel-bound ebMS MEPs are expected to be among the most common ebMS MEPs that use more than one underlying transport protocol MEP instance between the Initiating MSH and the Responding MSH. In this sense, they may be qualified as asynchronous.

- The **Two-Way/Push-and-Push MEP** composes the choreographies of two One-Way/Push MEPs in opposite directions, the User Message unit of the second referring to the User Message unit of the first via eb:RefToMessageId. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPush`.
- The **Two-Way/Push-and-Pull MEP** composes the choreography of a One-Way/Push MEP followed by the choreography of a One-Way/Pull MEP, both initiated from the same MSH (Initiator). The User Message unit in the "pulled" message must refer to the previously "pushed" User Message unit. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull`.
- The **Two-Way/Pull-and-Push MEP** composes the choreography of a One-Way/Pull MEP followed by the choreography of a One-Way/Push MEP, with both MEPs initiated from the same MSH. The User Message unit in the "pushed" message must refer to the previously "pulled" User Message unit. This MEP is identified by the URI `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pullAndPush`.

In all the above MEPs, the messages labels are respectively "request" and "reply". The two last MEPs support asynchronous exchanges where one party is constrained in terms of addressing or connectivity capability.

3. Message Pulling and Partitioning

3.1. 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 shifts sides. For these reasons, a receiver may want (a) to retain control over the transfer procedure of the underlying protocol by initiating transfers, and/or (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 Channels (MPCs)

Message Pulling is defined in an abstract way by the One-Way/Pull ebMS MEP (see Section 2.2.6). This MEP allows an MSH to initiate the transfer of a message as a receiver. When used in combination with the One-Way/Push ebMS MEP, it allows an MSH full control over initiating asynchronous transfers with another MSH in both directions, engaging in a client-server type of interaction with the remote MSH, without any need to open a TCP/IP 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 its 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 requests. 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 Channels (see Section 3.4) allow for partitioning the flow of messages from an MSH to another MSH into separate flows, so that each one of these flows can be controlled independently by either MSH, in terms of transfer priorities. A Sending MSH MUST be able to determine whether a submitted message should be pulled or pushed, and to which Message Partition Channel (MPC) it must be assigned. Similarly, the Receiving MSH is aware of which MPC(s) should be pulled from, and which ones will be used for push. This knowledge is based on an agreement shared between parties prior to the exchanges, and modeled in this specification as the P-Mode operation set (see Section 4).

3.2. Supporting Message Pulling

Using Message pulling requires the ability of an MSH to support the One-Way/Pull MEP. The PullRequest signal that initiates this MEP is described in Section 5.2.3.1. Because there is always at least one MPC open between a Sending MSH and a Receiving MSH–the default MPC–the Pull mode can be supported regardless of the ability to support several MPCs.

When sending a PullRequest signal, the name of the MPC to pull messages from must be specified (in eb:PullRequest/@mpc attribute), unless the default value is to be assumed.

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 for the Consumer on the Initiator side. The message is associated with an MPC. If no MPC name is provided by the submitter, or if the MSH implementation has not been provided with a way to determine this association by itself, the default MPC is used. The MEP associated with this message (e.g. as specified by P-Mode.MEP; see Section 4.2) is a One-Way/Pull.

On Initiating MSH side:

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

On Responding 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 MPC for sending, a warning signal with short description: "EmptyMessagePartitionChannel" (see Section 6.7.1) MUST be sent back instead.

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

On Initiating MSH side:

5. Receive: the pulled message is available for processing by the MSH. The header @mpc attribute indicates from which MPC it has been pulled, and is the same as the value of @mpc in the corresponding 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:

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1">
       <eb:SignalMessage>
              <eb:MessageInfo>
                     <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
                     <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
              </eb:MessageInfo>
              <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
       </eb:SignalMessage>
</eb:Messaging>
</S11:Header>
<S11:Body/>
</S11:Envelope>
```

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

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
   <eb:UserMessage mpc="http://sender.example.com/mpc123">
      <eb:MessageInfo>
         <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
         <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
         <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:PartyInfo>
         ...
      </eb:PartyInfo>
      <eb:CollaborationInfo>
         ...
      </eb:CollaborationInfo>
      <eb:PayloadInfo>
         ...
      </eb:PayloadInfo>
   </eb:UserMessage>
</eb:Messaging>
</S11:Header>
<S11:Body>
...
</S11:Body>
</S11:Envelope>
```

3.3. Combining Pulling with Security and Reliability

Reliability of a pulled message is usually associated with the reliability of the corresponding PullRequest signal. The reliability of the One-Way/Pull MEP instance is addressed in Section 8.3.

Security for the PullRequest signal is described in details in Section 7.11.

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

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
```

```
        <eb:SignalMessage>
                <eb:MessageInfo>
                        <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
                        <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
                </eb:MessageInfo>
                <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
        </eb:SignalMessage>
</eb:Messaging>
<wss:Security>
        ...
</wss:Security>
<wsr:Request S11:mustUnderstand="1">
        ...
        <ReplyPattern>
                <Value>Response</Value>
        </ReplyPattern>
        <AckRequested/>
        ...
</wsr:Request>
</S11:Header>
<S11:Body/>
</S11:Envelope>
```

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

```
<S11:Envelope>
<S11:Header>
<eb:Messaging S11:mustUnderstand="1" >
   <eb:UserMessage mpc="http://sender.example.com/mpc123">
      <eb:MessageInfo>
         <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
         <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
         <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
      </eb:MessageInfo>
      <eb:PartyInfo>
      ...
      </eb:PartyInfo>
      <eb:CollaborationInfo>
      ...
      </eb:CollaborationInfo>
      <eb:PayloadInfo>
      ...
      </eb:PayloadInfo>
   </eb:UserMessage>
</eb:Messaging>
<wsr:Response S11:mustUnderstand="1">
        ...
</wsr:Response>
<wss:Security>
        ...
</wss:Security>
</S11:Header>
<S11:Body>
...
</S11:Body>
</S11: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. A wsr:Request header could be present, in addition to wsr:Response, in case some specific reliability requirement is associated with the pulled message (see Section 8.3).

3.4. Message Partition Channels

3.4.1. Concept and Purpose

Message Partition Channels (MPCs) 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, MPCs 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 MPC 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 MPC will be used.

   Notes:

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

   Although MPCs are most obviously beneficial to message pulling operations, MPCs may be used in association with pushed messages as well. The benefits of doing so, listed above, apply to the push case as well.

Example*: A pair of business partners – a large buyer and a small supplier - have decided to create two MPCs for transferring messages sent by the buyer. Urgent messages that require immediate processing (e.g. high priority Purchase Orders, and updates to prior Purchase Orders) are assigned to one MPC; and less urgent messages (payments, catalog requests, confirmations, acknowledgments of receipts, etc.)are assigned to the other MPC. The buyer determines the level of urgency of a posting, which may or 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" MPC; then, once that is exhausted, only the messages from the less urgent MPC. This way, the low-capacity Receiving MSH (supplier) is able to prioritize the reception of its messages, 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. Such filtering could be implemented in a Sending MSH and/or in a Receiving MSH as a complement to, or instead of, different MPCs. The notion of MPC 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 MPC, without a need to communicate and process filter expressions for each message transfer.

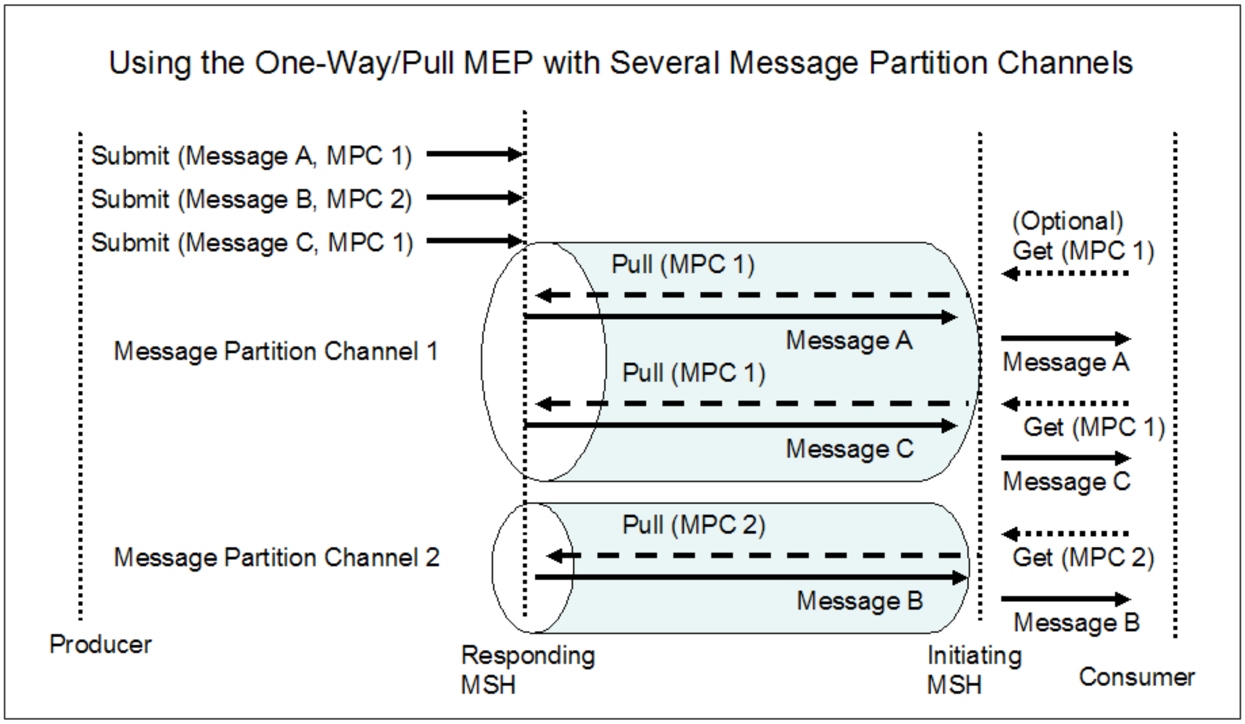
Using the One-Way/Pull MEP with Several Message Partition Channels
Submit (Message A, MPC 1)
Submit (Message B, MPC 2)
Submit (Message C, MPC 1)
Message Partition Channel 1
Message Partition Channel 2
Producer
Pull (MPC 1)
Message A
Pull (MPC 1)
Message C
Pull (MPC 2)
Message B
Responding MSH
Initiating MSH
(Optional) Get (MPC 1)
Message A
Get (MPC 1)
Message C
Get (MPC 2)
Message B
Consumer

Figure 5: One-Way/Pull with Message Partition Channels

Figure 5 illustrates how MPCs 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. Messages on MPC 1 are "pulled" in priority by the Consumer side.

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

- The MEPs in which these messages participate. Messages over the same MPC 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). MPCs do not have any routing or addressing capability.

Before it is transferred from a Sending MSH to a Receiving MSH, regardless of whether it is pushed or pulled, a message is always assigned to an MPC. If no explicit assignment is requested (e.g. by the message Producer at Submit time or per configuration of the MSH), the default MPC name "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultMPC" is assigned.

3.4.2. Some Use Cases

Figure 6 illustrates various cases in using MPCs.

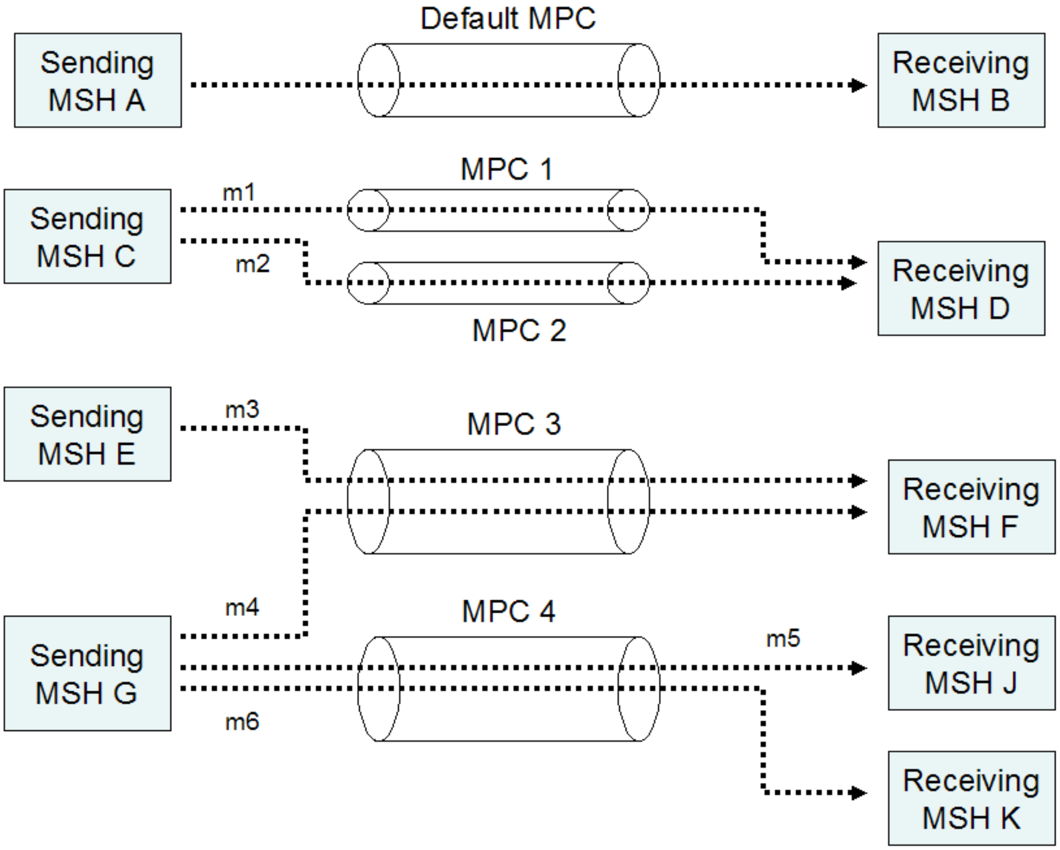

Figure 6: Message Partition Channel Use Cases

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 MPC has been explicitly defined or assigned. All messages transferred from A to B – whether pushed or pulled – will implicitly use the default MPC.

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

MPC 3 is shared by two Sending MSHs, E and G. The effect of using this MPC 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 from which to pull first.

MPC 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 the 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, for example, two agencies equally qualified to process trouble tickets, indiscriminately pulling messages from the same MPC at the pace allowed by their processing capacity. MPC 4 may also be used by concurrent, pushed message flows. Using the same MPC 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).

3.4.3. Definition and Usage Requirements

An MPC 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 MPC this message is assigned to. MPC 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 MPC name, and assuming there is no other reason preventing them from transferring messages over this MPC (e.g. security).

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

> Note:
> As defined above, an MPC 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 MPC name (e.g. MPC 3 in the figure could also be renamed MPC 2). Formally speaking, all these messages would be transferred over the same MPC. There might be some business significance in deciding whether two pairs of MSHs that have unconnected message flows should use the same MPC 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 MPCs, not signal messages.

A PullRequest signal message always indicates in its header (see Section 5.2.3.1) the MPC on which the message must be pulled. If no MPC is explicitly identified, the default MPC MUST be pulled from. The pulled message sent in response MUST have been assigned to the indicated MPC.

The association of a message with an MPC 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 4). This is left to the implementation.

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

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

There is no specific quality of service associated with an MPC. Security and reliability remain associated with parties or with MSHs, in a way that is orthogonal to MPCs; although an implementation is free to associate QoS with MPCs as long as this conforms to an agreement between parties.

4. Processing Modes

An MSH is operating–either for sending or receiving messages–with knowledge of some contextual information that controls the way messages are processed. This 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 types of processing, an MSH generally supports several P-Modes.

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 or among parties. 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 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 set of all P-Modes that are supported by an MSH during operation, is called the P-Mode operation set of the MSH.

The association of a P-Mode with a message may be based on various criteria, usually dependent on header data (e.g. Service/Action, Conversation ID, or other message properties). Which security and/or which reliability protocol and parameters, as well as which MEP is being used when sending a message, is determined by the P-Mode associated with this message.

A data model for P-Modes is described in Appendix D. Although this specification does not require support for any particular representation of a P-Mode, a conformance profile for this specification may require support for a particular representation. An MSH MUST conform the processing of its messages to the values in the P-Mode associated with this message. The details of which P-Mode parameters must be supported by an implementation, is governed by the features associated with the conformance profile claimed by this implementation, i.e. by its profile feature set (see Appendix G on Conformance). An MSH MUST NOT process a message to normal completion if it has no matching P-Mode in its P-Mode operation set: i.e. the MSH MUST NOT deliver such a message when in Receiving role, or MUST NOT send it when in Sending role. When it cannot match a message to a P-Mode, an MSH MUST generate a ProcessingModeMismatch (EBMS:0010) error.

> Note:
> It is important to distinguish between Conformance Profiles (Appendix G) and P-Modes. A conformance profile qualifies an MSH implementation and does not vary with the usage made of the MSH. A P-Mode qualifies the dynamic exchange and processing of messages, and is generally user defined. It must be within the capabilities allowed by the conformance profile claimed by the MSH on which it is deployed.

4.1. Messaging Service Processing Model

Although different P-Modes may apply from one message to the other, the overall processing model remains the same for all messages. The P-Modes set may be seen as configuring the execution parameters for the general 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 7 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.

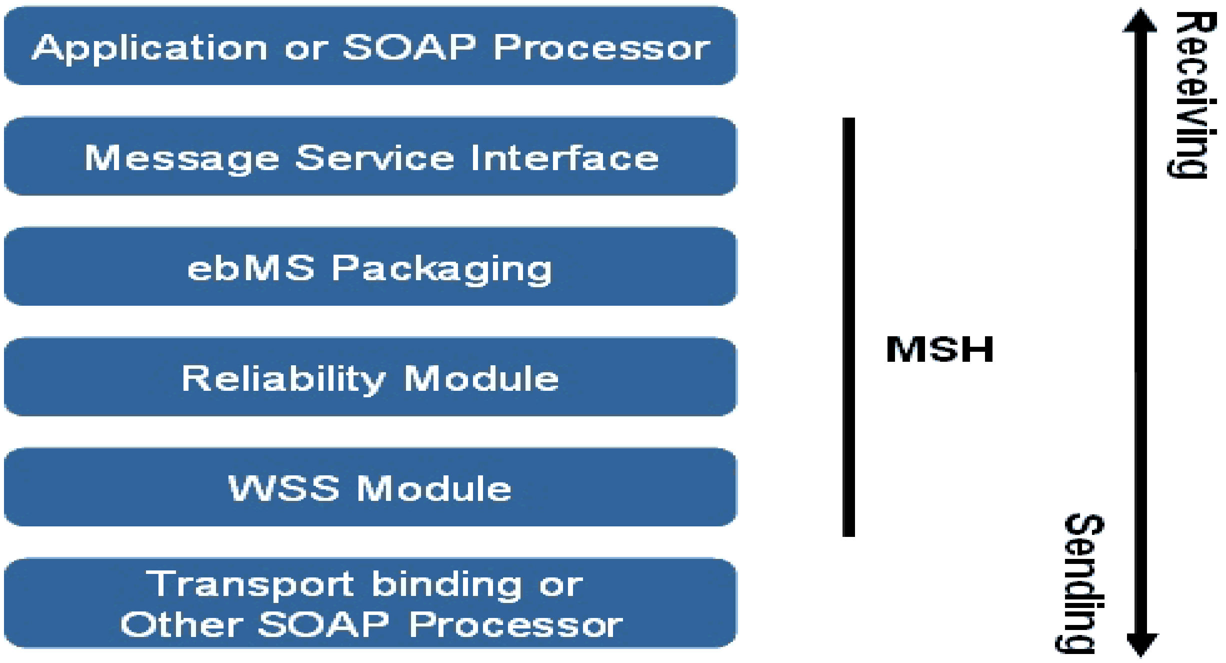
Application or SOAP Processor
Message Service Interface
ebMS Packaging
Reliability Module
WSS Module
Transport binding or Other SOAP Processor
MSH
Receiving
Sending

Figure 7: 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, although the processing order shown here is RECOMMENDED, especially in regard to Security and Reliability Modules.

