



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

OASIS Standard

1 October 2007

Specification URIs:

This Version:

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

Previous 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

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

37 reliable, secure delivery of business information. Furthermore, the specification defines a flexible
38 enveloping technique, permitting messages to contain payloads of any format type. This
39 versatility ensures legacy electronic business systems employing traditional syntaxes (i.e.
40 UN/EDIFACT, ASC X12, or HL7) can leverage the advantages of the ebXML infrastructure along
41 with users of emerging technologies.

42 **Status:**

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

46 Technical Committee members should send comments on this specification to the
47 ebxml-msg@lists.oasis-open.org list. Others should use the comment form at
48 http://www.oasis-open.org/committees/comments/form.php?wg_abbrev=ebxml-msg.

49 For information on whether any patents have been disclosed that may be essential to
50 implementing this specification, and any offers of patent licensing terms, please refer to the
51 Intellectual Property Rights section of the OASIS ebXML Messaging Services TC web page
52 (<http://www.oasis-open.org/committees/ebxml-msg/ipr.php>).

53 The non-normative errata page for this specification is located at [http://www.oasis-](http://www.oasis-open.org/committees/ebxml-msg/)
54 [open.org/committees/ebxml-msg/](http://www.oasis-open.org/committees/ebxml-msg/).

55

56 Notices

57 Copyright © OASIS® 1993–2007. All Rights Reserved.

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

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

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

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

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

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

85 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
86 might be claimed to pertain to the implementation or use of the technology described in this document or
87 the extent to which any license under such rights might or might not be available; neither does it
88 represent that it has made any effort to identify any such rights. Information on OASIS' procedures with
89 respect to rights in any document or deliverable produced by an OASIS Technical Committee can be
90 found on the OASIS website. Copies of claims of rights made available for publication and any
91 assurances of licenses to be made available, or the result of an attempt made to obtain a general license
92 or permission for the use of such proprietary rights by implementers or users of this OASIS Committee
93 Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no
94 representation that any information or list of intellectual property rights will at any time be complete, or
95 that any claims in such list are, in fact, Essential Claims.

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

Table of Contents

102		
103	1. Introduction.....	9
104	1.1. Background and Objectives.....	9
105	1.2. Scope.....	9
106	1.3. Web Services and Their Role in an eBusiness Messaging Framework.....	10
107	1.4. Caveats and Assumptions.....	10
108	1.5. General Rules for Normative Interpretation.....	11
109	1.6. XML Notation.....	11
110	1.7. Namespace Prefixes.....	11
111	1.8. Example Domains.....	12
112	1.9. Normative References.....	12
113	1.10. Non-Normative References.....	13
114	2. Messaging Model.....	14
115	2.1. Terminology and Concepts.....	14
116	2.1.1. Components of the Model.....	14
117	2.1.2. Message Terminology.....	15
118	2.1.3. Messaging Roles.....	15
119	2.1.4. Abstract Messaging Operations.....	16
120	2.2. Message Exchange Patterns.....	16
121	2.2.1. Rationale.....	16
122	2.2.2. General Definition	16
123	2.2.3. MEP Bindings.....	17
124	2.2.4. Relationship to SOAP MEPs	18
125	2.2.5. The One-Way/Push MEP.....	19
126	2.2.6. The One-Way/Pull MEP.....	19
127	2.2.7. The Two-Way/Sync MEP.....	20
128	2.2.8. Other Transport-Channel-Bound MEPs.....	21
129	3. Message Pulling and Partitioning.....	22
130	3.1. Objectives.....	22
131	3.2. Supporting Message Pulling.....	22
132	3.3. Combining Pulling with Security and Reliability.....	23
133	3.4. Message Partition Channels.....	24
134	3.4.1. Concept and Purpose.....	24
135	3.4.2. Some Use Cases.....	26
136	3.4.3. Definition and Usage Requirements.....	28
137	4. Processing Modes.....	29
138	4.1. Messaging Service Processing Model.....	29
139	4.2. Processing Mode Features.....	30
140	4.3. Default Features for Processing Mode.....	31
141	5. Message Packaging.....	33
142	5.1. Message Envelope and Message Parts.....	33
143	5.1.1. MIME Structure and SOAP Profile.....	33
144	5.1.2. MIME and XML Considerations.....	36
145	5.1.3. ebXML SOAP Envelope Extension.....	36

146	5.1.4. ebMS Header.....	38
147	5.1.5. Payload Containers.....	38
148	5.2. The eb:Messaging Container Element.....	39
149	5.2.1. eb:Messaging Element Specification.....	40
150	5.2.2. eb:Messaging/eb:UserMessage.....	41
151	5.2.3. eb:Messaging/eb:SignalMessage.....	46
152	5.2.4. Message Unit Bundling.....	46
153	5.3. Examples of ebMS Messages.....	47
154	5.3.1. UserMessage Example.....	47
155	5.3.2. PullRequest Message Example.....	49
156	5.3.3. Error Message Example.....	49
157	5.3.4. Receipt Message Example.....	50
158	5.3.5. "Bundled" Message Example.....	50
159	6. Error Handling.....	52
160	6.1. Terminology.....	52
161	6.2. Packaging of ebMS Errors.....	52
162	6.2.1. eb:Error Element.....	52
163	6.2.2. eb:Error/@origin.....	53
164	6.2.3. eb:Error/@category.....	53
165	6.2.4. eb:Error/@errorCode.....	53
166	6.2.5. eb:Error/@severity.....	53
167	6.2.6. eb:Error/@refToMessageInError.....	53
168	6.2.7. eb:Error/@shortDescription.....	53
169	6.2.8. eb:Error/Description.....	53
170	6.2.9. eb:Error/ErrorDetail.....	53
171	6.3. ebMS Error Message.....	53
172	6.4. Extensibility of the Error Element.....	54
173	6.4.1. Adding new ebMS Errors.....	54
174	6.5. Generating ebMS Errors.....	54
175	6.6. Error Reporting.....	54
176	6.7. Standard ebMS Errors.....	55
177	6.7.1. ebMS Processing Errors.....	55
178	6.7.2. Security Processing Errors.....	56
179	6.7.3. Reliable Messaging Errors.....	56
180	7. Security Module.....	57
181	7.1. Security Element.....	57
182	7.2. Signing Messages.....	57
183	7.3. Signing SOAP with Attachments Messages.....	58
184	7.4. Encrypting Messages.....	58
185	7.5. Encrypting SOAP with Attachments Messages.....	58
186	7.6. Signing and Encrypting Messages.....	58
187	7.7. Security Token Authentication.....	58
188	7.8. Security Policy Errors.....	58
189	7.9. Secured Message Examples.....	59
190	7.9.1. Digitally Signed and Encrypted ebXML Message.....	59

191	7.9.2. Digitally Signed and Encrypted ebXML SOAP with Attachments Message.....	61
192	7.9.3. Digitally Signed Receipt Signal Message.....	63
193	7.10. Message Authorization	64
194	7.11. Securing the PullRequest Signal.....	66
195	7.11.1. Authentication.....	66
196	7.11.2. Authorization.....	66
197	7.11.3. Preventing Replay Attacks.....	66
198	7.12. Countermeasure Technologies.....	66
199	7.12.1. Persistent Digital Signature.....	66
200	7.12.2. Persistent Signed Receipt.....	66
201	7.12.3. Non-Persistent Authentication.....	67
202	7.12.4. Non-Persistent Integrity.....	67
203	7.12.5. Persistent Confidentiality.....	67
204	7.12.6. Non-Persistent Confidentiality.....	67
205	7.12.7. Persistent Authorization.....	67
206	7.12.8. Non-Persistent Authorization.....	67
207	7.13. Security Considerations.....	67
208	8. Reliable Messaging Module.....	69
209	8.1. The Reliable Messaging Model.....	69
210	8.1.1. Message Processing.....	69
211	8.1.2. The Reliable Messaging Processor in the MSH.....	69
212	8.2. Reliable Delivery of ebMS Messages.....	71
213	8.2.1. Reliability Contracts for the RMP.....	71
214	8.2.2. Reliability Contracts for the MSH.....	72
215	8.2.3. Reliability for Signal Messages.....	73
216	8.2.4. Handling of Delivery Failures.....	73
217	8.3. Reliability of ebMS MEPs.....	74
218	8.3.1. Reliability of the One-Way/Push MEP.....	74
219	8.3.2. Reliability of the One-Way/Pull MEP.....	75
220	8.3.3. Reliability of the Two-Way/Sync MEP.....	76
221	8.3.4. Reliability of Other Transport-Channel-Bound MEPs.....	77
222	APPENDIX A. The ebXML SOAP Extension Element Schema.....	78
223	APPENDIX B. Reliable Messaging Bindings.....	82
224	B.1. WS-Reliability Binding.....	82
225	B.1.1. Operations and Contracts Binding.....	82
226	B.1.2. Complement to the Reliability of the One-Way/Push MEP.....	82
227	B.1.3. Complement to the Reliability of the One-Way/Pull MEP.....	82
228	B.1.4. Complement to the Reliability of the Two-Way/Sync MEP.....	83
229	B.2. WS-ReliableMessaging Binding.....	85
230	B.2.1. Operations and Contracts Binding.....	85
231	B.2.2. Complement to the Reliability of the One-Way/Push MEP.....	86
232	B.2.3. Complement to the Reliability of the One-Way/Pull MEP.....	87
233	B.2.4. Complement to the Reliability of the Two-Way/Sync MEP.....	87
234	APPENDIX C. SOAP Format and Bindings.....	89
235	C.1. Using SwA with SOAP-1.1.....	89

236	C.2. Using SwA with SOAP-1.2.....	90
237	C.3. SMTP Binding.....	91
238	APPENDIX D. Processing Modes.....	92
239	D.1. Objectives and Usage.....	92
240	D.2. Model for Processing Modes.....	93
241	D.2.1. Notation.....	94
242	D.3. Processing Mode Parameters.....	95
243	D.3.1. General P-Mode Parameters.....	95
244	D.3.2. PMode[1].Protocol.....	96
245	D.3.3. PMode[1].BusinessInfo.....	96
246	D.3.4. PMode[1].ErrorHandling.....	96
247	D.3.5. PMode[1].Reliability.....	97
248	D.3.6. PMode[1].Security.....	98
249	APPENDIX E. P-Mode Values and ebMS MEP Bindings.....	100
250	E.1. P-Mode Values and the One-Way/Push MEP.....	100
251	E.2. P-Mode Values and the One-Way/Pull MEP.....	101
252	E.3. P-Mode Values and the Two-Way/Sync MEP.....	102
253	APPENDIX F. Compatibility Mapping to ebMS 2.0.....	103
254	F.1. Objectives and Approach.....	103
255	F.2. Compatibility Mapping Rules.....	103
256	F.2.1. (CM1) Header Mapping Rules.....	103
257	F.2.2. (CM2) Payload Mapping Rules.....	104
258	F.2.3. (CM3) Reliability Mapping Rules.....	104
259	F.2.4. (CM4) MEP Mapping Rules.....	105
260	F.2.5. (CM5) Signal Mapping Rules.....	107
261	F.2.6. (CM6) Processing Mode Mapping Rules.....	108
262	APPENDIX G. Conformance.....	109
263	APPENDIX H. Acknowledgments.....	110
264	APPENDIX I. Revision History.....	111
265		

266 **Table of Figures**

267 Figure 1: Entities of the Messaging Model and Their Interactions..... 15

268 Figure 2: One-Way/Push MEP..... 19

269 Figure 3: One-Way/Pull MEP..... 20

270 Figure 4: Two-Way/Sync MEP..... 21

271 Figure 5: One-Way/Pull with Message Partition Channels..... 26

272 Figure 6: Message Partition Channel Use Cases..... 27

273 Figure 7: Component Relationships..... 30

274 Figure 8: User Message Structure..... 34

275 Figure 9: Signal Message Structure..... 35

276 Figure 10: Sending an ebMS Message Reliably..... 70

277 Figure 11: Sending an ebMS MEP Response Message Reliably..... 71

278 Figure 12: Reliable One-Way/Push MEP..... 75

279 Figure 13: Reliable One-Way/Pull MEP..... 76

280 Figure 14: Reliable Two-Way/Sync MEP..... 77

281 Figure 15: P-Mode Structure for Two-Way/Push-and-Pull MEP..... 94

282

283

1. Introduction

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

290

1.1. Background and Objectives

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

299 It is becoming critical for broad adoption among all partners – large or small - of a supply-chain, to
300 handle differences in message flow capacity, intermittent connectivity, lack of static IP addresses or
301 firewall restrictions. Such new capabilities played an important role in the motivation that led to ebMS
302 3.0, along with the need to integrate and profile the emerging SOAP-based QoS-supporting standards.
303 The message header profiling that provided, in ebMS 2.0, a standard business-level header, has also
304 been extended to better address the diversity of back-end binding models, as well as the emerging trend
305 in business activity monitoring, the eBusiness side of which a message handler should be able to
306 support.

307 The ebXML messaging framework is not a restrictive one: business messages, identified as the
308 'payloads' of ebXML messages, are not limited to XML documents. Traditional EDI formats may also be
309 transported by ebMS. These payloads can take any digital form—XML, ASC X12, HL7, AIAG E5,
310 database tables, binary image files, etc. Multiple payloads, possibly of different MIME types, can be
311 transported in a single ebMS message. An objective of ebXML Messaging protocol is to be capable of
312 being carried over any available transfer protocol. This version of the specification provides bindings to
313 HTTP and SMTP, but other protocols to which SOAP may bind can also be used. The choice of an XML
314 framework rather reflects confidence in a growing XML-based Web infrastructure and development tools
315 infrastructure, the components of which can be leveraged and reused by developers.

316

1.2. Scope

317 The ebXML infrastructure is composed of several independent, but related, components. Some
318 references and bindings to other ebXML specifications in this document should be interpreted as aids to
319 integration, rather than as a requirement to integrate or to use in combination. For example, ebMS may
320 refer to the [ebCPPA] specification, rather than require its use. The ebMS relies on a concept of
321 "Agreement", the concrete representation of which (e.g. CPA or other configuration information) is left for
322 implementers to decide.

323 The ebMS defines messaging functions, protocol and envelope intended to operate over SOAP (SOAP
324 1.1 or SOAP 1.2, and SOAP with Attachments). Binding to lower transport layers such as HTTP and
325 SMTP relies on standard SOAP bindings when these exist, and ebMS only specifies some complement
326 to these, as required.

327 This document, Part 1: Core Features, supports networking topologies in which there are limitations on
328 initiating message transfer, but with only a point-to-point MSH topology, in which no intermediaries are
329 present. A forthcoming Part 2, containing Advanced Features, may take into account topologies that
330 contain intermediaries (e.g. hub, multi-hop), as well as those in which the ultimate MSH acts as a SOAP
331 intermediary.

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

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

343 **1.3. Web Services and Their Role in an eBusiness Messaging** 344 **Framework**

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

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

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

370 **1.4. Caveats and Assumptions**

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

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

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

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

379 This specification presents some alternatives regarding underlying specifications (e.g. SOAP 1.1/1.2,
380 WSS1.0/1.1, and Web Services specifications that support the reliability function). This does not imply
381 that a conforming implementation must support them all, nor that it is free to support any option. The

382 definition of conformance profiles - out of scope for this document, and to be described in an adjunct
383 OASIS document - will complement this specification by asserting which option(s) must be supported in
384 order to claim support for a particular conformance profile. Conformance to compatible profiles is a
385 prerequisite to interoperability. See Appendix G for more details on conformance profiles.

386 1.5. General Rules for Normative Interpretation

387 The key words *MUST*, *MUST NOT*, *REQUIRED*, *SHALL*, *SHALL NOT*, *SHOULD*, *SHOULD NOT*,
388 *RECOMMENDED*, *MAY*, and *OPTIONAL* in this document are to be interpreted as described in
389 [RFC2119].

390 For any given module described in this specification, an implementation **MUST** satisfy **ALL** of the
391 following conditions to be considered a conforming implementation of that module:

- 392 1. It supports all the mandatory syntax, features and behavior (as identified by the [RFC2119] key
393 words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL** and **SHALL NOT**) defined in the section that
394 specifies that module.
- 395 2. When the keywords **MUST**, **SHALL**, or **REQUIRED** are used to qualify a feature, support for this
396 feature--either message content or implementation behavior--is mandatory in an implementation
397 with a conformance profile that requires this feature.
- 398 3. It complies with the following interpretation of the keywords **OPTIONAL** and **MAY**: When these
399 keywords apply to the behavior of the implementation, the implementation is free to support
400 these behaviors or not, as meant in [RFC2119]. When these keywords apply to message
401 contents relevant to a module of features, a conforming implementation of such a module **MUST**
402 be capable of processing these optional message contents according to the described ebXML
403 semantics.
- 404 4. If it has implemented optional syntax, features and/or behavior defined in this specification, it
405 **MUST** be capable of interoperating with another implementation that has not implemented the
406 optional syntax, features and/or behavior. It **MUST** be capable of processing the prescribed
407 failure mechanism for those optional features it has chosen to implement.
- 408 5. It is capable of interoperating with another implementation that has chosen to implement optional
409 syntax, features and/or behavior, defined in this specification, it has chosen not to implement.
410 Handling of unsupported features **SHALL** be implemented in accordance with the prescribed
411 failure mechanism defined for the feature.

412 1.6. XML Notation

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

416 1.7. Namespace Prefixes

417 This table maps various prefixes that appear in XML examples to their intended corresponding
418 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/wsr/2004/06/ws-reliability-1.1.xsd

Prefix	Namespace
wsrx	http://docs.oasis-open.org/ws-rx/wsrn/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

419

420 1.8. Example Domains

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

425 1.9. Normative References

- 426 [HTTP11] R. Fielding, et al, *Hypertext Transfer Protocol -- HTTP/1.1*, 1999.
 427 <<http://www.ietf.org/rfc/rfc2616.txt>>
- 428 [IANAMEDIA] Various, *MIME Media Types*, Various. <<http://www.iana.org/assignments/media-types/>>
- 429 [RFC2045] N Freed, et al, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of*
 430 *Internet Message Bodies*, 1996. <<http://www.ietf.org/rfc/rfc2045.txt>>
- 431 [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, 1997.
 432 <<http://www.ietf.org/rfc/rfc2119.txt>>
- 433 [RFC2387] E. Levinson, *The MIME Multipart/Related Content-type*, 1998.
 434 <<http://www.ietf.org/rfc/rfc2387.txt>>
- 435 [RFC2392] E. Levinson, *Content-ID and Message-ID Uniform Resource Locators*, 1998.
 436 <<http://www.ietf.org/rfc/rfc2392.txt>>
- 437 [RFC2396] T. Berners-Lee, et al, *Uniform Resource Identifiers (URI): Generic Syntax*, 1998.
 438 <<http://www.ietf.org/rfc/rfc2396.txt>>
- 439 [RFC2822] P. Resnick, ed., *Internet Message Format*, 2001. <<http://www.ietf.org/rfc/rfc2822.txt>>
- 440 [SMTP] J. Klensin, ed., *Simple Mail Transfer Protocol*, 2001.
 441 <<http://www.ietf.org/rfc/rfc2821.txt>>
- 442 [SOAP11] D. Box, et al, *Simple Object Access Protocol (SOAP) 1.1*, 2000.
 443 <<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>>
- 444 [SOAP12] M. Gudgin, et al, *SOAP Version 1.2 Part 1: Messaging Framework*, 2003.
 445 <<http://www.w3.org/TR/soap12-part1/>>
- 446 [SOAPATTACH] J. Barton, et al, *SOAP Messages with Attachments*, 2000.
 447 <<http://www.w3.org/TR/SOAP-attachments>>
- 448 [UTF8] F. Yergeau, *UTF-8, a transformation format of ISO 10646*, 1998.
 449 <<http://www.ietf.org/rfc/rfc2279.txt>>
- 450 [WSIAP10] Chris Ferris, et al, eds, *Attachments Profile Version 1.0*, 2004. <[http://www.ws-](http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html)
 451 [i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html](http://www.ws-i.org/Profiles/AttachmentsProfile-1.0-2004-08-24.html)>
- 452 [WSIBSP10] Abbie Barbir, et al, eds, *Basic Security Profile Version 1.0*, 2005. <[http://www.ws-](http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html)
 453 [i.org/Profiles/BasicSecurityProfile-1.0.html](http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html)>
- 454 [WSR11] Kazunori Iwasa, et al, eds, *WS-Reliability 1.1*, 2004. <[http://docs.oasis-](http://docs.oasis-open.org/wsm/ws-reliability/v1.1/wsm-ws_reliability-1.1-spec-os.pdf)
 455 [open.org/wsm/ws-reliability/v1.1/wsm-ws_reliability-1.1-spec-os.pdf](http://docs.oasis-open.org/wsm/ws-reliability/v1.1/wsm-ws_reliability-1.1-spec-os.pdf)>
- 456 [WSRM11] D. Davis, et al, eds, *Web Services Reliable Messaging (WS-ReliableMessaging)*
 457 *Version 1.1*, 2007. <<http://docs.oasis-open.org/ws-rx/wsrn/v1.1/wsrn.pdf>>
- 458 [WSRMP11] D. Davis, et al, eds, *Web Services Reliable Messaging Policy (WS-RM Policy) Version*
 459 *1.1*, 2007. <<http://docs.oasis-open.org/ws-rx/wsrmp/v1.1/wsrmp.pdf>>

- 460 [WSS10] Anthony Nadalin, et al, eds., *Web Services Security: SOAP Message Security 1.0*,
 461 2004. <[http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-
 security-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-

 462 security-1.0.pdf)>
- 463 [WSS10-USER] P. Hallam-Baker, et al, eds., *Web Services Security UsernameToken Profile 1.0*, 2004.
 464 <<http://docs.oasis-open.org/wss/2004/01/>>
- 465 [WSS10-X509] P. Hallam-Baker, et al, eds., *Web Services Security X.509 Certificate Token Profile*,
 466 2004. <<http://docs.oasis-open.org/wss/2004/01/>>
- 467 [WSS11] Anthony Nadalin, et al, eds., *Web Services Security: SOAP Message Security 1.1*,
 468 2005. <<http://docs.oasis-open.org/wss/v1.1/>>
- 469 [WSS11-USER] A. Nadalin, et al, eds., *Web Services Security UsernameToken Profile 1.1*, 2006.
 470 <<http://docs.oasis-open.org/wss/v1.1/>>
- 471 [WSS11-X509] A. Nadalin, et al, eds., *Web Services Security X.509 Certificate Token Profile 1.1*, 2006.
 472 <<http://docs.oasis-open.org/wss/v1.1/>>
- 473 [XML10] Tim Bray, et al, eds., *Extensible Markup Language (XML) 1.0 (Third Edition)*, 2004.
 474 <<http://www.w3.org/TR/2004/REC-xml-20040204/>>
- 475 [XMLDSIG] Donald Eastlake, et al, eds, *XML-Signature Syntax and Processing*, 2002.
 476 <<http://www.w3.org/TR/xmlsig-core/>>
- 477 [XMLENC] D. Eastlake, et al, *XML Encryption Syntax and Processing*, 2002.
 478 <<http://www.w3.org/TR/xmlenc-core/>>
- 479 [XMLNS] Tim Bray, et al, eds, *Namespaces in XML*, 1999. <[http://www.w3.org/TR/REC-xml-
 names/](http://www.w3.org/TR/REC-xml-

 480 names/)>
- 481 [XMLSCHEMA] Henry S. Thompson, et al, eds., *XML Schema Part 1: Structures Second Edition*, 2004.
 482 <<http://www.w3.org/TR/xmlschema-1/>>
- 483 [XPATH] James Clark, et al, eds., *XML Path Language (XPath) Version 1.0*, 1999.
 484 <<http://www.w3.org/TR/xpath>>
 485

486 1.10. Non-Normative References

- 487 [ebBP-SIG] OASIS ebXML Business Process TC, *ebXML Business Signals Schema*, 2006.
 488 <<http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0>>
- 489 [ebCPPA] OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee,
 490 *Collaboration-Protocol Profile and Agreement Specification Version 2.0*, 2002.
 491 <<http://www.oasis-open.org/committees/download.php/204/ebcpp-2.0.pdf>>
- 492 [ebCPPA21] OASIS ebXML Collaboration Protocol Profile and Agreement Technical Committee,
 493 *Collaboration-Protocol Profile and Agreement Specification Version 2.1*, 2005.
 494 <[http://www.oasis-open.org/committees/download.php/12208/ebcpp-2.1-april-5-2005-
 draft.doc](http://www.oasis-open.org/committees/download.php/12208/ebcpp-2.1-april-5-2005-

 495 draft.doc)>
- 496 [ebRISK] ebXML Security Team, *ebXML Technical Architecture Risk Assessment v1.0*, 2001.
 497 <<http://ebxml.org/specs/secRISK.pdf>>
- 498 [ISO6523] Unknown, *Identification of organization identification schemes*, 1998.
 499 <[http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=25773
 >](http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=25773

 500 >)>
- 501 [ISO9735] Unknown, *EDIFACT*, 1988.
 502 <<http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=17592>>
- 503 [QAFW] Karl Dubost, et al, eds, *QA Framework: Specification Guidelines*, 2005.
 504 <<http://www.w3.org/TR/qaframe-spec/>>
- 505 [RFC2246] T. Dierks, et al, *The TLS Protocol Version 1.0*, 1999.
 506 <<http://www.ietf.org/rfc/rfc2246.txt>>
- 507 [RFC2402] S. Kent, et al, *IP Authentication Header*, 1998. <<http://www.ietf.org/rfc/rfc2402.txt>>
- 508 [RFC2606] D. Eastlake, et al, *Reserved Top Level DNS Names*, 1999.

509 <<http://www.ietf.org/rfc/rfc2606.txt>>
510 [RFC2617] J. Franks, et al, *HTTP Authentication: Basic and Digest Access Authentication*, 1999.
511 <<http://www.ietf.org/rfc/rfc2617.txt>>
512 [RFC3023] M. Murata, et al, *XML Media Types*, 2001. <<http://www.ietf.org/rfc/rfc3023.txt>>
513 [SOAPEMAIL] H. M. Mountain, et al, *SOAP Version 1.2 Email Binding*, 2002.
514 <<http://www.w3.org/TR/soap12-email>>
515 [WSPOLICY] A. Vedamuthu, et al, eds, *Web Services Policy 1.5: Framework*, 2007.
516 <<http://www.w3.org/TR/ws-policy/>>
517 [WSSECPOL] A. Nadalin, et al, eds, *WS-SecurityPolicy 1.2*, 2007. <[http://docs.oasis-open.org/ws-
518 sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf](http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf)>
519

520

2. Messaging Model

521

2.1. Terminology and Concepts

522

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

523

524

2.1.1. Components of the Model

525

The ebMS messaging model assumes the following components:

526

- **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).

527

528

529

530

531

- **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).

532

533

534

- **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.

535

536

537

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

538

Notes:

539

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.

540

541

542

543

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.

544

545

546

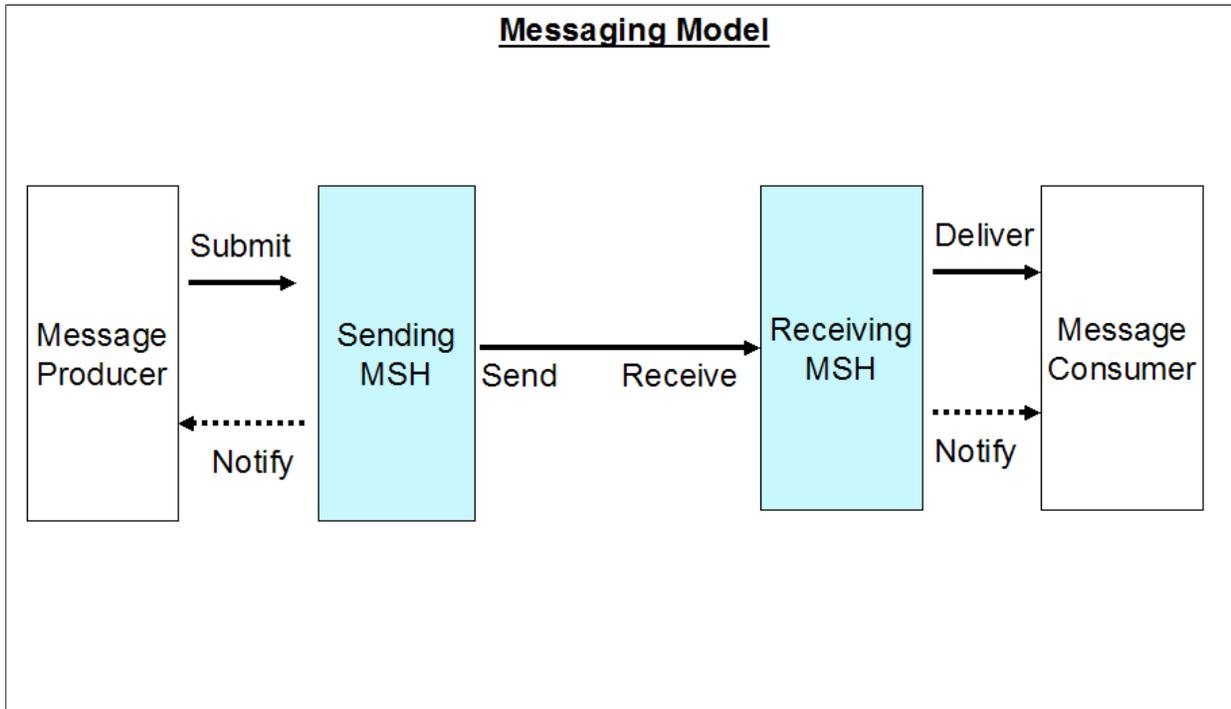


Figure 1: Entities of the Messaging Model and Their Interactions

547

2.1.2. Message Terminology

548

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

549

550

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:

551

552

- 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.

553

554

555

556

- 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.

557

558

559

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`).

560

561

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.

562

563

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.

564

565

2.1.3. Messaging Roles

566

The Messaging Model assumes the following roles for an MSH:

567

- **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.)

568

569

570

571

572

- **Receiving:** An MSH acting in the Receiving role performs the functions associated with the

573 receiving and processing of an ebMS user message. The abstract operations Receive, Deliver
574 and Notify are supported by this role. (Note that even in a Receiving role, an MSH MAY be
575 required to generate and send ebMS Signal Messages related to the reception of messages,
576 such as error messages or PullRequest signals.)

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

579 **2.1.4. Abstract Messaging Operations**

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

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

593 **2.2. Message Exchange Patterns**

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

597 **2.2.1. Rationale**

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

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

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

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

618 2.2.2. General Definition

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

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

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

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

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

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

646 2.2.3. MEP Bindings

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

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

664 A transport channel binding is a critical complement to an MEP, to be agreed on in order for partners to
665 interoperate. The rationale in using different transport channel bindings for an ebMS MEP is to
666 accommodate different connectivity constraints (e.g. firewall restrictions, intermittent availability, non-
667 static IP address) by dictating how each message transfer is initiated over the underlying protocol.

668 Because such connectivity constraints usually exist independently from the details of the transport
669 protocol, the transport channel binding is the right level to address them. The transport channel bindings
670 identified in this specification are:

- 671 • **Push:** maps an MEP User message to the 1st leg of an underlying 2-way transport protocol, or of
672 a 1-way protocol. This binding is identified by the URI
673 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push>.
- 674 • **Pull:** maps an MEP User message to the second leg of an underlying two-way transport protocol,
675 as a result of an ebMS Pull Signal sent over the first leg. This binding is identified by the URI
676 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/pull>.
- 677 • **Sync:** maps an exchange of two User messages respectively to the first and second legs of a
678 two-way underlying transport protocol. This binding is identified by the URI
679 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/sync>.

680 Notes:

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

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

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

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

708 **2.2.4. Relationship to SOAP MEPs**

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

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

715 Expressing the transport-channel-binding in terms of SOAP MEPs is only helpful if there is a published,
716 non-ambiguous, standard way for these to map to the underlying protocol(s). This is currently only the

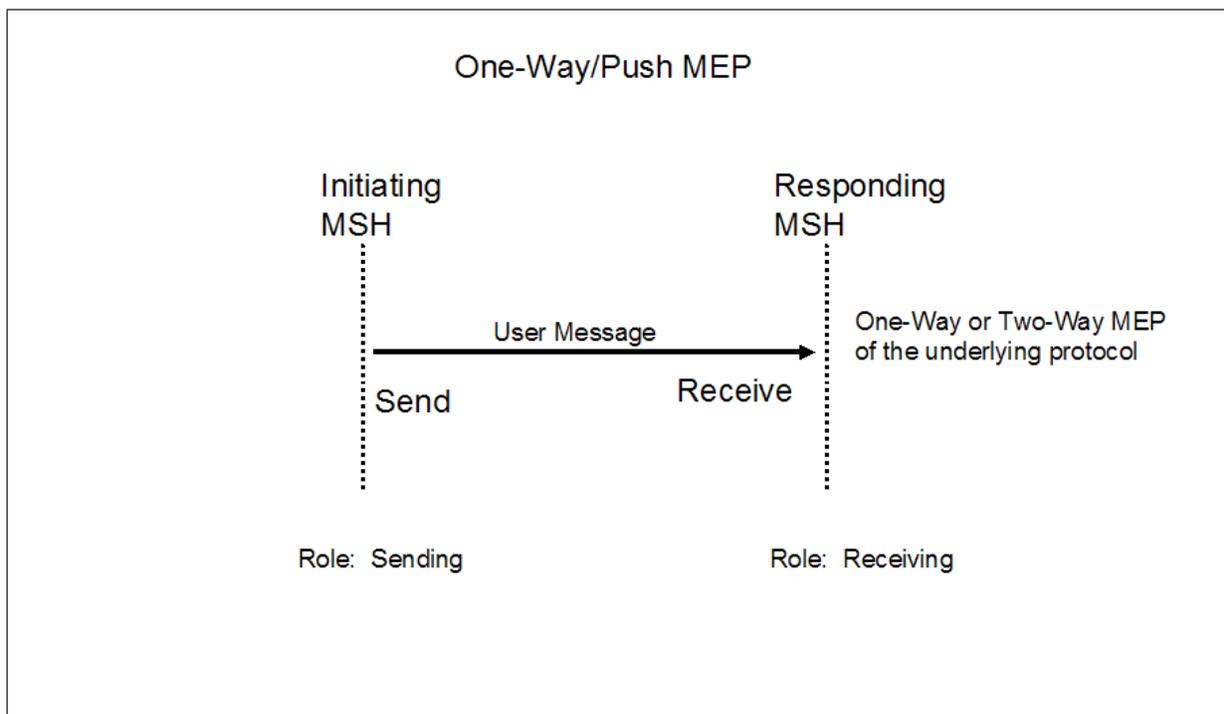
717 case for some SOAP MEPs and some transport protocols. Consequently, this specification has chosen
718 to express its transport-channel-bindings directly in terms of how to use the channels of the transport
719 protocol, abstracting such a transport as either "One-Way" or "Two-Way".

720 2.2.5. The One-Way/Push MEP

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

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

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



733 Figure 2: One-Way/Push MEP

734

734 2.2.6. The One-Way/Pull MEP

735 This transport-channel-bound MEP involves the transfer of a single ebMS User Message unit (label:
736 "oneway"). This MEP is initiated by the Receiving MSH, over a two-way underlying transport protocol.
737 The first leg of the protocol exchange carries a Pull Signal message. The second leg returns the pulled
738 User Message unit. To conform to this MEP the pulled User Message unit MUST NOT include an
739 eb:RefToMessageId element. In case no message is available for pulling, an ebMS error signal of
740 severity level "warning" and short description of "EmptyMessagePartitionChannel", as listed in Section
741 6.7.1, MUST be returned over the response leg. Figure 3 illustrates this MEP.

776

3. Message Pulling and Partitioning

777

3.1. Objectives

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

- 784 • Message Pulling
- 785 • Message Partition Channels (MPCs)

786 Message Pulling is defined in an abstract way by the One-Way/Pull ebMS MEP (see Section 2.2.6). This
787 MEP allows an MSH to initiate the transfer of a message as a receiver. When used in combination with
788 the One-Way/Push ebMS MEP, it allows an MSH full control over initiating asynchronous transfers with
789 another MSH in both directions, engaging in a client-server type of interaction with the remote MSH,
790 without any need to open a TCP/IP port to incoming requests. This MEP also supports exchanges with a
791 partner that is intermittently connected: instead of periodically polling for partner presence, a sending
792 MSH will simply wait for the partner MSH to pull its messages.

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

797 Message Partition Channels (see Section 3.4) allow for partitioning the flow of messages from an MSH to
798 another MSH into separate flows, so that each one of these flows can be controlled independently by
799 either MSH, in terms of transfer priorities. A Sending MSH MUST be able to determine whether a
800 submitted message should be pulled or pushed, and to which Message Partition Channel (MPC) it must
801 be assigned. Similarly, the Receiving MSH is aware of which MPC(s) should be pulled from, and which
802 ones will be used for push. This knowledge is based on an agreement shared between parties prior to
803 the exchanges, and modeled in this specification as the P-Mode operation set (see Section 4).

3.2. Supporting Message Pulling

804
805 Using Message pulling requires the ability of an MSH to support the One-Way/Pull MEP. The
806 PullRequest signal that initiates this MEP is described in Section 5.2.3.1. Because there is always at
807 least one MPC open between a Sending MSH and a Receiving MSH—the default MPC—the Pull mode
808 can be supported regardless of the ability to support several MPCs.

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

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

813 On Responding MSH side:

- 814 1. Submit: submission of message data to the MSH by the Producer party, intended for the
815 Consumer on the Initiator side. The message is associated with an MPC. If no MPC name is
816 provided by the submitter, or if the MSH implementation has not been provided with a way to
817 determine this association by itself, the default MPC is used. The MEP associated with this
818 message (e.g. as specified by P-Mode.MEP; see Section 4.2) is a One-Way/Pull.

819 On Initiating MSH side:

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

822 On Responding MSH side:

- 823 3. Reception of the PullRequest signal. For every PullRequest signal received the Responder MSH
824 (acting in Sending role) selects a previously submitted message. It is RECOMMENDED to select
825 messages according to a FIFO policy with respect to the Submit operation. If there is no user
826 message available in the specified MPC for sending, a warning signal with short description:
827 "EmptyMessagePartitionChannel" (see Section 6.7.1) MUST be sent back instead.
- 828 4. Send: the selected message is sent over the SOAP Response to the PullRequest.

829 **On Initiating MSH side:**

- 830 5. Receive: the pulled message is available for processing by the MSH. The header @mpc attribute
831 indicates from which MPC it has been pulled, and is the same as the value of @mpc in the
832 corresponding PullRequest signal.
- 833 6. Deliver: after processing of ebMS headers, delivery of the pulled message data to the Consumer
834 of the MSH.

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

```
836 <S11:Envelope>
837 <S11:Header>
838 <eb:Messaging S11:mustUnderstand="1">
839   <eb:SignalMessage>
840     <eb:MessageInfo>
841       <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
842       <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
843     </eb:MessageInfo>
844     <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
845   </eb:SignalMessage>
846 </eb:Messaging>
847 </S11:Header>
848 <S11:Body/>
849 </S11:Envelope>
```

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

```
851 <S11:Envelope>
852 <S11:Header>
853 <eb:Messaging S11:mustUnderstand="1" >
854   <eb:UserMessage mpc="http://sender.example.com/mpc123">
855     <eb:MessageInfo>
856       <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
857       <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
858       <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
859     </eb:MessageInfo>
860     <eb:PartyInfo>
861       ...
862     </eb:PartyInfo>
863     <eb:CollaborationInfo>
864       ...
865     </eb:CollaborationInfo>
866     <eb:PayloadInfo>
867       ...
868     </eb:PayloadInfo>
869   </eb:UserMessage>
870 </eb:Messaging>
871 </S11:Header>
872 <S11:Body>
873   ...
874 </S11:Body>
875 </S11:Envelope>
```

876 3.3. Combining Pulling with Security and Reliability

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

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

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

```
883 <S11:Envelope>
```

```

884 <S11:Header>
885 <eb:Messaging S11:mustUnderstand="1" >
886   <eb:SignalMessage>
887     <eb:MessageInfo>
888       <eb:Timestamp>2006-10-01T10:01:00</eb:Timestamp>
889       <eb:MessageId>UUID-4@receiver.example.com</eb:MessageId>
890     </eb:MessageInfo>
891     <eb:PullRequest mpc="http://sender.example.com/mpc123"/>
892   </eb:SignalMessage>
893 </eb:Messaging>
894 <wss:Security>
895   ...
896 </wss:Security>
897 <wsr:Request S11:mustUnderstand="1">
898   ...
899   <ReplyPattern>
900     <Value>Response</Value>
901   </ReplyPattern>
902   <AckRequested/>
903   ...
904 </wsr:Request>
905 </S11:Header>
906 <S11:Body/>
907 </S11:Envelope>

```

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

```

910 <S11:Envelope>
911 <S11:Header>
912 <eb:Messaging S11:mustUnderstand="1" >
913   <eb:UserMessage mpc="http://sender.example.com/mpc123">
914     <eb:MessageInfo>
915       <eb:Timestamp>2006-10-01T10:02:00</eb:Timestamp>
916       <eb:MessageId>UUID-5@sender.example.com</eb:MessageId>
917       <eb:RefToMessageId>UUID-4@receiver.example.com</eb:RefToMessageId>
918     </eb:MessageInfo>
919     <eb:PartyInfo>
920       ...
921     </eb:PartyInfo>
922     <eb:CollaborationInfo>
923       ...
924     </eb:CollaborationInfo>
925     <eb:PayloadInfo>
926       ...
927     </eb:PayloadInfo>
928   </eb:UserMessage>
929 </eb:Messaging>
930 <wsr:Response S11:mustUnderstand="1">
931   ...
932 </wsr:Response>
933 <wss:Security>
934   ...
935 </wss:Security>
936 </S11:Header>
937 <S11:Body>
938   ...
939 </S11:Body>
940 </S11:Envelope>

```

941 **Note:**
942 In the above example, the reliability header, which assumes the use of WS-Reliability, is
943 a Response element. It contains the reliability acknowledgment for the PullRequest
944 signal. In this example there is no wsr:Request reliability header. A wsr:Request header
945 could be present, in addition to wsr:Response, in case some specific reliability
946 requirement is associated with the pulled message (see Section 8.3).

947 **3.4. Message Partition Channels**

948 **3.4.1. Concept and Purpose**

949 Message Partition Channels (MPCs) allow for partitioning the flow of messages from a Sending MSH to a
950 Receiving MSH into several flows that can be controlled separately and consumed differently. They also

951 allow for merging flows from several Sending MSHs, into a unique flow that will be treated as such by a
952 Receiving MSH. In particular, MPCs allow for:

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

961 Notes:

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

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

968 **Example:** *A pair of business partners – a large buyer and a small supplier - have decided to create two*
969 *MPCs for transferring messages sent by the buyer. Urgent messages that require immediate processing*
970 *(e.g. high priority Purchase Orders, and updates to prior Purchase Orders) are assigned to one MPC;*
971 *and less urgent messages (payments, catalog requests, confirmations, acknowledgments of receipts,*
972 *etc.) are assigned to the other MPC. The buyer determines the level of urgency of a posting, which may*
973 *or may not be manifested inside the message. Per an agreement with the buyer, the supplier will pull and*
974 *process first all messages from the "urgent" MPC; then, once that is exhausted, only the messages from*
975 *the less urgent MPC. This way, the low-capacity Receiving MSH (supplier) is able to prioritize the*
976 *reception of its messages, focusing its resources on the most urgent messages and avoiding the*
977 *overhead and risk in managing (persistence, recovery, security) less urgent but important messages that*
978 *it cannot process in the short term.*

979 Any more complex filtering mechanism that requires checking a filter condition on header data, is out of
980 scope of this specification. Such filtering could be implemented in a Sending MSH and/or in a Receiving
981 MSH as a complement to, or instead of, different MPCs. The notion of MPC is a simple and robust
982 solution with low interoperability risk: it allows for partitioning messages based on prior agreement
983 between producer and consumer on which type of message will use which MPC, without a need to
984 communicate and process filter expressions for each message transfer.

985

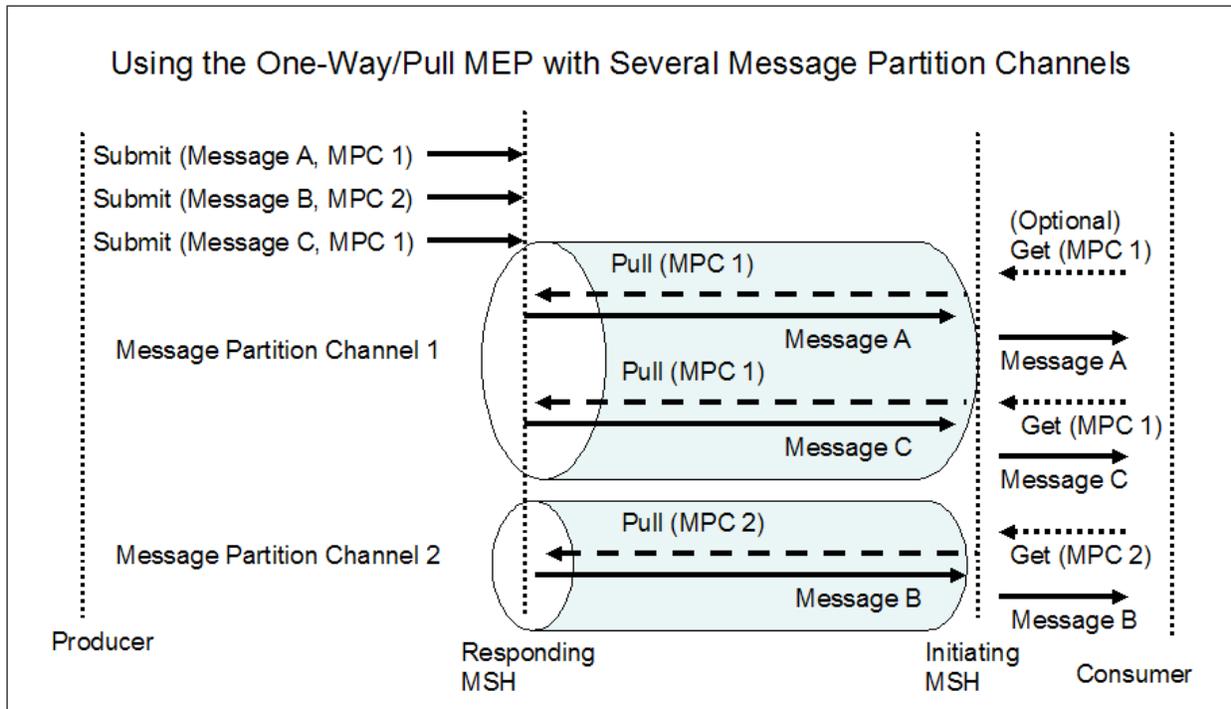


Figure 5: One-Way/Pull with Message Partition Channels

987 Figure 5 illustrates how MPCs and the One-Way/Pull MEP can be used by a Consumer party to control
 988 the order of the messages it wants to receive and process. Messages on MPC 1 are "pulled" in priority
 989 by the Consumer side.

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

- 993 • The MEPs in which these messages participate. Messages over the same MPC can either be
 994 pulled or pushed, based on the different MEPs that govern the transfer of these messages.
- 995 • The regular addressing means used for sending messages (e.g. URL of Receiving MSH when
 996 pushing messages). MPCs do not have any routing or addressing capability.

997 Before it is transferred from a Sending MSH to a Receiving MSH, regardless of whether it is pushed or
 998 pulled, a message is always assigned to an MPC. If no explicit assignment is requested (e.g. by the
 999 message Producer at Submit time or per configuration of the MSH), the default MPC is
 1000 "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultMPC" is assigned.

1001 3.4.2. Some Use Cases

1002 Figure 6 illustrates various cases in using MPCs.

1003

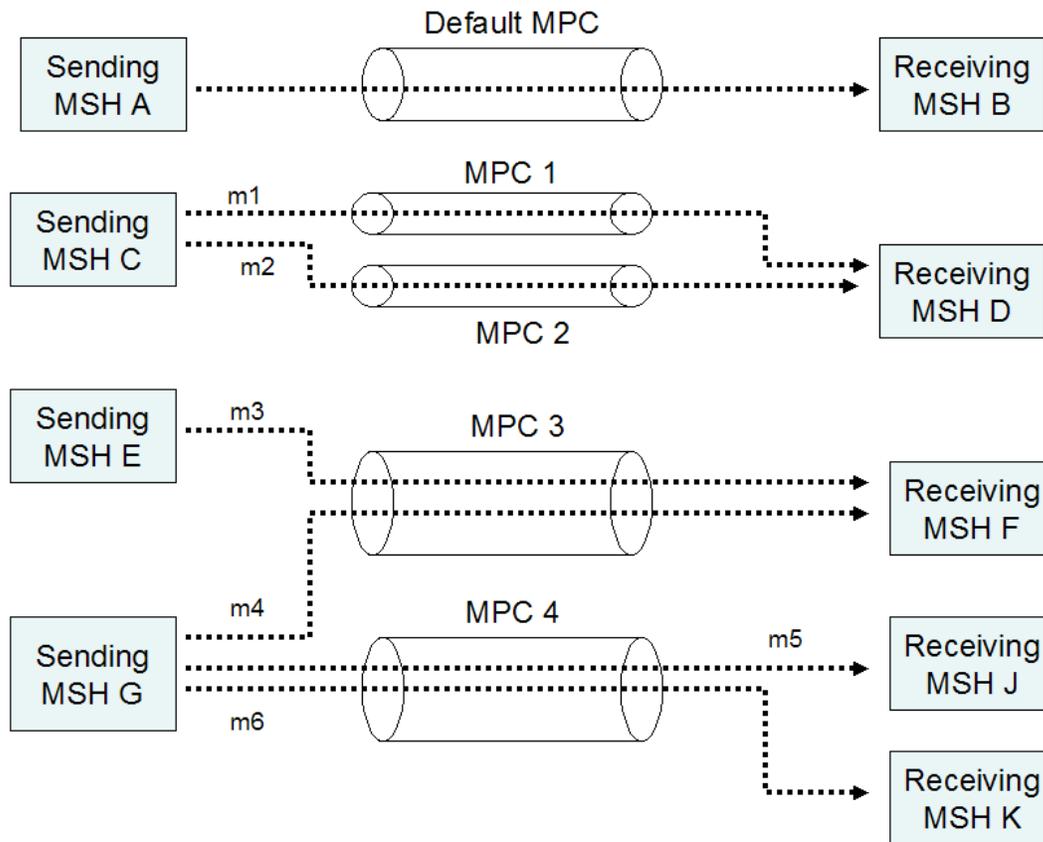


Figure 6: Message Partition Channel Use Cases

1004 In the figure above, each arrow represents the transfer of a user message, which could be either pushed or pulled.
1005

1006 Between MSHs A and B, no MPC has been explicitly defined or assigned. All messages transferred from
1007 A to B – whether pushed or pulled – will implicitly use the default MPC.