- **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 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 Protocol Bindings** - These are the actual transport protocol bindings. This specification defines bindings for HTTP and SMTP in Appendix C, and supports the addition of other protocols.

4.2. Processing Mode Features

The P-Mode is partitioned into functional groups called P-Mode features. Each P-Mode feature covers one of the 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 Channels (MPCs).

The 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 all messages between two MSHs similarly. 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 them, 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 them, 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.BusinessInfo**: includes all message-relevant data related to a collaboration between two parties. It also indicates which MPCs are to be used by these parties. This feature will complement or validate message data that is expected to be provided by the application on a per-message basis for these header elements:
  - eb:UserMessage/eb:PartyInfo
  - eb:UserMessage/eb:CollaborationInfo
  - eb:UserMessage/eb:MessageProperties
- **P-Mode.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. It also may include the policy chosen for raising ebMS Errors from the errors generated by functional modules (Reliability, Security). This P-Mode feature must define reporting mode parameters that will allow a Receiving MSH to decide:
  - whether an error generated on reception of a message must be returned as response over the same SOAP MEP. (e.g. errorHandling.report.asResponse = true/false).
  - whether an error generated on reception of a message must be returned to sender or to a third party over a new SOAP MEP. (e.g. errorHandling.report.ReceiverErrorsTo = <URL>).
  - whether the Consumer and/or Producer (e.g. errorHandling.Report.ProcessErrorNotifyConsumer) of a message must be notified of an error generated on reception of the message.

In this specification, a P-Mode feature is abstractly considered to apply to both sending and receiving roles, although implementations may choose to represent only the subset relevant to the role in which they operate. A single P-Mode instance is also intended to govern all messages involved in an ebMS MEP. (The ebMS MEP and its transport channel binding are attributes of a P-Mode.) Because messages involved in an MEP (e.g. request and reply) may use different qualities of service, a single P-Mode may use different vectors of values for its parameters, depending on the message in the MEP. An outline of the data model for P-Modes is given in Appendix D.

Agreeing on a P-Mode operation 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 5.2.2.7).

4.3. Default Features for Processing Mode

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

- **Default P-Mode.MEP**: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay`
- **Default P-Mode.MEPbinding**: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push`
- **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.BusinessInfo**: In the absence of any application input at message level as well as for this P-Mode feature, the following default header element values will be used (shown here for a message sent by an Initiator to a Responder party). Any of these may be overridden by application input.
  - eb:UserMessage/eb:PartyInfo: The eb:From element contains a PartyId with value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultFrom`
The eb:To element contains a PartyId with value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultTo`
  - eb:UserMessage/eb:CollaborationInfo: Contains no eb:AgreementRef. The eb:Service element has the value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service`
The eb:Action element has the value: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test`
(Section 5.2.2 details the semantics of these values.)
The eb:ConversationId element has the value: 1.
The default MPC is in use.
  - eb:UserMessage/eb:MessageProperties: This element is absent.
  - eb:UserMessage/eb:PayloadInfo: This element is absent.
- **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, such as notification to local Consumer or Producer.

In the absence of a user-agreed P-Mode feature, it is RECOMMENDED that an MSH operate 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 Appendix G) that are not compatible with the default P-Mode feature. For example, such an incompatibility would occur for the default P-Mode.MEP with a conformance profile that only requires the 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.

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 1.6. 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 both multipart and 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 itself comprises two sections: (a) the SOAP Body element within the SOAP Envelope, and in case of MIME packaging, (b) zero or more additional MIME parts containing additional application-level payloads. The SOAP Body and MIME parts are also referred to as ebMS Payload Containers. The SOAP Body is the only payload container that requires XML-structured content, though non-XML content may be included within an appropriately typed (binary or otherwise) element inside the Body.

The general structure and composition of an ebMS User Message is described in Figure 8, and a Signal Message in Figure 9.

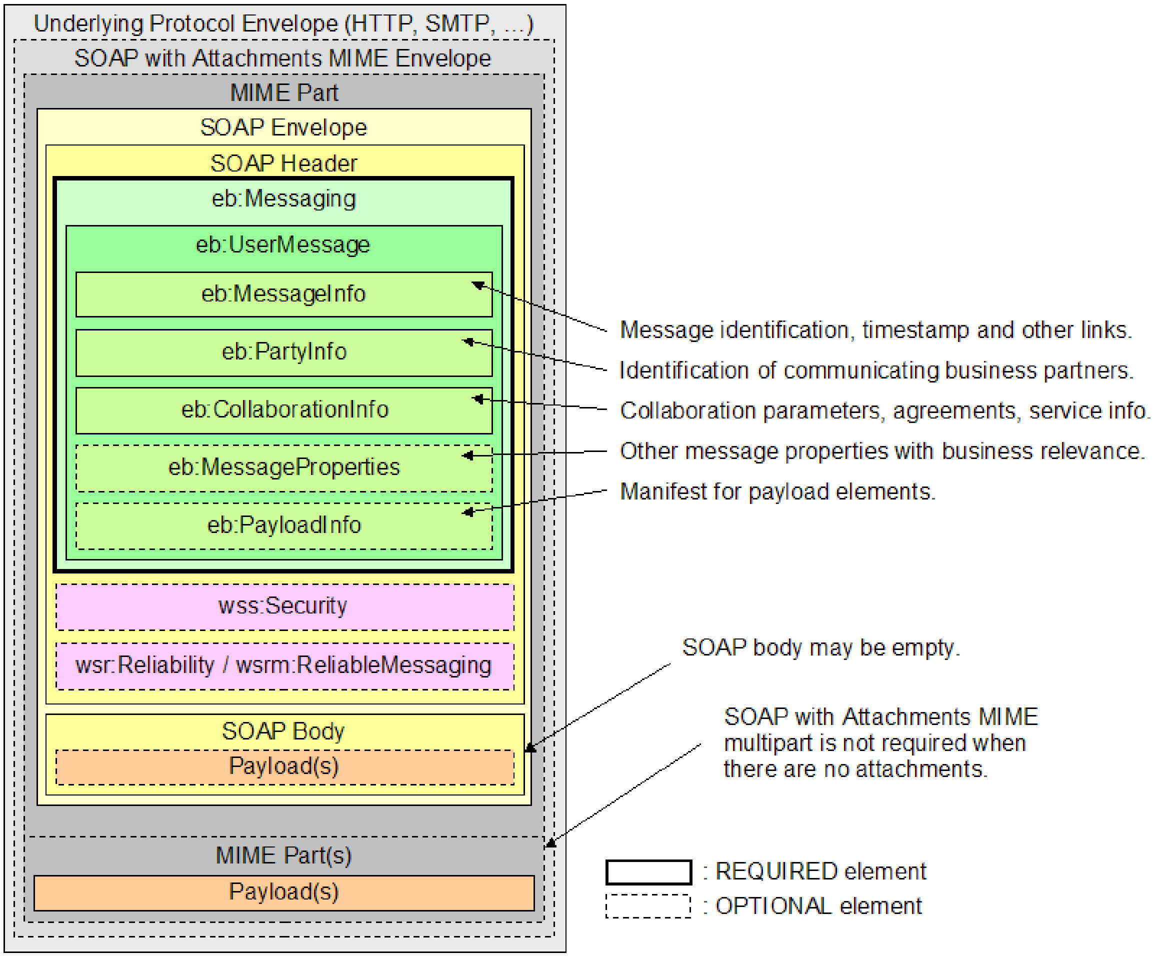
Underlying Protocol Envelope (HTTP, SMTP, ...)
SOAP with Attachments MIME Envelope
MIME Part
SOAP Envelope
SOAP Header
eb:Messaging
eb:UserMessage
eb:MessageInfo
eb:PartyInfo
eb:CollaborationInfo
eb:MessageProperties
eb:PayloadInfo
wss:Security
wsr:Reliability / wsrm:ReliableMessaging
SOAP Body
Payload(s)
MIME Part(s)
Payload(s)
Message identification, timestamp and other links.
Identification of communicating business partners.
Collaboration parameters, agreements, service info.
Other message properties with business relevance.
Manifest for payload elements.
SOAP body may be empty.
SOAP with Attachments MIME multipart is not required when there are no attachments.
: REQUIRED element
: OPTIONAL element

Figure 8: User Message Structure

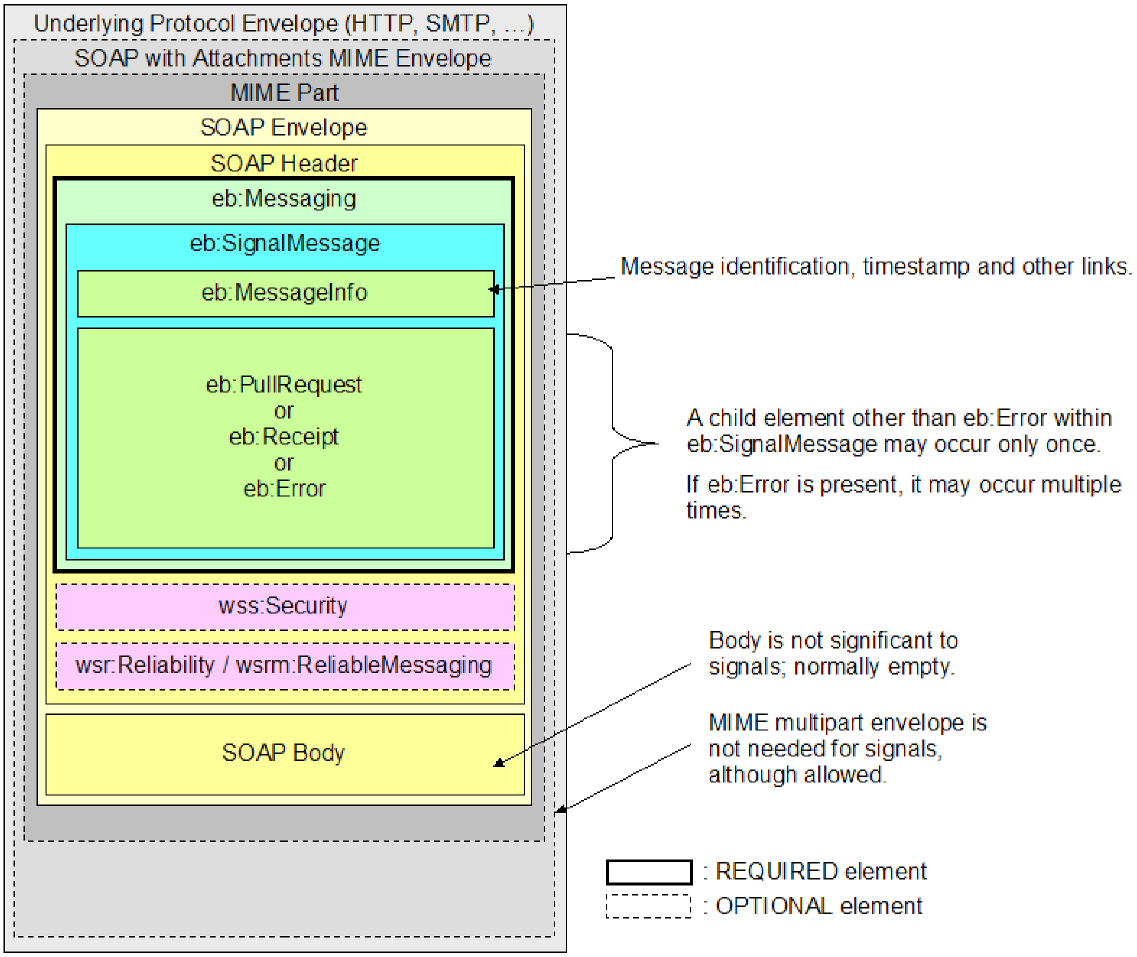
Underlying Protocol Envelope (HTTP, SMTP, ...)
SOAP with Attachments MIME Envelope
MIME Part
SOAP Envelope
SOAP Header
eb:Messaging
eb:SignalMessage
eb:MessageInfo
eb:PullRequest
or
eb:Receipt
or
eb:Error
wss:Security
wsr:Reliability / wsrm:ReliableMessaging
SOAP Body
Message identification, timestamp and other links.
A child element other than eb:Error within eb:SignalMessage may occur only once.
If eb:Error is present, it may occur multiple times.
Body is not significant to signals; normally empty.
MIME multipart envelope is not needed for signals, although allowed.
: REQUIRED element
: OPTIONAL element

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, it is possible for a Sending MSH implementation to generate an ebMS message from a well-formed input SOAP message simply by adding an eb:Messaging header; likewise, some Receiving MSH implementation could deliver a well-formed SOAP message as output by removing (and processing) the eb:Messaging header.

All MIME headers of the Message Package MUST conform with the SOAP Messages with Attachments [SOAPATTACH] W3C Note. In addition, the Content-Type MIME header of the Message Package MUST contain a type parameter whose value matches the MIME media type of the MIME body part containing the SOAP Envelope document. In accordance with the [SOAP11] specification, the MIME media type of the SOAP 1.1 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]) 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 (the SOAP Envelope).

5.1.2. MIME and XML Considerations

This section contains further MIME- and XML-specific packaging requirements and guidance.

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 parameter are present, the XML prolog for the SOAP Message SHALL contain the encoding declaration, and 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 that 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. A 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://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
```

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:

```
<S11:Envelope xmlns:S11="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/">
       <S11:Header/>
       <S11:Body/>
</S11:Envelope>
```

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

The schemaLocation for the namespace described in Section 5.1.3.1 is:

```
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd
```

Separate schemaLocation attributes are RECOMMENDED. For example:

```
<S11:Envelope xmlns:S11="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/">
  <S11:Header>
    <eb:Messaging xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/"
      xsi:schemaLocation=
       "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-
3_0-200704.xsd">
      <eb:UserMessage>
        <eb:MessageInfo >...</eb:MessageInfo>
             ...
        <eb:PayloadInfo >...</eb:PayloadInfo>
             ...
      </eb:UserMessage>
    </eb:Messaging>
  </S11:Header>
  <S11:Body>
       ...
  </S11:Body>
</S11:Envelope>
```

5.1.3.3. SOAP Header Element

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

5.1.3.4. SOAP Body Element

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

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

5.1.3.5. ebXML SOAP Extensions

An ebMS Message extends the SOAP Message with the extension element eb:Messaging, where "eb" is the namespace prefix 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. These are not qualified under the ebMS namespace.

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 [SOAP12] body. The Content-Type header MAY contain a "charset" parameter. For example:

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

The MIME charset parameter 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 value of the MIME charset parameter SHALL be equivalent to the encoding declaration of the SOAP Message. If provided, the MIME charset parameter 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"

<S11:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
   <S11:Header>
      <eb:Messaging>
      …
      </eb:Messaging>
   </S11:Header>
   <S11:Body>
   …
   </S11:Body>
</S11: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.

There SHOULD also be no additional MIME attachments that are not Payload Containers (i.e., that are not referenced by an eb:PayloadInfo element, as described in Section 5.2.2.12); but if any such attachments are present, they are outside the scope of MSH processing. An MSH MUST NOT process application data that is not referenced by eb:PayloadInfo.

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. 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:

```
<S11:Envelope xmlns:S11="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/">
   <S11:Header
xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
      <eb:Messaging S11:mustUnderstand="1">
         <eb:UserMessage>
            ...
            <eb:PayloadInfo>
               ...
            </eb:PayloadInfo>
            ...
         </eb:UserMessage>
      </eb:Messaging>
   </S11:Header>
   <S11:Body>
      ...
   </S11:Body>
</S11: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.

In the case of a User message, the ebXML header block contains an eb:UserMessage child element:

```
<eb:Messaging>
       <eb:UserMessage>
             <eb:MessageInfo>
             <!-- some headers here like Timestamp and MessageId -->
             </eb:MessageInfo>
             <!-- header elements of the ebMS user message -->
       </eb:UserMessage>
</eb:Messaging>
```

In the case of a Signal message, the ebXML header block (eb:Messaging) contains at least one eb:SignalMessage child element:

```
<eb:Messaging>
       <eb:SignalMessage>
             <eb:MessageInfo>
                 <!-- some headers here like Timestamp and MessageId -->
            </eb:MessageInfo>
             <eb:signalname>
                  <!-- header elements of this ebMS signal message -->
             </eb:signalname>
       </eb:SignalMessage>
</eb:Messaging>
```

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

5.2.1. eb:Messaging Element Specification

The eb:Messaging element has the following attributes:

- `eb:Messaging/@S11: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 type of Signal message. It contains all header information for the Signal message. If this element is not present, an element describing a User message MUST be present. Three types of Signal messages are specified in this document: Pull signal (eb:PullRequest), Error signal (eb:Error) and Receipt signal (eb:Receipt).

Both eb:UserMessage element and eb:SignalMessage element MAY be present within the eb:Messaging element.

Example ebMS Message Header:

```
<!-- (contained within S11:Header) -->

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

    <eb:UserMessage>

      <eb:MessageInfo>
        <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
        <eb:MessageId>UUID-2@example.com</eb:MessageId>
        <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
      </eb:MessageInfo>

      <eb:PartyInfo>
        <eb:From>
          <eb:PartyId>uri:example.com</eb:PartyId>
            <eb:Role>http://example.org/roles/Buyer</eb:Role>
        </eb:From>

        <eb:To>
          <eb:PartyId type="someType">QRS543</eb:PartyId>
          <eb:Role>http://example.org/roles/Seller</eb:Role>
        </eb:To>
      </eb:PartyInfo>

      <eb:CollaborationInfo>
        <eb:AgreementRef>http://registry.example.com/cpa/123456
        </eb:AgreementRef>
        <eb:Service type="MyServiceTypes">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@example.com">
          <eb:Schema location="http://example.org/bar.xsd" 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>

```

5.2.2. eb:Messaging/eb:UserMessage

This element has the following attributes:

- `eb:Messaging/eb:UserMessage/@mpc`: This OPTIONAL attribute contains a URI that identifies the Message Partition Channel to which the message is assigned. The absence of this element indicates the use of the default MPC. When the message is pulled, the value of this attribute MUST indicate the MPC 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.

5.2.2.1. eb:Messaging/eb:UserMessage/eb:MessageInfo

This element has the following children 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 C.3).

5.2.2.2. 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.

5.2.2.3. 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. If it occurs multiple times, each instance MUST identify the same organization.
- eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role: The REQUIRED eb:Role element occurs once, and 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, with a default value of http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultRole. Other possible values are subject to partner agreement.

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

```
<eb:From>
  <eb:PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">
123456789</eb:PartyId>
  <eb:PartyId type="SCAC">RDWY</eb:PartyId>
  <eb:Role>http://example.org/roles/Buyer</eb:Role>
</eb:From>
```

5.2.2.4. 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, @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 , EDIFACT or ANSI ASC X12 registries. Technical specifications for the first two registries can be found at and [ISO6523] and [ISO9735], respectively. Further discussion of PartyId types and methods of construction can be found in an appendix of [ebCPPA21]. The value of any given @type attribute MUST be unique within a list of PartyId elements.