1008 MSHs C and D have been configured to use MPCs 1 and 2 (in addition to the default MPC). Messages
1009 sent may be assigned to either one of these MPCs. In case these messages are pulled, MSH D may
1010 choose from which MPC to pull first.

1011 MPC 3 is shared by two Sending MSHs, E and G. The effect of using this MPC is to define on the
1012 Receiving MSH F a merged inflow of messages from E and G, which may be presented to the Consumer
1013 as a single flow. If messages m3 and m4 are pulled, MSH F has control over which MSH from which to
1014 pull first.

1015 MPC 4 is used by MSH G to send either to MSH J or MSH K. When combined with message pulling, this
1016 use case allows for various scenarios. For example, the message flow might initially go exclusively from
1017 G to J. In case MSH J fails, another MSH (K) may immediately take over the message flow without any
1018 change on the sender side (assuming K is authorized) nor any knowledge by K of where the initial flow
1019 was intended for. Or, two Receiving MSHs (J and K) that are remote from each other but used by
1020 equivalent applications may split the processing of messages submitted to the same Sending MSH G.
1021 This may be, for example, two agencies equally qualified to process trouble tickets, indiscriminately
1022 pulling messages from the same MPC at the pace allowed by their processing capacity. MPC 4 may also
1023 be used by concurrent, pushed message flows. Using the same MPC does not introduce any
1024 dependency between the processing of m5 and m6 in J and K, but may be associated with a particular
1025 business meaning (i.e. is meaningful to Consumers of J and K).

1026 3.4.3. Definition and Usage Requirements

1027 An MPC is a flow of messages from a set of Sending MSHs to a set of Receiving MSHs, in the sense
1028 given in flow networks theory. It is identified by a name—a string of characters—that is assigned to every
1029 message of the flow. For every message it sends or receives, an MSH must be aware of which MPC this
1030 message is assigned to. MPC is a dynamic notion, the elements of which do not need to be fully defined
1031 prior to initiating this flow. For example, additional MSHs (either Sending or Receiving) may join the flow
1032 at any time, assuming they have knowledge of the MPC name, and assuming there is no other reason
1033 preventing them from transferring messages over this MPC (e.g. security).

1034 The association between a user message and an MPC is apparent in the ebMS header of the message
1035 (see Section 5.2). Except for the default MPC, the MPC name must appear in the header of a user
1036 message transferred over this MPC.

1037 Note:

1038 As defined above, an MPC may involve more than a Sending MSH and a Receiving
1039 MSH. In particular, two unrelated pairs of Sending/Receiving MSHs (e.g. in the previous
1040 figure, C and D on the one hand, E and F on the other hand) could transfer messages
1041 using the same MPC name (e.g. MPC 3 in the figure could also be renamed MPC 2).
1042 Formally speaking, all these messages would be transferred over the same MPC. There
1043 might be some business significance in deciding whether two pairs of MSHs that have
1044 unconnected message flows should use the same MPC to transfer these messages,
1045 even though as far as the MSHs are concerned, they will process these two separate
1046 sub-flows of messages independently from each other.

1047 Only user messages may be assigned to MPCs, not signal messages.

1048 A PullRequest signal message always indicates in its header (see Section 5.2.3.1) the MPC on which the
1049 message must be pulled. If no MPC is explicitly identified, the default MPC MUST be pulled from. The
1050 pulled message sent in response MUST have been assigned to the indicated MPC.

1051 The association of a message with an MPC must be done either at Submit time, e.g. requested by the
1052 message Producer; or at any time between Submit and Send, e.g. based on configuration or processing
1053 mode (see Section 4). This is left to the implementation.

1054 Support for assigning messages to MPCs—e.g. by automatically mapping messages submitted by a
1055 Producer to a particular MPC based on some rules, queries or filters—is out of scope of this specification.
1056 Similarly, there is no requirement on what criteria (e.g. query expression, FIFO policy) can be used to
1057 select messages when pulling messages from an MPC. This specification only describes the properties
1058 of MPCs, and how their use affects the message protocol. It does not prescribe a particular way to
1059 implement MPCs or to use them.

1060 A message associated with an MPC could fail to be transferred for various reasons (transport issue,
1061 security, intermediaries, etc.) and therefore could be removed from the MPC at any time. In other words,
1062 there is no additional delivery contract for messages over an MPC, other than that specified by the
1063 reliability agreement.

1064 There is no specific quality of service associated with an MPC. Security and reliability remain associated
1065 with parties or with MSHs, in a way that is orthogonal to MPCs; although an implementation is free to
1066 associate QoS with MPCs as long as this conforms to an agreement between parties.

1067

4. Processing Modes

1068 An MSH is operating—either for sending or receiving messages—with knowledge of some contextual
1069 information that controls the way messages are processed. This contextual information that governs the
1070 processing of a particular message is called Processing Mode (or P-Mode). Because different messages
1071 may be subject to different types of processing, an MSH generally supports several P-Modes.

1072 A P-Mode represents some MSH input data that typically is not provided on a per-message basis, but
1073 that is common to a set of messages exchanged between or among parties. To this extent, the P-Mode
1074 may be interpreted as configuration data for a deployed MSH. On a Sending MSH, together with the
1075 information provided by the application layer for each submitted message, the P-Mode fully determines
1076 the content of the message header. For example, the "security" part of the P-Mode will specify
1077 certificates and keys, as well as which messages will be subject to these. This in turn will determine the
1078 content of the Security header. The set of all P-Modes that are supported by an MSH during operation,
1079 is called the P-Mode operation set of the MSH.

1080 The association of a P-Mode with a message may be based on various criteria, usually dependent on
1081 header data (e.g. Service/Action, Conversation ID, or other message properties). Which security and/or
1082 which reliability protocol and parameters, as well as which MEP is being used when sending a message,
1083 is determined by the P-Mode associated with this message.

1084 A data model for P-Modes is described in Appendix D. Although this specification does not require
1085 support for any particular representation of a P-Mode, a conformance profile for this specification may
1086 require support for a particular representation. An MSH MUST conform the processing of its messages to
1087 the values in the P-Mode associated with this message. The details of which P-Mode parameters must
1088 be supported by an implementation, is governed by the features associated with the conformance profile
1089 claimed by this implementation, i.e. by its profile feature set (see Appendix G on Conformance). An MSH
1090 MUST NOT process a message to normal completion if it has no matching P-Mode in its P-Mode
1091 operation set: i.e. the MSH MUST NOT deliver such a message when in Receiving role, or MUST NOT
1092 send it when in Sending role. When it cannot match a message to a P-Mode, an MSH MUST generate a
1093 ProcessingModeMismatch (EBMS:0010) error.

1094 Note:

1095 It is important to distinguish between Conformance Profiles (Appendix G) and P-Modes.
1096 A conformance profile qualifies an MSH implementation and does not vary with the
1097 usage made of the MSH. A P-Mode qualifies the dynamic exchange and processing of
1098 messages, and is generally user defined. It must be within the capabilities allowed by the
1099 conformance profile claimed by the MSH on which it is deployed.

4.1. Messaging Service Processing Model

1100 Although different P-Modes may apply from one message to the other, the overall processing model
1101 remains the same for all messages. The P-Modes set may be seen as configuring the execution
1102 parameters for the general model.

1104 The ebXML Messaging Service may be conceptually broken down into the following three parts:

- 1105 1. an abstract Service Interface,
- 1106 2. functions provided by the MSH and
- 1107 3. the mapping to underlying transport service(s).

1108 Figure 7 depicts a logical arrangement of the functional modules existing within one possible
1109 implementation of the ebXML Messaging Services architecture. These modules are arranged in a
1110 manner to indicate their inter-relationships and dependencies.

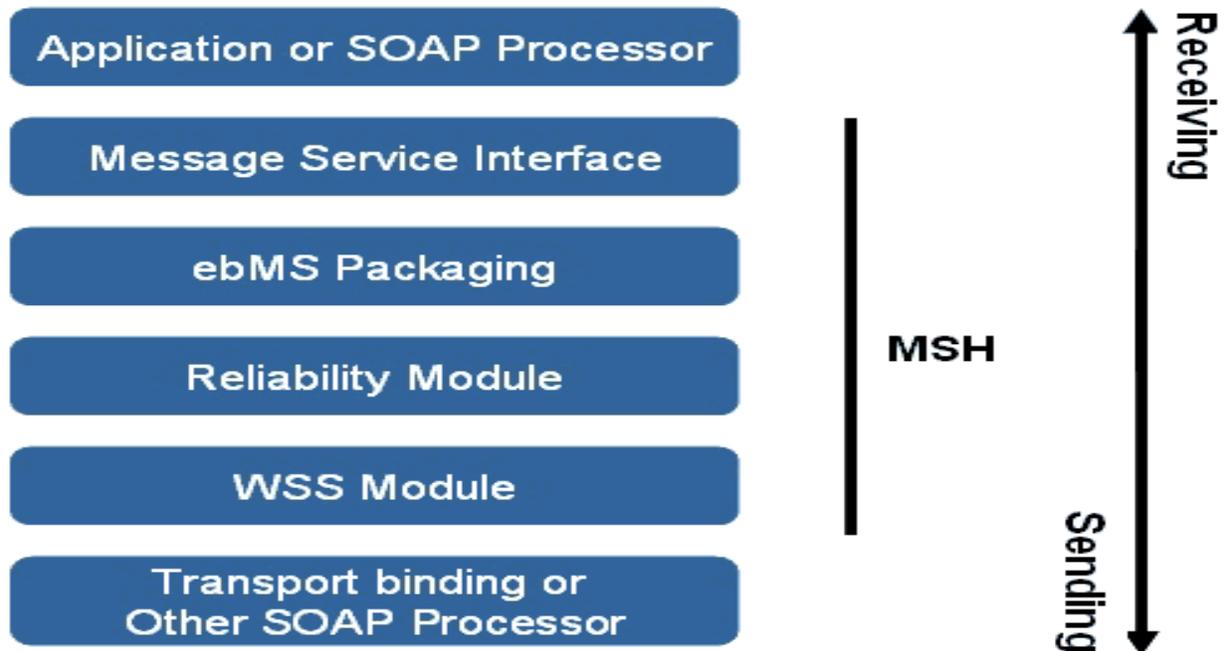


Figure 7: Component Relationships

1111

1112 Following is a description of each module illustrated above. It should be noted that the stack diagram
 1113 above is abstract, and this specification does not mandate that implementations adopt the architecture
 1114 suggested by it, although the processing order shown here is RECOMMENDED, especially in regard to
 1115 Security and Reliability Modules.

- 1116 • **Application or SOAP Processor** - This is where the business logic for a message exchange /
 1117 business process exists.
- 1118 • **Messaging Service Interface** - This is the interface through which messages are channelled
 1119 between the MSH core and the ebXML Application.
- 1120 • **ebMS Packaging** - Handling, (de)enveloping and execution of Payload Services are performed
 1121 by this module.
- 1122 • **Reliable Message Processing** - This module fulfills the Quality of Service requirements for a
 1123 message.
- 1124 • **Web Services Security Processing** - Encryption/decryption of any SOAP message content and
 1125 generation/verification of any digital signatures occurs in this module.
- 1126 • **Transport Protocol Bindings** - These are the actual transport protocol bindings. This
 1127 specification defines bindings for HTTP and SMTP in Appendix C, and supports the addition of
 1128 other protocols.

1129 4.2. Processing Mode Features

1130 The P-Mode is partitioned into functional groups called P-Mode features. Each P-Mode feature covers
 1131 one of the functional areas that is critical to achieving interoperability between two partners: security,
 1132 reliability, transport, business collaboration, error reporting, Message Exchange Patterns (MEPs) and
 1133 Message Partition Channels (MPCs).

1134 The main P-Mode features are here identified by names of the form: P-Mode.<featurename>:

- 1135 • **P-Mode.Protocol:** includes all transport related information that is necessary to achieve
1136 transport-level interoperability. This feature determines the type of transport involved (e.g. HTTP,
1137 SMTP, FTP) between two MSHs, and related configuration parameters. This feature usually
1138 treats all messages between two MSHs similarly. It also includes information about which SOAP
1139 version is to be used (SOAP 1.1 or SOAP 1.2).
- 1140 • **P-Mode.Reliability:** includes all reliability contracts, or references to them, that will govern the
1141 reliability of messages exchanged. This feature determines the content of the reliability headers.
- 1142 • **P-Mode.Security:** includes all security contracts, or references to them, including the security
1143 context and related resources (certificates, SAML assertions, etc.) that govern the message
1144 exchange. This feature determines the content of the wsse:Security header.
- 1145 • **P-Mode.BusinessInfo:** includes all message-relevant data related to a collaboration between
1146 two parties. It also indicates which MPCs are to be used by these parties. This feature will
1147 complement or validate message data that is expected to be provided by the application on a
1148 per-message basis for these header elements:
 - 1149 • eb:UserMessage/eb:PartyInfo
 - 1150 • eb:UserMessage/eb:CollaborationInfo
 - 1151 • eb:UserMessage/eb:MessageProperties
- 1152 • **P-Mode.ErrorHandling:** defines how each ebMS Error type is to be reported by this MSH. E.g. if
1153 the reporting is done using ebMS signal messages, it defines the address of the destination
1154 MSH. It also may include the policy chosen for raising ebMS Errors from the errors generated by
1155 functional modules (Reliability, Security). This P-Mode feature must define reporting mode
1156 parameters that will allow a Receiving MSH to decide:
 - 1157 • whether an error generated on reception of a message must be returned as response
1158 over the same SOAP MEP. (e.g. errorHandling.report.asResponse = true/false).
 - 1159 • whether an error generated on reception of a message must be returned to sender or to
1160 a third party over a new SOAP MEP. (e.g. errorHandling.report.ReceiverErrorsTo =
1161 <URL>).
 - 1162 • whether the Consumer and/or Producer (e.g.
1163 errorHandling.Report.ProcessErrorNotifyConsumer) of a message must be notified of an
1164 error generated on reception of the message.

1165 In this specification, a P-Mode feature is abstractly considered to apply to both sending and receiving
1166 roles, although implementations may choose to represent only the subset relevant to the role in which
1167 they operate. A single P-Mode instance is also intended to govern all messages involved in an ebMS
1168 MEP. (The ebMS MEP and its transport channel binding are attributes of a P-Mode.) Because messages
1169 involved in an MEP (e.g. request and reply) may use different qualities of service, a single P-Mode may
1170 use different vectors of values for its parameters, depending on the message in the MEP. An outline of
1171 the data model for P-Modes is given in Appendix D.

1172 Agreeing on a P-Mode operation set is essential for two parties in order for their MSHs to interoperate.
1173 P-Modes are the MSH-level expression of a prior agreement between partners. A reference to such an
1174 agreement may be present in the message header (see eb:AgreementRef element in Section 5.2.2.7).

1175 **4.3. Default Features for Processing Mode**

1176 In order to facilitate interoperability testing, or during the early phase of a deployment, it may be useful to
1177 drive message exchanges without relying on user-agreed P-Modes, without interfacing with any
1178 application, and (initially) without the added complexity of security and reliability features. To this end, a
1179 default semantics of each P-Mode feature is defined as follows:

- 1180 • **Default P-Mode.MEP:** [http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/oneWay](http://docs.oasis-open.org/ebxml-
1181 msg/ebms/v3.0/ns/core/200704/oneWay)
- 1182 • **Default P-Mode.MEPbinding:** [http://docs.oasis-open.org/ebxml-
msg/ebms/v3.0/ns/core/200704/push](http://docs.oasis-open.org/ebxml-
1183 msg/ebms/v3.0/ns/core/200704/push)
- 1184 • **Default P-Mode.Protocol:** HTTP 1.1 transport is assumed, with default configuration (on

- 1185 standard port), using SOAP 1.2.
- 1186 • **Default P-Mode.Reliability:** No reliable messaging assumed (no reliability header will be
1187 present).
- 1188 • **Default P-Mode.Security:** No secure messaging assumed (no security header will be present.)
- 1189 • **Default P-Mode.BusinessInfo:** In the absence of any application input at message level as well
1190 as for this P-Mode feature, the following default header element values will be used (shown here
1191 for a message sent by an Initiator to a Responder party). Any of these may be overridden by
1192 application input.
- 1193 • **eb:UserMessage/eb:PartyInfo:** The eb:From element contains a PartyId with value:
1194 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultFrom>
1195 The eb:To element contains a PartyId with value:
1196 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultTo>
 - 1197 • **eb:UserMessage/eb:CollaborationInfo:** Contains no eb:AgreementRef. The eb:Service
1198 element has the value:
1199 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service>
1200 The eb:Action element has the value:
1201 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test>
1202 (Section 5.2.2 details the semantics of these values.)
1203 The eb:ConversationId element has the value: 1.
1204 The default MPC is in use.
 - 1205 • **eb:UserMessage/eb:MessageProperties:** This element is absent.
 - 1206 • **eb:UserMessage/eb:PayloadInfo:** This element is absent.
- 1207 • **Default P-Mode.ErrorHandling:** No reporting via ebMS message is required. The MSH may
1208 handle error reporting in a way that does not involve the partner MSH, such as notification to
1209 local Consumer or Producer.
- 1210 In the absence of a user-agreed P-Mode feature, it is RECOMMENDED that an MSH operate based on
1211 the above default semantics for this feature except in the following cases:
- 1212 1. The MSH is designed to conform to this specification along profiles (see Appendix G) that are
1213 not compatible with the default P-Mode feature. For example, such an incompatibility would
1214 occur for the default P-Mode.MEP with a conformance profile that only requires the One-
1215 Way/Pull MEP.
 - 1216 2. The MSH has been pre-configured to operate with a non-default P-Mode feature. This would be
1217 the case when an MSH is distributed along with a predefined P-Mode feature, e.g. built-in
1218 security. This amounts to using a user-defined P-Mode feature.
- 1219 A Sending MSH and a Receiving MSH may use a mix of default and non-default P-Mode features.

1220 5. Message Packaging

1221 5.1. Message Envelope and Message Parts

1222 5.1.1. MIME Structure and SOAP Profile

1223 In the ebMS SOAP header eb:Messaging, the prefix "eb" is an example prefix that corresponds to the
1224 ebMS 3.0 namespace, as defined in Section 1.6. The ebMS Message can be packaged as a plain
1225 [SOAP11] or [SOAP12] message, or within a MIME multipart to allow payloads or attachments to be
1226 included. Because either packaging option can be used, implementations MUST support both multipart
1227 and non-multipart messages.

1228 The ebMS Message MAY contain SOAP extension elements other than the eb:Messaging header block.
1229 For example, header blocks supporting message reliability and message security MAY be produced and
1230 consumed by an MSH in order to fulfill deployment requirements for those features.

1231 An ebMS Message is packaged as a SOAP 1.1 or 1.2 message independent from communications
1232 protocols. When represented as a MIME multipart message envelope, this envelope MUST be structured
1233 in compliance with the SOAP Messages with Attachments [SOAPATTACH] W3C Note, referred to as a
1234 Message Package.

1235 There are two logical sections within the Message Package:

- 1236 • The first section is the ebMS Header (i.e. The eb:Messaging SOAP header block), itself
1237 contained in the SOAP Header.
- 1238 • The second section is the ebMS Payload, which itself comprises two sections: (a) the SOAP
1239 Body element within the SOAP Envelope, and in case of MIME packaging, (b) zero or more
1240 additional MIME parts containing additional application-level payloads. The SOAP Body and
1241 MIME parts are also referred to as ebMS Payload Containers. The SOAP Body is the only
1242 payload container that requires XML-structured content, though non-XML content may be
1243 included within an appropriately typed (binary or otherwise) element inside the Body.

1244 The general structure and composition of an ebMS User Message is described in Figure 8, and a Signal
1245 Message in Figure 9.

1247

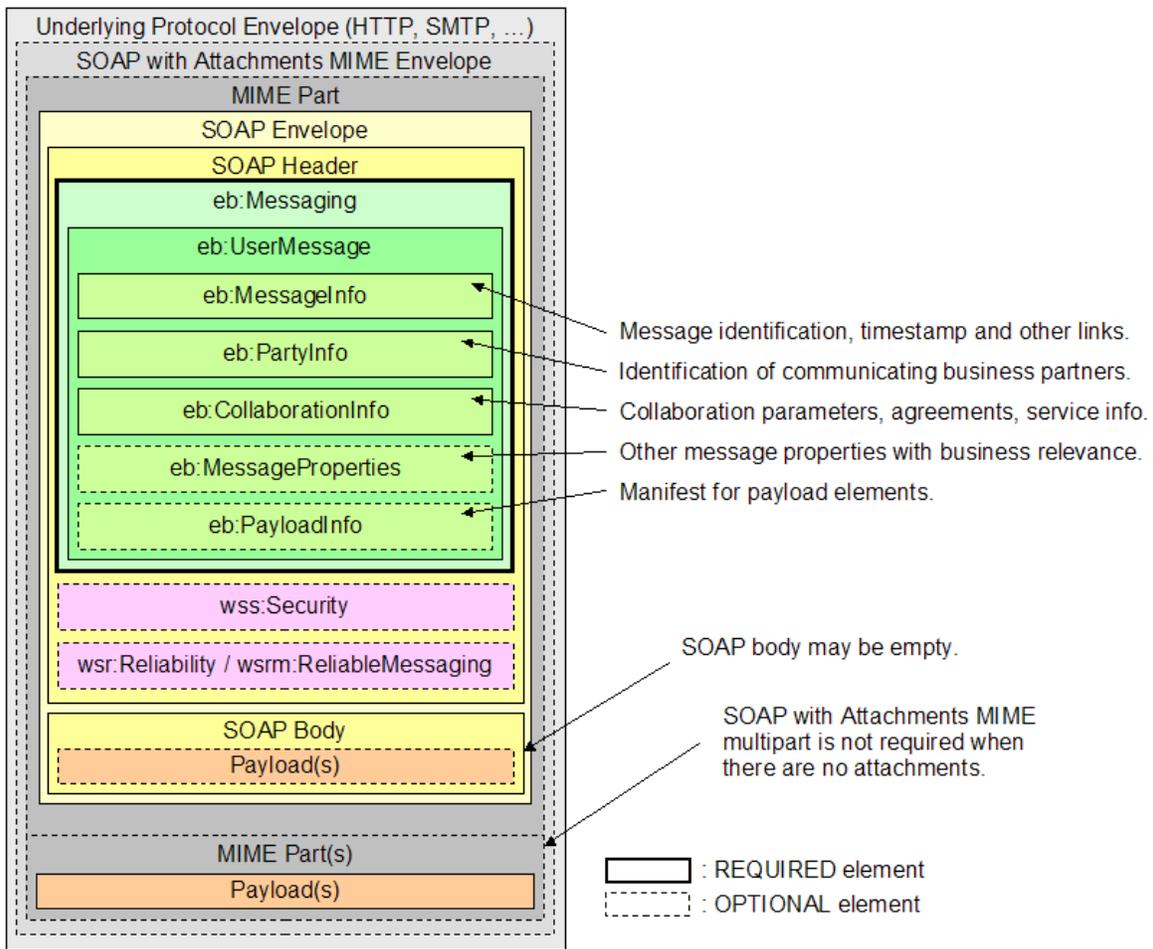


Figure 8: User Message Structure

1248

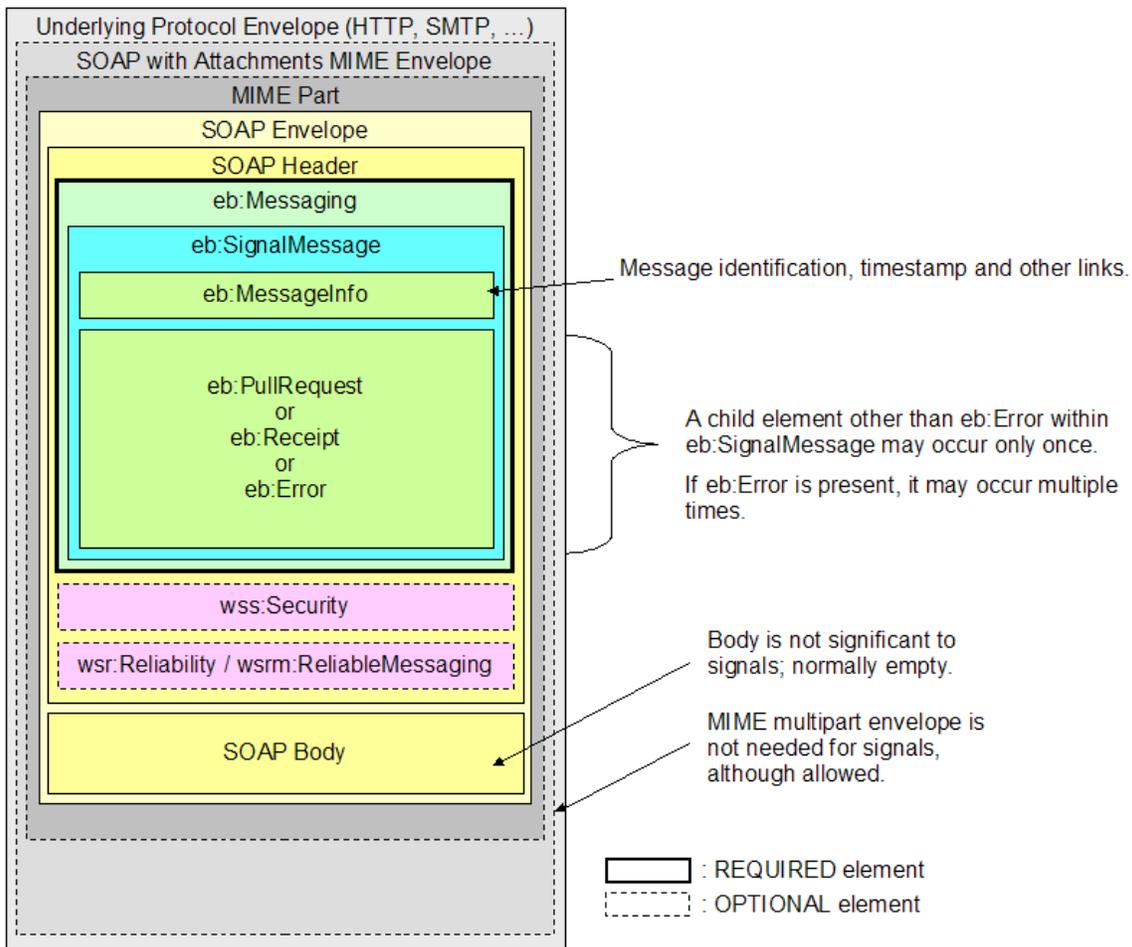


Figure 9: Signal Message Structure

1249 The processing of the SOAP eb:Messaging header block is done according to the SOAP processing
 1250 semantics: an MSH behaves as a SOAP processor or SOAP node that MUST understand this header
 1251 block. Other header blocks (except for those relevant to reliability and security of an ebMS Message)
 1252 are not affected by the ebXML processing. Consequently, it is possible for a Sending MSH implementation to
 1253 generate an ebMS message from a well-formed input SOAP message simply by adding an eb:Messaging
 1254 header; likewise, some Receiving MSH implementation could deliver a well-formed SOAP message as
 1255 output by removing (and processing) the eb:Messaging header.

1256 All MIME headers of the Message Package MUST conform with the SOAP Messages with Attachments
 1257 [SOAPATTACH] W3C Note. In addition, the Content-Type MIME header of the Message Package MUST
 1258 contain a type parameter whose value matches the MIME media type of the MIME body part containing
 1259 the SOAP Envelope document. In accordance with the [SOAP11] specification, the MIME media type of
 1260 the SOAP 1.1 Message has the value "text/xml". It is STRONGLY RECOMMENDED that the initial
 1261 headers contain a Content-ID MIME header structured in accordance with MIME [RFC2045], and in
 1262 addition to the required parameters for the Multipart/Related media type, the start parameter (OPTIONAL
 1263 in MIME Multipart/Related [RFC2387]) be present. This permits more robust error detection. The
 1264 following fragment is an example of the MIME headers for the multipart/related Message Package:

1265 Example 1. MIME Header fragment for the multipart/related Message Package

```
1266 Content-Type: multipart/related; type="text/xml";
1267 boundary="boundaryValue"; start="<messagepackage-123@example.com>"
1268 --boundaryValue
```

1269 `Content-ID: messagepackage-123@example.com`

1270 Because implementations MUST support non-multipart messages, an ebMS Message with no payload
1271 may be sent either as a plain SOAP message or as a [SOAPATTACH] multipart message with only one
1272 body part (the SOAP Envelope).

1273 **5.1.2. MIME and XML Considerations**

1274 This section contains further MIME- and XML-specific packaging requirements and guidance.

1275 **5.1.2.1. Additional MIME Parameters**

1276 Any MIME part described by this specification MAY contain additional MIME headers in conformance
1277 with the MIME [RFC2045] specification. Implementations MAY ignore any MIME header not defined in
1278 this specification. Implementations MUST ignore any MIME header they do not recognize. For example,
1279 an implementation could include Content-Length in a message. However, a recipient of a message with
1280 Content-Length could ignore it.

1281 **5.1.2.2. Reporting MIME Errors**

1282 If a MIME error is detected in the Message Package then it MUST be reported as specified in SOAP with
1283 Attachments [SOAPATTACH].

1284 **5.1.2.3. XML Prolog**

1285 The SOAP Message's XML Prolog, if present, MAY contain an XML declaration. This specification has
1286 defined no additional comments or processing instructions appearing in the XML prolog. For example:

```
1287 Content-Type: text/xml; charset="UTF-8"  
1288 <?xml version="1.0" encoding="UTF-8" ?>  
1289
```

1290 **5.1.2.4. XML Declaration**

1291 The XML declaration MAY be present in a SOAP Message. If present, it MUST contain the version
1292 specification required by the XML Recommendation [XML10] and MAY contain an encoding declaration.
1293 The semantics described below MUST be implemented by a compliant ebXML Message Service.

1294 **5.1.2.5. Encoding Declaration**

1295 If both the encoding declaration and the MIME root part charset parameter are present, the XML prolog
1296 for the SOAP Message SHALL contain the encoding declaration, and SHALL be equivalent to the
1297 charset attribute of the MIME Content-Type of the root part (see Section 5.1.4). If provided, the encoding
1298 declaration MUST NOT contain a value conflicting with the encoding used when creating the SOAP
1299 Message. It is RECOMMENDED that UTF-8 be used when encoding the SOAP Message. If the
1300 character encoding cannot be determined by an XML processor using the rules specified in section 4.3.3
1301 of XML [XML10], the XML declaration and its contained encoding declaration SHALL be provided in the
1302 ebXML SOAP Header Document. **Note:** The encoding declaration is not required in an XML document
1303 according to XML v1.0 specification [XML10].

1304 **5.1.3. ebXML SOAP Envelope Extension**

1305 In conformance with the [XML10] specification, all extension element content is namespace qualified. A
1306 namespace declaration (xmlns pseudo-attribute) for the ebXML SOAP extension may be included in the
1307 SOAP Envelope or Header element, or directly in the ebXML SOAP extension element.

1308 **5.1.3.1. namespace Pseudo Attribute**

1309 The namespace declaration for the ebXML SOAP Envelope extension (xmlns pseudo attribute) (see

1310 [XMLNS]) has a REQUIRED value of:

1311 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/>

1312 5.1.3.2. xsi:schemaLocation attribute

1313 The SOAP namespace:

1314 <http://schemas.xmlsoap.org/soap/envelope/>

1315 resolves to a W3C XML Schema specification. It is STRONGLY RECOMMENDED that ebXML MSH
1316 implementations include the XMLSchema-instance namespace qualified schemaLocation attribute in the
1317 SOAP Envelope element, to indicate to validating parsers a location of the schema document that should
1318 be used to validate the document. Failure to include the schemaLocation attribute could prevent XML
1319 schema validation of received messages.

1320 For example:

```
1321 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"  
1322   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1323   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/  
1324     http://schemas.xmlsoap.org/soap/envelope/">  
1325   <S11:Header/>  
1326   <S11:Body/>  
1327 </S11:Envelope>
```

1328 In addition, the ebXML SOAP Header extension element content MAY be similarly qualified, so as to
1329 identify the location where validating parsers can find the schema document containing the ebXML
1330 namespace-qualified SOAP extension element definition. The ebXML SOAP extension element schema,
1331 found in Appendix A, has been defined using the W3C Recommendation version of the XML Schema
1332 specification [XMLSCHEMA]. The XMLSchema-instance namespace qualified schemaLocation attribute
1333 should include a mapping of the ebXML SOAP Envelope extension namespace to its schema document
1334 in the same element that declares the ebXML SOAP Envelope extensions namespace.

1335 The schemaLocation for the namespace described in Section 5.1.3.1 is:

1336 http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd

1337 Separate schemaLocation attributes are RECOMMENDED. For example:

```
1338 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"  
1339   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1340   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/  
1341     http://schemas.xmlsoap.org/soap/envelope/">  
1342   <S11:Header>  
1343     <eb:Messaging xmlns:eb="http://docs.oasis-open.org/ebxml-  
1344     msg/ebms/v3.0/ns/core/200704/"  
1345       xsi:schemaLocation=  
1346         "http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/  
1347         http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-  
1348         3_0-200704.xsd">  
1349       <eb:UserMessage>  
1350         <eb:MessageInfo >...</eb:MessageInfo>  
1351         ...  
1352         <eb:PayloadInfo >...</eb:PayloadInfo>  
1353         ...  
1354       </eb:UserMessage>  
1355     </eb:Messaging>  
1356   </S11:Header>  
1357   <S11:Body>  
1358     ...  
1359   </S11:Body>  
1360 </S11:Envelope>
```

1361 5.1.3.3. SOAP Header Element

1362 The SOAP Header element is the first child element of the SOAP Envelope element. It MUST have a
1363 namespace qualifier that matches the SOAP Envelope namespace declaration for the namespace

1364 "http://schemas.xmlsoap.org/soap/envelope/".

1365 **5.1.3.4. SOAP Body Element**

1366 The SOAP Body element is the second child element of the SOAP Envelope element. It MUST have a
1367 namespace qualifier that matches the SOAP Envelope namespace declaration for the namespace
1368 "http://schemas.xmlsoap.org/soap/envelope/".

1369 Note:

1370 Unlike ebMS v2, ebXML Messaging 3.0 does not define or make use of any elements
1371 within the SOAP Body, which is wholly reserved for user-specified payload data.

1372 **5.1.3.5. ebXML SOAP Extensions**

1373 An ebMS Message extends the SOAP Message with the extension element eb:Messaging, where "eb" is
1374 the namespace prefix for ebMS 3.0.

1375 Other headers that support some aspects of ebMS messaging, such as the security header
1376 (wsse:Security) and reliability headers, may be present. These are not qualified under the ebMS
1377 namespace.

1378 **5.1.4. ebMS Header**

1379 In case of MIME packaging, the root body part of the Message Package is the SOAP message, as
1380 defined in the SOAP Messages with Attachments [SOAPATTACH] W3C Note. This root part always
1381 contains the ebMS header.

1382 The MIME Content-Type header for the root part MUST have the value "text/xml" to match the MIME
1383 media type of the MIME body part containing the [SOAP11] Message document, or
1384 "application/soap+xml" in the case of a [SOAP12] body. The Content-Type header MAY contain a
1385 "charset" parameter. For example:

```
1386 Content-Type: text/xml; charset="UTF-8"
```

1387 The MIME charset parameter identifies the character set used to create the SOAP Message. The
1388 semantics of this attribute are described in the "charset parameter / encoding considerations" of text/xml
1389 as specified in [RFC3023]. The list of valid values can be found at [IANAMEDIA].

1390 If both are present, the value of the MIME charset parameter SHALL be equivalent to the encoding
1391 declaration of the SOAP Message. If provided, the MIME charset parameter MUST NOT contain a value
1392 conflicting with the encoding used when creating the SOAP Message.

1393 For maximum interoperability it is RECOMMENDED UTF-8 [UTF8] be used when encoding this
1394 document. Due to the processing rules defined for media types derived from text/xml [RFC3023], this
1395 MIME attribute has no default.

1396 The following fragment represents an example of a root part, for a MIME packaging of ebMS:

```
1397 Content-ID: <messagepackage-123@example.com>  
1398 Content-Type: text/xml; charset="UTF-8"  
1399  
1400 <S11:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">  
1401   <S11:Header>  
1402     <eb:Messaging>  
1403       ...  
1404     </eb:Messaging>  
1405   </S11:Header>  
1406   <S11:Body>  
1407     ...  
1408   </S11:Body>  
1409 </S11:Envelope>
```

1410 **5.1.5. Payload Containers**

1411 In addition to the SOAP Body, other Payload Containers MAY be present within a Message Package in
1412 conformance with the SOAP Messages with Attachments [SOAPATTACH] specification.

1413 If there is no application payload within the Message Package, then the SOAP Body MUST be empty,
 1414 and there MUST NOT be additional Payload Containers.

1415 There SHOULD also be no additional MIME attachments that are not Payload Containers (i.e., that are
 1416 not referenced by an eb:PayloadInfo element, as described in Section 5.2.2.12); but if any such
 1417 attachments are present, they are outside the scope of MSH processing. An MSH MUST NOT process
 1418 application data that is not referenced by eb:PayloadInfo.

1419 The contents of each Payload Container (including the SOAP Body) MUST be identified in the
 1420 /eb:Messaging/eb:UserMessage/eb:PayloadInfo element.

1421 The ebXML Messaging Service Specification makes no provision, nor limits in any way, the structure or
 1422 content of application payloads. Payloads MAY be simple, plain-text objects or complex, nested,
 1423 multipart objects. The specification of the structure and composition of payload objects is the prerogative
 1424 of the organization defining the business process or information exchange using the ebXML Messaging
 1425 Service.

1426 **Example of SOAP Message containing an ebMS header:**

```

1427 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
1428   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1429   xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
1430     http://schemas.xmlsoap.org/soap/envelope/"
1431   <S11:Header
1432     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
1433     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
1434       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
1435     <eb:Messaging S11:mustUnderstand="1">
1436       <eb:UserMessage>
1437         ...
1438         <eb:PayloadInfo>
1439           ...
1440         </eb:PayloadInfo>
1441         ...
1442       </eb:UserMessage>
1443     </eb:Messaging>
1444   </S11:Header>
1445   <S11:Body>
1446     ...
1447   </S11:Body>
1448 </S11:Envelope>
  
```

1449 **5.2. The eb:Messaging Container Element**

1450 The REQUIRED eb:Messaging element is a child of the SOAP Header. It is a container for either a User
 1451 message or a Signal message.

1452 In the case of a User message, the ebXML header block contains an eb:UserMessage child element:

```

1453 <eb:Messaging>
1454   <eb:UserMessage>
1455     <eb:MessageInfo>
1456       <!-- some headers here like Timestamp and MessageId -->
1457     </eb:MessageInfo>
1458     <!-- header elements of the ebMS user message -->
1459   </eb:UserMessage>
1460 </eb:Messaging>
  
```

1461 In the case of a Signal message, the ebXML header block (eb:Messaging) contains at least one
 1462 eb:SignalMessage child element:

```

1463 <eb:Messaging>
1464   <eb:SignalMessage>
1465     <eb:MessageInfo>
1466       <!-- some headers here like Timestamp and MessageId -->
1467     </eb:MessageInfo>
1468     <eb:signalname>
1469       <!-- header elements of this ebMS signal message -->
1470     </eb:signalname>
1471   </eb:SignalMessage>
  
```

1472 </eb:Messaging>

1473 For example, *signalname* can be "PullRequest".

1474 5.2.1. eb:Messaging Element Specification

1475 The eb:Messaging element has the following attributes:

- 1476 • eb:Messaging/@S11:mustUnderstand: indicates whether the contents of the element
1477 MUST be understood by the MSH. This attribute is REQUIRED, with namespace qualified to the
1478 SOAP namespace (<http://schemas.xmlsoap.org/soap/envelope/>). It MUST have value of '1' (true)
1479 indicating the element MUST be understood or rejected.

1480 The eb:Messaging element has the following children elements:

- 1481 • eb:Messaging/eb:UserMessage: The OPTIONAL UserMessage element contains all header
1482 information for a User message. If this element is not present, an element describing a Signal
1483 message MUST be present.
- 1484 • eb:Messaging/eb:SignalMessage/eb:[*signalname*]: The OPTIONAL element is named
1485 after a type of Signal message. It contains all header information for the Signal message. If this
1486 element is not present, an element describing a User message MUST be present. Three types
1487 of Signal messages are specified in this document: Pull signal (eb:PullRequest), Error signal
1488 (eb:Error) and Receipt signal (eb:Receipt).

1489 Both eb:UserMessage element and eb:SignalMessage element MAY be present within the eb:Messaging
1490 element.

1491

1492 Example ebMS Message Header:

```
1493 <!-- (contained within S11:Header) -->
1494
1495 <eb:Messaging S11:mustUnderstand="1" >
1496
1497 <eb:UserMessage>
1498
1499 <eb:MessageInfo>
1500 <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
1501 <eb:MessageId>UUID-2@example.com</eb:MessageId>
1502 <eb:RefToMessageId>UUID-1@example.com</eb:RefToMessageId>
1503 </eb:MessageInfo>
1504
1505 <eb:PartyInfo>
1506 <eb:From>
1507 <eb:PartyId>uri:example.com</eb:PartyId>
1508 <eb:Role>http://example.org/roles/Buyer</eb:Role>
1509 </eb:From>
1510
1511 <eb:To>
1512 <eb:PartyId type="someType">QRS543</eb:PartyId>
1513 <eb:Role>http://example.org/roles/Seller</eb:Role>
1514 </eb:To>
1515 </eb:PartyInfo>
1516
1517 <eb:CollaborationInfo>
1518 <eb:AgreementRef>http://registry.example.com/cpa/123456
1519 </eb:AgreementRef>
1520 <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
1521 <eb:Action>NewPurchaseOrder</eb:Action>
1522 <eb:ConversationId>4321</eb:ConversationId>
1523 </eb:CollaborationInfo >
1524
1525 <eb:MessageProperties>
1526 <eb:Property name="ProcessInst">PurchaseOrder:123456
1527 </eb:Property>
1528 <eb:Property name="ContextID"> 987654321
1529 </eb:Property>
1530 </eb:MessageProperties >
1531
1532 <eb:PayloadInfo>
1533 <eb:PartInfo href="cid:foo@example.com">
1534 <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>
```

```

1535         <eb:Description xml:lang="en-US">Purchase Order for 100,000 foo
1536 widgets</eb:Description>
1537     </eb:PartInfo>
1538     <eb:PartInfo href="#idref">
1539     </eb:PartInfo>
1540 </eb:PayloadInfo>
1541
1542 </eb:UserMessage>
1543 </eb:Messaging>
1544

```

1545 **5.2.2. eb:Messaging/eb:UserMessage**

1546 This element has the following attributes:

- 1547 • eb:Messaging/eb:UserMessage/@mpc: This OPTIONAL attribute contains a URI that
1548 identifies the Message Partition Channel to which the message is assigned. The absence of this
1549 element indicates the use of the default MPC. When the message is pulled, the value of this
1550 attribute MUST indicate the MPC requested in the PullRequest message.

1551 This element has the following children elements:

- 1552 • eb:Messaging/eb:UserMessage/eb:MessageInfo: This REQUIRED element occurs once,
1553 and contains data that identifies the message, and relates to other messages' identifiers.
- 1554 • eb:Messaging/eb:UserMessage/eb:PartyInfo: This REQUIRED element occurs once,
1555 and contains data about originating party and destination party.
- 1556 • eb:Messaging/eb:UserMessage/eb:CollaborationInfo: This REQUIRED element
1557 occurs once, and contains elements that facilitate collaboration between parties.
- 1558 • eb:Messaging/eb:UserMessage/eb:MessageProperties: This OPTIONAL element
1559 occurs at most once, and contains message properties that are user-specific. As parts of the
1560 header such properties allow for more efficient monitoring, correlating, dispatching and validating
1561 functions (even if these are out of scope of ebMS specification) which would otherwise require
1562 payload access.
- 1563 • eb:Messaging/eb:UserMessage/eb:PayloadInfo: This OPTIONAL element occurs at
1564 most once, and identifies payload data associated with the message, whether included as part of
1565 the message as payload document(s) contained in a Payload Container, or remote resources
1566 accessible via a URL. The purpose of the PayloadInfo is (a) to make it easier to directly extract a
1567 particular payload associated with this User message, (b) to allow an application to determine
1568 whether it can process the payload without having to parse it.

1569 **5.2.2.1. eb:Messaging/eb:UserMessage/eb:MessageInfo**

1570 This element has the following children elements:

- 1571 • eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:Timestamp: The REQUIRED
1572 Timestamp element has a value representing the date at which the message header was
1573 created, and is conforming to a dateTime (see [XMLSCHEMA]). It MUST be expressed as UTC.
1574 Indicating UTC in the Timestamp element by including the 'Z' identifier is optional.
- 1575 • eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:MessageId: This REQUIRED
1576 element has a value representing – for each message - a globally unique identifier conforming to
1577 MessageId [RFC2822]. Note: In the Message-Id and Content-Id MIME headers, values are
1578 always surrounded by angle brackets. However references in mid: or cid: scheme URI's and the
1579 MessageId and RefToMessageId elements MUST NOT include these delimiters.
- 1580 • eb:Messaging/eb:UserMessage/eb:MessageInfo/eb:RefToMessageId: This
1581 OPTIONAL element occurs at most once. When present, it MUST contain the MessageId value
1582 of an ebMS Message to which this message relates, in a way that conforms to the MEP in use
1583 (see Section C.3).

1584 **5.2.2.2. eb:Messaging/eb:UserMessage/eb:PartyInfo**

1585 This element has the following children elements:

- 1586 • eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From: The REQUIRED element
1587 occurs once, and contains information describing the originating party.
- 1588 • eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To: The REQUIRED element occurs
1589 once, and contains information describing the destination party.

1590 **5.2.2.3. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From**

1591 This element has the following children elements:

- 1592 • eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId: The
1593 REQUIRED PartyId element occurs one or more times. If it occurs multiple times, each instance
1594 MUST identify the same organization.
- 1595 • eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role: The REQUIRED
1596 eb:Role element occurs once, and identifies the authorized role (fromAuthorizedRole or
1597 toAuthorizedRole) of the Party sending (when present as a child of the From element) or
1598 receiving (when present as a child of the To element) the message. The value of the Role
1599 element is a non-empty string, with a default value of
1600 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/defaultRole> .
1601 Other possible values are subject to partner agreement.

1602 **Example:** The following fragment demonstrates usage of the From element.

```
1603 <eb:From>  
1604   <eb:PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">  
1605     123456789</eb:PartyId>  
1606     <eb:PartyId type="SCAC">RDWY</eb:PartyId>  
1607     <eb:Role>http://example.org/roles/Buyer</eb:Role>  
1608   </eb:From>
```

1609 **5.2.2.4. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:PartyId**

1610 This element has a string value content that identifies a party, or that is one of the identifiers of this party.

1611 It has a single attribute, @type. The type attribute indicates the domain of names to which the string in
1612 the content of the PartyId element belongs. It is RECOMMENDED that the value of the type attribute be
1613 a URI. It is further RECOMMENDED that these values be taken from the EDIRA , EDIFACT or ANSI ASC
1614 X12 registries. Technical specifications for the first two registries can be found at and [ISO6523] and
1615 [ISO9735], respectively. Further discussion of PartyId types and methods of construction can be found in
1616 an appendix of [ebCPPA21]. The value of any given @type attribute MUST be unique within a list of
1617 PartyId elements.

1618 An example of PartyId element is:

```
1619 <eb:PartyId type="urn:oasis:names:tc:ebxml-cppa:partyid-type:duns">  
1620   123456789</eb:PartyId>
```

1621 If the eb:PartyId/@type attribute is not present, the content of the PartyId element MUST be a URI
1622 [RFC2396], otherwise the Receiving MSH SHOULD report a "ValueInconsistent" error with severity
1623 "error". It is strongly RECOMMENDED that the content of the eb:PartyId element be a URI.

1624 **5.2.2.5. eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To**

1625 This element has the same children elements as

1626 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From, above in Section 5.2.2.3.

1627 **Example:** The following fragment demonstrates usage of the To element.

```
1628 <eb:To>  
1629   <eb:PartyId>mailto:joe@example.com</eb:PartyId>  
1630   <eb:Role>http://example.org/roles/Seller</eb:Role>
```

1631 `</eb:To>`

1632 **5.2.2.6. eb:Messaging/eb:UserMessage/eb:CollaborationInfo**

1633 This element has the following children elements:

- 1634 • `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef`: This
1635 OPTIONAL element occurs zero or once. The AgreementRef element is a string that identifies
1636 the entity or artifact governing the exchange of messages between the parties.
- 1637 • `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service`: This
1638 REQUIRED element occurs once. It is a string identifying the service that acts on the message
1639 and it is specified by the designer of the service.
- 1640 • `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action`: This REQUIRED
1641 element occurs once. The element is a string identifying an operation or an activity within a
1642 Service that may support several of these.
- 1643 • `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationId`: This
1644 REQUIRED element occurs once. The element is a string identifying the set of related messages
1645 that make up a conversation between Parties.

1646 **5.2.2.7. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:AgreementRef**

1647 AgreementRef is a string value that identifies the agreement that governs the exchange. The P-Mode
1648 under which the MSH operates for this message should be aligned with this agreement.