An example of PartyId element is:

```
<eb:PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">
123456789</eb:PartyId>
```

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

5.2.2.5. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To

This element has the same children elements as eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From, above in Section 5.2.2.3.

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>
```

5.2.2.6. 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.

5.2.2.7. 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 should 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 containing 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://registry.example.com/cpas/our_cpa.xml</eb:AgreementRef>
```

If a CPA is referred to and a Receiving MSH detects an inconsistency, then it MUST report it with an "`ValueInconsistent`" error of severity "`error`". If the `AgreementRef` is not recognized, then the Receiving MSH MUST report it as a "`ValueNotRecognized`" error of severity "`error`".

The AgreementRef element may have two attributes:

- `@type:` This OPTIONAL attribute 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; this choice is left to profiles of this specification. If the type attribute is not present, the content of the eb:AgreementRef element MUST be a URI. If it is not a URI, then the MSH MUST report a "ValueInconsistent" error of severity "error".
- `@pmode:` This OPTIONAL attribute allows for explicit association of a message with a P-Mode. When used, its value contains the PMode.ID parameter.

5.2.2.8. 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.

Examples of the Service element include:

```
<eb:Service>urn:example.org:services:SupplierOrderProcessing</eb:Service>
```

```
<eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
```

The Service element MAY contain 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 a "`ValueInconsistent`" error of severity "`error`".

When the value of the element is `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/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.

5.2.2.9. 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 "ValueNotRecognized" error of severity "error".

When the value of this element is http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test, then the eb:Service element MUST have the value http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/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.

5.2.2.10. 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.

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.

5.2.2.11. 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 two attributes:

- @name: The value of this REQUIRED attribute must be agreed upon between partners.
- @type: This OPTIONAL attribute allows for resolution of conflicts between properties with the same name, and may also help with Property grouping, e.g. various elements of an address.

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 message 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>
```

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

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 a MIME attachment, an XML element within the SOAP Body, or another resource which may be obtained by resolving a URL, according to the value of the href attribute, described below.

5.2.2.13. 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 the [RFC2392] Content-ID URI of the payload object referenced, an xml:id fragment identifier, or the URL of an externally referenced resource; for example, "cid:foo@example.com" 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. This metadata MAY be used to validate the Payload Part to which it refers, but the MSH is NOT REQUIRED to do so. The Schema element contains the following attributes:
  - (a) namespace - the OPTIONAL target namespace of the schema
  - (b) location – the REQUIRED URI of the schema
  - (c) version – an OPTIONAL version identifier of the schema.
- `eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties`
  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 REQUIRED @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 meta-data that qualifies or abstracts the payload data. 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:PartProperties>
```

```
  <eb:Property name="Description">Purchase Order for 11
widgets</eb:Property>
  <eb:Property name="MimeType">application/xml</eb:Property>
</eb:PartProperties>
```

Full PayloadInfo Example:

```
<eb:PayloadInfo>
  <eb:PartInfo href="cid:foo@example.com">
    <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
    <eb:PartProperties>
      <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
      <eb:Property name="MimeType">application/xml</eb:Property>
    </eb:PartProperties>
  </eb:PartInfo>
  <eb:PartInfo href="#goo_payload_id">
    <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
    <eb:PartProperties>
      <eb:Property name="Description">Purchase Order for 100 widgets</eb:Property>
      <eb:Property name="MimeType">application/xml</eb:Property>
    </eb:PartProperties>
  </eb:PartInfo>
</eb:PayloadInfo>
```

5.2.3. 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 defines the nature of the ebMS signal. There is only one eb:[SignalName] child element when [SignalName]=PullRequest or [SignalName]=Receipt. There may be several children elements when SignalName=Error.

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 interpretation of the signal is concerned.

5.2.3.1. eb:Messaging/eb:SignalMessage/eb:PullRequest

This element has the following attribute:

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

5.2.3.2. eb:Messaging/eb:SignalMessage/eb:Error

The eb:Error element MAY occur zero or more times. For its complete specification, refer to Section 6.

5.2.3.3. eb:Messaging/eb:SignalMessage/eb:Receipt

The eb:Receipt element MAY occur zero or one times; and, if present, SHOULD contain a single ebbpsig:NonRepudiationInformation child element, as defined in the ebBP Signal Schema [ebBP-SIG]. The value of eb:MessageInfo/eb:RefToMessageId MUST refer to the message for which this signal is a receipt.

5.2.4. Message Unit Bundling

When the eb:Messaging element contains multiple children elements, i.e. multiple Message Units (either User Message Units or Signal Message Units), this is called Message Unit bundling. The following general rules govern Message Unit bundling:

> Note:
> Other use cases for bundling may be considered in a forthcoming Part 2 of this specification, resulting in changes to these rules, potentially allowing for multiple User

Message Units or multiple Signal Message Units of the same type.

- The eb:Messaging element MUST NOT contain more than one eb:UserMessage element.
- The eb:Messaging element MAY contain multiple eb:SignalMessage elements, in addition to an optional eb:UserMessage element, but MUST NOT contain more than one Signal Message Unit of the same type.

The following is a non-exhaustive list of valid bundling cases:

(a) eb:Messaging element with the following children:

- an eb:UserMessage element
- an eb:SignalMessage element with an eb:PullRequest child

(b) eb:Messaging element with the following children:

- an eb:UserMessage element
- an eb:SignalMessage element with one or more eb:Error children

(c) eb:Messaging element with the following children:

- an eb:UserMessage element
- an eb:SignalMessage element with an eb:PullRequest child
- an eb:SignalMessage element (distinct from the previous one) with one or more eb:Error children

(d) eb:Messaging element with the following children:

- an eb:SignalMessage element with an eb:PullRequest child
- an eb:SignalMessage element (distinct from the previous one) with an eb:Receipt child

With regard to MEP transport channel bindings, the following restrictions must be observed:

- An ebMS Message containing an eb:SignalMessage/eb:PullRequest element cannot bind to the back-channel of the underlying transport protocol, regardless of its bundling context (bundling cases (a) , (c) or (d)) i.e. even if it is also a User Message. For example, such a message can neither appear as "reply" in the Sync transport channel binding, nor as a "oneway" in the Pull transport channel binding.
- An ebMS Message containing an eb:SignalMessage/eb:PullRequest element and a User Message unit (case (a) or case (c)) cannot act as a "request" in the Sync transport channel binding, as the semantics of this combination would require sending back two User Messages units over the back-channel, which is a bundling case not supported in this release.

5.3. Examples of ebMS Messages

The following listings provide examples of various types of ebMS messages: UserMessage, PullRequest Signal, Error Signal, Receipt Signal and a "bundled" message.

> Note:
> The examples are depicted using the SOAP 1.1 envelope; however, SOAP 1.2 could have been used instead, with the appropriate namespace adjustment. In that case, the contents of the eb:Messaging header would be the same, with the exception of the attribute `eb:Messaging/@S11:mustUnderstand`, which becomes `eb:Messaging/@S12:mustUnderstand`, having a boolean value of "true" instead of the integer value "1".

5.3.1. UserMessage Example

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

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>

```

```
   <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
      <eb:UserMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">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:part@example.com">
               <eb:Schema location="http://registry.example.org/po.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order for 11 Widgets</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>
   </eb:Messaging>

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

The following is an example of a possible Response to the above User Message:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>

   <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
      <eb:UserMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@responder.example.com</eb:MessageId>
            <eb:RefToMessageId>UUID-1@requester.example.com</eb:RefToMessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
            <eb:Action>PurchaseOrderResponse</eb:Action>
            <eb:ConversationId>4321</eb:ConversationId>
         </eb:CollaborationInfo>
         <eb:PayloadInfo>
            <eb:PartInfo href="cid:part@example.com">
               <eb:Schema location="http://registry.example.org/poc.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order Confirmation</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </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://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>

   <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
      <eb:SignalMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PullRequest mpc="http://msh.example.com/mpc123" />
      </eb:SignalMessage>
   </eb:Messaging>

</S11:Header>

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

5.3.3. Error Message Example

The following is an example of an Error Signal Message:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>

  <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
     <eb:SignalMessage>
        <eb:MessageInfo>
           <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
           <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:Error origin="ebMS" category="Content"
                  errorCode="EBMS:0001" severity="failure"
                  refToMessageInError="UUID-1@sender.example.com">
           <eb:Description xml:lang="en">Value not recognized</eb:Description>
        </eb:Error>
        <eb:Error origin="Security" category="Processing" errorCode="0101"
                  severity="failure" refToMessageInError="UUID-23@esender.fxample.com">
           <eb:Description xml:lang="en">Failed Authentication</eb:Description>
        </eb:Error>
     </eb:SignalMessage>

  </eb:Messaging>
```

```

</S11:Header>

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

5.3.4. Receipt Message Example

The following is an example a Receipt Signal Message, as described in Section 5.2.3.3:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>
  <eb:Messaging xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0">

    <eb:SignalMessage>
      <eb:MessageInfo>
        <eb:Timestamp>2006-07-01T13:42:37.429Z</eb:Timestamp>
        <eb:MessageId>uiwtoruiopwr2543890@b.example.com</eb:MessageId>
        <eb:RefToMessageId>uiopfdsmnf4898965563434@a.example.com</eb:RefToMessageId>
      </eb:MessageInfo>

      <eb:Receipt>
        <ebbpsig:NonRepudiationInformation>
          <ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
            <ds:Reference URI="http://b.example.com/doc45/#b">
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>fX/iNylcUHNLV4lCE0eC7aEGP28=</ds:DigestValue>
            </ds:Reference>
          </ebbpsig:MessagePartNRInformation>
          <ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
            <ds:Reference URI="http:/a.example.com/doc23/#a">
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
              <ds:DigestValue>fX/iNylcUHNLV4lCE0eC7aEGP28=</ds:DigestValue>
            </ds:Reference>
          </ebbpsig:MessagePartNRInformation>
        </ebbpsig:NonRepudiationInformation>
      </eb:Receipt>

    </eb:SignalMessage>

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

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

5.3.5. "Bundled" Message Example

The following is an example a User Message unit bundled with both PullRequest and Error Signal Message units, as described in Section 5.2.4:

```
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
           xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
<S11:Header>

  <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">

     <eb:SignalMessage>
        <eb:MessageInfo>
           <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
           <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:Error origin="ebMS" category="Content"
                  errorCode="EBMS:0001" severity="failure"
                  refToMessageInError="UUID-1@sender.example.com">
           <eb:Description xml:lang="en">Value not recognized</eb:Description>
        </eb:Error>
        <eb:Error origin="Security" category="Processing" errorCode="0101"
                  severity="failure" refToMessageInError="UUID-23@esender.fxample.com">
           <eb:Description xml:lang="en">Failed Authentication</eb:Description>
        </eb:Error>
     </eb:SignalMessage>

      <eb:SignalMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PullRequest mpc="http://msh.example.com/mpc123" />
      </eb:SignalMessage>

      <eb:UserMessage>
         <eb:MessageInfo>
            <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
            <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>
         </eb:MessageInfo>
         <eb:PartyInfo>
            <eb:From>
               <eb:PartyId>uri:requester.example.com</eb:PartyId>
               <eb:Role>http://example.org/roles/Buyer</eb:Role>
            </eb:From>
            <eb:To>
               <eb:PartyId type="someType">QRS543</eb:PartyId>
               <eb:Role>http://example.org/roles/Seller</eb:Role>
            </eb:To>
         </eb:PartyInfo>
         <eb:CollaborationInfo>
            <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
            <eb:Service type="MyServiceTypes">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@example.com">
               <eb:Schema location="http://registry.example.org/bar.xsd" version="2.0"/>
               <eb:PartProperties>
                  <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
                  <eb:Property name="MimeType">application/xml</eb:Property>
               </eb:PartProperties>
            </eb:PartInfo>
         </eb:PayloadInfo>
      </eb:UserMessage>

  </eb:Messaging>

</S11:Header>

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

6. Error Handling

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.

6.1. 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.

6.2. Packaging of ebMS Errors

6.2.1. 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 origin="ebMS" category="Unpackaging"
          shortDescription="InvalidHeader"
          errorCode="EBMS:0009" severity="fatal">
    <eb:Description xml:lang="en"> … </eb:Description>
</eb:Error>
```

6.2.2. eb:Error/@origin

This OPTIONAL attribute identifies the functional module within which the error occurred. This module could be the the ebMS Module, the Reliability Module, or the Security Module. Possible values for this attribute include "ebMS", "reliability", and "security". The use of other modules, and thus their corresponding @origin values, may be specified elsewhere, such as in a forthcoming Part 2 of this specification.

6.2.3. eb:Error/@category

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

6.2.4. eb:Error/@errorCode

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

6.2.5. 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.

6.2.6. eb:Error/@refToMessageInError

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

6.2.7. eb:Error/@shortDescription

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

6.2.8. 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.

6.2.9. 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.

6.3. 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.

For an example of an ebXML Error Message, see Section 5.3.3.

6.4. Extensibility of the Error Element

6.4.1. 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:".

6.5. 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).

6.6. Error Reporting

There are three primary means of Error Reporting:

- Reporting with Fault Sending: An MSH may generate a SOAP Fault for reporting ebMS processing errors of severity "failure", which prevent further message processing. This Fault must comply with SOAP Fault processing, i.e. be sent back as an HTTP response in case the message in error was over an HTTP request. In case of ebMS processing errors (see Section 6.7.1), the Fault message MUST also include the eb:SignalMessage/eb:Error element in the eb:Messaging header.
- 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. The reporting message must always be combined with a SOAP Fault unless the severity is "warning".

***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*.

Appendix E shows possible options for combining error reporting with ebMS MEPs, when binding to a two-way protocol such as HTTP. It also shows how these combinations can be controlled with P-Mode parameters.

6.7. Standard ebMS Errors

This section defines the standard error codes expected to be generated and processed by a conformant MSH. They are segmented according to the stage of processing they are likely to occur: during reliable message processing, security processing, and general ebMS processing.

6.7.1. 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	EmptyMessagePartitionChannel	warning	Communication	There is no message available for pulling from this MPC 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 or another header (e.g. reliability, security) expected by the MSH is not compatible with the expected content, based on the associated P-Mode.
EBMS:0011	ExternalPayloadError	failure	Content	The MSH is unable to resolve an external payload reference (i.e. a Part that is not contained within the ebMS Message, as identified by a PartInfo/href URI).

6.7.2. Security Processing Errors

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

Error Code	Short Description	Recommended Severity	Category Value	Description or Semantics
EBMS:0101	FailedAuthentication	failure	Processing	The signature in the Security header intended for the "ebms" SOAP actor, could not be validated by the Security module.
EBMS:0102	FailedDecryption	failure	Processing	The encrypted data reference the Security header intended for 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.

6.7.3. 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.

7. 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.

7.1. 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 C 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.

7.2. 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.

7.3. 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.

7.4. 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. It may also encrypt select child elements of the eb:Messaging header, leaving other elements unencrypted. For example, the eb:PartyInfo section may be used to aid in message routing before decryption of other elements has occurred. Therefore, when third-party routing of a message is expected, 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 be encrypted.

7.5. 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 be encrypted.

7.6. Signing and Encrypting Messages

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

7.7. Security Token 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 is RECOMMENDED to be supported, and MAY include support for wsse:PasswordText-type passwords. 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.

An MSH MAY support other types of Security Tokens, as allowed by the WS-Security family of standards.

7.8. 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.

7.9. Secured Message Examples

7.9.1. 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"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
    <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
        <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2006-10-31T17:36:20.656Z</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 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 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 S11: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>
    </S11:Header>
    <S11: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>
    </S11:Body>
</S11:Envelope>
```

7.9.2. 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"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
    <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
        <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
            <eb:UserMessage>
                <eb:MessageInfo>
                    <eb:Timestamp>2006-10-28T17:29:59.119Z</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 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 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 S11: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/oasis-wss-SwAProfile-
1.1#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/oasis-
wss-SwAProfile-1.1#Attachment-Ciphertext-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/oasis-
wss-SwAProfile-1.1#Attachment-Content-Signature-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>
    </S11:Header>
    <S11:Body/>
</S11: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--
```

7.9.3. Digitally Signed Receipt Signal Message

The following is an example of a signed Receipt for the User Message shown above in Section 7.9.1. Note the correlations to that message in the eb:RefToMessageId and ds:Reference elements.

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

<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
  <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/">
    <eb:Messaging id="ThisebMessage" S11:mustUnderstand="1">

      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:Timestamp>2006-10-31T18:02:37.429Z</eb:Timestamp>
          <eb:MessageId>UUID-3@msh-server.example.com</eb:MessageId>
          <eb:RefToMessageId>UUID-2@msh-server.example.com</eb:RefToMessageId>
        </eb:MessageInfo>

        <eb:Receipt>
          <ebbpsig:NonRepudiationInformation
       xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ebbpsig:MessagePartNRInformation>
              <ebbpsig:MessagePartIdentifier>ebMessage</ebbpsig:MessagePartIdentif
ier>
              <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:DigestValue>
              </ds:Reference>
            </ebbpsig:MessagePartNRInformation>
            <ebbpsig:MessagePartNRInformation>
              <ebbpsig:MessagePartIdentifier>body</ebbpsig:MessagePartIdentifier>
              <ds:Reference URI="#body">
                <ds:Transforms>
                  <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#"/>
                </ds:Transforms>
                <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                <ds:DigestValue>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestValue>
              </ds:Reference>
            </ebbpsig:MessagePartNRInformation>
          </ebbpsig:NonRepudiationInformation>
        </eb:Receipt>
      </eb:SignalMessage>

    </eb:Messaging>

    <wsse:Security S11: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>

      <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="#ThisebMessage">
            <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:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>T24okA0MUh5iBNMG6tk8QAKZ+lFMmY1rcPnkOr9j3fHRGM2qqUnoB
                           ydOTnClcEMzPZbnlhdNYZYmab1lqa4N5ynLjwlM4kp0uMip9hapij
                           wL67aBnUeHiFmUau0x9DBOdKZTVa1QQ92106gej2YPDt3VKIlLLT2
                           c8O4TfayGvuY= </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>
  </S11:Header>
  <S11:Body/>
</S11:Envelope>
```

7.10. Message Authorization

Message authorization is defined here as authorizing the processing of a message in conformance with the parameters of the P-Mode associated with this message. This includes authorizing the access to some ebMS resources such as:

- "delivery" resources as identified by eb:Service and eb:Action
- Message Partition Channel (MPC) that a Pull signal is accessing for pulling messages.

This is different from simply authorizing a received message for further processing by the MSH, which can be achieved by processing the Security header described earlier in Section 7, regardless of ebMS-specific resources claimed by the message. A message could successfully be authenticated by the security module (see Section 4.1), yet not be authorized to pull from a particular MPC, or to effect delivery of data to a particular Service. For implementations in which there is limited interaction between processing modules of the MSH - e.g. in case of an architecture based on composing SOAP nodes, the Security header MAY be consumed by the WSS module before reaching the ebMS message processor. (Even if the header is forwarded, it may be impractical to require an ebMS processor implementation to parse it.)

This specification provides a resource-level authorization mechanism. Since any resource a message may claim access to is identified by the P-Mode associated with the message, this is equivalent to authorizing the association of the message with the P-Mode.

For this purpose, a second wsse:Security header, which contains only an authentication token, MAY be present. This specification describes in particular one token option, not exclusively of others: the wsse:UsernameToken profile. This secondary Security header may itself be secured (e.g. encrypted) by the main Security header.

In the P-Mode model (see Appendix D) such tokens are represented as the PMode.Initiator.Authorization parameter set (for authorizing the initiator of an MEP) and the PMode.Responder.Authorization parameter set.

This header is not intended to be processed or consumed by the same WSS module as the "main" Security header, but is targeted further along to the "ebms" actor - typically a role played by the ebMS header processor, which has knowledge of the association between these tokens and the P-Modes that govern the message processing.

The following example shows a PullRequest message for which this type of authorization is required. Both security headers (shown here as a SOAP1.1 message) are present, with one of them - the secondary header - targeted to the "ebms" actor. This Pull signal can effect message delivery from MPC "http://msh.example.com/mpc123" only if its credentials match the authorization parameters of at least one P-Mode associated with pulling messages on this MPC.