1649 The value of an AgreementRef element MUST be unique within a namespace mutually agreed by the two
1650 parties. This could be a concatenation of the From and To PartyId values, a URI containing the Internet
1651 domain name of one of the parties, or a namespace offered and managed by some other naming or
1652 registry service. It is RECOMMENDED that the AgreementRef be a URI. The AgreementRef MAY
1653 reference an instance of a CPA as defined in [ebCPPA].

1654 An example of the AgreementRef element follows:

```
1655 <eb:AgreementRef>http://registry.example.com/cpas/our_cpa.xml</eb:AgreementRef>
```

1656 If a CPA is referred to and a Receiving MSH detects an inconsistency, then it MUST report it with an
1657 "ValueInconsistent" error of severity "error". If the AgreementRef is not recognized, then the
1658 Receiving MSH MUST report it as a "ValueNotRecognized" error of severity "error".

1659 The AgreementRef element may have two attributes:

- 1660 • `@type`: This OPTIONAL attribute indicates how the parties sending and receiving the message
1661 will interpret the value of the reference (e.g. the value could be "ebcpa2.1" for parties using a
1662 CPA-based agreement representation). There is no restriction on the value of the type attribute;
1663 this choice is left to profiles of this specification. If the type attribute is not present, the content of
1664 the `eb:AgreementRef` element MUST be a URI. If it is not a URI, then the MSH MUST report a
1665 "ValueInconsistent" error of severity "error".
- 1666 • `@pmode`: This OPTIONAL attribute allows for explicit association of a message with a P-Mode.
1667 When used, its value contains the PMode.ID parameter.

1668 **5.2.2.8. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service**

1669 This element identifies the service that acts on the message. Its actual semantics is beyond the scope of
1670 this specification. The designer of the service may be a standards organization, or an individual or
1671 enterprise.

1672 Examples of the Service element include:

```
1673 <eb:Service>urn:example.org:services:SupplierOrderProcessing</eb:Service>
```

1674

```
1675 <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
```

1676 The Service element MAY contain a single `@type` attribute, that indicates how the parties sending and

1677 receiving the message will interpret the value of the element. There is no restriction on the value of the
1678 type attribute. If the type attribute is not present, the content of the Service element MUST be a URI (see
1679 [RFC2396]). If it is not a URI then the MSH MUST report a "ValueInconsistent" error of severity
1680 "error".

1681 When the value of the element is `http://docs.oasis-open.org/ebxml-`
1682 `msg/ebms/v3.0/ns/core/200704/service`, then the receiving MSH MUST NOT deliver this
1683 message to the Consumer. With the exception of this delivery behavior, and unless indicated otherwise
1684 by the `eb:Action` element, the processing of the message is not different from any other user message.

1685 **5.2.2.9. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action**

1686 This element is a string identifying an operation or an activity within a Service. Its actual semantics is
1687 beyond the scope of this specification. Action SHALL be unique within the Service in which it is defined.
1688 The value of the Action element is specified by the designer of the service.

1689 An example of the Action element follows:

```
1690 <eb:Action>NewOrder</eb:Action>
```

1691 If the value of either the Service or Action element is unrecognized by the Receiving MSH, then it MUST
1692 report a "ValueNotRecognized" error of severity "error".

1693 When the value of this element is `http://docs.oasis-open.org/ebxml-`
1694 `msg/ebms/v3.0/ns/core/200704/test`, then the `eb:Service` element MUST have the value
1695 `http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/service`. Such a
1696 value for the `eb:Action` element only indicates that the user message is sent for testing purposes and
1697 does not require any specific handling by the MSH.

1698 **5.2.2.10. eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:ConversationId**

1699 This element is a string identifying the set of related messages that make up a conversation between
1700 Parties.

1701 If a CPA is referred to by `eb:AgreementRef`, the number of conversations related to this CPA MUST
1702 comply with CPA requirements. The value of `eb:ConversationId` MUST uniquely identify a
1703 conversation within the context of this CPA.

1704 An example of the ConversationId element follows:

```
1705 <eb:ConversationId>20001209-133003-28572</eb:ConversationId>
```

1706 The Party initiating a conversation determines the value of the ConversationId element that SHALL be
1707 reflected in all messages pertaining to that conversation. The actual semantics of this value is beyond
1708 the scope of this specification. Implementations SHOULD provide a facility for mapping between their
1709 identification scheme and a ConversationId generated by another implementation.

1710 **5.2.2.11. eb:Messaging/eb:UserMessage/eb:MessageProperties**

1711 This element has zero or more `eb:Property` child elements.

1712 An `eb:Property` element is of `xs:anySimpleType` (e.g. string, URI) and has two attributes:

- 1713 • `@name`: The value of this REQUIRED attribute must be agreed upon between partners.
- 1714 • `@type`: This OPTIONAL attribute allows for resolution of conflicts between properties with the
1715 same name, and may also help with Property grouping, e.g. various elements of an address.

1716 Its actual semantics is beyond the scope of this specification. The element is intended to be consumed
1717 outside the ebMS-specified functions. It may contain some information that qualifies or abstracts
1718 message data, or that allows for binding the message to some business process. A representation in the
1719 header of such properties allows for more efficient monitoring, correlating, dispatching and validating
1720 functions (even if these are out of scope of ebMS specification) that do not require payload access.

1721 Example:

```

1722 <eb:MessageProperties>
1723   <eb:Property name="ContextId">C1234</eb:Property>
1724   <eb:Property name="processinstanceID">3A4-1234</eb:Property>
1725   <eb:Property name="transactionID">45764321</eb:Property>
1726 </eb:MessageProperties>

```

1727 **5.2.2.12. eb:Messaging/eb:UserMessage/eb:PayloadInfo**

1728 Each PayloadInfo element identifies payload data associated with the message. The purpose of the
 1729 PayloadInfo is:

- 1730 • to make it easier to extract particular payload parts associated with this ebMS Message,
- 1731 • and to allow an application to determine whether it can process these payload parts, without
 1732 having to parse them.

1733 The PayloadInfo element has the following child element:

- 1734 • eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo
 1735 This element occurs zero or more times. The PartInfo element is used to reference a MIME
 1736 attachment, an XML element within the SOAP Body, or another resource which may be obtained
 1737 by resolving a URL, according to the value of the href attribute, described below.

1738 **5.2.2.13. eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo**

1739 This element has the following attribute:

- 1740 • eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/@href
 1741 This OPTIONAL attribute has a value that is the [RFC2392] Content-ID URI of the payload object
 1742 referenced, an xml:id fragment identifier, or the URL of an externally referenced resource; for
 1743 example, "cid:foo@example.com" or "#idref". The absence of the attribute href in the element
 1744 eb:PartInfo indicates that the payload part being referenced is the SOAP Body element itself. For
 1745 example, a declaration of the following form simply indicates that the entire SOAP Body is to be
 1746 considered a payload part in this ebMS message:

```

1747 <eb:PayloadInfo>
1748   <eb:PartInfo/>
1749 </eb:PayloadInfo>

```

1750 Any other namespace-qualified attribute MAY be present. A Receiving MSH MAY choose to ignore any
 1751 foreign namespace attributes other than those defined above.

1752 The designer of the business process or information exchange using ebXML Messaging decides what
 1753 payload data is referenced by the Manifest and the values to be used for xlink:role.

1754 This element has the following child elements:

- 1755 • eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:Schema
 1756 This element occurs zero or more times. It refers to schema(s) that define the instance document
 1757 identified in the parent PartInfo element. If the item being referenced has schema(s) of some
 1758 kind that describe it (e.g. an XML Schema, DTD and/or a database schema), then the Schema
 1759 element SHOULD be present as a child of the PartInfo element. It provides a means of
 1760 identifying the schema and its version defining the payload object identified by the parent
 1761 PartInfo element. This metadata MAY be used to validate the Payload Part to which it refers, but
 1762 the MSH is NOT REQUIRED to do so. The Schema element contains the following attributes:
 - 1763 • (a) namespace - the OPTIONAL target namespace of the schema
 - 1764 • (b) location – the REQUIRED URI of the schema
 - 1765 • (c) version – an OPTIONAL version identifier of the schema.
- 1766 • eb:Messaging/eb:UserMessage/eb:PayloadInfo/eb:PartInfo/eb:PartProperties
 1767 This element has zero or more eb:Property child elements. An eb:Property element is of
 1768 xs:anySimpleType (e.g. string, URI) and has a REQUIRED @name attribute, the value of which
 1769 must be agreed between partners. Its actual semantics is beyond the scope of this specification.

1770 The element is intended to be consumed outside the ebMS specified functions. It may contain
1771 meta-data that qualifies or abstracts the payload data. A representation in the header of such
1772 properties allows for more efficient monitoring, correlating, dispatching and validating functions
1773 (even if these are out of scope of ebMS specification) that do not require payload access.

1774 Example:

```
1775 <eb:PartProperties>  
1776   <eb:Property name="Description">Purchase Order for 11  
1777   widgets</eb:Property>  
1778   <eb:Property name="MimeType">application/xml</eb:Property>  
1779 </eb:PartProperties>
```

1780 Full PayloadInfo Example:

```
1781 <eb:PayloadInfo>  
1782   <eb:PartInfo href="cid:foo@example.com">  
1783     <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>  
1784     <eb:PartProperties>  
1785       <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>  
1786       <eb:Property name="MimeType">application/xml</eb:Property>  
1787     </eb:PartProperties>  
1788   </eb:PartInfo>  
1789   <eb:PartInfo href="#goo_payload_id">  
1790     <eb:Schema location="http://example.org/bar.xsd" version="2.0"/>  
1791     <eb:PartProperties>  
1792       <eb:Property name="Description">Purchase Order for 100 widgets</eb:Property>  
1793       <eb:Property name="MimeType">application/xml</eb:Property>  
1794     </eb:PartProperties>  
1795   </eb:PartInfo>  
1796 </eb:PayloadInfo>
```

1797 **5.2.3. eb:Messaging/eb:SignalMessage**

1798 This element is an alternative to the eb:UserMessage element. It has two child elements:

- 1799 • eb:Messaging/eb:SignalMessage/eb:MessageInfo
1800 This REQUIRED element is similar to eb:MessageInfo as defined for user messages.
- 1801 • eb:Messaging/eb:SignalMessage/eb:[SignalName]
1802 This REQUIRED element defines the nature of the ebMS signal. There is only one
1803 eb:[SignalName] child element when [SignalName]=PullRequest or [SignalName]=Receipt.
1804 There may be several children elements when SignalName=Error.

1805 An ebMS signal does not require any SOAP Body: if the SOAP Body is not empty, it MUST be ignored by
1806 the MSH, as far as interpretation of the signal is concerned.

1807 **5.2.3.1. eb:Messaging/eb:SignalMessage/eb:PullRequest**

1808 This element has the following attribute:

- 1809 • eb:Messaging/eb:SignalMessage/eb:PullRequest/@mpc
1810 This OPTIONAL attribute identifies the Message Partition Channel from which the message is to
1811 be pulled. The absence of this attribute indicates the default MPC.

1812 **5.2.3.2. eb:Messaging/eb:SignalMessage/eb:Error**

1813 The eb:Error element MAY occur zero or more times. For its complete specification, refer to Section 6.

1814 **5.2.3.3. eb:Messaging/eb:SignalMessage/eb:Receipt**

1815 The eb:Receipt element MAY occur zero or one times; and, if present, SHOULD contain a single
1816 ebbpsig:NonRepudiationInformation child element, as defined in the ebBP Signal Schema [ebBP-SIG].
1817 The value of eb:MessageInfo/eb:RefToMessageId MUST refer to the message for which this signal is a
1818 receipt.

1819 5.2.4. Message Unit Bundling

1820 When the eb:Messaging element contains multiple children elements, i.e. multiple Message Units (either
1821 User Message Units or Signal Message Units), this is called Message Unit bundling. The following
1822 general rules govern Message Unit bundling:

1823 Note:

1824 Other use cases for bundling may be considered in a forthcoming Part 2 of this
1825 specification, resulting in changes to these rules, potentially allowing for multiple User
1826 Message Units or multiple Signal Message Units of the same type.

- 1827 • The eb:Messaging element MUST NOT contain more than one eb:UserMessage element.
- 1828 • The eb:Messaging element MAY contain multiple eb:SignalMessage elements, in addition to an
1829 optional eb:UserMessage element, but MUST NOT contain more than one Signal Message Unit
1830 of the same type.

1831 The following is a non-exhaustive list of valid bundling cases:

1832 (a) eb:Messaging element with the following children:

- 1833 • an eb:UserMessage element
- 1834 • an eb:SignalMessage element with an eb:PullRequest child

1835 (b) eb:Messaging element with the following children:

- 1836 • an eb:UserMessage element
- 1837 • an eb:SignalMessage element with one or more eb:Error children

1838 (c) eb:Messaging element with the following children:

- 1839 • an eb:UserMessage element
- 1840 • an eb:SignalMessage element with an eb:PullRequest child
- 1841 • an eb:SignalMessage element (distinct from the previous one) with one or more
1842 eb:Error children

1843 (d) eb:Messaging element with the following children:

- 1844 • an eb:SignalMessage element with an eb:PullRequest child
- 1845 • an eb:SignalMessage element (distinct from the previous one) with an
1846 eb:Receipt child

1847 With regard to MEP transport channel bindings, the following restrictions must be observed:

- 1848 • An ebMS Message containing an eb:SignalMessage/eb:PullRequest element cannot bind to the
1849 back-channel of the underlying transport protocol, regardless of its bundling context (bundling
1850 cases (a) , (c) or (d)) i.e. even if it is also a User Message. For example, such a message can
1851 neither appear as "reply" in the Sync transport channel binding, nor as a "oneway" in the Pull
1852 transport channel binding.
- 1853 • An ebMS Message containing an eb:SignalMessage/eb:PullRequest element and a User
1854 Message unit (case (a) or case (c)) cannot act as a "request" in the Sync transport channel
1855 binding, as the semantics of this combination would require sending back two User Messages
1856 units over the back-channel, which is a bundling case not supported in this release.

1857 5.3. Examples of ebMS Messages

1858 The following listings provide examples of various types of ebMS messages: UserMessage, PullRequest
1859 Signal, Error Signal, Receipt Signal and a "bundled" message.

1860 Note:

1861 The examples are depicted using the SOAP 1.1 envelope; however, SOAP 1.2 could
1862 have been used instead, with the appropriate namespace adjustment. In that case, the
1863 contents of the eb:Messaging header would be the same, with the exception of the
1864 attribute `eb:Messaging/@S11:mustUnderstand`, which becomes

1865 eb:Messaging/@S12:mustUnderstand, having a boolean value of "true" instead of
1866 the integer value "1".

1867 5.3.1. UserMessage Example

1868 The following is an example of an ebMS Request User Message packaged in a SOAP 1.1 envelope:

```
1869 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"  
1870     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">  
1871 <S11:Header>  
1872  
1873     <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1874     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/  
1875     http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">  
1876     <eb:UserMessage>  
1877         <eb:MessageInfo>  
1878             <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>  
1879             <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>  
1880         </eb:MessageInfo>  
1881         <eb:PartyInfo>  
1882             <eb:From>  
1883                 <eb:PartyId>uri:requester.example.com</eb:PartyId>  
1884                 <eb:Role>http://example.org/roles/Buyer</eb:Role>  
1885             </eb:From>  
1886             <eb:To>  
1887                 <eb:PartyId type="someType">QRS543</eb:PartyId>  
1888                 <eb:Role>http://example.org/roles/Seller</eb:Role>  
1889             </eb:To>  
1890         </eb:PartyInfo>  
1891         <eb:CollaborationInfo>  
1892             <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>  
1893             <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>  
1894             <eb:Action>NewPurchaseOrder</eb:Action>  
1895             <eb:ConversationId>4321</eb:ConversationId>  
1896         </eb:CollaborationInfo>  
1897         <eb:MessageProperties>  
1898             <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>  
1899             <eb:Property name="ContextID"> 987654321</eb:Property>  
1900         </eb:MessageProperties>  
1901         <eb:PayloadInfo>  
1902             <eb:PartInfo href="cid:part@example.com">  
1903                 <eb:Schema location="http://registry.example.org/po.xsd" version="2.0"/>  
1904                 <eb:PartProperties>  
1905                     <eb:Property name="Description">Purchase Order for 11 Widgets</eb:Property>  
1906                     <eb:Property name="MimeType">application/xml</eb:Property>  
1907                 </eb:PartProperties>  
1908             </eb:PartInfo>  
1909         </eb:PayloadInfo>  
1910     </eb:UserMessage>  
1911 </eb:Messaging>  
1912 </S11:Header>  
1913 <S11:Body>  
1914 </S11:Body>  
1915 </S11:Envelope>
```

1917

1918 The following is an example of a possible Response to the above User Message:

```
1919 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"  
1920     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">  
1921 <S11:Header>  
1922  
1923     <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1924     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/  
1925     http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">  
1926     <eb:UserMessage>  
1927         <eb:MessageInfo>  
1928             <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>  
1929             <eb:MessageId>UUID-2@responder.example.com</eb:MessageId>  
1930             <eb:RefToMessageId>UUID-1@requester.example.com</eb:RefToMessageId>  
1931         </eb:MessageInfo>  
1932         <eb:PartyInfo>  
1933             <eb:From>  
1934                 <eb:PartyId type="someType">QRS543</eb:PartyId>
```

```

1935         <eb:Role>http://example.org/roles/Seller</eb:Role>
1936     </eb:From>
1937     <eb:To>
1938         <eb:PartyId>uri:requester.example.com</eb:PartyId>
1939         <eb:Role>http://example.org/roles/Buyer</eb:Role>
1940     </eb:To>
1941 </eb:PartyInfo>
1942 <eb:CollaborationInfo>
1943     <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
1944     <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
1945     <eb:Action>PurchaseOrderResponse</eb:Action>
1946     <eb:ConversationId>4321</eb:ConversationId>
1947 </eb:CollaborationInfo>
1948 <eb:PayloadInfo>
1949     <eb:PartInfo href="cid:part@example.com">
1950         <eb:Schema location="http://registry.example.org/poc.xsd" version="2.0"/>
1951         <eb:PartProperties>
1952             <eb:Property name="Description">Purchase Order Confirmation</eb:Property>
1953             <eb:Property name="MimeType">application/xml</eb:Property>
1954         </eb:PartProperties>
1955     </eb:PartInfo>
1956 </eb:PayloadInfo>
1957 </eb:UserMessage>
1958 </eb:Messaging>
1959 </S11:Header>
1960 <S11:Body>
1961 </S11:Body>
1962 </S11:Envelope>
1963

```

1964 5.3.2. PullRequest Message Example

1965 The following is an example of a PullRequest Signal Message:

```

1966 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
1967     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
1968 <S11:Header>
1969     <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1970     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
1971     http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
1972     <eb:SignalMessage>
1973         <eb:MessageInfo>
1974             <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
1975             <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
1976         </eb:MessageInfo>
1977         <eb:PullRequest mpc="http://msh.example.com/mpc123" />
1978     </eb:SignalMessage>
1979 </eb:Messaging>
1980 </S11:Header>
1981 <S11:Body/>
1982 </S11:Envelope>
1983
1984
1985

```

1986

1987 5.3.3. Error Message Example

1988 The following is an example of an Error Signal Message:

```

1989 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
1990     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
1991 <S11:Header>
1992     <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1993     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
1994     http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
1995     <eb:SignalMessage>
1996         <eb:MessageInfo>
1997             <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
1998             <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
1999         </eb:MessageInfo>
2000         <eb:Error origin="ebMS" category="Content"
2001             errorCode="EBMS:0001" severity="failure"
2002

```

```

2003         refToMessageInError="UUID-1@sender.example.com">
2004         <eb:Description xml:lang="en">Value not recognized</eb:Description>
2005     </eb:Error>
2006     <eb:Error origin="Security" category="Processing" errorCode="0101"
2007         severity="failure" refToMessageInError="UUID-23@sender.fxample.com">
2008         <eb:Description xml:lang="en">Failed Authentication</eb:Description>
2009     </eb:Error>
2010 </eb:SignalMessage>
2011
2012 </eb:Messaging>
2013
2014 </S11:Header>
2015
2016 <S11:Body/>
2017 </S11:Envelope>

```

2018 **5.3.4. Receipt Message Example**

2019 The following is an example a Receipt Signal Message, as described in Section 5.2.3.3:

```

2020 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
2021     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
2022 <S11:Header>
2023     <eb:Messaging xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2024         xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2025         http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd"
2026     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
2027     xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0">
2028
2029     <eb:SignalMessage>
2030         <eb:MessageInfo>
2031             <eb:Timestamp>2006-07-01T13:42:37.429Z</eb:Timestamp>
2032             <eb:MessageId>uiwtoruioipwr2543890@b.example.com</eb:MessageId>
2033             <eb:RefToMessageId>uiopfdsmnf4898965563434@a.example.com</eb:RefToMessageId>
2034         </eb:MessageInfo>
2035
2036         <eb:Receipt>
2037             <ebbpsig:NonRepudiationInformation>
2038                 <ebbpsig:MessagePartNRInformation>
2039                     <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
2040                     <ds:Reference URI="http://b.example.com/doc45/#b">
2041                         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2042                         <ds:DigestValue>fX/iNylcUHNLV4lCE0eC7aEGP28=</ds:DigestValue>
2043                     </ds:Reference>
2044                 </ebbpsig:MessagePartNRInformation>
2045                 <ebbpsig:MessagePartNRInformation>
2046                     <ebbpsig:MessagePartIdentifier></ebbpsig:MessagePartIdentifier>
2047                     <ds:Reference URI="http://a.example.com/doc23/#a">
2048                         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2049                         <ds:DigestValue>fX/iNylcUHNLV4lCE0eC7aEGP28=</ds:DigestValue>
2050                     </ds:Reference>
2051                 </ebbpsig:MessagePartNRInformation>
2052             </ebbpsig:NonRepudiationInformation>
2053         </eb:Receipt>
2054     </eb:SignalMessage>
2055
2056 </eb:Messaging>
2057 </S11:Header>
2058
2059 <S11:Body/>
2060 </S11:Envelope>

```

2062 **5.3.5. "Bundled" Message Example**

2063 The following is an example a User Message unit bundled with both PullRequest and Error Signal
2064 Message units, as described in Section 5.2.4:

```

2065 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
2066     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
2067 <S11:Header>
2068     <eb:Messaging S11:mustUnderstand="1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2069         xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2070

```

```

2071 http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
2072
2073 <eb:SignalMessage>
2074   <eb:MessageInfo>
2075     <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
2076     <eb:MessageId>UUID-2@receiver.example.com</eb:MessageId>
2077   </eb:MessageInfo>
2078   <eb:Error origin="ebMS" category="Content"
2079     errorCode="EBMS:0001" severity="failure"
2080     refToMessageInError="UUID-1@sender.example.com">
2081     <eb:Description xml:lang="en">Value not recognized</eb:Description>
2082   </eb:Error>
2083   <eb:Error origin="Security" category="Processing" errorCode="0101"
2084     severity="failure" refToMessageInError="UUID-23@esender.fxample.com">
2085     <eb:Description xml:lang="en">Failed Authentication</eb:Description>
2086   </eb:Error>
2087 </eb:SignalMessage>
2088
2089 <eb:SignalMessage>
2090   <eb:MessageInfo>
2091     <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
2092     <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
2093   </eb:MessageInfo>
2094   <eb:PullRequest mpc="http://msh.example.com/mpc123" />
2095 </eb:SignalMessage>
2096
2097 <eb:UserMessage>
2098   <eb:MessageInfo>
2099     <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
2100     <eb:MessageId>UUID-1@requester.example.com</eb:MessageId>
2101   </eb:MessageInfo>
2102   <eb:PartyInfo>
2103     <eb:From>
2104       <eb:PartyId>uri:requester.example.com</eb:PartyId>
2105       <eb:Role>http://example.org/roles/Buyer</eb:Role>
2106     </eb:From>
2107     <eb:To>
2108       <eb:PartyId type="someType">QRS543</eb:PartyId>
2109       <eb:Role>http://example.org/roles/Seller</eb:Role>
2110     </eb:To>
2111   </eb:PartyInfo>
2112   <eb:CollaborationInfo>
2113     <eb:AgreementRef>http://registry.example.com/cpa/123456</eb:AgreementRef>
2114     <eb:Service type="MyServiceTypes">QuoteToCollect</eb:Service>
2115     <eb:Action>NewPurchaseOrder</eb:Action>
2116     <eb:ConversationId>4321</eb:ConversationId>
2117   </eb:CollaborationInfo>
2118   <eb:MessageProperties>
2119     <eb:Property name="ProcessInst">PurchaseOrder:123456</eb:Property>
2120     <eb:Property name="ContextID"> 987654321</eb:Property>
2121   </eb:MessageProperties>
2122   <eb:PayloadInfo>
2123     <eb:PartInfo href="cid:foo@example.com">
2124       <eb:Schema location="http://registry.example.org/bar.xsd" version="2.0"/>
2125       <eb:PartProperties>
2126         <eb:Property name="Description">Purchase Order for 11 widgets</eb:Property>
2127         <eb:Property name="MimeType">application/xml</eb:Property>
2128       </eb:PartProperties>
2129     </eb:PartInfo>
2130   </eb:PayloadInfo>
2131 </eb:UserMessage>
2132
2133 </eb:Messaging>
2134
2135 </S11:Header>
2136
2137 <S11:Body/>
2138 </S11:Envelope>

```

2139

6. Error Handling

2140 Error handling must take into account the composed nature of an MSH, which includes relatively
2141 independent (SOAP) modules such as those handling reliability and security. Error reporting is also
2142 subject to the same connectivity constraints as the exchange of regular messages. This calls for a more
2143 comprehensive error model. With regard to different ways to report errors, this model must allow for a
2144 clear distinction between what is relevant to an agreement, and what is relevant to immutable
2145 interoperability requirements.