```
<?xml version="1.0" encoding="UTF-8"?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/">

  <S11:Header
      xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-secext-1.0.xsd">

    <eb:Messaging S11:mustUnderstand="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
      <eb:SignalMessage>
        <eb:MessageInfo>
          <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
          <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
        </eb:MessageInfo>
        <eb:PullRequest mpc="http://msh.example.com/mpc123" />
      </eb:SignalMessage>
    </eb:Messaging>

    <wsse:Security S11:mustUnderstand="1">
      <!-- main security header -->
    </wsse:Security>

    <wsse:Security S11:mustUnderstand="1" actor="ebms"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd">
      <!-- authorization security header (here non encrypted) -->
      <wsse:UsernameToken wsu:Id="ebms-1234">
        <wsse:Username>acme</wsse:Username>
        <wsse:Password Type="...">xyz123</wsse:Password>
        <wsu:Created> ... </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>

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

Permission to use a P-Mode for processing a received message is granted or denied at the time the P-Mode authorization parameters are compared with the credentials in the message.

7.11. Securing the PullRequest Signal

7.11.1. 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].

7.11.2. Authorization

The processing of a PullRequest signal received by a Sending MSH MAY be authorized based on any of the following, or combination of the following, mechanisms:

(a) Digital signature validation by the Security (WSS) module (see Sections 7.2 and 7.3),

(b) A WSS authentication token addressed to the "default" actor/role (see Section 7.7).

(c) A WSS authentication token addressed to the "ebms" actor/role (see Section 7.10).

(d) A transfer-protocol-level identity-authentication mechanism, such as those described in Section 7.11.1.

7.11.3. Preventing Replay 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.

7.12. Countermeasure Technologies

7.12.1. 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.

7.12.2. Persistent Signed Receipt

An ebMS Message that has been digitally signed MAY be acknowledged with a message containing an eb:Receipt Signal (described in Section 5.2.3.3), that itself is digitally signed in the manner described in the previous section. The Receipt Signal MUST contain the information necessary to provide nonrepudiation of receipt of the original message; that is, an XML Digital Signature Reference element list consistent with that contained in the Web Services Security Signature element of the original message.

7.12.3. 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.

7.12.4. 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.

7.12.5. 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.

7.12.6. 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.

7.12.7. Persistent Authorization

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

7.12.8. 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.

7.13. 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.

8. Reliable Messaging Module

8.1. The Reliable Messaging Model

The reliable delivery of messages has two aspects:

1. a contractual aspect regarding delivery conditions and error notification, where the contracting parties are the MSHs and the entities using the MSH - the message Producer and Consumer.
2. a protocol aspect, that describes the reliability mechanism "on the wire".

This section emphasizes the contractual aspect. The details of the protocol aspect depend on the specifics of the reliability module and its binding, described in Appendix B.

8.1.1. Message Processing

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.

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 message processing flows do not exclude the insertion of such additional steps, which are depicted in Figure 7 and described in Section 4.1.

8.1.2. The Reliable Messaging Processor in the MSH

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 (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. This other component is normally the module that is processing the ebMS header and its packaging, as described in the Processing Model (Section 4.1). On the sender side, this module is abstracted as the RM-Producer. On the receiver side, it is abstracted as 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 message received reliably. This response is sent back reliably.
- **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 message sent reliably (e.g. a notification telling that the message was not delivered).

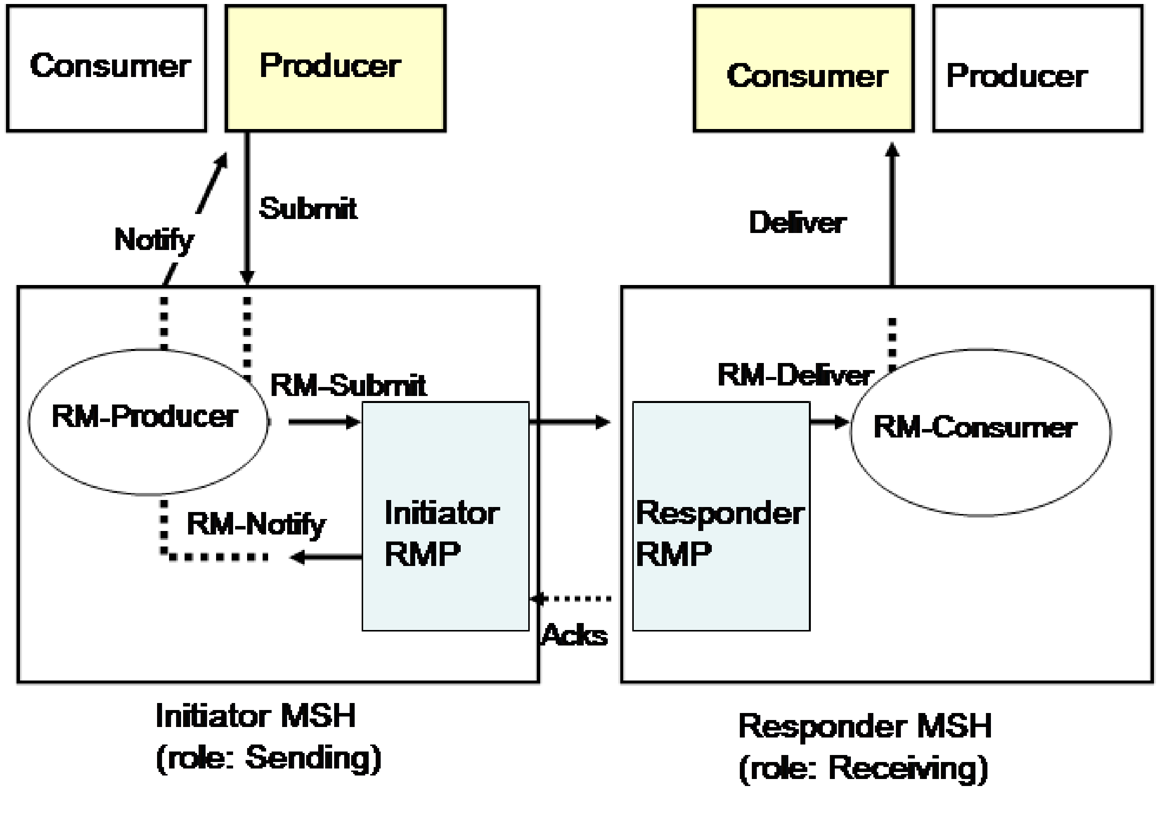

Figure 10: Sending an ebMS Message Reliably

Figure 10 shows the operations involved when sending a request reliably. As indicated in Section 8.3, this sequence of operations applies either to the User Message in the One-Way/Push MEP, the PullRequest Signal of a One-Way/Pull MEP, or the first leg of a Two-Way/Sync MEP.

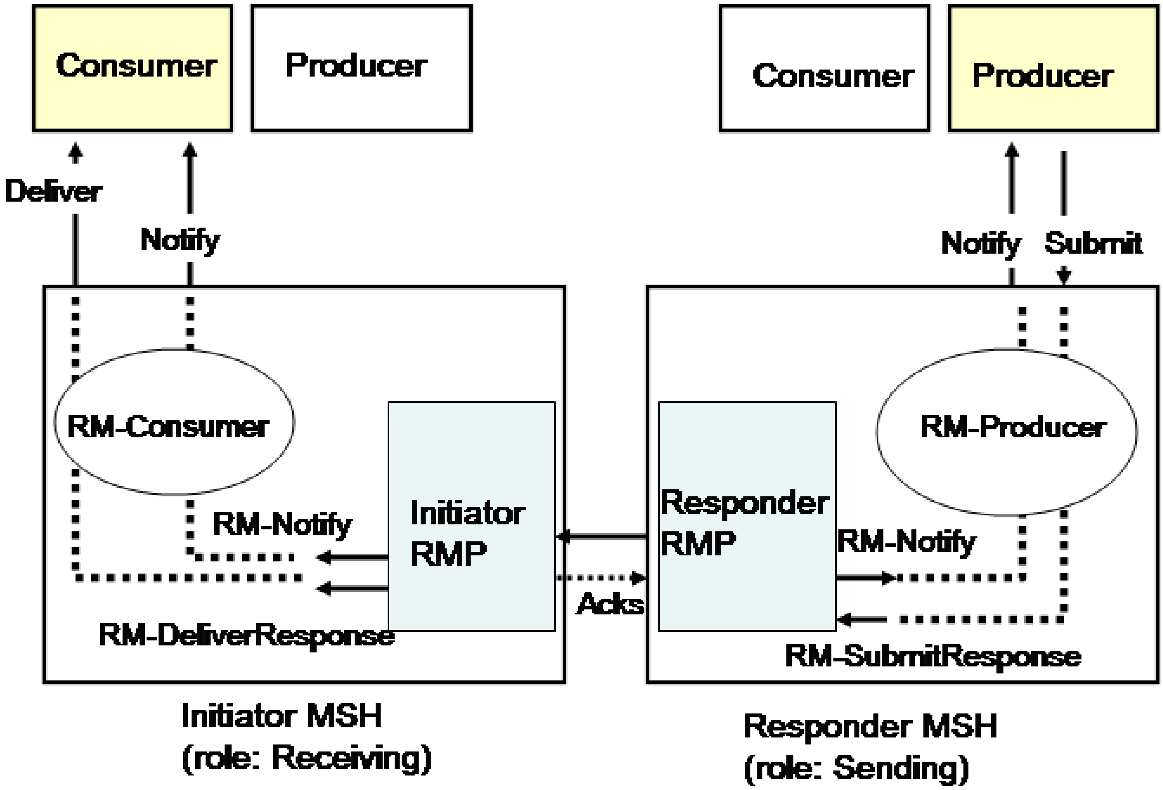

Figure 11: Sending an ebMS MEP Response Message Reliably

Figure 11 shows the abstract operations and components involved when sending a response reliably. As indicated in Section 8.3, this sequence of operations applies ither to a pulled user message in the One-Way/Pull MEP or the response user message in a Two-Way/Sync MEP. Note that depending on the reliability processing mode (P-Mode.Reliability), awareness of delivery failure may occur on either side.

8.2. Reliable Delivery of ebMS Messages

Because the reliability function is supported by a module (RMP) within the MSH, the contractual aspect has to be considered at two levels: (a) between the RMP and the MSH internals, and (b) between the MSH and its Consumer/Producer entities (e.g. an application).

8.2.1. Reliability Contracts for the RMP

Depending on the reliability required for a request message, an RMP must support some or all of the following contracts:

- **At-Least-Once RM-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 RM-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 RM-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, as illustrated in Figure 11. In such a case they are expressed in the above contracts with RM-SubmitResponse and RM-DeliverResponse operations (instead of RM-Submit and RM-Deliver, respectively), and the Responder and Initiator RMPs switch roles.

These contracts may be combined; e.g. Exactly-Once results from the combination of At-Least-Once and At-Most-Once.

In order to support these reliability contracts, both Initiator and Responder RMPs MUST use a reliability protocol independent from the transport protocol and that provides end-to-end acknowledgment and message resending capabilities. The details and parameters associated with these protocol functions are described in Appendix B.

8.2.2. Reliability Contracts for the MSH

Because reliability quality of service (QoS) must have significance for the user 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) a delivery failure notification is communicated (Notify operation) to either the Producer or the Consumer.
- **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. An ebMS message is a duplicate of another if it has same eb:MessageId value.
- **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 Section 8.3.
- 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 sending 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 and reported either to the Producer or the Consumer, depending on the P-Mode.ErrorHandling.
- Have sufficient control on which RM sequence is used when submitting a message (RM-Submit), so that an RM sequence may be mapped to an ebMS conversation (eb:ConversationId).

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

8.2.3. Reliability for Signal Messages

Messages that have eb:CollaborationInfo/eb:Service set to "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/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 to 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, the Producer MAY be notified (Notify) of the failure depending on the reporting mode, as for regular user messages.

8.2.4. Handling of Delivery Failures

Delivery is an abstract operation that may be implemented in various ways. It is the responsibility of an implementation or product to clearly state at what point in its processing it considers that a message is delivered. Such a statement amounts to defining a concrete "binding" to the Deliver operation, that a user can rely on for interpreting the reliability contracts defined and required in this specification, relative to this implementation.

There are two options when supporting the At-Least-Once delivery contract:

1. Delivery failures are always notified to the Producer (the sending side).
2. Delivery failures are always notified, though either to the Producer or to the Consumer (the receiving side), depending on the nature of the failure.

It is part of an agreement between parties to decide which notification option (1 or 2) must be enforced. An MSH implementation may also be limited in its ability to support option 1. Conformance profiles for this specification may require either option to be supported.

Delivery Failures (DFs) may be caused by network failure, or by processing failure on either side. In the remaining part of this section, the following is assumed:

- An MSH is always aware of processing failures that occur locally or that have been communicated to it, and it is always able to report these to its local party (Producer or Consumer) in some way. E.g. a message processing failure in a Receiving RMP can always be notified to the Consumer.
- A DF that needs to be communicated from MSH to MSH should not itself rely on the transfer of an Error message (or a Fault), as such message may precisely also fail to be transferred. It is safer that it relies on the "non-transfer" of a message, such as a missing Acknowledgment.

  Note:
  By relying on the non-reception of an Acknowledgment for notifying DF, "false" DFs may occur (in case of Acknowledgment loss), but the case where a message fails to be delivered unknowingly from the Producer (false delivery) cannot occur. False DF - which can never be completely eliminated - can always be detected outside the reliable messaging (RM) layer, in a tractable and less urgent way - e.g. the sending party may synchronize on a daily basis by communicating its list of assumed delivery failures, for confirmation by receiver. The Status Request feature (which may be described in a forthcoming Part 2 of the ebMS specification) could facilitate this function.

Restrictions in the ability to support notification option 1 usually depend on the semantics of Acknowledgment that is supported by the RMP. Three cases are to be considered:

Case 1: The acknowledgment is "on receipt" (as in WS-ReliableMessaging) and has no delivery semantics. In that case:

- DF notifications to the Producer rely on lack of acknowledgments for network failures (non-reception of a User message)
- DF notifications to the Producer rely on Error messages (or Faults) for any other failure occurring after reception, on Consumer side.

For reasons mentioned above, this acknowledgment semantics does not generally support option 1. However, in the case of the HTTP binding with no intermediaries present, non-delivery due to processing failure can still be indicated in a reliable way to the sending side (and will trump the acknowledgment), as either a SOAP Fault is received on the HTTP response or the HTTP response fails.

The requirements for this transport-specific solution to option 1 which is reliable only for non-delivered pushed messages (as opposed to pulled) are detailed in Appendix B.

Case 2: The acknowledgment is "on MSH-delivery" (supported in WS-Reliability). In that case, notification option 1 can be supported as well as option 2. In order for option 1 to be supported, an RMP must implement RM-Deliver operation so that it is only considered successful (worthy of sending an acknowledgment) if the Deliver operation from MSH to Consumer also succeeds. It is RECOMMENDED that an implementation support this acknowledgment semantics.

Case 3: The acknowledgment is "on RM-delivery" (supported in WS-Reliability). In case the condition in Case 2 is not supported by an RMP implementation, RM-Delivery is only concerning the RMP module and does not coincide with MSH delivery. Acknowledgments are "on RM-delivery" only.

Support for option 1 may be accomplished by relying on the transport-specific solution mentioned in Case 1. This solution is easier to implement here, as it only concerns the module processing the ebMS header (not the RMP implementation), as described in Appendix B.

8.3. Reliability of ebMS MEPs

This section describes the reliability model for MEPs. For a concrete enumeration of all reliability options for MEPs in the context of an HTTP binding, see Appendix E, which also shows how these combinations can be controlled with P-Mode parameters.

8.3.1. Reliability of the One-Way/Push MEP

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.

  Note:
  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 this sequence, or when it is invoked but does not complete successfully.

Figure 12 illustrates the message flow for this reliable MEP.

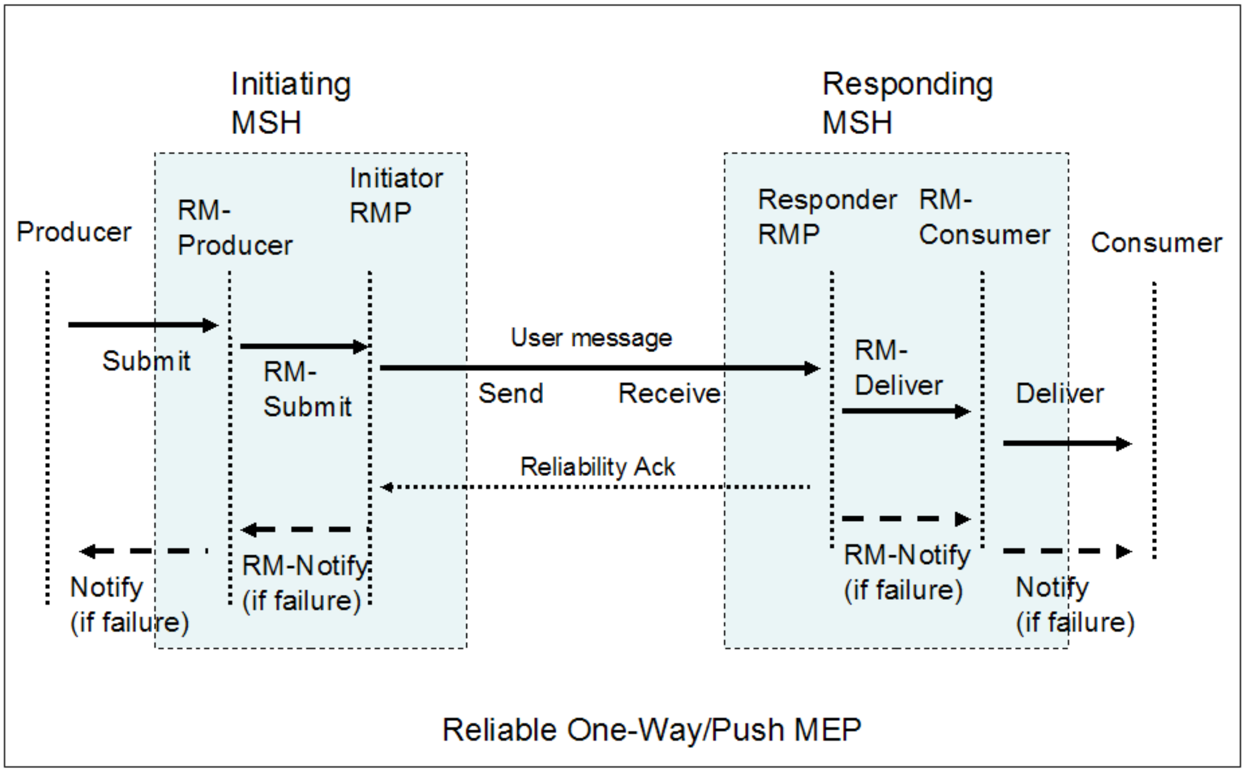

Figure 12: Reliable One-Way/Push MEP

The way in which the Reliability Acknowledgment binds to the underlying protocol - e.g. as a separate HTTP request, or on the back-channel of a previous message - is controlled by the P-Mode parameter Reliability.AtLeastOnce.ReplyPattern.