2146 Error generation and error reporting are treated here as orthogonal concepts. While the generation of
2147 errors is a matter of conformance, the reporting of errors may be subject to an agreement. Consequently,
2148 the way errors are to be reported is specified in the P-Mode (P-Mode.ErrorHandling feature) that results
2149 from such an agreement.

2150

6.1. Terminology

- 2151 • **Fault:** A Fault always means a SOAP Fault. It must be generated and processed according to
2152 the [SOAP11] or [SOAP12] specification.
- 2153 • **Error:** An error that is not a SOAP Fault, and occurs in one of the defined modules (ebMS
2154 Module, Reliability Module, Security Module).
- 2155 • **ebMS Error:** This is a particular case of Error, which is generated by the ebMS Module in
2156 conformity with this specification.
- 2157 • **Reliability Error:** This is a particular case of Error, generated by the Reliability Module.
- 2158 • **Security Error:** This is a particular case of Error, generated by the Security Module.
- 2159 • **Escalated ebMS Error:** This is an ebMS Error that originates in a module other than the ebMS
2160 Module (i.e. Security module, or Reliability module).
- 2161 • **ebMS Error Generation:** The operation of creating an ebMS Error object based on some failure
2162 or warning condition.
- 2163 • **ebMS Error Reporting:** The operation of communicating an ebMS Error object to some other
2164 entity.
- 2165 • **Message-in-error:** A flawed message causing an error of some kind.

2166

6.2. Packaging of ebMS Errors

2167

6.2.1. eb:Error Element

2168 An ebMS Error is represented by an eb:Error XML infoset, regardless of the way it is reported. Each error
2169 raised by an MSH has the following properties:

- 2170 • origin (optional attribute)
- 2171 • category (optional attribute)
- 2172 • errorCode (required attribute)
- 2173 • severity (required attribute)
- 2174 • refToMessageInError (optional attribute)
- 2175 • shortDescription (optional attribute)
- 2176 • Description (optional element)
- 2177 • ErrorDetail (optional element)

2178

2179 Example:

2180

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

```
2181         shortDescription="InvalidHeader"  
2182         errorCode="EBMS:0009" severity="fatal">  
2183         <eb:Description xml:lang="en"> ... </eb:Description>  
2184     </eb>Error>
```

2185 **6.2.2. eb>Error/@origin**

2186 This OPTIONAL attribute identifies the functional module within which the error occurred. This module
2187 could be the the ebMS Module, the Reliability Module, or the Security Module. Possible values for this
2188 attribute include "ebMS", "reliability", and "security". The use of other modules, and thus their
2189 corresponding @origin values, may be specified elsewhere, such as in a forthcoming Part 2 of this
2190 specification.

2191 **6.2.3. eb>Error/@category**

2192 This OPTIONAL attribute identifies the type of error related to a particular origin. For example: Content,
2193 Packaging, UnPackaging, Communication, InternalProcess.

2194 **6.2.4. eb>Error/@errorCode**

2195 This REQUIRED attribute is a unique identifier for the type of error.

2196 **6.2.5. eb>Error/@severity**

2197 This REQUIRED attribute indicates the severity of the error. Valid values are: warning, failure.

2198 The **warning** value indicates that a potentially disabling condition has been detected, but no message
2199 processing and/or exchange has failed so far. In particular, if the message was supposed to be delivered
2200 to a consumer, it would be delivered even though a warning was issued. Other related messages in the
2201 conversation or MEP can be generated and exchanged in spite of this problem.

2202 The **failure** value indicates that the processing of a message did not proceed as expected, and cannot
2203 be considered successful. If, in spite of this, the message payload is in a state of being delivered, the
2204 default behavior is not to deliver it, unless an agreement states otherwise (see OpCtx-ErrorHandling).
2205 This error does not presume the ability of the MSH to process other messages, although the
2206 conversation or the MEP instance this message was involved in is at risk of being invalid.

2207 **6.2.6. eb>Error/@refToMessageInError**

2208 This OPTIONAL attribute indicates the MessageId of the message in error, for which this error is raised.

2209 **6.2.7. eb>Error/@shortDescription**

2210 This OPTIONAL attribute provides a short description of the error that can be reported in a log, in order
2211 to facilitate readability.

2212 **6.2.8. eb>Error/Description**

2213 This OPTIONAL element provides a narrative description of the error in the language defined by the
2214 xml:lang attribute. The content of this element is left to implementation-specific decisions.

2215 **6.2.9. eb>Error/ErrorDetail**

2216 This OPTIONAL element provides additional details about the context in which the error occurred. For
2217 example, it may be an exception trace.

2218 **6.3. ebMS Error Message**

2219 When reported as messages, ebMS Errors are packaged as ebMS Signal Messages. Several eb>Error

2220 elements MAY be present under eb:SignalMessage. If this is the case, and if eb:RefToMessageId is
2221 present as a child of eb:SignalMessage/eb:MessageInfo, then every eb:Error element MUST be related
2222 to the ebMS message (message-in-error) identified by eb:RefToMessageId.
2223 If the element eb:SignalMessage/eb:MessageInfo does not contain eb:RefToMessageId, then the
2224 eb:Error element(s) MUST NOT be related to a particular ebMS message.
2225 For an example of an ebXML Error Message, see Section 5.3.3.

2226 6.4. Extensibility of the Error Element

2227 6.4.1. Adding new ebMS Errors

2228 The errorCode attribute (eb:Messaging/eb:SignalMessage/eb:Error/@errorCode) must be an identifier
2229 that is unique within the scope of an MSH. ebMS Errors in addition to those specified here may be added
2230 by creating new errorCode values. The value of the errorCode attribute must begin with the five
2231 characters "EBMS:".

2232 6.5. Generating ebMS Errors

2233 This specification identifies key ebMS Errors, as well as the conditions under which they must be
2234 generated. Some of these error-raising conditions include the escalation as ebMS Errors of either Faults
2235 or Errors generated by Reliability and Security modules. These modules could be those contained in the
2236 MSH raising the Error, or those contained in a remote MSH communicating with the MSH raising the
2237 Error. Except for some cases defined in this specification, Error escalation policies are left to an
2238 agreement between users, represented in the processing mode of an MSH (P-Mode.ErrorHandling).

2239 6.6. Error Reporting

2240 There are three primary means of Error Reporting:

- 2241 • Reporting with Fault Sending: An MSH may generate a SOAP Fault for reporting ebMS
2242 processing errors of severity "failure", which prevent further message processing. This Fault
2243 must comply with SOAP Fault processing, i.e. be sent back as an HTTP response in case the
2244 message in error was over an HTTP request. In case of ebMS processing errors (see Section
2245 6.7.1), the Fault message MUST also include the eb:SignalMessage/eb:Error element in the
2246 eb:Messaging header.
- 2247 • Reporting with Notification: An out-of-band transfer of error information from MSH to some entity
2248 (message producer, consumer, or any other entity, be it local or remote). In case of notification to
2249 the message Producer or Consumer, such reporting action is abstracted by the "Notify" operation
2250 in the messaging model.
- 2251 • Error message: an ebMS signal message sent from one MSH to another, which contains at least
2252 one eb:Error element. Such a reporting action is modeled by Send and Receive abstract
2253 operations over such a message. The reporting message must always be combined with a
2254 SOAP Fault unless the severity is "warning".

2255 **Example of different options in reporting errors raised on a Sending MSH:** Some error detected on
2256 a submitted message and before it is even packaged, would normally be locally notified to the message
2257 Producer, and not even reported to the destination MSH. However, in case this message was part of a
2258 larger exchange that is holding its state waiting for completion on the receiving side, the preferred policy
2259 could state that the message-in-error be also reported (using an error message) to the Receiving MSH. If
2260 the Receiving MSH is getting its messages as responses to PullRequest signals, such ebMS errors can
2261 be transmitted as responses to these signals. If user messages are pushed sender to receiver, it could
2262 be decided that errors generated on the sender side will be pushed like any regular message.

2263 **Example of different options in reporting errors raised on a Receiving MSH:** If a Receiving MSH
2264 detects an error in a received message, the reporting policy may vary depending on the context and the
2265 ability of parties to process such errors. For example, the error-raising Receiving MSH may just notify its
2266 own Consumer party, or send back an error message to the Sending MSH, or both. The usual common

2267 requirement in all these cases, is that the error be reported somehow, and complies with the *eb>Error*
 2268 *element structure*.

2269 Appendix E shows possible options for combining error reporting with ebMS MEPs, when binding to a
 2270 two-way protocol such as HTTP. It also shows how these combinations can be controlled with P-Mode
 2271 parameters.

2272 6.7. Standard ebMS Errors

2273 This section defines the standard error codes expected to be generated and processed by a conformant
 2274 MSH. They are segmented according to the stage of processing they are likely to occur: during reliable
 2275 message processing, security processing, and general ebMS processing.

2276 6.7.1. ebMS Processing Errors

2277 The table below describes the Errors that may occur within the ebMS Module itself (ebMS Errors that are
 2278 not Escalated Errors), i.e. with `@origin="ebms"`. These errors MUST be supported by an MSH, meaning
 2279 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).

2280

2281 6.7.2. Security Processing Errors

2282 The table below describes the Errors that originate within the Security Module, i.e. with
 2283 @origin="security". These errors MUST be escalated by an MSH, meaning generated appropriately, or
 2284 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.

2285

2286 6.7.3. Reliable Messaging Errors

2287 The table below describes the Errors that originate within the Reliable Messaging Module, i.e. with
 2288 @origin="reliability". These errors MUST be escalated by an MSH, meaning generated appropriately, or
 2289 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.

2290

2291

7. Security Module

2292 The ebXML Messaging Service, by its very nature, presents certain security risks. A Messaging Service
2293 may be at risk by means of:

- 2294 • Unauthorized access
- 2295 • Data integrity and/or confidentiality attacks (e.g. through man-in-the-middle attacks)
- 2296 • Denial-of-Service and spoofing

2297 Each security risk is described in detail in the ebXML Technical Architecture Risk Assessment Technical
2298 Report [ebRISK].

2299 Each of these security risks may be addressed in whole, or in part, by the application of one, or a
2300 combination, of the countermeasures described in this section. This specification describes a set of
2301 profiles, or combinations of selected countermeasures, selected to address key risks based upon
2302 commonly available technologies. Each of the specified profiles includes a description of the risks that
2303 are not addressed.

2304 Application of countermeasures SHOULD be balanced against an assessment of the inherent risks and
2305 the value of the asset(s) that might be placed at risk.

7.1. Security Element

2307 Web Services Security 1.0 [WSS10] or 1.1 [WSS11] can be utilized to secure an ebMS message. Web
2308 Services Security provides three mechanisms to secure messages: ability to send security tokens as part
2309 of a message, message integrity and message confidentiality.

2310 Zero or one Security elements per target, belonging to the Web Services Security-defined namespace,
2311 MAY be present as a child of the SOAP Header. The Security element MUST be namespace qualified in
2312 accordance with Web Services Security. The structure and content of the Security element MUST
2313 conform to the Web Services Security specification and the Web Services Security SOAP Messages with
2314 Attachments Profile [SOAPATTACH].

2315 To promote interoperability the security element MUST conform to the WS-I Basic Security Profile
2316 Version 1.0 [WSIBSP10], and WS-I Attachments Profile Version 1.0 [WSIAP10].

2317 Note

2318 An MSH implementation may elect to leverage WSS 1.0 and/or or WSS 1.1. Note that
2319 the security of attachment defined in WSS 1.1 is not only applicable to SOAP 1.1;
2320 security of attachment is orthogonal to the SOAP version, even though all examples in
2321 the WSS 1.1 specification depict only the SOAP 1.1 variant when securing attachments.
2322 In other words, an MSH may secure a SOAP 1.2 with Attachments message in the same
2323 way a SOAP 1.1 with Attachment can be secured in WSS 1.1. Refer to Section C for
2324 complete details of the ebMS SOAP binding.

2325 This specification outlines the use of Web Services Security x.509 Certificate Token Profile [WSS10-
2326 X509] or [WSS11-X509] and the Web Services Security Username Token Profile [WSS10-USER] or
2327 [WSS11-USER]. An MSH implementation MAY choose to support other Web Services Security Profiles.

7.2. Signing Messages

2329 Signing of ebMS Messages is defined in Web Services Security [WSS10] and [WSS11]. Support for
2330 WSS X.509 Certificate Token Profile is REQUIRED to sign a message.

2331 It is REQUIRED that compliant MSH implementations support Detached Signatures as defined by the
2332 XML Signature Specification [XMLDSIG].

2333 An MSH implementation MAY support Enveloped Signatures as defined by the XML Signature
2334 Specification. Enveloped Signatures add an additional level of security in detecting the addition of XML
2335 elements to the SOAP Header. The use of Enveloped Signatures may limit the ability of intermediaries to
2336 process messages.

2337 To ensure the integrity of the user-specified payload data and ebMS message headers it is
2338 RECOMMENDED that the entire eb:Messaging Container Element and the SOAP Body be included in
2339 the signature.

2340 **7.3. Signing SOAP with Attachments Messages**

2341 Application payloads that are built in conformance with the [SOAPATTACH] specification may be
2342 signed. To sign a SOAP with Attachment message the Security element must be built in accordance with
2343 WSS 1.1.

2344 It is REQUIRED that compliant MSH implementations support the Attachment-Content-Only transform. It
2345 is RECOMMENDED that compliant MSH implementations support the Attachment-Complete transform.

2346 To ensure the integrity of the user-specified payload data and ebMS headers it is RECOMMENDED that
2347 the entire eb:Messaging Container Element, and all MIME Body parts of included payloads are included
2348 in the signature.

2349 **7.4. Encrypting Messages**

2350 Encryption of ebMS Messages is defined in Web Services Security [WSS10] and [WSS11]. Support for
2351 Web Services Security X.509 Certificate Token Profile is REQUIRED to encrypt message.

2352 An MSH Implementation may encrypt the eb:Messaging Container Element. It may also encrypt select
2353 child elements of the eb:Messaging header, leaving other elements unencrypted. For example, the
2354 eb:PartyInfo section may be used to aid in message routing before decryption of other elements has
2355 occurred. Therefore, when third-party routing of a message is expected, it is RECOMMENDED that the
2356 eb:PartyInfo section not be encrypted. To ensure the confidentiality of the user-specified payload data, it
2357 is RECOMMENDED that the SOAP Body be encrypted.

2358 **7.5. Encrypting SOAP with Attachments Messages**

2359 Application payloads that are built in conformance with the [SOAPATTACH] specification may be
2360 encrypted. To encrypt a SOAP with Attachment message the Security element must be built in
2361 accordance to WSS 1.1. To ensure the confidentiality of the user-specified payload data it is
2362 RECOMMENDED that the MIME Body parts of included payloads be encrypted.

2363 **7.6. Signing and Encrypting Messages**

2364 When both signature and encryption are required of the MSH, the message MUST be signed prior to
2365 being encrypted.

2366 **7.7. Security Token Authentication**

2367 In constrained environments where management of XML digital signatures is not possible, an
2368 authentication alternative that is based on Web Services Security Username Token Profile is
2369 RECOMMENDED to be supported, and MAY include support for wsse:PasswordText-type passwords.
2370 The value of the wsse:UserName element is an implementation issue. The "user" may represent the
2371 MSH itself, or may represent a party using the MSH. In the latter case, there is no requirement that this
2372 user name be identical to some eb:From/PartyId value.

2373 An MSH MAY support other types of Security Tokens, as allowed by the WS-Security family of
2374 standards.

2375 **7.8. Security Policy Errors**

2376 A responding MSH MAY respond with an error if a received ebMS message does not meet the security
2377 policy of the responding MSH. For example, a security policy might indicate that messages with unsigned
2378 parts of the SOAP Body or eb:Messaging Container element are unauthorized for further processing. If a
2379 responding MSH receives a message with unsigned data within the SOAP Body and error MAY be

2380 returned to the initiating MSH.

2381 7.9. Secured Message Examples

2382 7.9.1. Digitally Signed and Encrypted ebXML Message

```
2383 Mime-Version: 1.0
2384 Content-Type: text/xml
2385 Content-Transfer-Encoding: binary
2386 SOAPAction: ""
2387 Content-Length: 7205
2388
2389 <?xml version="1.0" encoding="UTF-8"?>
2390 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
2391   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2392   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2393   xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2394   http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
2395   <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
2396   msg/ebms/v3.0/ns/core/200704/">
2397     <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
2398       <eb:UserMessage>
2399         <eb:MessageInfo>
2400           <eb:Timestamp>2006-10-31T17:36:20.656Z</eb:Timestamp>
2401           <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
2402           <eb:RefToMessageId>UUID-1@msh-
2403   server.example.com</eb:RefToMessageId>
2404         </eb:MessageInfo>
2405         <eb:PartyInfo>
2406           <eb:From>
2407             <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
2408             <eb:Role>http://example.org/roles/Buyer</eb:Role>
2409           </eb:From>
2410           <eb:To>
2411             <eb:PartyId type="someType">QRS543</eb:PartyId>
2412             <eb:Role>http://example.org/roles/Seller</eb:Role>
2413           </eb:To>
2414         </eb:PartyInfo>
2415         <eb:CollaborationInfo>
2416           <eb:AgreementRef>http://msh-
2417   server.example.com/cpa/123456</eb:AgreementRef>
2418           <eb:Service type="someType">QuoteToCollect</eb:Service>
2419           <eb:Action>NewPurchaseOrder</eb:Action>
2420           <eb:ConversationId>2a81ffbd-0d3d-4cbd-8601-
2421   d916e0ed2fe2</eb:ConversationId>
2422         </eb:CollaborationInfo>
2423         <eb:MessageProperties>
2424           <eb:Property
2425   name="ProcessInst">PurchaseOrder:123456</eb:Property>
2426           <eb:Property name="ContextID">987654321</eb:Property>
2427         </eb:MessageProperties>
2428         <eb:PayloadInfo>
2429           <eb:PartInfo href="#enc">
2430             <eb:Description xml:lang="en-US">PO Image</eb:Description>
2431           </eb:PartInfo>
2432         </eb:PayloadInfo>
2433       </eb:UserMessage>
2434     </eb:Messaging>
2435     <wsse:Security S11:mustUnderstand="1"
2436       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2437   wssecurity-secext-1.0.xsd"
2438       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2439   wssecurity-utility-1.0.xsd">
2440       <wsse:BinarySecurityToken
2441         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2442   wss-soap-message-security-1.0#Base64Binary"
2443         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2444   wss-x509-token-profile-1.0#X509v3"
2445         wsu:Id="signingCert">...</wsse:BinarySecurityToken>
2446       <wsse:BinarySecurityToken
2447         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2448   wss-soap-message-security-1.0#Base64Binary"
2449         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2450   wss-x509-token-profile-1.0#X509v3"
```

```

2451         wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
2452         <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
2453         <enc:EncryptionMethod
2454 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
2455         xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2456 wss-wssecurity-secext-1.0.xsd"/>
2457         <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
2458         <wsse:SecurityTokenReference>
2459         <wsse:Reference URI="#encryptionCert"/>
2460         </wsse:SecurityTokenReference>
2461         </KeyInfo>
2462         <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
2463         <CipherValue>F3HmZ2Ldyn0umLCx/8Q9B9e8OoslJx9i9hOWQjh6JJWYqDLbd
2464 g0QVFiVT1LVjazlThS9m9rkRtpkhCUIY1xjFKtDsuIIAW8cLzV7IHkVoDtQ7ihJc8hY1lEESX9qZN65Jgy
2465 Aa3BYgW9ipjGHtNgZ9RzUdzKdeY74DFm27R6m8b0=</CipherValue>
2466         </CipherData>
2467         <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
2468         <DataReference URI="#enc"/>
2469         </ReferenceList>
2470         </enc:EncryptedKey>
2471         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2472         <ds:SignedInfo>
2473         <ds:CanonicalizationMethod
2474 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
2475         <ds:SignatureMethod
2476 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
2477         <ds:Reference URI="#ebMessage">
2478         <ds:Transforms>
2479         <ds:Transform
2480 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
2481         </ds:Transforms>
2482         <ds:DigestMethod
2483 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2484         <ds:DigestValue>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestVal
2485 ue>
2486         </ds:Reference>
2487         <ds:Reference URI="#body">
2488         <ds:Transforms>
2489         <ds:Transform
2490 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
2491         </ds:Transforms>
2492         <ds:DigestMethod
2493 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2494         <ds:DigestValue>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestVal
2495 ue>
2496         </ds:Reference>
2497         </ds:SignedInfo>
2498         <ds:SignatureValue>
2499         T24okA0MUh5iBNMG6tk8QAKZ+lFMmY1rcPnkOr9j3fHRGM2qqUnoBydOTnClcE
2500 MzPZbnlhDn
2501         YZYmabllqa4N5ynLjw1M4kp0uMip9hapijwL67aBnUeHiFmUau0x9DBOdKZTVa
2502 1QQ92106ge
2503         j2YPDt3VKI1LLT2c8O4TfayGvuY= </ds:SignatureValue>
2504         <ds:KeyInfo>
2505         <wsse:SecurityTokenReference
2506         xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
2507 200401-wss-wssecurity-secext-1.0.xsd">
2508         <wsse:Reference URI="#signingCert"/>
2509         </wsse:SecurityTokenReference>
2510         </ds:KeyInfo>
2511         </ds:Signature>
2512         </wsse:Security>
2513     </S11:Header>
2514     <S11:Body wsu:Id="body"
2515     xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2516 wssecurity-utility-1.0.xsd">
2517     <EncryptedData Id="enc" Type="http://www.w3.org/2001/04/xmlenc#Content"
2518     xmlns="http://www.w3.org/2001/04/xmlenc#">
2519     <EncryptionMethod
2520 Algorithm="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc"/>
2521     <CipherData>
2522     <CipherValue>tjOgUPMmQwd6hXiHuv142swqv4dTYiBfmg8u1SuFVRC3yfNlokshv
2523 oxs1/qQoqNlprDiSOxsxsFvg1la7dehjmWb0owuvU2deleKr5KPcSApnG+kTvNrtg==</CipherValue>
2524     </CipherData>
2525     </EncryptedData>
2526     </S11:Body>
2527 </S11:Envelope>