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 for the Consumer on the Initiator side.

On Initiator MSH side:

- Step (2): Generation of a 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.

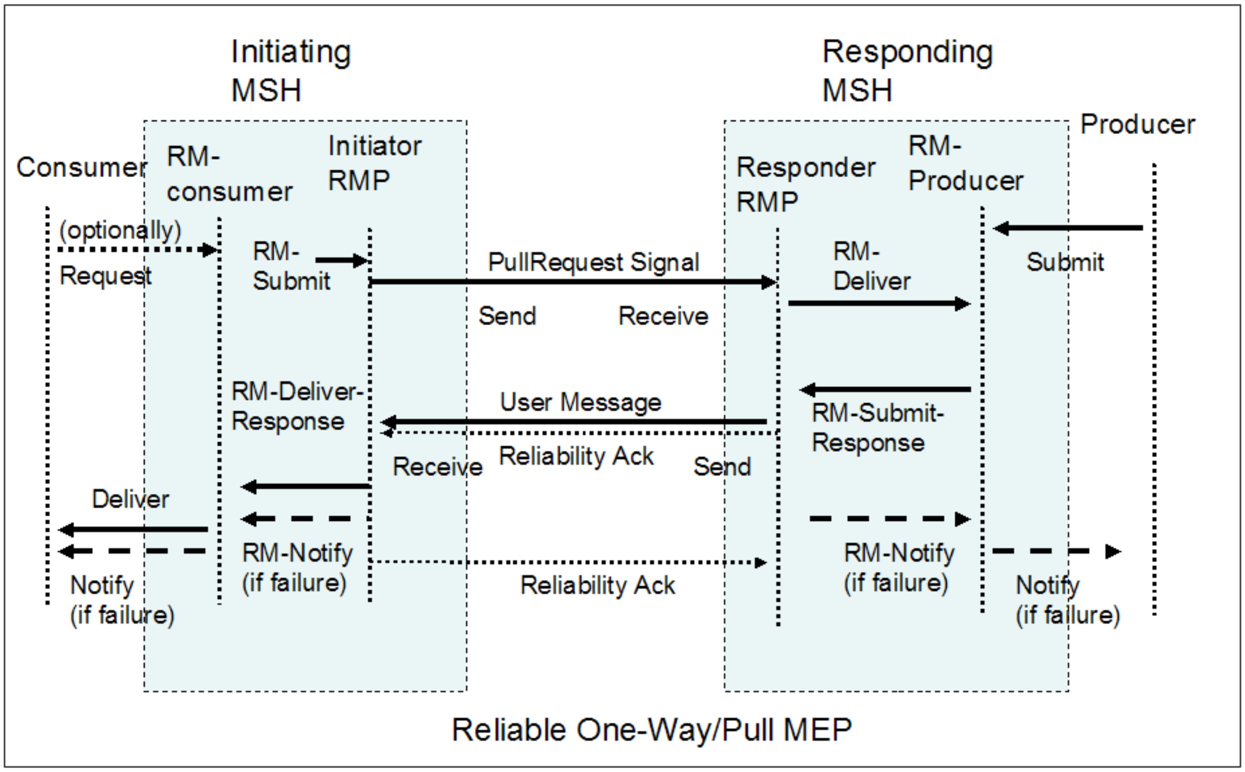

Figure 13: Reliable One-Way/Pull MEP

The way in which the Reliability Acknowledgments are bound to the underlying protocol is controlled by the P-Mode parameter Reliability.AtLeastOnce.ReplyPattern.

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 P-Mode.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 Two-Way/Sync MEP

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 for 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.

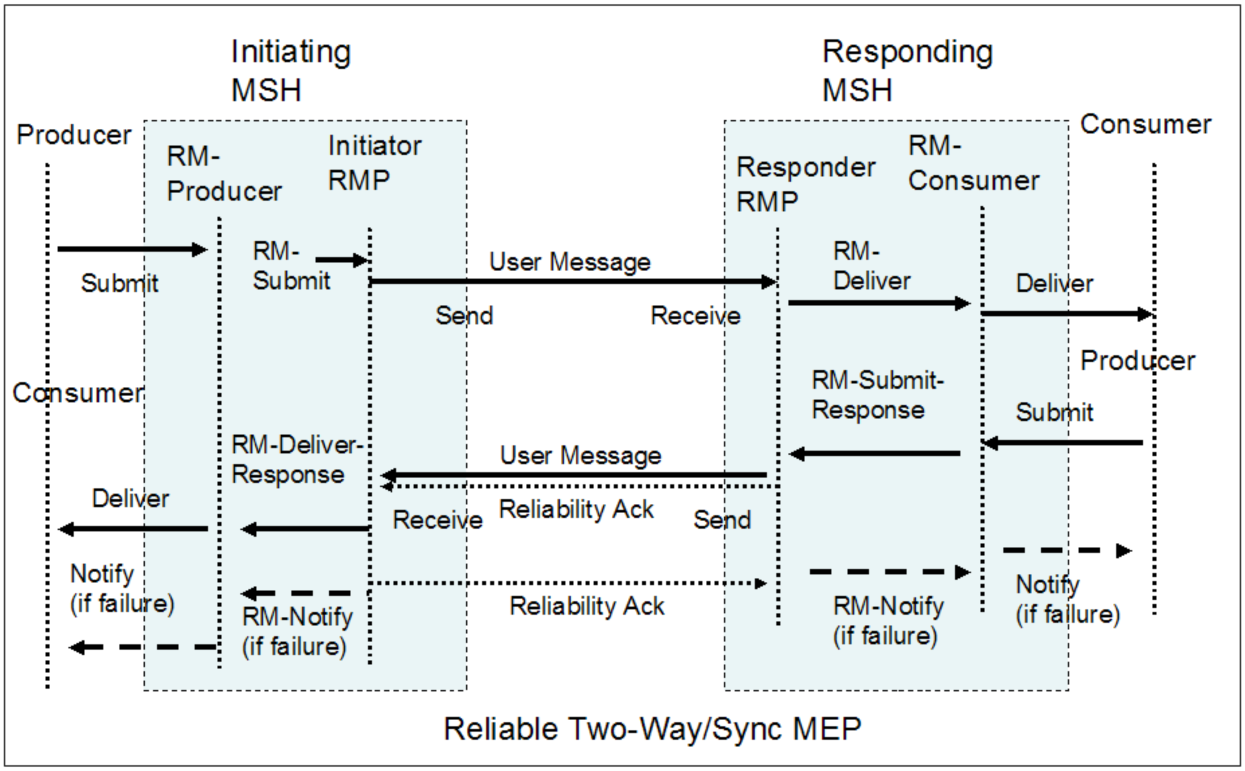

Figure 14: Reliable Two-Way/Sync MEP

The way in which the Reliability Acknowledgments are bound to the underlying protocol is controlled by the P-Mode parameter Reliability.AtLeastOnce.ReplyPattern.

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.

8.3.4. Reliability of Other Transport-Channel-Bound MEPs

Each one of the MEPs defined in Section 2.2.8: Two-Way/Push-and-Push, Two-Way/Push-and-Pull, and Two-Way/Pull-and-Push, has been characterized as having a message choreography equivalent to a sequence of two of the previous MEPs (e.g. Two-Way/Push-and-Pull has a choreography equivalent to One-Way/Push + One-Way/Pull). The reliability of these more complex MEPs may be handled by composing reliable versions of these simpler exchanges, which are described in Sections 8.3.1, 8.3.2 and 8.3.3. It can be noted that the reliable Two-Way/Push-and-Push MEP will not make use of the RM-SubmitResponse operation.

APPENDIX A. The ebXML SOAP Extension Element Schema

Following is the XML schema that describes the eb:Messaging header, as described in Section 5.2. This copy is provided for convenience only, and is non-normative. The normative version of the schema may be found in a separate file, at http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
xmlns:tns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
targetNamespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xsd:annotation>
      <xsd:appinfo>Schema for ebMS-3 XML Infoset</xsd:appinfo>
      <xsd:documentation xml:lang="en">
         This schema defines the XML Infoset of ebMS-3 headers. These headers are
         placed within the SOAP Header element of either a SOAP 1.1 or SOAP 1.2
         message.
      </xsd:documentation>
   </xsd:annotation>
   <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
schemaLocation="http://schemas.xmlsoap.org/soap/envelope/"/>
   <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
schemaLocation="http://www.w3.org/2003/05/soap-envelope"/>
   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
   <xsd:element name="Messaging" type="Messaging"/>
   <xsd:complexType name="Messaging">
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
   The eb:Messaging element is the top element of ebMS-3 headers, and it is
   placed within the SOAP Header element (either SOAP 1.1 or SOAP 1.2). The
   eb:Messaging element may contain several instances of eb:SignalMessage
   and eb:UserMessage elements. However in the core part of the ebMS-3
   specification, only one instance of either eb:UserMessage or eb:SignalMessage
   must be present. The second part of ebMS-3 specification may need to include
   multiple instances of either eb:SignalMessage, eb:UserMessage or both.
   Therefore, this schema is allowing multiple instances of eb:SignalMessage
   and eb:UserMessage elements for part 2 of the ebMS-3 specification. Note
   that the eb:Messaging element cannot be empty (at least one of
   eb:SignalMessage or eb:UserMessage element must present).
         </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element name="SignalMessage" type="SignalMessage" minOccurs="0"
maxOccurs="unbounded"/>
         <xsd:element name="UserMessage" type="UserMessage" minOccurs="0"
maxOccurs="unbounded"/>
         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="tns:headerExtension"/>
   </xsd:complexType>
   <xsd:complexType name="SignalMessage">
      <xsd:annotation>
         <xsd:documentation xml:lang="en">
   In the core part of ebMS-3 specification, an eb:Signal Message is allowed to
   contain eb:MessageInfo and at most one Receipt Signal, at most one
eb:PullRequest
   element, and/or a series of eb:Error elements. In part 2 of the ebMS-3
   specification, new signals may be introduced, and for this reason,
   an extensibility point is added here to the eb:SignalMessage element to
   allow it to contain any elements.
         </xsd:documentation>
      </xsd:annotation>
      <xsd:sequence>
         <xsd:element name="MessageInfo" type="MessageInfo"/>
         <xsd:element name="PullRequest" type="PullRequest" minOccurs="0"/>
         <xsd:element name="Receipt" type="Receipt" minOccurs="0"/>
         <xsd:element name="Error" type="Error" minOccurs="0"
maxOccurs="unbounded"/>
         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Error">
      <xsd:sequence>
         <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
         <xsd:element name="ErrorDetail" type="xsd:token" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="category" type="xsd:token" use="optional"/>
      <xsd:attribute name="refToMessageInError" type="xsd:token" use="optional"/>
      <xsd:attribute name="errorCode" type="xsd:token" use="required"/>
      <xsd:attribute name="origin" type="xsd:token" use="optional"/>
      <xsd:attribute name="severity" type="xsd:token" use="required"/>
      <xsd:attribute name="shortDescription" type="xsd:token" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="PullRequest">
      <xsd:sequence>
         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="pullAttributes"/>
   </xsd:complexType>
   <xsd:complexType name="Receipt">
      <xsd:sequence>
         <xsd:any namespace="##other" processContents="lax"
maxOccurs="unbounded"/>
      </xsd:sequence>
   </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="MessageProperties" type="tns:MessageProperties"
minOccurs="0"/>
         <xsd:element name="PayloadInfo" type="tns:PayloadInfo" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="MessageInfo">
      <xsd:sequence>
         <xsd:element name="Timestamp" type="xsd:dateTime"/>
         <xsd:element name="MessageId" type="tns:non-empty-string"/>
         <xsd:element name="RefToMessageId" type="tns:non-empty-string"
minOccurs="0"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="PartyInfo">
      <xsd:sequence>
         <xsd:element name="From" type="tns:From"/>
         <xsd:element name="To" type="tns:To"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="PartyId">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="From">
      <xsd:sequence>
         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
         <xsd:element name="Role" type="tns:non-empty-string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="To">
      <xsd:sequence>
         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
         <xsd:element name="Role" type="tns:non-empty-string"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="CollaborationInfo">
      <xsd:sequence>
         <xsd:element name="AgreementRef" type="tns:AgreementRef" minOccurs="0"/>
         <xsd:element name="Service" type="tns:Service"/>
         <xsd:element name="Action" type="xsd:token"/>
         <xsd:element name="ConversationId" type="xsd:token"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Service">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"
use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="AgreementRef">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="type" type="tns:non-empty-string"
use="optional"/>
            <xsd:attribute name="pmode" type="tns:non-empty-string"
use="optional"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="PayloadInfo">
      <xsd:sequence>
         <xsd:element name="PartInfo" type="tns:PartInfo" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="PartInfo">
      <xsd:sequence>
         <xsd:element name="Schema" type="tns:Schema" minOccurs="0"/>
         <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
         <xsd:element name="PartProperties" type="tns:PartProperties"
minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="href" type="xsd:token"/>
   </xsd:complexType>
   <xsd:complexType name="Schema">
      <xsd:attribute name="location" type="xsd:anyURI" use="required"/>
      <xsd:attribute name="version" type="tns:non-empty-string" use="optional"/>
      <xsd:attribute name="namespace" type="tns:non-empty-string" use="optional"/>
   </xsd:complexType>
   <xsd:complexType name="Property">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute name="name" type="tns:non-empty-string"
use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:complexType name="PartProperties">
      <xsd:sequence>
         <xsd:element name="Property" type="tns:Property" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="MessageProperties">
      <xsd:sequence>
         <xsd:element name="Property" type="Property" maxOccurs="unbounded"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:attributeGroup name="headerExtension">
      <xsd:attribute name="id" type="xsd:ID" use="optional"/>
      <xsd:attribute ref="S11:mustUnderstand" use="optional">
         <xsd:annotation>
            <xsd:documentation>
       if SOAP 1.1 is being used, this attribute is required
           </xsd:documentation>
         </xsd:annotation>
      </xsd:attribute>
      <xsd:attribute ref="S12:mustUnderstand" use="optional">
         <xsd:annotation>
            <xsd:documentation>
       if SOAP 1.2 is being used, this attribute is required
           </xsd:documentation>
         </xsd:annotation>
      </xsd:attribute>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
   </xsd:attributeGroup>
   <xsd:attributeGroup name="pullAttributes">
      <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
   </xsd:attributeGroup>
   <xsd:complexType name="Description">
      <xsd:simpleContent>
         <xsd:extension base="tns:non-empty-string">
            <xsd:attribute ref="xml:lang" use="required"/>
         </xsd:extension>
      </xsd:simpleContent>
   </xsd:complexType>
   <xsd:simpleType name="non-empty-string">
      <xsd:restriction base="xsd:string">
         <xsd:minLength value="1"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

APPENDIX B. Reliable Messaging Bindings

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 G).

B.1. WS-Reliability Binding

B.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).

B.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).

B.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.

B.1.4. Complement to the Reliability of the Two-Way/Sync 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.

B.2. WS-ReliableMessaging Binding

> Note
> This section is based on the Committee Specification (11 April 2007) of the WS-ReliableMessaging Version 1.1 specification [WSRM11]. It is possible that updates will be required in order to conform with the final release of WS-ReliableMessaging as OASIS Standard. However, it is expected that such updates, if any, will be minor and can be handled via the errata process.

B.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 on the Source side. The MSH must escalate such faults as DysfunctionalReliability ebMS errors (EBMS:0201).
- WSRM-Required fault, when received by an RMS, must be communicated to the MSH on Source side. The MSH must escalate such faults as ProcessingModeMismatch (EBMS:0010). It is recommended to report the RM Error code in the ErrorDetail element of EBMS:0010.
- InvalidAcknowledgment and UnknownSequence, when received by the RMD, must be communicated to the MSH on Destination side. 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 map to equivalent delivery assurance definitions in the WS-RM specification. Although WS-RM does not mandate support for these delivery assurances (DAs), and only specifies the protocol aspect, a conformance profile supporting reliable messaging requires the use of a WS-RM implementation (RMD) that supports at least some of these DAs as extensions.

It is RECOMMENDED that all messages transmitted over a same sequence use the same MPC. This becomes 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. MAY be specified in the Processing Mode, as extensions to the PMode.Reliability group (see Appendix D).

Sequence acknowledgments 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 either be created by a CreateSequence message carried over an HTTP response, the HTTP request being initiated by this MSH, or be offered (using wsrm:Offer) by the CreateSequence used for opening a sequence for sending Pull signals reliably.

Either one of the two following options MUST be used, in order to enable MSH interoperability based on WS-ReliableMessaging, regarding the reliability contracts for messages exchanged between two MSHs:

1. The reliability contract and parameters apply equally to all messages sent between two MSHs. All messages exchanged in the same direction between two MSHs are subject to the same reliability quality of service. In such a case, the P-Modes.Reliability parameters associated with each one of these messages must not conflict with this common quality of service.

2. The reliability contract and parameters MAY vary from one message to the other. In that case, the scope of application of a reliability contract MUST be the sequence, meaning all messages within the same sequence are subject to the same reliability contract.

When support for case (2) above is required, the source of a sequence (RMS) must be able to indicate which delivery assurance is associated with this sequence, so that the RMD implements the expected DA. Indeed, although both MSHs share knowledge of the reliability contracts associated with each message (P-Mode.reliability), the RMD has no access to the ebMS header, and can only rely on the sequence number. In order to avoid the constraint of using predefined sequence numbers, the association DA-sequence must be dynamically supported by an RMS. Consequently, an implementation of WS-ReliableMessaging that supports case (2) MUST also support the extension of the wsrm:CreateSequence element with a child element structured as a policy assertion as defined in [WSRMP11], i.e. either one of the following:

```
(a) <wsrmp:AtLeastOnceDelivery wsrmp:InOrder='true|false'/>
(b) <wsrmp:AtMostOnceDelivery wsrmp:InOrder='true|false'/>
(c) <wsrmp:ExactlyOnceDelivery wsrmp:InOrder='true|false'/>
```

The above extensions MUST also be supported in wsrm:Accept/{any} and understood, in case of a conformance profile that requires support for reliable One-Way/Pull or reliable Two-Way/Sync. It is also RECOMMENDED that the above extensions be supported in wsrm:Offer/{any} and understood.

The above DA assertion (a) must match a P-Mode.Reliability with parameters AtMostOnce.Contract = "false", AtLeastOnce.Contract = "true"; and its attribute @wsrmp:InOrder must match the InOrder.Contract value.

The above DA assertion (b) must match a P-Mode.Reliability with parameters AtMostOnce.Contract = "true", AtLeastOnce.Contract = "false"; and its attribute @wsrmp:InOrder must match the InOrder.Contract value.

The above DA assertion (c) must match a P-Mode.Reliability with parameters AtMostOnce.Contract = "true", AtLeastOnce.Contract = "true"; and its attribute @wsrmp:InOrder must match the InOrder.Contract value.

Additional reliability parameters – if any, e.g. resending frequency, etc. - associated with each one of the reliability contracts (At-Least-Once, At-Most-Once, In-Order) are to be defined in P-Mode.Reliability extensions and known from both parties prior to the exchange with no need to be transmitted via the RM protocol. When receiving a CreateSequence message with the above extension specifying a reliability contract, the RMD MUST be able to resolve it to a single set of additional parameters governing this mode of reliability. For example, the P-Modes of all messages sent under At-Least-Once should have same values for the set of PMode.Reliability parameters related to this contract (AcksTo, AcksOnDelivery, ReplyPattern and any other custom parameters such as those controlling message resending, if any), as well as for the NotifyProducerDeliveryFailures parameter about failure reporting.

Because acknowledgments in WS-ReliableMessaging are on receipt, the Reliability.AckOnDelivery parameter in the P-Mode of messages sent reliably MUST be "false".

B.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 sequence be initiated by the RMS sending a wsrm:CreateSequence message, as opposed to responding to an wsrm:Offer.

In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Response", then the CreateSequence/AcksTo element MUST contain an WS-Addressing anonymous IRI.

In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Callback", then the CreateSequence/AcksTo element MUST contain an URI specified in an additional P-Mode.Reliability parameter.

The P-Mode.Reliability.AtLeastOnce.ReplyPattern MUST NOT have value "Poll",

When an underlying two-way protocol is used, any pair of sequence lifecycle message (CreateSequence/CreateSequenceResponse, CloseSequence/CloseSequenceResponse, TerminateSequence/ TerminateSequenceResponse) SHOULD be exchanged over a single request-response MEP of the protocol.

It is RECOMMENDED that the Initiator MSH be made aware of a delivery failure from the Responder MSH to its Consumer (NotifyProducerDeliveryFailures = "true"). 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.

B.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.

When initiating an instance of the One-Way/Pull MEP, and if it is expected – based on P-Modes deployed - that pulled message may be sent reliably, then the PullRequest signal itself MUST be sent under At-Least-Once delivery (see Section 8). Acknowledgments for Pull signals should be sent over the second leg of the One-Way/Pull MEP (PMode.Reliability.AtLeastOnce.ReplyPattern ="Response"), bundled with the pulled ebMS user message. However the frequency of acknowledgments may not need be on a per message basis.

In case pulled messages must be sent reliably, the following requirements apply:

- When a sequence is initiated (CreateSequence) to be associated with PullRequest signals intended for the same MPC, then the wsrm:Offer MUST be present in the CreateSequence element. The offered sequence SHOULD be used for sending back pulled messages reliably.
- When no more messages have to be pulled reliably from an MPC, it is RECOMMENDED that the Sending MSH closes and terminate the associated sequences. When the Sending MSH decides to terminate a reliable sequence of pulled messages, a CloseSequence message or a TerminateSequence SHOULD be sent over a pulled message, e.g. piggybacked over the EmptyMessagePartitionChannel warning (EBMS:0006).

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.

B.2.4. Complement to the Reliability of the Two-Way/Sync MEP

In the reliable Two-Way/Sync MEP, either:

- The request message alone is sent reliably, in which case the requirements and recommendations for the One-Way/Push also apply here.
- Or both the request and the reply are sent reliably. The response alone SHALL NOT be sent reliably.

In case both request and reply are sent reliably, it is RECOMMENDED that both sequences are established and discarded in a coordinated way. The same rules apply as for the reliability of the One-way Pull MEP. The in-bound sequence termination SHOULD be terminated on the initiative of the MEP

Initiator, after the out-bound sequence is terminated.

APPENDIX C. SOAP Format and Bindings

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 [HTTP11]and SMTP [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.).

C.1. Using SwA with SOAP-1.1

The following example shows an ebMS-3 message using SOAP 1.1 with attachments. 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@cars.example.com".
- 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://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
  <S11:Header>
     <eb:Messaging S11:mustUnderstand="1">
         …
           <eb:PayloadInfo>
              <eb:PartInfo href="cid:car-photo@cars.example.com" />
              <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@cars.example.com"/>
     </t:Data>
  </S11:Body>
</S11:Envelope>
```

```
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <car-photo@cars.example.com>

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

--MIME_boundary—
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <picture-of-owner@cars.example.com>

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

Example 1: SOAP-1.1 with Attachment

C.2. Using SwA with SOAP-1.2

The following (Example 2) shows the same message given in Example 1 above, 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://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
  <S12:Header>
     <eb:Messaging 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@cars.example.com>

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

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <picture-of-owner@cars.example.com>

...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

C.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 C.

> Note:
> This binding applies only to the ebMS One-Way/Push or Two-Way/Push-and-Push MEPs. An SMTP binding for the other ebMS MEPs involving the Pull or Synchronous transfer features would require an SMTP binding of the SOAP Request-Response MEP; for example, [SOAPEMAIL]. Use of such bindings are out of scope of this specification, and may be detailed in a forthcoming Part 2 of this specification.

APPENDIX D. Processing Modes

D.1. Objectives and Usage

A Processing Mode (or P-Mode) is a collection of parameters that determine how messages are exchanged between a pair of MSHs with respect to quality of service, transmission mode, and error handling.

A P-Mode may be viewed and used in two ways:

- It is an agreement between two parties as to how messages must be processed, on both the sending and receiving sides. Both MSHs must be able to associate the same P-Mode with a message, as this is necessary for consistent processing (of security, reliability, message exchange pattern, etc.) end-to-end.
- It is configuration data for a Sending MSH, as well as for a Receiving MSH.

Several P-Mode instances may be used to govern the processing of different messages between two MSHs. A P-Mode is usually associated with a class of messages that is identified by some common header values – e.g. the class of messages sharing same values for eb:Service, eb:Action, and eb:AgreementRef.

More abstractly, a P-Mode is said to be *deployed* on an MSH when it is governing the processing of an associated class of messages on the MSH.

Before a message is sent, the Sending MSH must be able to determine which P-Mode is used for this message. The process to determine this is left to each implementation – here are three examples:

Example 1: Several P-Modes have been deployed on the Sending MSH, one for each triple Service/Action/AgreementRef that is expected to be used in messages. When a message is submitted to a Sending MSH via an API, the Service, Action and AgreementRef to be put in the message header are also passed as arguments, along with the payload. The Sending MSH selects the P-Mode to be used for this message based on the values for Service/Action/AgreementRef, and completes the message header using other parameter values from the matched P-Mode (e.g. MPC, Role, PartyId, and the right content for the Reliable Messaging and Security headers). On the receiving side, the MSH will also associate the same P-Mode with this message.

Example 2: Several P-Modes have been deployed on the Sending MSH, and are given an ID (see PMode.ID below). When a message is submitted to a Sending MSH via an API, the ID of the P-Mode it is associated with is explicitly provided, along with the payload. The Sending MSH then completes the message header using parameter values from the associated P-Mode (e.g. MPC, AgreementRef, Role, Service, Action…). When sending the message, the MSH also adds the P-Mode.ID in the header (as value of the AgreementRef/@pmode attribute), so that the association with the appropriate P-Mode is done unambiguously and faster by the Receiving MSH.

Example 3: A P-Mode has been deployed on the Sending MSH, which is a constrained device with a light conformance profile. Because this device is always supposed to process messages in the same way, the P-Mode is largely hard-coded in the implementation and only a few parameters are left for users to decide as their configuration choice.

This specification is only concerned with defining an abstract model for the P-Mode. It enumerates parameters and states their semantics w/r to the features described in the specification. This P-Mode data model is not concerned with a detailed representation for these parameters and their content, which is left to a P-Mode representation choice. The objective of these parameters is to represent abstract controls for these specification features, which can be used as a basis for configuring an implementation or can be communicated between parties via a concrete representation on which they need to agree.

For example, the parameter: PMode[1].Security.X509.Signature.Certificate simply assumes that the implementation is given a way to identify and access a certificate for the signature function. The representation details for this certificate identification are left to another document to specify – e.g. a P-Mode binding to WS-Policy [WSPOLICY] assertions (such as WS-SecurityPolicy [WSSECPOL]).

A P-Mode, or set of P-Modes, may also be represented as parts of a CPA document, the details of which are out of scope of this Appendix.

In order to promote the portability of P-Mode representations across MSH implementations, a conformance profile may require support for a particular P-Mode representation.

An implementation may decide to extend the P-Mode data model specified here, with additional parameters. Conversely, depending on its conformance profile an implementation may only need to support a subset of the P-Mode parameters described here.

D.2. Model for Processing Modes

A P-Mode actually governs the transmission of all the messages involved in an ebMS MEP between two MSHs. P-Mode parameters are grouped into six functional categories, also called P-Mode features (see Section 4):

- **General Parameters:** as a P-Mode concerns all messages in an ebMS MEP, these parameters are not associated with any particular message in the MEP, but are attributes of the entire MEP.
- **Protocol**: defines protocol-related parameters necessary for interoperating, that are associated with a particular message of the MEP.
- **BusinessInfo**: defines the business profile of a user message in terms of business header elements and their values (e.g. Service, Action) or other items with business significance (payload profile, MPC).
- **ErrorHandling**: defines the mode of handling and of reporting of errors associated with the message in this leg.
- **Reliability**: defines the reliability contracts and their parameters, applying to the message in this leg.
- **Security**: defines the security level expected for the message in the exchange, and provides related security context data.

Because messages in the same MEP may be subject to different requirements - e.g. the reliability, security and error reporting of a response may not be the same as for a request – the P-Mode will be divided into "legs". Each user message label in an ebMS MEP is associated with a P-Mode leg. Each P-Mode leg has a full set of parameters of the six categories above (except for General Parameters), even though in many cases parameters will have same value across the MEP legs. Signal messages that implement transport channel bindings (such as PullRequest) are also controlled by the same categories of parameters, except for BusinessInfo group.

The following figure illustrates the general structure of a P-Mode for a Two-Way/Push-and-Pull MEP; for example, a PurchaseOrder business transaction that includes the pair PurchaseOrderRequest + PurchaseOrderConfirm. Its binding channel is "Push-and-Pull" e.g. because the buyer cannot receive incoming requests.

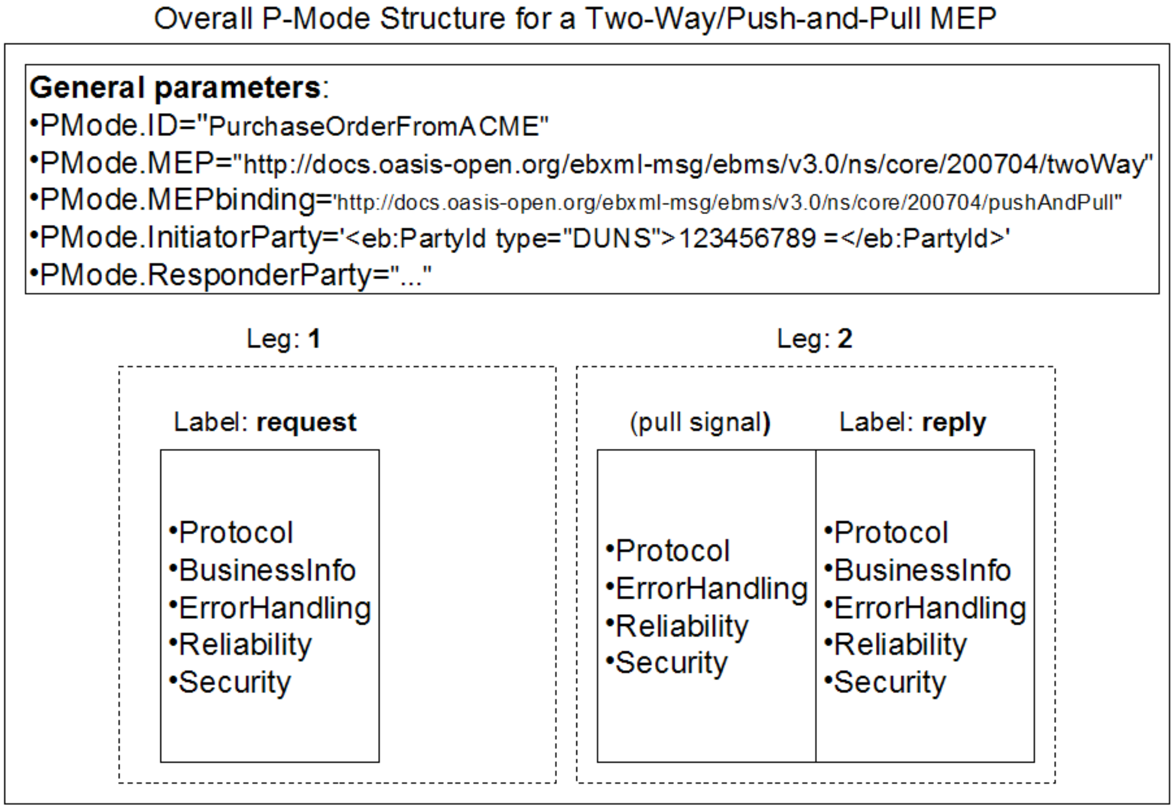
Overall P-Mode Structure for a Two-Way/Push-and-Pull MEP
General parameters:
•PMode.ID="PurchaseOrderFromACME"
•PMode.MEP="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/twoWay"
•PMode.MEPbinding="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pushAndPull"
•PMode.InitiatorParty='<eb:PartyId type="DUNS">123456789 =</eb:PartyId>'
•PMode.ResponderParty="..."
Leg: 1
Label: request
•Protocol
•BusinessInfo
•ErrorHandling
•Reliability
•Security
Leg: 2
(pull signal)
•Protocol
•ErrorHandling
•Reliability
•Security
Label: reply
•Protocol
•BusinessInfo
•ErrorHandling
•Reliability
•Security

Figure 15: P-Mode Structure for Two-Way/Push-and-Pull MEP

In the above illustration, each leg of the MEP may have different P-Mode parameters, although in many cases these parameters will be identical from one leg to the other. Because the P-Mode specifies the MEP transport channel binding, it may also specify a set of parameters for the Pull signal, which may be subject to specific requirements (reliability, security/authorization).

> Note:
> In general, a Pull signal cannot be precisely targeted to a particular MEP, but instead to an MPC. For this reason, all Pull signals for a particular MPC will usually share similar P-Mode parameters.

D.2.1. Notation

Consider a PurchaseOrder business transaction as defined above.

- The P-Mode associated with this type of transaction between two partners, may be called: **PurchaseOrder.PMode**.
- An index notation is used to identify the legs of an MEP. The part of the P-Mode that relates to Leg 1 of the PurchaseOrder MEP ("request" label), will be called **PurchaseOrder.PMode[request]**. .A number representing the occurrence order may be used instead of the leg label, e.g. **PurchaseOrder.PMode[1]**. This is appropriate for a MEP in which the legs are strictly serialized over time.
- In case there are two sets of P-Mode parameters associated with a leg, as for the pulled "reply", the part of the P-Mode that concerns the user message in leg 2 is noted: **PurchaseOrder.PMode[2][u]**, while the part of the P-Mode that concerns the (pull) signal message in leg 2 is noted: **PurchaseOrder.PMode[2][s]**.

D.3. Processing Mode Parameters

P-Mode parameters define how a message should be processed. These parameters either define elements that are expected to be found in the message, or processing behavior expected for this message (e.g. level of reliability, error reporting). Every parameter in this section does not need to be given a value when defining   a P-Mode. In such a case, either the corresponding header element can take any value for a message processed under this P-Mode, or the MSH behavior this parameter controls is not constrained by the P-Mode. It is also possible to associate multiple authorized values (or a range of values) with a parameter in a P-Mode (e.g. multiple MPC values).

D.3.1. General P-Mode Parameters

The general P-Mode parameters (i.e. not specific to any single message in the MEP) are:

- **PMode.ID**: (optional) The identifier for the P-Mode, e.g. the name of the business transaction: PurchaseOrderFromACME. This identifier is user-defined and optional, for the convenience of P-Mode management. It must uniquely identify the P-Mode among all P-Modes deployed on the same MSH, and may be absent if the P-Mode is identified by other means, e.g. embedded in a larger structure that is itself identified, or has parameter values distinct from other P-Modes used on the same MSH. If the ID is specified, the AgreementRef/@pmode attribute value is also expected to be set in associated messages.
- **PMode.Agreement**: The reference to the agreement governing this message exchange (maps to eb:AgreementRef in message header).
- **PMode.MEP**: The type of ebMS MEP associated with this P-Mode. The value must be a URI, e.g: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay`.
- **PMode.MEPbinding**: The transport channel binding assigned to the MEP (push, pull, sync, push-and-push, push-and-pull, pull-and-push, pull-and-pull, …). The value must be a URI, e.g: `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push`.
- **PMode.Initiator.Party**: 1.(PMode.Initiator and its subelements are optional if PMode.Responder is present.) Qualifies the party initiating the MEP (see Section 2.2.3). A user message initiating an MEP instance under this P-Mode must have its eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From element contain the same PartyId elements as the PartyId elements defined in this parameter. Any user message sent to the initiator must have its eb:PartyInfo/eb:To map to or be compatible with this parameter.
- **PMode.Initiator.Role**: Name of the role assumed by the party sending the first message of this MEP. Either the message element `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role` or the element `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To/eb:Role` of each message in this MEP must have this value, depending on the direction of message transfer.
- **PMode.Initiator.Authorization.username** and **PMode.Initiator.Authorization.password:** Describe authorization information for messages sent by Initiator. These parameters need to be matched by a wsse:UsernameToken element in a message (in a security header only intended for authorization) for this message to be processed successfully on receiver side – here by Responder MSH.
- **PMode.Responder.Party**: (PMode.Responder and its subelements are optional if PMode.Initiator is present.) Qualifies the party responding to the initiator party in this MEP. Any user message sent to the responder must have its eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To element contain the same PartyId elements as the PartyId elements defined in this parameter.
- **PMode.Responder.Role**: Name of the role assumed by the party receiving the first message of this MEP. Either the message element `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role` or the element `eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To/eb:Role` of each message in this MEP must have this value, depending on the direction of message transfer.
- **PMode.Responder.Authorization.username** and **PMode.Responder.Authorization.password:** Describe authorization information for messages sent by Responder. These parameters need to be matched by a wsse:UsernameToken element in a message (in a security header only intended for authorization) for this message to be processed successfully on receiver side – here by Initiator MSH.

The P-Mode parameters that are specific to a P-Mode leg (here, associated with leg 1 of an MEP) are grouped into five categories: Protocol, BusinessInfo, ErrorHandling, Reliability, and Security:

D.3.2. PMode[1].Protocol

- **PMode[1].Protocol.Address**: the value of this parameter represents the address (endpoint URL) of the Receiver MSH (or Receiver Party) to which Messages under this P-Mode leg are to be sent. Note that a URL generally determines the transport protocol (for example, if the endpoint is an email address, then the transport protocol must be SMTP; if the address scheme is "http", then the transport protocol must be HTTP).
- **PMode[1].Protocol.SOAPVersion**: this parameter indicates the SOAP version to be used (1.1 or 1.2). In some implementations, this parameter may be constrained by the implementation, and not set by users.

D.3.3. PMode[1].BusinessInfo

Note:
This set of parameters only applies to user messages.