```

2528

2529 7.9.2. Digitally Signed and Encrypted ebXML SOAP with Attachments 2530 Message

```
2531 Mime-Version: 1.0
2532 Content-Type: multipart/related; type="text/xml";
2533     boundary="----=_Part_2_6825397.1130520599536"
2534 SOAPAction: ""
2535 Content-Length: 7860
2536
2537 -----_Part_2_6825397.1130520599536
2538 Content-Type: text/xml
2539 Content-Transfer-Encoding: binary
2540
2541 <?xml version="1.0" encoding="UTF-8"?>
2542 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
2543     xmlns:xsd="http://www.w3c.org/2001/XMLSchema"
2544     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2545     xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2546     http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
2547     <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
2548     msg/ebms/v3.0/ns/core/200704/">
2549         <eb:Messaging id="ebMessage" S11:mustUnderstand="1">
2550             <eb:UserMessage>
2551                 <eb:MessageInfo>
2552                     <eb:Timestamp>2006-10-28T17:29:59.119Z</eb:Timestamp>
2553                     <eb:MessageId>UUID-2@msh-server.example.com</eb:MessageId>
2554                     <eb:RefToMessageId>UUID-1@msh-
2555     server.example.com</eb:RefToMessageId>
2556                 </eb:MessageInfo>
2557                 <eb:PartyInfo>
2558                     <eb:From>
2559                         <eb:PartyId>uri:msh-server.example.com</eb:PartyId>
2560                         <eb:Role>http://example.org/roles/Buyer</eb:Role>
2561                     </eb:From>
2562                     <eb:To>
2563                         <eb:PartyId type="someType">QRS543</eb:PartyId>
2564                         <eb:Role>http://example.org/roles/Seller</eb:Role>
2565                     </eb:To>
2566                 </eb:PartyInfo>
2567                 <eb:CollaborationInfo>
2568                     <eb:AgreementRef>http://msh-
2569     server.example.com/cpa/123456</eb:AgreementRef>
2570                     <eb:Service type="someType">QuoteToCollect</eb:Service>
2571                     <eb:Action>NewPurchaseOrder</eb:Action>
2572                     <eb:ConversationId>782a5c5a-9dad-4cd9-9bbe-
2573     94c0d737f22b</eb:ConversationId>
2574                 </eb:CollaborationInfo>
2575                 <eb:MessageProperties>
2576                     <eb:Property
2577     name="ProcessInst">PurchaseOrder:123456</eb:Property>
2578                     <eb:Property name="ContextID">987654321</eb:Property>
2579                 </eb:MessageProperties>
2580                 <eb:PayloadInfo>
2581                     <eb:PartInfo href="cid:PO_Image@example.com">
2582                         <eb:Description xml:lang="en-US">PO Image</eb:Description>
2583                     </eb:PartInfo>
2584                 </eb:PayloadInfo>
2585             </eb:UserMessage>
2586         </eb:Messaging>
2587         <wsse:Security S11:mustUnderstand="1"
2588             xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2589     wssecurity-secext-1.0.xsd"
2590             xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2591     wssecurity-utility-1.0.xsd">
2592             <wsse:BinarySecurityToken
2593                 EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2594     wss-soap-message-security-1.0#Base64Binary"
2595                 ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2596     wss-x509-token-profile-1.0#x509v3"
2597                 wsu:Id="signingCert">...</wsse:BinarySecurityToken>
2598             <wsse:BinarySecurityToken
2599                 EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2600     wss-soap-message-security-1.0#Base64Binary"
```

```

2601         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2602 wss-x509-token-profile-1.0#X509v3"
2603         wsu:Id="encryptionCert">...</wsse:BinarySecurityToken>
2604         <enc:EncryptedKey xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
2605         <enc:EncryptionMethod
2606 Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
2607         xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
2608 wss-wssecurity-secext-1.0.xsd"/>
2609         <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
2610         <wsse:SecurityTokenReference>
2611         <wsse:Reference URI="#encryptionCert"/>
2612         </wsse:SecurityTokenReference>
2613         </KeyInfo>
2614         <CipherData xmlns="http://www.w3.org/2001/04/xmlenc#">
2615         <CipherValue>jJRbQBjzYpfdCkPk5F7jUoFjw6Ls6DQ8D9sdI62fwjW9Um/g9
2616 QfivLeVzvSndgnthfEBC1Z6loKiuEF5/Ztw/tFrRgkboR7EBG5XaJUnt0rt8iCChy4PfxCEhH1KjFgTJhU
2617 bXxNW3FxSLkouCn2qIBDrJqwZXAiistt29JrANcc=</CipherValue>
2618         </CipherData>
2619         <ReferenceList xmlns="http://www.w3.org/2001/04/xmlenc#">
2620         <DataReference URI="#encrypted-attachment"/>
2621         </ReferenceList>
2622         </enc:EncryptedKey>
2623         <EncryptedData Id="encrypted-attachment" MimeType="image/jpeg"
2624         Type="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-
2625 1.1#Attachment-Content-Only"
2626         xmlns="http://www.w3.org/2001/04/xmlenc#">
2627         <EncryptionMethod
2628 Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
2629         <CipherData>
2630         <CipherReference URI="cid:PO_Image@example.com">
2631         <Transforms>
2632         <Transform
2633         Algorithm="http://docs.oasis-open.org/wss/oasis-
2634 wss-SwAProfile-1.1#Attachment-Ciphertext-Transform"
2635         xmlns="http://www.w3.org/2000/09/xmldsig#" />
2636         </Transforms>
2637         </CipherReference>
2638         </CipherData>
2639         </EncryptedData>
2640         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2641         <ds:SignedInfo>
2642         <ds:CanonicalizationMethod
2643 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
2644         <ds:SignatureMethod
2645 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
2646         <ds:Reference URI="#ebMessage">
2647         <ds:Transforms>
2648         <ds:Transform
2649 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
2650         </ds:Transforms>
2651         <ds:DigestMethod
2652 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
2653         <ds:DigestValue>xUISuIq5eVxy3FL/4yCrZoEZrTM=</ds:DigestVal
2654 ue>
2655         </ds:Reference>
2656         <ds:Reference URI="cid:PO_Image@example.com">
2657         <ds:Transforms>
2658         <ds:Transform
2659         Algorithm="http://docs.oasis-open.org/wss/oasis-
2660 wss-SwAProfile-1.1#Attachment-Content-Signature-Transform"
2661         />
2662         </ds:Transforms>
2663         <ds:DigestMethod
2664 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
2665         <ds:DigestValue>R4hCV4K4I5QZdSsrP4Krlu46hFo=</ds:DigestVal
2666 ue>
2667         </ds:Reference>
2668         </ds:SignedInfo>
2669         <ds:SignatureValue>
2670         BGnJV/b7EUbAesn7GmNhZ8yYN6Zo06uz29E5r9GHxDW+MUH4wksgA654w+sB0r
2671 Wl8xNranag
2672         3dhKoHbaRERzYHDGqIVfIRqgEwOrHwhz4h7uoLX4yxOU6G9T/gily67Q3pENGP
2673 mVowzoppHm
2674         /yd/A2T0+v4vso20aJiSieEIZSQ= </ds:SignatureValue>
2675         <ds:KeyInfo>
2676         <wsse:SecurityTokenReference

```

```

2677         xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
2678 200401-wss-wssecurity-secext-1.0.xsd">
2679         <wsse:Reference URI="#signingCert"/>
2680         </wsse:SecurityTokenReference>
2681         </ds:KeyInfo>
2682         </ds:Signature>
2683     </wsse:Security>
2684 </S11:Header>
2685 <S11:Body/>
2686 </S11:Envelope>
2687
2688 -----_Part_2_6825397.1130520599536
2689 Content-Type: application/octet-stream
2690 Content-Transfer-Encoding: base64
2691 Content-Id: <PO_Image@example.com>
2692 Content-Description: WSS XML Encryption message; type="image/jpeg"
2693
2694 VEhmwb4FHqQH8m5PKqVu8H0/bq2yUF
2695
2696 -----_Part_2_6825397.1130520599536--

```

2697 7.9.3. Digitally Signed Receipt Signal Message

2698 The following is an example of a signed Receipt for the User Message shown above in Section 7.9.1.
2699 Note the correlations to that message in the eb:RefToMessageId and ds:Reference elements.

```

2700 Mime-Version: 1.0
2701 Content-Type: text/xml
2702 Content-Transfer-Encoding: binary
2703 SOAPAction: ""
2704 Content-Length: 7205
2705
2706 <?xml version="1.0" encoding="UTF-8"?>
2707 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
2708   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
2709   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2710   xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2711 http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
2712   <S11:Header xmlns:eb="http://docs.oasis-open.org/ebxml-
2713 msg/ebms/v3.0/ns/core/200704/">
2714     <eb:Messaging id="ThisebMessage" S11:mustUnderstand="1">
2715
2716       <eb:SignalMessage>
2717         <eb:MessageInfo>
2718           <eb:Timestamp>2006-10-31T18:02:37.429Z</eb:Timestamp>
2719           <eb:MessageId>UUID-3@msh-server.example.com</eb:MessageId>
2720           <eb:RefToMessageId>UUID-2@msh-server.example.com</eb:RefToMessageId>
2721         </eb:MessageInfo>
2722
2723         <eb:Receipt>
2724           <ebbpsig:NonRepudiationInformation
2725             xmlns:ebbpsig="http://docs.oasis-open.org/ebxml-bp/ebbp-signals-2.0"
2726             xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2727             <ebbpsig:MessagePartNRInformation>
2728               <ebbpsig:MessagePartIdentifier>ebMessage</ebbpsig:MessagePartIdentif
2729 ier>
2730               <ds:Reference URI="#ebMessage">
2731                 <ds:Transforms>
2732                   <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
2733 c14n#"/>
2734                 </ds:Transforms>
2735                 <ds:DigestMethod
2736 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2737                 <ds:DigestValue>Ae0PLUKJUnUyAMXkLQD/WwKiFiI=</ds:DigestValue>
2738               </ds:Reference>
2739             </ebbpsig:MessagePartNRInformation>
2740             <ebbpsig:MessagePartNRInformation>
2741               <ebbpsig:MessagePartIdentifier>body</ebbpsig:MessagePartIdentifier>
2742             <ds:Reference URI="#body">
2743               <ds:Transforms>
2744                 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
2745 c14n#"/>
2746               </ds:Transforms>
2747               <ds:DigestMethod
2748 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
2749               <ds:DigestValue>kNY6X7LnRTwxXXBzSw07tcA0KSU=</ds:DigestValue>

```

```

2750         </ds:Reference>
2751         </ebbbsig:MessagePartNRInformation>
2752         </ebbbsig:NonRepudiationInformation>
2753         </eb:Receipt>
2754         </eb:SignalMessage>
2755     </eb:Messaging>
2756
2757     <wsse:Security S11:mustUnderstand="1"
2758       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2759 wssecurity-secext-1.0.xsd"
2760       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2761 wssecurity-utility-1.0.xsd">
2762       <wsse:BinarySecurityToken
2763         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2764 soap-message-security-1.0#Base64Binary"
2765         ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-
2766 token-profile-1.0#X509v3"
2767         wsu:Id="signingCert">...</wsse:BinarySecurityToken>
2768
2769       <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
2770         <ds:SignedInfo>
2771           <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
2772 c14n#" />
2773           <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
2774 sha1" />
2775           <ds:Reference URI="#ThisebMessage">
2776             <ds:Transforms>
2777               <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
2778             </ds:Transforms>
2779             <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
2780             <ds:DigestValue>Ae0PLUKJUUnUyAMXkLQD/WwKiFiI=</ds:DigestValue>
2781           </ds:Reference>
2782         </ds:SignedInfo>
2783         <ds:SignatureValue>T24okA0MUh5iBNMG6tk8QAKZ+lFMmYlrcPnkOr9j3fHRGM2qqUnoB
2784 ydOTnClcEMzPZbnlhdNYZYmabl1qa4N5ynLjwlm4kp0uMip9hapij
2785 wL67aBnUeHiFmUau0x9DBOdKZTVa1QQ92106gej2YPDt3VKI1LLT2
2786 c804TfayGvuY= </ds:SignatureValue>
2787
2788         <ds:KeyInfo>
2789           <wsse:SecurityTokenReference
2790             xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2791 wssecurity-secext-1.0.xsd">
2792             <wsse:Reference URI="#signingCert" />
2793           </wsse:SecurityTokenReference>
2794         </ds:KeyInfo>
2795       </ds:Signature>
2796     </wsse:Security>
2797   </S11:Header>
2798   <S11:Body/>
2799 </S11:Envelope>

```

2800 7.10. Message Authorization

2801

2802 Message authorization is defined here as authorizing the processing of a message in conformance with
2803 the parameters of the P-Mode associated with this message. This includes authorizing the access to
2804 some ebMS resources such as:

- 2805 • "delivery" resources as identified by eb:Service and eb:Action
- 2806 • Message Partition Channel (MPC) that a Pull signal is accessing for pulling messages.

2807 This is different from simply authorizing a received message for further processing by the MSH, which
2808 can be achieved by processing the Security header described earlier in Section 7, regardless of ebMS-
2809 specific resources claimed by the message. A message could successfully be authenticated by the
2810 security module (see Section 4.1), yet not be authorized to pull from a particular MPC, or to effect
2811 delivery of data to a particular Service. For implementations in which there is limited interaction between
2812 processing modules of the MSH - e.g. in case of an architecture based on composing SOAP nodes, the
2813 Security header MAY be consumed by the WSS module before reaching the ebMS message processor.
2814 (Even if the header is forwarded, it may be impractical to require an ebMS processor implementation to
2815 parse it.)

2816 This specification provides a resource-level authorization mechanism. Since any resource a message
 2817 may claim access to is identified by the P-Mode associated with the message, this is equivalent to
 2818 authorizing the association of the message with the P-Mode.

2819 For this purpose, a second wsse:Security header, which contains only an authentication token, MAY be
 2820 present. This specification describes in particular one token option, not exclusively of others: the
 2821 wsse:UsernameToken profile. This secondary Security header may itself be secured (e.g. encrypted) by
 2822 the main Security header.

2823 In the P-Mode model (see Appendix D) such tokens are represented as the PMode.Initiator.Authorization
 2824 parameter set (for authorizing the initiator of an MEP) and the PMode.Responder.Authorization
 2825 parameter set.

2826 This header is not intended to be processed or consumed by the same WSS module as the "main"
 2827 Security header, but is targeted further along to the "ebms" actor - typically a role played by the ebMS
 2828 header processor, which has knowledge of the association between these tokens and the P-Modes that
 2829 govern the message processing.

2830 The following example shows a PullRequest message for which this type of authorization is required.
 2831 Both security headers (shown here as a SOAP1.1 message) are present, with one of them - the
 2832 secondary header - targeted to the "ebms" actor. This Pull signal can effect message delivery from MPC
 2833 "http://msh.example.com/mpc123" only if its credentials match the authorization parameters of at least
 2834 one P-Mode associated with pulling messages on this MPC.

```

2835 <?xml version="1.0" encoding="UTF-8"?>
2836 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/">
2837
2838   <S11:Header
2839     xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
2840     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
2841     wssecurity-secext-1.0.xsd">
2842
2843     <eb:Messaging S11:mustUnderstand="1"
2844       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2845       xsi:schemaLocation="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/
2846       http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd">
2847       <eb:SignalMessage>
2848         <eb:MessageInfo>
2849           <eb:Timestamp>2006-07-25T12:19:05</eb:Timestamp>
2850           <eb:MessageId>UUID-2@initiator.example.com</eb:MessageId>
2851         </eb:MessageInfo>
2852         <eb:PullRequest mpc="http://msh.example.com/mpc123" />
2853       </eb:SignalMessage>
2854     </eb:Messaging>
2855
2856     <wsse:Security S11:mustUnderstand="1">
2857       <!-- main security header -->
2858     </wsse:Security>
2859
2860     <wsse:Security S11:mustUnderstand="1" actor="ebms"
2861       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
2862       utility-1.0.xsd">
2863       <!-- authorization security header (here non encrypted) -->
2864       <wsse:UsernameToken wsu:Id="ebms-1234">
2865         <wsse:Username>acme</wsse:Username>
2866         <wsse:Password Type="...">xyz123</wsse:Password>
2867         <wsu:Created> ... </wsu:Created>
2868       </wsse:UsernameToken>
2869     </wsse:Security>
2870
2871   </S11:Header>
2872   <S11:Body />
2873 </S11:Envelope>

```

2874 Permission to use a P-Mode for processing a received message is granted or denied at the time the P-
 2875 Mode authorization parameters are compared with the credentials in the message.

2876 **7.11. Securing the PullRequest Signal**

2877 **7.11.1. Authentication**

2878 A Sending MSH MUST be able to authenticate a Receiving MSH that sends a PullRequest. When
2879 authentication is required for a particular Receiving MSH, it is RECOMMENDED that the Sending MSH
2880 use security at the SOAP protocol level (WSS). In case a Receiving MSH is not able to use SOAP level
2881 security, other authentication mechanisms MAY be used, e.g. the HTTP Basic or Digest Access
2882 Authentication schemes [RFC2617].

2883 **7.11.2. Authorization**

2884 The processing of a PullRequest signal received by a Sending MSH MAY be authorized based on any of
2885 the following, or combination of the following, mechanisms:

- 2886 (a) Digital signature validation by the Security (WSS) module (see Sections 7.2 and 7.3),
- 2887 (b) A WSS authentication token addressed to the "default" actor/role (see Section 7.7).
- 2888 (c) A WSS authentication token addressed to the "ebms" actor/role (see Section 7.10).
- 2889 (d) A transfer-protocol-level identity-authentication mechanism, such as those described in
2890 Section 7.11.1.

2891 **7.11.3. Preventing Replay Attacks**

2892 Malignant duplication and reuse of a PullRequest signals could lead to transfer of user messages to an
2893 unauthorized destination in spite of valid claims in the signal message. In order to prevent this attack, it
2894 is RECOMMENDED to (1) use At-Most-Once reliability so that duplicate elimination would eliminate
2895 PullRequest duplicates, (2) enforce the integrity of reliability headers by proper compliance with WSS.

2896 **7.12. Countermeasure Technologies**

2897 **7.12.1. Persistent Digital Signature**

2898 The only available technology that can be applied to the purpose of digitally signing an ebMS Message
2899 (the ebXML SOAP Header and Body and its associated payload objects) is provided by technology that
2900 conforms to the Web Services Security and Web Services Security SOAP Messages with Attachments
2901 Profile. An XML Signature conforming to these specifications can selectively sign portions of an XML
2902 document(s), permitting the documents to be augmented (new element content added) while preserving
2903 the validity of the signature(s).

2904 If signatures are being used to digitally sign an ebMS Message then Web Services Security and Web
2905 Services Security SOAP Messages with Attachments Profile MUST be used to bind the ebXML SOAP
2906 Header and Body to the ebXML Payload Container(s) or data elsewhere on the web that relate to the
2907 message.

2908 An ebMS Message requiring a digital signature SHALL be signed following the process defined in this
2909 section of the specification and SHALL be in full compliance with Web Services Security and Web
2910 Services Security SOAP Messages with Attachments Profile.

2911 **7.12.2. Persistent Signed Receipt**

2912 An ebMS Message that has been digitally signed MAY be acknowledged with a message containing an
2913 eb:Receipt Signal (described in Section 5.2.3.3), that itself is digitally signed in the manner described in
2914 the previous section. The Receipt Signal MUST contain the information necessary to provide
2915 nonrepudiation of receipt of the original message; that is, an XML Digital Signature Reference element
2916 list consistent with that contained in the Web Services Security Signature element of the original
2917 message.

2918 **7.12.3. Non-Persistent Authentication**

2919 Non-persistent authentication is provided by the communications channel used to transport the ebMS
2920 Message. This authentication MAY be either in one direction or bi-directional. The specific method will
2921 be determined by the communications protocol used. For instance, the use of a secure network protocol,
2922 such as TLS [RFC2246] or IPSec [RFC2402] provides the sender of an ebMS Message with a way to
2923 authenticate the destination for the TCP/IP environment.

2924 **7.12.4. Non-Persistent Integrity**

2925 A secure network protocol such as TLS or IPSec MAY be configured to provide for digests and
2926 comparisons of the packets transmitted via the network connection.

2927 **7.12.5. Persistent Confidentiality**

2928 Persistent confidentiality is provided by technology that conforms to Web Services Security and Web
2929 Services Security SOAP Messages with Attachments Profile. Encryption conforming to these
2930 specifications can provide persistent, selective confidentiality of elements within an ebMS Message
2931 including the SOAP Header.

2932 **7.12.6. Non-Persistent Confidentiality**

2933 A secure network protocol, such as TLS or IPSEC, provides transient confidentiality of a message as it is
2934 transferred between two ebXML adjacent MSH nodes.

2935 **7.12.7. Persistent Authorization**

2936 Persistent authorization MAY be provided using Web Services Security: SAML Token Profile.

2937 **7.12.8. Non-Persistent Authorization**

2938 A secure network protocol such as TLS or IPSEC MAY be configured to provide for bilateral
2939 authentication of certificates prior to establishing a session. This provides for the ability for an ebXML
2940 MSH to authenticate the source of a connection and to recognize the source as an authorized source of
2941 ebMS Messages.

2942 **7.13. Security Considerations**

2943 Implementers should take note, there is a vulnerability present even when an Web Services Security is
2944 used to protect to protect the integrity and origin of ebMS Messages. The significance of the vulnerability
2945 necessarily depends on the deployed environment and the transport used to exchange ebMS Messages.

2946 The vulnerability is present because ebXML messaging is an integration of both XML and MIME
2947 technologies. Whenever two or more technologies are conjoined there are always additional (sometimes
2948 unique) security issues to be addressed. In this case, MIME is used as the framework for the message
2949 package, containing the SOAP Envelope and any payload containers. Various elements of the SOAP
2950 Envelope make reference to the payloads, identified via MIME mechanisms. In addition, various labels
2951 are duplicated in both the SOAP Envelope and the MIME framework, for example, the type of the content
2952 in the payload. The issue is how and when all of this information is used.

2953 Specifically, the MIME Content-ID: header is used to specify a unique, identifying label for each payload.
2954 The label is used in the SOAP Envelope to identify the payload whenever it is needed. The MIME
2955 Content-Type: header is used to identify the type of content carried in the payload; some content types
2956 may contain additional parameters serving to further qualify the actual type. This information is available
2957 in the SOAP Envelope.

2958 The MIME headers are not protected, even when a Web Services Security based digital signature and/or
2959 Web Services Security based encryption is applied. Thus, an ebMS Message may be at risk depending
2960 on how the information in the MIME headers is processed as compared to the information in the SOAP

2961 Envelope.

2962 The Content-ID: MIME header is critical. An adversary could easily mount a denial-of-service attack by
2963 mixing and matching payloads with the Content-ID: headers. As with most denial-of-service attacks, no
2964 specific protection is offered for this vulnerability. However, it should be detected since the digest
2965 calculated for the actual payload will not match the digest included in the SOAP Envelope when the
2966 digital signature is validated.

2967 The presence of the content type in both the MIME headers and SOAP Envelope is a problem. Ordinary
2968 security practices discourage duplicating information in two places. When information is duplicated,
2969 ordinary security practices require the information in both places to be compared to ensure they are
2970 equal. It would be considered a security violation if both sets of information fail to match.

2971 An adversary could change the MIME headers while a message is en route from its origin to its
2972 destination and this would not be detected when the security services are validated. This threat is less
2973 significant in a peer-to-peer transport environment as compared to a multi-hop transport environment. All
2974 implementations are at risk if the ebMS Message is ever recorded in a long-term storage area since a
2975 compromise of that area puts the message at risk for modification.

2976 The actual risk depends on how an implementation uses each of the duplicate sets of information. If any
2977 processing beyond the MIME parsing for body part identification and separation is dependent on the
2978 information in the MIME headers, then the implementation is at risk of being directed to take unintended
2979 or undesirable actions. How this might be exploited is best compared to the common programming
2980 mistake of permitting buffer overflows: it depends on the creativity and persistence of the adversary.

2981 Thus, an implementation could reduce the risk by ensuring that the unprotected information in the MIME
2982 headers is never used except by the MIME parser for the minimum purpose of identifying and separating
2983 the body parts. This version of the specification makes no recommendation regarding whether or not an
2984 implementation should compare the duplicate sets of information nor what action to take based on the
2985 results of the comparison.

2986

8. Reliable Messaging Module

2987

8.1. The Reliable Messaging Model

2988

The reliable delivery of messages has two aspects:

2989

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.

2990

2991

2. a protocol aspect, that describes the reliability mechanism "on the wire".

2992

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.

2993

2994

8.1.1. Message Processing

2995

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.

2996

2997

2998

2999

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.

3000

3001

3002

3003

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

3004

On Sending side:

3005

1. processing of ebMS headers (the ebMS-qualified headers are added to the message).

3006

2. processing of reliability headers (the headers are added to the message).

3007

On Receiving side:

3008

1. processing of reliability headers (the headers are removed from the message).

3009

2. processing of ebMS headers (the ebMS-qualified headers are removed from the message).

3010

Note

3011

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.

3012

3013

3014

8.1.2. The Reliable Messaging Processor in the MSH

3015

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).

3016

3017

3018

3019

3020

3021

3022

3023

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).