- **PMode[1].BusinessInfo.Service**: Name of the service to which the User message is intended to be delivered. Its content should map to the element `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service`.
- **PMode[1].BusinessInfo.Action**: Name of the action the User message is intended to invoke. Its content should map to the element `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action`
- **PMode[1].BusinessInfo.Properties[]**: The value of this parameter is a list of properties. A property is a data structure that consists of four values: the property name, which can be used as an identifier of the property (e.g. a required property named "messagetype" can be noted as: Properties[messagetype].required="true"); the property description; the property data type; and a Boolean value, indicating whether the property is expected or optional, within the User message. This parameter controls the contents of the element `eb:Messaging/eb:UserMessage/eb:MessageProperties`.
- **PMode[1].BusinessInfo.PayloadProfile[]**: This parameter allows for specifying some constraint or profile on the payload. It specifies a list of payload parts. A payload part is a data structure that consists of five properties: name (or Content-ID) that is the part identifier, and can be used as an index in the notation PayloadProfile[]; MIME data type (text/xml, application/pdf, etc.); name of the applicable XML Schema file if the MIME data type is text/xml; maximum size in kilobytes; and a Boolean value indicating whether the part is expected or optional, within the User message. The message payload(s) must match this profile.
- **PMode[1].BusinessInfo.PayloadProfile.maxSize**:: This parameter allows for specifying a maximum size in kilobytes for the entire payload, i.e. for the total of all payload parts.
- **PMode[1].BusinessInfo.MPC**: The value of this parameter is the identifier of the MPC (Message Partition Channel) to which the message is assigned. It maps to the attribute `eb:Messaging/eb:UserMessage/@mpc`.

D.3.4. PMode[1].ErrorHandling

Note:
This P-Mode group concerns errors generated by the reception of the message (for either a User message or a Signal message, unless indicated otherwise) sent over leg 1 of the MEP.

- **PMode[1].ErrorHandling.Report.SenderErrorsTo**: This parameter indicates the address, or comma-separated list of addresses, to which to send ebMS errors generated by the MSH that was trying to send the message in error.
- **PMode[1].ErrorHandling.Report.ReceiverErrorsTo**: This parameter indicates the address, or comma-separated list of addresses, to which to send ebMS errors generated by the MSH that receives the message in error; e.g. this may be the address of the MSH sending the message in error.

- **PMode[1].ErrorHandling.Report.AsResponse**: This Boolean parameter indicates whether (if "true") errors generated from receiving a message in error are sent over the back-channel of the underlying protocol associated with the message in error, or not.
- **PMode[1].ErrorHandling.Report.ProcessErrorNotifyConsumer**: This Boolean parameter indicates whether (if "true") the Consumer (application/party) of a User Message matching this P-Mode should be notified when an error occurs in the Receiving MSH, during processing of the received User message.
- **PMode[1].ErrorHandling.Report.ProcessErrorNotifyProducer**: This Boolean parameter indicates whether (if "true") the Producer (application/party) of a User Message matching this P-Mode should be notified when an error occurs in the Sending MSH, during processing of the User Message to be sent.
- **PMode[1].ErrorHandling.Report.DeliveryFailuresNotifyProducer**: This Boolean parameter indicates whether (if "true") the Producer (application/party) of a User Message matching this P-Mode must always be notified when the delivery to Consumer failed, or whether (if "false"), in some cases, it is sufficient to notify the Consumer only (Report.ProcessErrorNotifyConsumer="true"). This assumes that Reliability.AtLeastOnce.Contract is "true". This also assumes that the Sending MSH implementation has the ability to determine or to be made aware of all cases of non-delivery that occur after the message has been received by the Receiving MSH.

D.3.5. PMode[1].Reliability

- **PMode[1].Reliability.AtLeastOnce.Contract**: If "true", this Boolean parameter indicates that the "At-Least-Once" reliability contract (see Section 8.2.2) is to be used between MSH and Consumer (Guaranteed Delivery). It also indicates that this contract applies to ebMS signals (see Section 8.2.1) – e.g. PullRequest – between the receiving reliability module and the next MSH component (e.g. RM-Consumer).
- **PMode[1].Reliability.AtLeastOnce.Contract.AckOnDelivery**: This Boolean parameter indicates the semantics of acknowledgments that are generated by the reliability module. It is usually constrained by the implementation and not set by users. For User messages: if "true", the acknowledgment is only sent after the message has been delivered by the MSH to the Consumer entity (see Case 2 in Section 8.2.4). If "false", the only guarantee for the sender when receiving an acknowledgment is that the User message has been well received (see Case 1 or 3 in Section 8.2.4), and made available for further processing within the MSH. For Signal messages – e.g. PullRequest: if "true", indicates that Signal messages are acknowledged only if delivered (see Section 8.2.1) from the receiving reliability module to the next MSH component (Case 3 in Section 8.2.4), i.e. to the RM-Consumer (see 8.1.2). If "false", the message acknowledgment only guarantees receipt of the signal (Case 1 in Section 8.2.4).
- **PMode[1].Reliability.AtLeastOnce.Contract.AcksTo**: This parameter is a URI that specifies where acknowledgments are to be sent. It may contain an anonymous URI (defined in WS-Addressing). If absent, acknowledgments are to be sent to the same URI associated with the MSH sending messages reliably.
- **PMode[1].Reliability.AtLeastOnce.Contract.AckResponse**: This Boolean is true when an Acknowledgment must be sent, for a response that is sent reliably.
- **PMode[1].Reliability.AtLeastOnce.ReplyPattern**: This parameter indicates whether a reliability acknowledgment is to be sent as a callback, synchronously in the response (back-channel of underlying protocol), or as response of separate ack pulling. Three values are possible for this parameter, when using WS-Reliability: "Response", "Callback", or "Poll".
- **PMode[1].Reliability.AtMostOnce.Contract**: If "true", this Boolean parameter indicates that "At-Most-Once" (or duplicate elimination) should be enforced when receiving a message. The contract is for delivery between MSH and Consumer for User messages (see Section 8.2.2), and between reliability module and next MSH component for Signal messages (see Section 8.2.1).
- **PMode[1].Reliability.InOrder.Contract**: If "true", this Boolean parameter indicates that this message is part of an ordered sequence. It only concerns User messages (delivery contract between MSH and Consumer application, see Section 8.2.2).
- **PMode[1].Reliability.StartGroup**: This parameter is a Boolean that may be used to indicate if messages matching this P-Mode must be associated with a new reliability group or sequence. For example, a particular Service and Action may have the application semantics of initiating a new

ordered sequence of messages.

- **PMode[1].Reliability.Correlation**: This parameter tells how to correlate a message matching this P-Mode with an existing reliability group or sequence. It is a comma-separated list of XPath elements relative to the eb:Messaging header. Each one of these XPaths identifies an element or attribute inside eb:UserMessage or eb:SignalMessage, and may include predicates. For example, "eb:UserMessage/eb:CollaborationInfo/eb:ConversationId, eb:UserMessage/eb:MessageProperties/eb:Property[@name=\"ProcessInstance\"] will correlate all messages that share the same ConversationId and have the same value for the message property named "ProcessInstance". In case there is no ongoing group or sequence associated with the values in Reliability.Correlation for a message under this P-Mode, then a new group/sequence is started.
- **PMode[1].Reliability.TerminateGroup**: This parameter is a Boolean value that may be used to indicate if messages matching this P-Mode must cause the closure of the reliability group or sequence with which they correlate.

D.3.6. PMode[1].Security

- **PMode[1].Security.WSSVersion**: This parameter has two possible values, 1.0 and 1.1. The value of this parameter represents the version of WS-Security to be used.
- **PMode[1].Security.X509.Sign**: The value of this parameter is a list of the names of XML elements (inside the SOAP envelope) that should be signed, as well as whether or not attachments should also be signed. The list is represented in two sublists that extend this parameter: **Sign.Element[]** and **Sign.Attachment[]**. An element within the Element[] list could be specified either by its XML name or by its qualified name (its XML name and the namespace to which it belongs). An element within the Attachment[] list is identified by the Content-Id.
- **PMode[1].Security.X509.Signature.Certificate**: The value of this parameter identifies the public certificate to use when verifying signed data.
- **PMode[1].Security.X509.Signature.HashFunction**: The value of this parameter identifies the algorithm that is used to compute the digest of the message being signed. The definitions for these values are in the [XMLDSIG] specification.
- **PMode[1].Security.X509.Signature.Algorithm**: The value of this parameter identifies the algorithm that is used to compute the value of the digital signature. The definitions for these values are found in the [XMLDSIG] or [XMLENC] specifications.
- **PMode[1].Security. X509.Encryption.Encrypt**: The value of this parameter lists the names of XML elements(inside the SOAP envelope) that should be encrypted, as well as whether or not attachments should also be encrypted. The list is represented in two sublists that extend this parameter: **Encrypt.Element[]** and **Encrypt.Attachment[]**. An element within these lists is identified as in **Security.X509.Sign** lists**.**
- **PMode[1].Security.X509.Encryption.Certificate**: The value of this parameter identifies the public certificate to use when encrypting data.
- **PMode[1].Security.X509.Encryption.Algorithm**: The value of this parameter identifies the encryption algorithm to be used. The definitions for these values are found in the [XMLENC] specification.
- **PMode[1].Security.X509.Encryption.MinimumStrength**: The integer value of this parameter describes the effective strength the encryption algorithm MUST provide in terms of "effective" or random bits. The value is less than the key length in bits when check bits are used in the key. So, for example the 8 check bits of a 64-bit DES key would not be included in the count, and to require a minimum strength the same as supplied by DES would be reported by setting MinimumStrength to 56.
- **PMode[1].Security.UsernameToken.username**: The value of this parameter is the username to include in a WSS Username Token.
- **PMode[1].Security.UsernameToken.password**: The value of this parameter is the password to use inside a WSS Username Token.
- **PMode[1].Security.UsernameToken.Digest**: The Boolean value of this parameter indicates whether a password digest should be included in the WSS UsernameToken element.
- **PMode[1].Security.UsernameToken.Nonce**: The Boolean value of this parameter indicates whether the WSS UsernameToken element should contain a Nonce element.
- **PMode[1].Security.UsernameToken.Created**: The Boolean value of this parameter indicates

whether the WSS UsernameToken element should have a Created timestamp element.

- **PMode[1].Security.PModeAuthorize**: The Boolean value of this parameter indicates whether messages on this MEP leg must be authorized for processing under this P-Mode. If the parameter is "true" this implies that either PMode.Responder.Authorization.{username/password}, if the message is sent by Responder, or PMode.Initiator.Authorization if the message is sent by Initiator, must be used for this purpose, as specified in Section 7.10. For example, when set to "true" for a PullRequest message sent by the Initiator, the pulling will only be authorized over the MPC indicated by this Pull signal if (a) the MPC is the same as specified in the P-Mode leg for the pulled message, and (b) the signal contains the right credentials (e.g. username/password).
- **PMode[1].Security.SendReceipt**: The Boolean value of this parameter indicates whether a signed receipt (Receipt ebMS signal) containing a digest of the message must be sent back.
- **PMode[1].Security.SendReceipt.ReplyPattern**: This parameter indicates whether the Receipt signal is to be sent as a callback (value "callback"), or synchronously in the back-channel response (value "response"). If not present, any pattern may be used.

APPENDIX E. P-Mode Values and ebMS MEP Bindings

This section describes the effect that various Processing Mode values have on the binding of each ebMS MEP to HTTP.

E.1. P-Mode Values and the One-Way/Push MEP

The following table illustrates how the One-Way/Push MEP binds to HTTP, depending on the values of P-Mode parameters that affect message content.

No combination of P-Mode values other than those listed below are expected to be used. Valid combinations not explicitly represented in the table below are mentioned in "notes" as variants of the most common ones.

MEP: One-way / Push	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6
Reliability.AtLeastOnce.Contract:	False	False	True	True	True	True
Reliability.AtLeastOnce.ReplyPattern	N/A	N/A	Response	Response	Callback	Callback
ErrorHandling.Report.AsResponse	False	True	False	True	False	True
HTTP Request (pushed message)	UserMessage	UserMessage	UserMessage + RM header (with AckRequested element if WS-Reliability)	UserMessage + RM header (see case 3)	UserMessage + RM header (see case 3)	UserMessage + RM header (see case 3)
HTTP Response	No SOAP envelope except if SOAP Fault.[a]	No SOAP envelope except if ebMS error on the UserMessage: an ebMS header for Error SignalMessage.[a],[b]	SOAP header with RM Ack[a],[c]	SOAP header with RM Ack[c], plus an ebMS header for Error SignalMessage, if any.[a],[b]	Same as Case 1	Same as Case 2

[a] A SOAP Fault may be included if the request was in error. This Fault is combined with an ebMS error message (eb:Messaging/eb:SignalMessage/eb:Error) unless it is generated by the Security or Reliability module.

[b] The ebMS error message may or may not be combined with a SOAP Fault, depending on its severity.

[c] Acks may be grouped so that an Ack is not sent back for every UserMessage.

E.2. P-Mode Values and the One-Way/Pull MEP

The following table illustrates how the One-Way/Pull MEP binds to HTTP, depending on the values of P-Mode parameters that affect message content.

No combination of P-Mode values other than those listed below are expected to be used. Valid combinations not explicitly represented in the table below are mentioned in "notes" as variants of the most common ones.

MEP: **One-way / Pull**	**Case 1**	**Case 2**	**Case 3**
[1][s].Reliability.AtLeastOnce.Contract:	False	True	True
[1][s].Reliability.AtLeastOnce.ReplyPattern	N/A	Response	Response
[1][s].ErrorHandling.Report.AsResponse	True[d]	True	True
HTTP Request (PullRequest signal)	PullRequest signal	PullRequest signal + RM header (with AckRequested element if WS-Reliability)	PullRequest signal + RM header (see case 2)
[1][u].Reliability.AtLeastOnce.Contract:	False	True[e]	True[e]
[1][u].Reliability.AtLeastOnce.ReplyPattern	N/A	None (in case no ack required for pulled message)	Callback (the pulled message must be acknowledged on a separate MEP)
HTTP Response (pulled message)	Pulled UserMessage[f]	SOAP header with RM Ack[g] of pull signal + Pulled UserMessage[f]	SOAP header with RM Ack[g] of pull signal + Pulled UserMessage[f]
A second HTTP Request in same direction as previous HTTP Request (For example, the next PullRequest signal.)	N/A	N/A	RM header containing Ack + possibly other SOAP headers/body.

[d] A possible case where value is False – all other values being same - is not reported here.

[e] A possible case where the pulled message is not sent reliably while the pull signal is, would be of little relevance – not detailed here. Conversely, reliable sending of the pulled message requires reliable sending of the pull signal.

[f] or else an ebMS error (with or without SOAP Fault) if the Pull signal had an error.

[g] Acks may be grouped so that an Ack is not sent back for every UserMessage.

E.3. P-Mode Values and the Two-Way/Sync MEP

The following table illustrates how the Two-Way/Sync MEP binds to HTTP, depending on the values of P-Mode parameters that affect message content.

No combination of P-Mode values other than those listed below are expected to be used. Valid combinations not explicitly represented in the table below are mentioned in "notes" as variants of the most common ones.

MEP: **Two-way / Sync**	**Case 1**	**Case 2**	**Case 3**	**Case 4**
[1].Reliability.AtLeastOnce.Contract:	False	True	True	True
[1].Reliability.AtLeastOnce.ReplyPattern	N/A	Response[h]	Response[i]	Response[i]
[1].ErrorHandling.Report.AsResponse	True[j]	True[j]	True[k]	True[k]
HTTP Request (request message)	UserMessage (request)	UserMessage + RM header (with AckRequested element if WS-Reliability)	UserMessage + RM header (see case 2)	UserMessage + RM header (see case 2)
[2].Reliability.AtLeastOnce.Contract:	False	False	True[k]	True[k]
[2].Reliability.AtLeastOnce.ReplyPattern	N/A	N/A	None (in case no ack required)	callback
HTTP Response (reply message)	UserMessage (reply)[l]	SOAP header with RM Ack[m] of request + UserMessage reply[l]	SOAP header with RM Ack[m] of request + UserMessage reply[l]	SOAP header with RM Ack[m] of request + UserMessage reply[l]
HTTP Request in same direction as previous HTTP Request (not belonging to this MEP)	N/A	N/A	N/A	RM header containing Ack + possibly other SOAP headers/body

[h] A possible case where the reply pattern is callback instead of response is not reported here.

[i] the pattern for acknowledging the request must be "response" in case the reply must also be sent reliably. In that case, Acks should not be grouped.

[j] A possible case where value is False – all other values being same - is not reported here.

[k] The reply may not be sent reliably if the request is not.

[l] or else an ebMS error (with or without SOAP Fault) if the request had an error.

[m]Acks may be grouped so that an Ack is not sent back for every UserMessage.

APPENDIX F. Compatibility Mapping to ebMS 2.0

F.1. Objectives and Approach

The reliance in V3 on recent SOAP-based specifications that cover security and reliability, could not be reconciled with preserving seamless backward compatibility with ebMS V2. In order to provide backward compatibility guidelines for implementations, this section defines mapping rules between V2 and V3 that establish an equivalence of header structures and processing features. These mapping rules define a *compatibility mapping*.

The primary intent of the compatibility mapping rules is to define a semantic bridge between V2 and V3 artifacts and features. Although these rules may appear like translation rules, e.g. for converting a V2 header into a V3 header, it is clear that some backward-compatible V3 implementations will not use them that way. Processing both V2 and V3 may be achieved without run-time conversion of messages or of features from one version to the other. For example, a messaging gateway may support separately both versions, and deal with two separate processing flows that would join only at the application interface level. Even in such a case, the rules are useful to define an equivalence between V2 and V3 processing flows and their configuration (quality of service, error handling, etc.), as well as to define how the business header elements of one version map to the other version. These rules help in interpreting agreements (e.g. CPA) that have initially been defined for one version, so that they can be used or rewritten for the other version.

A conformance profile that requires backward compatibility is defined in a companion document ("ebMS V3 Conformance Profiles"). Implementations or products that conform to this backward-compatibility profile must be able to:

- receive and process ebMS 2 messages (with features within "core" and "reliable messaging" modules).
- generate and send ebMS 2 messages (with features within "core" and "reliable messaging" modules).

F.2. Compatibility Mapping Rules

The compatibility mapping (CM) does not necessarily cover all feature allowed by ebMS 2, but a significant subset of these. It is made of mapping rules that are grouped into mapping modules (CM1 to CM6) that are briefly described below :

CM rules:

- CM1: Header mapping rules
- CM2: Payload mapping rules
- CM3: Reliability mapping rules
- CM4: MEP mapping rules
- CM5: Signal mapping rules
- CM6: Processing mode mapping rules

Note: For a concise notation, the namespace prefixes eb2 and eb3 below respectively qualify V2 and V3 message artifacts.

F.2.1. (CM1) Header Mapping Rules

Although the ebMS headers from V2 and from V3 do not share the same XML schema, there is a large overlap between their elements. Only eb2:TimeToLive has no counterpart in the eb3 header, although it has a counterpart in a reliability header based on WS-Reliability.

F.2.1.1. Rule CM1-a: Mapping General Message Information

eb2:MessageHeader/eb2:MessageData element maps to eb3/Messaging/eb3:MessageInfo, along with their contained elements (Timestamp, MessageId, RefToMessageId).

Depending on its usage, the optional eb2:TimeToLive would map differently to an eb3 header. In case it has some application semantics (e.g. validity period of the enclosed business document), such a value can be added in V3 as eb3:Messageproperties/eb3:property/@name="timetolive". However, it has no MSH semantics in V3, unlike in V2 where it controls delivery. Implementing similar semantics would be done as an extension to V3. In case eb2:TimeToLive is used as a reliability feature (e.g. expected maximum time during which reliability mechanisms are expected to operate on the message before declaring failure) then it should map to message ExpiryTime (see rule CM3-c).

F.2.1.2. Rule CM1-b: Mapping Party Information

eb2:MessageHeader/eb2:From maps to eb3:PartyInfo/eb3:From, along with their sub-elements.

Similarly, eb2:MessageHeader/eb2:To maps to eb3:PartyInfo/eb3:To, along with their sub-elements.

F.2.1.3. Rule CM1-c: Mapping Collaboration Information

eb2:ConversationId, eb2:Service, eb2:Action respectively map to eb3:CollaborationInfo/eb3:ConversationId, eb3:CollaborationInfo/eb3:Service and eb3:CollaborationInfo/eb3:Action.

F.2.1.4. Rule CM1-d: Mapping Agreement Reference

eb2:MessageHeader/eb2:CPAId maps to eb3:CollaborationInfo/eb3:AgreementRef.

F.2.2. (CM2) Payload Mapping Rules

F.2.2.1. Rule CM2-a: Mapping Attachments

Every attachment (MIME part) in V2 maps to a similar attachment in V3. The SOAP Body should not be used in V3. If a V3 message that must map to a V2 message has a non-empty SOAP Body, the child XML document must be mapped to a separate MIME part in V2.

F.2.3. (CM3) Reliability Mapping Rules

These rules define how some V2 header elements map to a separate reliability header in V3, and vice-versa. When the reliability quality of service is not apparent in the V3 reliability header (e.g. in case V3 uses WS-ReliableMessaging protocol), these rules rely on the P-Mode.Reliability parameters to determine the reliability elements in ebMS2 header.

F.2.3.1. Rule CM3-a: Acknowledgments

V2: AckRequested element maps to: in V3, wsrm:Request/AckRequested (if using WS-Reliability), optional wsrm:AckRequested header if using WS-ReliableMessaging (not necessary to get acknowledgments).

V2: Acknowledgment element maps to: in V3, wsrm:Response/SequenceReplies (if using WS-Reliability), wsrm:SequenceAcknowledgment if using WS-ReliableMessaging.

> Note:
> The meaning of acknowledgments may be different in V2 and V3. See Section 8.2.4 for the options in acknowledgment semantics, depending on which reliability module is used. In V2, the baseline semantics is "on receipt": the message has been safely stored in persistent storage or delivered to the application interface. In V3, the recommended semantics is: the message has been delivered to the application. It may however be similar to V2 semantics depending on the implementation (e.g. when using WS-ReliableMessaging). In V3 the P-Mode parameter Reliability.AtLeastOnce.AckOnDelivery specifies this semantics which in general depends on the implementation: when "false", it is similar to V2 (on receipt).

F.2.3.2. Rule CM3-b: Reliability Contracts

The reliability contracts At-Least-Once delivery, At-Most-Once delivery, In-Order delivery, that in V3 are specified in the P-Mode, and also in the message header in case WS-Reliability is used, respectively map to V2 header elements: eb2:AckRequested, eb2:DuplicateElimination, eb2:MessageOrder.

Any of the above reliability contracts requires the use of a reliable messaging module in V3, e.g. an implementation of WS-Reliability or of WS-ReliableMessaging.

The delivery failure notification in V2 (always required for non-acknowledged messages) is supported by WS-Reliability and therefore by V3 using WS-Reliability. Such failure notification is not explicitly mandated by WS-ReliableMessaging, or could take place on either side. In order to achieve the same notification policy as in V2, when used in V3 an implementation of WS-ReliableMessaging must be extended with the same notification capability.

> Note:
> The conditions under which delivery failure is notified to the From Party (in eb2) or message Producer (in eb3) may be different.

F.2.3.3. Rule CM3-c: Duplicate Elimination

eb2:MessageHeader/eb2:DuplicateElimination maps to wsrm:Request/wsrm:DuplicateElimination in WS-Reliability. It maps to the AtMostOnce delivery assurance definition in WS-ReliableMessaging, assuming an implementation of WS-ReliableMessaging that supports this delivery assurance.

F.2.3.4. Rule CM3-d: Use of Sequences and Sequence Numbers

An eb2 message that contains either AckRequested or DuplicateElimination or both, and no eb2:MessageOrder, may map to a V3 message (when using WS-Reliability) with no wsrm:SequenceNum – only a wsrm:MessageId/@groupId value, which is unique for every such message.

> Note:
> The elements that identify a message sent reliably in V3 (wsrm:SequenceNum, wsrm:MessageId/@groupId in WS-Reliability, or /wsrm:Sequence/wsrm:MessageNumber in WS-ReliableMessaging) do NOT map to the ebMS message ID element (i.e. eb2:MessageData/eb2:MessageId in V2, and eb3:MessageInfo/eb3:MessageId in V3).

F.2.3.5. Rule CM3-e: Message Ordering

In case message ordering is required:

eb2:MessageOrder maps to wsrm:Request/wsrm:MessageOrder.

eb2:SequenceNumber maps to wsrm:Request/wsrm:SequenceNum (with WS-Reliability).

The scope of a message sequence (and of the message ordering contract) is determined by eb2:ConversationId in V2, and by MessageId/@groupId in V3; i.e. sequence numbers must be unique within this scope.

The feature maps to the InOrder delivery assurance definition in WS-ReliableMessaging, assuming an implementation of WS-ReliableMessaging that supports this delivery assurance.

F.2.3.6. Rule CM3-f: Expiration Timeout

In case eb2:MessageHeader/eb2:MessageData/eb2:TimeToLive is used for expressing the maximum time during which reliability mechanisms are required to handle the message, it maps to wsrm:Request/wsrm:ExpiryTime.

F.2.4. (CM4) MEP Mapping Rules

Defines how V2 header elements that control the MEP in use and its mapping to the underlying protocol, map into V3 and vice versa. Also defines how CPA elements that control ebMS V2 MEPs map to P-Mode parameter and vice-versa.

F.2.4.1. Rule CM4-a: One-Way/Push With No Signals

In V3, this MEP, with no ebMS signal and no reliability acknowledgments on the response (back-channel), will map to a V2 message with no SyncReply element in eb2 header. RefToMessageId must not be used in the V3 message (it has a strict MEP semantics). The agreements map as follows:

V2 (CPA): syncReplyMode=none.

V3 (P-Mode): PMode.MEP="One-way", PMode.MEPbinding="push", PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern must NOT be "Response".

F.2.4.2. Rule CM4-b: One-Way/Push With Signals

One-Way / Push in V3, with ebMS signal and reliability acknowledgments on the response (back-channel), will map to a V2 message with SyncReply element in eb2 header. RefToMessageId must not be used in the V3 message (it has a strict MEP semantics). The agreements map as follows:

V2 (CPA): syncReplyMode= mshSignalsOnly.

V3 (P-Mode): PMode.MEP="One-way", PMode.MEPbinding="push", PMode.ErrorHandling.Report.AsResponse="true", PMode.Reliability.ReplyPattern="Response".

F.2.4.3. Rule CM4-c: Two-Way/Sync With No Signals

In V3, this MEP, with no ebMS signal and no reliability acknowledgments on the response (back-channel), will map to a V2 message (1st leg) with SyncReply element in eb2 header. In both versions, the response message refers to the request (leg 1) using RefToMessageId. The agreements map as follows:

V2 (CPA): (leg 1) syncReplyMode= responseOnly.

V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="sync", PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern may NOT be "Response".

F.2.4.4. Rule CM4-d: Two-Way/Sync With Signals

In V3, this MEP, with ebMS signal and reliability acknowledgments on the response (back-channel), will map to a V2 message (1st leg) with SyncReply element in eb2 header. In both versions, the response message refers to the request (leg 1) using RefToMessageId. The agreements map as follows:

V2 (CPA): (leg 1) syncReplyMode= signalsAndResponse

V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="sync", PMode.ErrorHandling.Report.AsResponse="true". PMode.Reliability.ReplyPattern ="Response".

F.2.4.5. Rule CM4-e: Two-Way/Push-and-Push

In V3, this MEP will map to an exchange of two messages in V2, where the second message refers to the first one using RefToMessageId (as in V3). The agreements map as follows:

Option 1: (signals may be sent back on underlying response)
V2 (CPA): (leg 1 and leg 2) syncReplyMode= mshSignalsOnly.
V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="Push-and-Push".
PMode.ErrorHandling.Report.AsResponse="true". PMode.Reliability.ReplyPattern="Response".

Option 2: (signals may NOT be sent back on underlying response)
V2 (CPA): (leg 1 and leg 2) syncReplyMode= none.
V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="Push-and-Push".
PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern different from "Response".

F.2.5. (CM5) Signal Mapping Rules

F.2.5.1. Rule CM5-a: Error Metadata Mapping

The metadata mapping of the Error elements in V2 and V3 is as follows. In some cases the semantics is close though not exactly same.

(a) Cases where a straight mapping exist from V2 to V3:

1. V2: Error/@severity (warning, error) maps to V3: eb:Error/@severity (respectively: warning, failure)
2. V2: Error/@codeContext maps to V3: eb:Error/@origin
3. V2: Error/@errorCode maps to V3: eb:Error/shortDescription
4. V2: Error/@location maps to V3: eb:Error/ErrorDetail
5. V2: Error/Description maps to V3: eb:Error/Description
6. V2: MessageData/RefToMessageId maps to V3: eb:Error/@refToMessageInError

(b) Cases where error element in V2 has no specified counterpart in V3:

1. V2: Error/@id. In V3 would map to: XML Id attribute.

(c) Cases where error element in V3 has no specified counterpart in V2:

1. V3: eb:Error/@errorCode
2. V3: eb:Error/@category

F.2.5.2. Rule CM5-b: Error Value Mapping

The value-equivalence between Errors in V2 and V3 is as follows, based on the semantics of these errors:

Note: the severity levels may not map in some cases, meaning that processing may continue in V3 while aborting in V2.

(a) Cases where a straight mapping exist from V2 to V3:

1. V2: ValueNotRecognized maps to V3: ValueNotRecognized
2. V2: NotSupported maps to V3: FeatureNotSupported
3. V2: DeliveryFailure maps to V3: DeliveryFailure
4. V2: MimeProblem maps to V3: MimeInconsistency

(b) Cases where a case by case mapping exist from V2 to V3:

1. V2: Inconsistent may map to V3: ValueInconsistent, in some cases InvalidHeader
2. V2: SecurityFailure maps to V3: FailedAuthentication or FailedDecryption
3. V2: OtherXML may map to V3: Other
4. V2: Unknown maps to (in most cases) V3: Other

(c) Cases where error value in V2 has no counterpart in V3:

1. V2: TimeToLiveExpired: no counterpart (not relevant).

(d) Cases where error value in V3 has no counterpart in V2:

1. V3: ConnectionFailure,
2. V3: EmptyMessagePartitionChannel
3. V3: ProcessingModeMismatch
4. V3: DysfunctionalReliability

F.2.5.3. Rule CM5-c: Ping and Pong Services

(a) Ping Service:

1. V2: Service element: urn:oasis:names:tc:ebxml-msg:service, and Action element

containing: Ping.

2. V3: Service element: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service, and Action element: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test

(b) Pong Service:

No corresponding Pong service in V3 Core specification. This feature may be defined in a forthcoming Part 2 (Advanced Features).

F.2.6. (CM6) Processing Mode Mapping Rules

These mapping rules, to be specified in a separate white paper, will define how the messaging subset of an existing CPA instance in V2 maps to a V3 P-Mode. They also provide guidance on how to represent a P-Mode with a CPA and related extensions.

APPENDIX G. Conformance

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 6).

APPENDIX H. Acknowledgments

The OASIS ebXML Messaging Services Technical Committee would like to acknowledge the contributions of its committee members, who at the time of publication were:

Hamid Ben Malek, Fujitsu Software <hbenmalek@us.fujitsu.com>
Jacques Durand, Fujitsu Software <jdurand@us.fujitsu.com>
Ric Emery, Axway Inc. <remery@us.axway.com>
Kazunori Iwasa, Fujitsu Limited <kiwasa@jp.fujitsu.com>
Ian Jones, British Telecommunications plc <ian.c.jones@bt.com>
Rajashekar Kailar, Centers for Disease Control and Prevention <kailar@bnetal.com>
Dale Moberg, Axway Inc. <dmoberg@us.axway.com>
Sacha Schlegel, Individual <sacha@schlegel.li>
Pete Wenzel, Sun Microsystems <pete.wenzel@sun.com>

In addition, the following former Technical Committee members contributed to this specification:

Doug Bunting, Sun Microsystems <doug.bunting@sun.com>
Matthew MacKenzie, Adobe Systems Incorporated <mattm@adobe.com>
Jeff Turpin, Axway Inc. <jturpin@us.axway.com>

APPENDIX I. Revision History

[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	05 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	01 Oct 2004	Matt MacKenzie	Integrated Reliable messaging, many editorial changes also.
WD 04	28 Sep 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.
WD 13	01 May 2006	Pete Wenzel	• Relabeled Figure 7. • Editorial corrections and clarifications throughout, provided by Jacques, Hamid, Ric, Dale. • Adjustments to MEP text from Jacques. • Rearranged Chapters to make more logical sense.
(CD 03) PR 01	09 May 2006	Pete Wenzel	• Updated Document Status.
WD 14	21 Aug 2006	Pete Wenzel	• New MEP Section from Jacques & Hamid (received 25 Jul 2006). • Revised Reliable Messaging Model Section from Jacques (posted 05 Aug 2006). • Changes based on some resolved Public Review issues, as documented in the issues list.

Rev	Date	By Whom	What
WD 15	03 Oct 2006	Pete Wenzel	- CORE-77, new Compatibility Appendix (13). - CORE-77, new Introduction Section 1.3. - CORE-16, Section 2.1.3. - CORE-26, Section 3.4.1. - New Processing Mode Model Appendix (12). - Updates to P-Mode Section (4). - CORE-47, Sections 5.2.1, 5.3.1. - CORE-56, partial. - CORE-73, Section 4.3. - CORE-15 (bundling options), Section 5.2, 5.2.1.
WD 16	14 Dec 2006	Pete Wenzel	- Fixed Appendix headings, Table of Figures hyperlinks. - Updates to WS-ReliableMessaging Binding, Nov 16 2006 from Jacques. - Updates to Compatibility Appendix, Version 0.6 (-h) from Jacques & Hamid, 21 Nov 2006. (incl. CORE-93) - New Message Authorization Section from Hamid & Jacques, 22 Nov 2006 and edits by Pete. - Editorial corrections throughout from Jacques, from 22 Nov 2006 email. - Updates to P-Mode Model from Hamid & Jacques (0.82, 29 Nov 2006). (incl. CORE-9, CORE-10, CORE-54, CORE-55, CORE-56) - CORE-46, Section 5.2.3.13. - CORE-50, Section 6.2.2. - CORE-53, Sections 8.1.1 & 4.1. - CORE-89, Section 1.3.

Rev	Date	By Whom	What
WD 17	07 Mar 2007	Pete Wenzel	• CORE-44, xml:id reference, Section 5.2.2.12. • CORE-87, Role now required, with default value, Section 5.2.2.3. • 12/19/2006 Edits from Jacques: CORE-9, Section 3.1; CORE-83, Section D.2.3; CORE-92+CORE-97, Section B.2; two-way definition in Section 2.2.3. • B.2 edits from Jacques (ebMS3-WS-ReliableMessaging-binding-4.pdf). • CORE-90, removed "wsswa" prefix. • CORE-92, reference latest RM draft. • 02/14/2007 Error Reporting update from Jacques, Section 6.6. • CORE-69, New Appendix (E), from Jacques (MEP-binding-cases-3.odt), and associated references. • 02/06/2007 Edits from Jacques, Sections 4, E.2.3. • Updates to P-Mode parameters (Appendix D), including CORE-88 Security rework (PMode-model-86.doc). • New Schema (Appendix A) from Hamid & Dale; CORE-2, CORE-3, CORE-4, CORE-5, CORE-6, CORE-22, CORE-48. • CORE-45, new ExternalReferenceError, Section 6.7.1. • CORE-52, Section 7.4 from Pete. • CORE-83 (Partial), Maximum message/payload size P-Mode parameters, D.3.3. • CORE-98, Fixed Examples. • CORE-91, New bundled message example, Section 5.3.5. • CORE-96, Receipt Signal added, Sections 5.2.3.3, 5.3.4 (example from Dale), 7.12.2, D.3.6. • CORE-103, Section 7.7, Username Token support not required. • Adjusted title page metadata, moved contributor lists to Appendix, per template requirements. • New Copyright text (04/15/2005 IPR Policy version).
WD 18	19 Mar 2007	Pete Wenzel	• 03/13/2007 Edits from Jacques, Sections 5.2.3.3, 2.2.2, Appendix I. • CORE-95, Updated Figure 15. • CORE-102, Section F.2.6. • 03/14/2007 Edits from Jacques, Appendix E. • CORE-105, Updated reference to WSRM CD7. • CORE-106, Signed Receipt Example, Section 7.9.3. • CORE-107, Updated schema and schemaLocation URIs.

Rev	Date	By Whom	What
WD 19	27 Mar 2007	Pete Wenzel	• CORE-110: Section 5.2.2.7, AgreementRef/@type now optional, default content is URI. @pmode spelling fixed in schema. • Section 5.2.3.3, Receipt signal contents SHOULD follow BPSS Signal (was MUST). • Section 7.9.3 example, removed "mid:" reference scheme. • CORE-107: Updated namespace and feature URIs throughout. • CORE-112: Section 5.2.2.12, removed attributes from PayloadInfo. • CORE-113: Section 5.2.2.13 and schema, clarify cardinality of eb:PartInfo/Schema attributes. • CORE-111: Section 5.1.3.6 and examples, removed Messaging/@eb:version.
CD 04	28 Mar 2007	Pete Wenzel	• Updated Document Status.
WD 20	18 Apr 2007	Pete Wenzel	• CORE-117: Updated SMTP Binding Note (Section C.3). • CORE-119: Changed "Flow" terminology to "Channel". • CORE-119: Updated namespace and schema to reflect @mpf renaming. • CORE-120: Filled-in Related Work section.
WD 21	24 Apr 2007	Pete Wenzel	• CORE-121: Corrected element/attribute capitalization, cardinality and qualification in schema, Section 5 and Examples.
WD 22	25 Apr 2007	Pete Wenzel	• CORE-119: Corrected instances of EmptyMessagePartitionFlow.
(CD 05) PR 02	26 Apr 2007	Pete Wenzel	• Updated Document Status. • Moved References Section to correct location.
WD 23	22 May 2007	Pete Wenzel	• CORE-105, Updated WSRM reference to its CS version. • CORE-108, Updated PartyId type references. • CORE-122, Added references to HTTP, SMTP. • CORE-123, Added URIs where appropriate. • CORE-124, Removed Conformance Definitions and Template. • CORE-126, Applied editorial corrections and clarifications. • Corrected and regenerated some Figures. • Editorial corrections (grammar, spelling) through the end of Section 3. • CORE-101: Corrections to examples based on validation results. • CORE-128: "Timestamp" and "ConversationId" element names in schema corrected.
WD 24	23 May 2007	Pete Wenzel	• CORE-130, Updated text referencing WS-I profiles.
CD 06	23 May 2007	Pete Wenzel	• Updated Document Status.
CS 01	31 May 2007	Pete Wenzel	• Updated Document Status.

Rev	Date	By Whom	What
WD 25	19 Jun 2007	Pete Wenzel	• CORE-128, Corrected additional element names.
WD 26	22 Jun 2007	Pete Wenzel	• CORE-131, Editorial corrections from Pim. • Editorial corrections (grammar, clarifications) throughout Sections 4 and 5.
CD 07	29 Jun 2007	Pete Wenzel	• Updated Document Status.
CS 02	12 Jul 2007	Pete Wenzel	• Updated Document Status.