3024

3025

3026

3027

3028

3029

3030 The abstract RM operations are defined as follows:

- 3031 • **RM-Submit**
3032 An abstract operation that transfers a SOAP message from an RM-Producer to an Initiator RMP,
3033 so that this message can be sent reliably.
- 3034 • **RM-Deliver**
3035 An abstract operation that transfers a SOAP message from a Responder RMP to its RM-
3036 Consumer, so that the payload from this message can later be delivered by the MSH.
- 3037 • **RM-SubmitResponse**
3038 An abstract operation that transfers a SOAP message from an RM-Producer to a Responder
3039 RMP as a response to a message received reliably. This response is sent back reliably.
- 3040 • **RM-DeliverResponse**
3041 An abstract operation that transfers a received SOAP response message from an Initiator RMP
3042 to its RM-Consumer.
- 3043 • **RM-Notify**
3044 An abstract operation that makes available to the RM-Producer or to the RM-Consumer a failure
3045 status of a message sent reliably (e.g. a notification telling that the message was not delivered).
3046

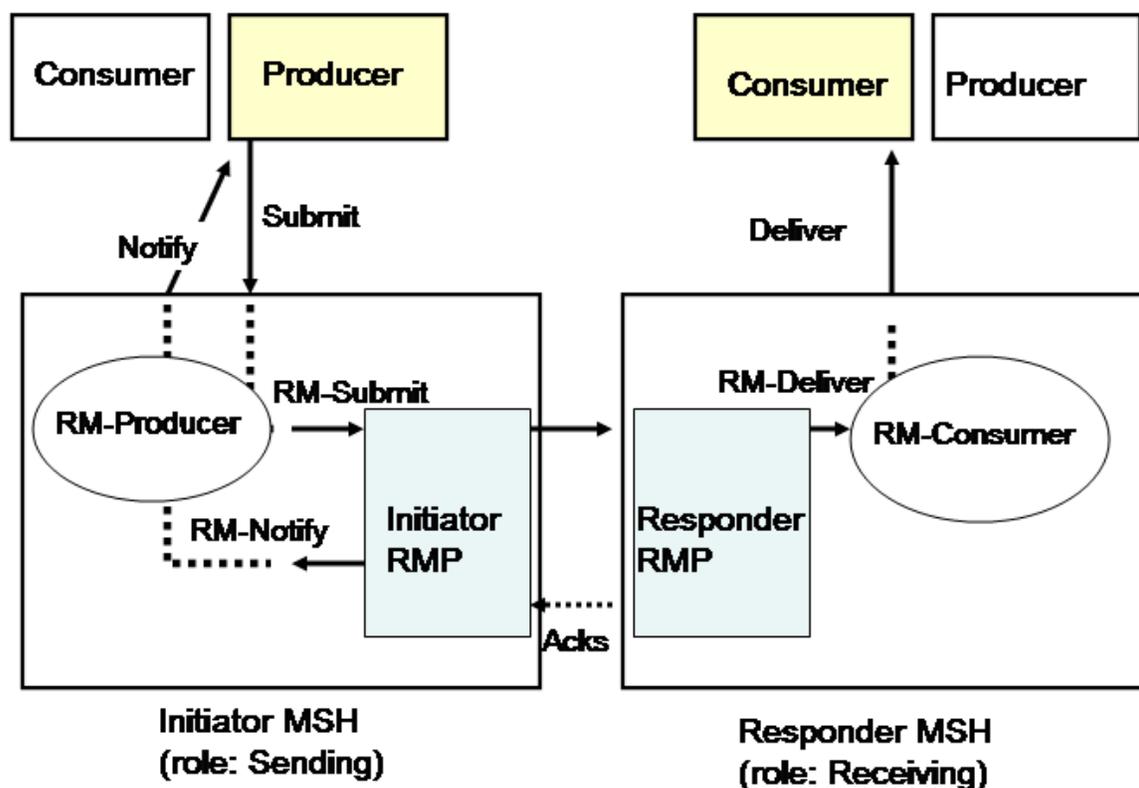


Figure 10: Sending an ebMS Message Reliably

3048 Figure 10 shows the operations involved when sending a request reliably. As indicated in Section 8.3,
3049 this sequence of operations applies either to the User Message in the One-Way/Push MEP, the
3050 PullRequest Signal of a One-Way/Pull MEP, or the first leg of a Two-Way/Sync MEP.
3051

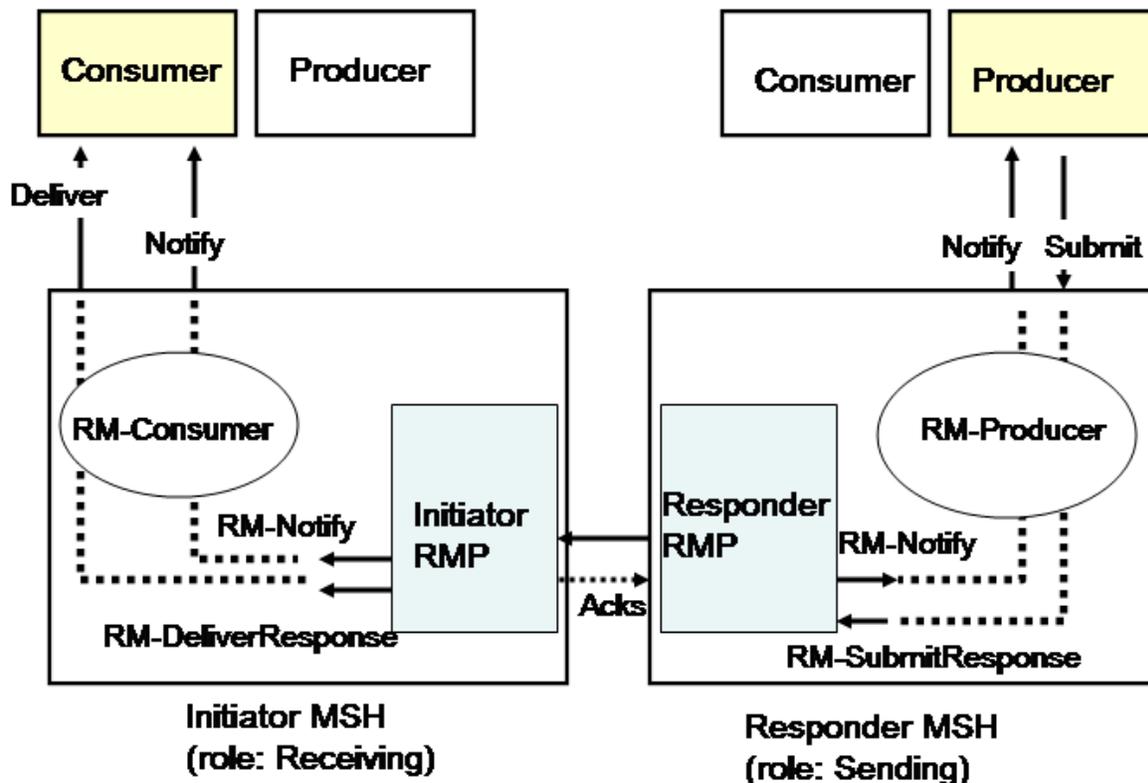


Figure 11: Sending an ebMS MEP Response Message Reliably

3053 Figure 11 shows the abstract operations and components involved when sending a response reliably.
 3054 As indicated in Section 8.3, this sequence of operations applies either to a pulled user message in the
 3055 One-Way/Pull MEP or the response user message in a Two-Way/Sync MEP. Note that depending on the
 3056 reliability processing mode (P-Mode.Reliability), awareness of delivery failure may occur on either side.

3057 8.2. Reliable Delivery of ebMS Messages

3058 Because the reliability function is supported by a module (RMP) within the MSH, the contractual aspect
 3059 has to be considered at two levels: (a) between the RMP and the MSH internals, and (b) between the
 3060 MSH and its Consumer/Producer entities (e.g. an application).

3061 8.2.1. Reliability Contracts for the RMP

3062 Depending on the reliability required for a request message, an RMP must support some or all of the
 3063 following contracts:

- 3064 • **At-Least-Once RM-Delivery**
 3065 When sending a message with this reliability requirement (RM-Submit invocation), one of the two
 3066 following outcomes shall occur: either (1) the Responder RMP successfully delivers (RM-Deliver
 3067 operation) the message to the RM-Consumer or (2) either the Initiator RMP or the Responder
 3068 RMP notifies (RM-Notify operation) respectively the RM Producer or the RM Consumer of a
 3069 delivery failure.
- 3070 • **At-Most-Once RM-Delivery**
 3071 Under this reliability requirement, a message submitted by an RM Producer (RM-Submit

3072 operation) to an Initiator RMP shall not be delivered more than once by the Responder RMP to
3073 its RM-Consumer. The notion of message duplicate is based on a notion of message ID that
3074 must be supported by the reliability specification being used.

3075 • **In-Order RM-Delivery**

3076 Under this reliability requirement, a sequence of messages submitted to an Initiator RMP
3077 (sequence of RM-Submit invocations) shall be delivered in the same order by the Responder
3078 RMP to its RM-Consumer.

3079 These contracts MAY also apply to response messages, as illustrated in Figure 11. In such a case they
3080 are expressed in the above contracts with RM-SubmitResponse and RM-DeliverResponse operations
3081 (instead of RM-Submit and RM-Deliver, respectively), and the Responder and Initiator RMPs switch
3082 roles.

3083 These contracts may be combined; e.g. Exactly-Once results from the combination of At-Least-Once and
3084 At-Most-Once.

3085 In order to support these reliability contracts, both Initiator and Responder RMPs MUST use a reliability
3086 protocol independent from the transport protocol and that provides end-to-end acknowledgment and
3087 message resending capabilities. The details and parameters associated with these protocol functions are
3088 described in Appendix B.

3089 **8.2.2. Reliability Contracts for the MSH**

3090 Because reliability quality of service (QoS) must have significance for the user of the MSH (Producer,
3091 Consumer), and not just for the internal components of the MSH (called RM-Producer and RM-
3092 Consumer) that interact with the RMP component, it is necessary to extend the above contracts and
3093 express them in terms of abstract MSH operations:

3094 • **At-Least-Once ebMS Delivery**

3095 When sending a message with this reliability requirement (Submit invocation), one of the two
3096 following outcomes shall occur: either (1) the Responder MSH successfully delivers (Deliver
3097 operation) the message to the Consumer or (2) a delivery failure notification is communicated
3098 (Notify operation) to either the Producer or the Consumer.

3099 • **At-Most-Once ebMS Delivery:**

3100 Under this reliability requirement, a message transmitted as the result of a Submit invocation on
3101 the Initiator MSH shall not be delivered more than once by the Responder MSH to its Consumer.
3102 An ebMS message is a duplicate of another if it has same eb:MessageId value.

3103 • **In-Order ebMS Delivery**

3104 Under this reliability requirement, a sequence of messages submitted to the Initiator MSH by its
3105 Producer shall be delivered by the Responder MSH in the same order to its Consumer.

3106 In order to fulfill the above QoS requirements, an MSH MUST do the following, in addition to interfacing
3107 with the reliability functions provided by the RMP:

- 3108 • Ensure a proper mapping between MSH abstract operations and RMP abstract operations. This
3109 mapping, which depends on the ebMS MEP being used, is described in Section 8.3.
- 3110 • Ensure the handling of additional failure cases that may happen outside the RMP processing
3111 and outside the transport layer. For example, in the case of At-Least-Once delivery, the sending
3112 MSH must ensure that if a message that has been submitted (Submit) fails before RM-Submit is
3113 invoked, then a delivery failure Error is generated, as would be the case if the message
3114 processing failed just after RM-Submit was invoked. Similarly, if a message fails to be delivered
3115 on receiver side (Deliver) even after RM-Deliver has been successfully invoked, then a delivery
3116 failure Error must be generated and reported either to the Producer or the Consumer, depending
3117 on the P-Mode.ErrorHandling.
- 3118 • Have sufficient control on which RM sequence is used when submitting a message (RM-Submit),
3119 so that an RM sequence may be mapped to an ebMS conversation (eb:ConversationId).

3120 Similar contracts apply to response messages (e.g. second leg of an ebMS Two-Way/Sync MEP), by
3121 switching Initiator MSH and Responder MSH in the above definitions.

3122 8.2.3. Reliability for Signal Messages

3123 Messages that have eb:CollaborationInfo/eb:Service set to "http://docs.oasis-open.org/ebxml-
3124 msg/ebms/v3.0/ns/core/200704/service" are not intended to be delivered (Deliver) to an MSH Consumer,
3125 although they may be submitted by an MSH Producer. They are intended for internal MSH consumption.
3126 They may also be subject to reliability contracts. In this case, the at-least-once contract is fulfilled with a
3127 successful RM-delivery. In case of at-least-once delivery, a failure to deliver MUST cause the generation
3128 of a delivery failure Error. If this message was submitted or initiated by an MSH Producer (Submit)
3129 instead of the MSH itself, the Producer MAY be notified (Notify) of the failure depending on the reporting
3130 mode, as for regular user messages.

3131 8.2.4. Handling of Delivery Failures

3132 Delivery is an abstract operation that may be implemented in various ways. It is the responsibility of an
3133 implementation or product to clearly state at what point in its processing it considers that a message is
3134 delivered. Such a statement amounts to defining a concrete "binding" to the Deliver operation, that a
3135 user can rely on for interpreting the reliability contracts defined and required in this specification, relative
3136 to this implementation.

3137 There are two options when supporting the At-Least-Once delivery contract:

- 3138 1. Delivery failures are always notified to the Producer (the sending side).
- 3139 2. Delivery failures are always notified, though either to the Producer or to the Consumer (the
3140 receiving side), depending on the nature of the failure.

3141 It is part of an agreement between parties to decide which notification option (1 or 2) must be enforced.
3142 An MSH implementation may also be limited in its ability to support option 1. Conformance profiles for
3143 this specification may require either option to be supported.

3144 Delivery Failures (DFs) may be caused by network failure, or by processing failure on either side. In the
3145 remaining part of this section, the following is assumed:

- 3146 • An MSH is always aware of processing failures that occur locally or that have been
3147 communicated to it, and it is always able to report these to its local party (Producer or
3148 Consumer) in some way. E.g. a message processing failure in a Receiving RMP can always be
3149 notified to the Consumer.
- 3150 • A DF that needs to be communicated from MSH to MSH should not itself rely on the transfer of
3151 an Error message (or a Fault), as such message may precisely also fail to be transferred. It is
3152 safer that it relies on the "non-transfer" of a message, such as a missing Acknowledgment.

3153 Note:

3154 By relying on the non-reception of an Acknowledgment for notifying DF, "false" DFs may
3155 occur (in case of Acknowledgment loss), but the case where a message fails to be
3156 delivered unknowingly from the Producer (false delivery) cannot occur. False DF - which
3157 can never be completely eliminated - can always be detected outside the reliable
3158 messaging (RM) layer, in a tractable and less urgent way - e.g. the sending party may
3159 synchronize on a daily basis by communicating its list of assumed delivery failures, for
3160 confirmation by receiver. The Status Request feature (which may be described in a
3161 forthcoming Part 2 of the ebMS specification) could facilitate this function.

3162 Restrictions in the ability to support notification option 1 usually depend on the semantics of
3163 Acknowledgment that is supported by the RMP. Three cases are to be considered:

3164 **Case 1:** The acknowledgment is "on receipt" (as in WS-ReliableMessaging) and has no delivery
3165 semantics. In that case:

- 3166 • DF notifications to the Producer rely on lack of acknowledgments for network failures (non-
3167 reception of a User message)
- 3168 • DF notifications to the Producer rely on Error messages (or Faults) for any other failure occurring
3169 after reception, on Consumer side.

3170 For reasons mentioned above, this acknowledgment semantics does not generally support option 1.

3171 However, in the case of the HTTP binding with no intermediaries present, non-delivery due to processing
3172 failure can still be indicated in a reliable way to the sending side (and will trump the acknowledgment), as
3173 either a SOAP Fault is received on the HTTP response or the HTTP response fails.

3174 The requirements for this transport-specific solution to option 1 which is reliable only for non-delivered
3175 pushed messages (as opposed to pulled) are detailed in Appendix B.

3176 **Case 2:** The acknowledgment is "on MSH-delivery" (supported in WS-Reliability). In that case,
3177 notification option 1 can be supported as well as option 2. In order for option 1 to be supported, an RMP
3178 must implement RM-Deliver operation so that it is only considered successful (worthy of sending an
3179 acknowledgment) if the Deliver operation from MSH to Consumer also succeeds. It is RECOMMENDED
3180 that an implementation support this acknowledgment semantics.

3181 **Case 3:** The acknowledgment is "on RM-delivery" (supported in WS-Reliability). In case the condition in
3182 Case 2 is not supported by an RMP implementation, RM-Delivery is only concerning the RMP module
3183 and does not coincide with MSH delivery. Acknowledgments are "on RM-delivery" only.

3184 Support for option 1 may be accomplished by relying on the transport-specific solution mentioned in
3185 Case 1. This solution is easier to implement here, as it only concerns the module processing the ebMS
3186 header (not the RMP implementation), as described in Appendix B.

3187 **8.3. Reliability of ebMS MEPs**

3188 This section describes the reliability model for MEPs. For a concrete enumeration of all reliability options
3189 for MEPs in the context of an HTTP binding, see Appendix E, which also shows how these combinations
3190 can be controlled with P-Mode parameters.

3191 **8.3.1. Reliability of the One-Way/Push MEP**

3192 The sequence of abstract operation invocations for a successful reliable instance of this MEP is as
3193 follows:

3194 **On Initiator MSH side:**

- 3195 • Step (1): **Submit**: submission of message data to the MSH by the Producer party.
- 3196 • Step (2): **RM-Submit**: after processing of ebXML headers, submission to the RMP.

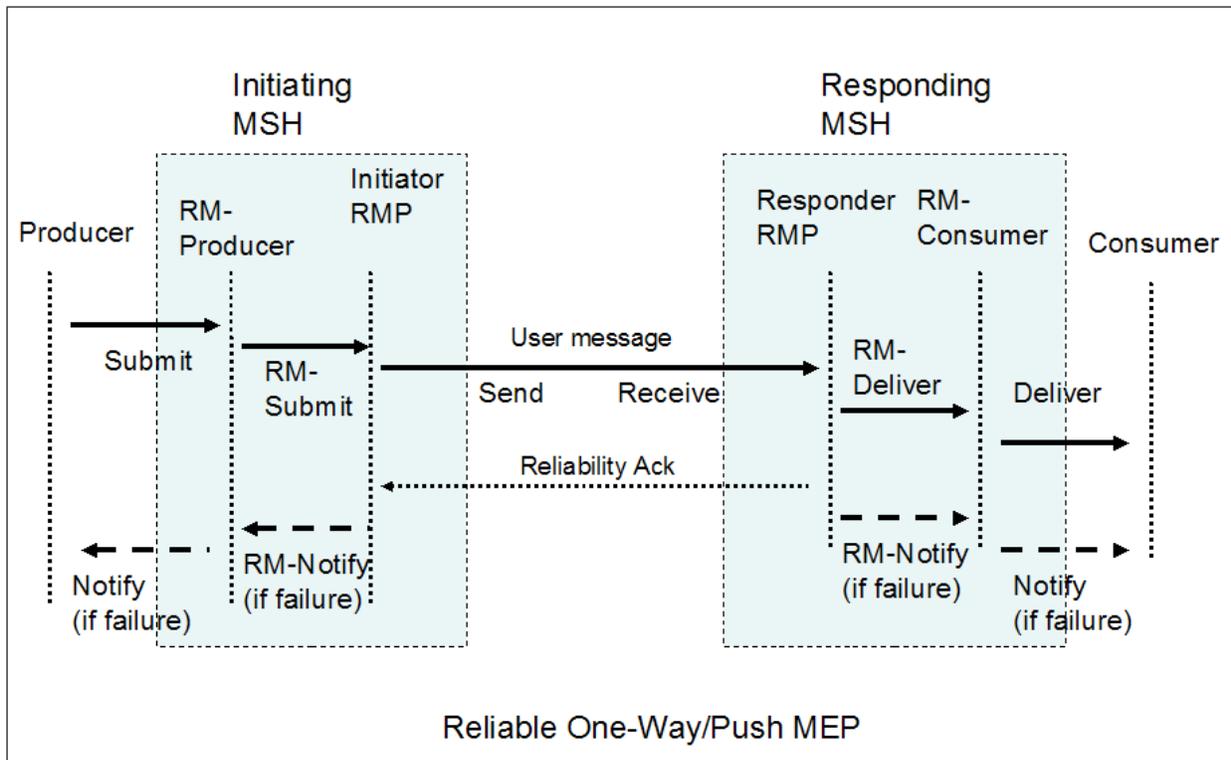
3197 **On Responder MSH side:**

- 3198 • Step (3): **RM-Deliver**: after processing of reliability headers, delivery to other MSH functions.
- 3199 • Step (4): **Deliver**: after processing of ebXML headers, delivery of message data to the Consumer
3200 of the MSH.

3201 **Note:**

3202 In case of delivery failure, either step (4) (Deliver) fails and Notify is invoked on
3203 Responder side, or both (3) and (4) fail and RM-Notify (then Notify) is invoked on either
3204 one of each side. A step "fails" either when it is not invoked in this sequence, or when it
3205 is invoked but does not complete successfully.

3206 Figure 12 illustrates the message flow for this reliable MEP.



3207 *Figure 12: Reliable One-Way/Push MEP*

3208 The way in which the Reliability Acknowledgment binds to the underlying protocol - e.g. as a separate
 3209 HTTP request, or on the back-channel of a previous message - is controlled by the P-Mode parameter
 3210 Reliability.AtLeastOnce.ReplyPattern.

3211 **8.3.2. Reliability of the One-Way/Pull MEP**

3212 The processing model is as follows, for a typical and successful reliable instance of this MEP:

3213 **On Responder MSH side:**

- 3214 • Step (1): **Submit**: submission of message data to the MSH by the Producer party, intended for
 3215 the Consumer on the Initiator side.

3216 **On Initiator MSH side:**

- 3217 • Step (2): Generation of a PullRequest signal by the MSH. **RM-Submit** is invoked on the Initiator
 3218 RMP for this signal.

3219 **On Responder MSH side:**

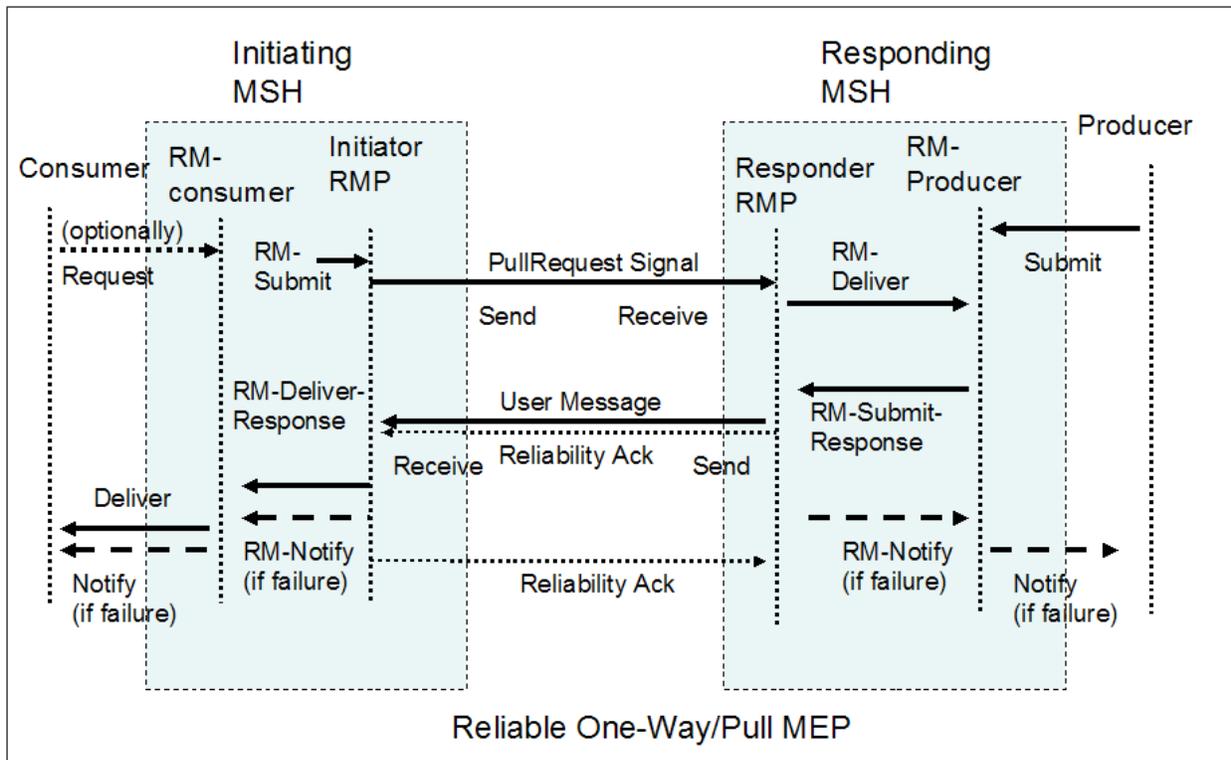
- 3220 • Step (3): Reception of the PullRequest signal by MSH functions. **RM-Deliver** is invoked on the
 3221 Responder RMP for this signal.
- 3222 • Step (4): Submission of the pulled message to the RMP. This results in an **RM-**
 3223 **SubmitResponse** invocation.

3224 **On Initiator MSH side:**

- 3225 • Step (5): **RM-DeliverResponse**: after processing of reliability headers of the pulled message,
 3226 delivery to the RM-Consumer.
- 3227 • Step (6): **Deliver**: after processing of ebMS headers, delivery of the pulled message data to the
 3228 Consumer of the MSH.

3229 Figure 13 illustrates the message flow for this reliable MEP.

3230



3231 *Figure 13: Reliable One-Way/Pull MEP*

3232 The way in which the Reliability Acknowledgments are bound to the underlying protocol is controlled by
 3233 the P-Mode parameter Reliability.AtLeastOnce.ReplyPattern.

3234 In this MEP, as well as in the Simple Request-reply MEP below, the same reliability contracts that apply
 3235 to the MEP request (here the PullRequest signal) MAY apply to the MEP response handled by RM-
 3236 SubmitResponse and RM-DeliverResponse operations.

3237 In such cases, when an MEP response is under reliability contract, the following requirements apply:

- 3238 • When the MEP response is under At-Least-Once reliability contract, then the MEP request
 3239 MUST also be under At-Least-Once reliability contract. In addition, if the MEP request is also
 3240 under At-Most-Once reliability contract, and if it has been delivered and responded to by the
 3241 Responder RMP, then if a duplicate of the MEP request is received later, a duplicate of the same
 3242 response that has been returned for the initial request MUST be returned for the duplicate
 3243 request. Note: depending on where a response delivery failure needs be notified (either on
 3244 Initiator or Responding side, based on P-Mode.Reliability content), an acknowledgment may or
 3245 may not need be returned for the response message by the Initiator RMP.
- 3246 • When the MEP response is under At-Most-Once delivery, then the MEP request MUST also be
 3247 under At-Most-Once delivery.

3248 8.3.3. Reliability of the Two-Way/Sync MEP

3249 The processing model is as follows, for a typical and successful instance of this MEP:

3250 **On Initiator MSH side:**

- 3251 • Step (1): **Submit**: submission of the request message data to the MSH by the Producer party.
- 3252 • Step (2): **RM-Submit**: submission of the request message to the Initiator RMP.

3253 **On Responder MSH side:**

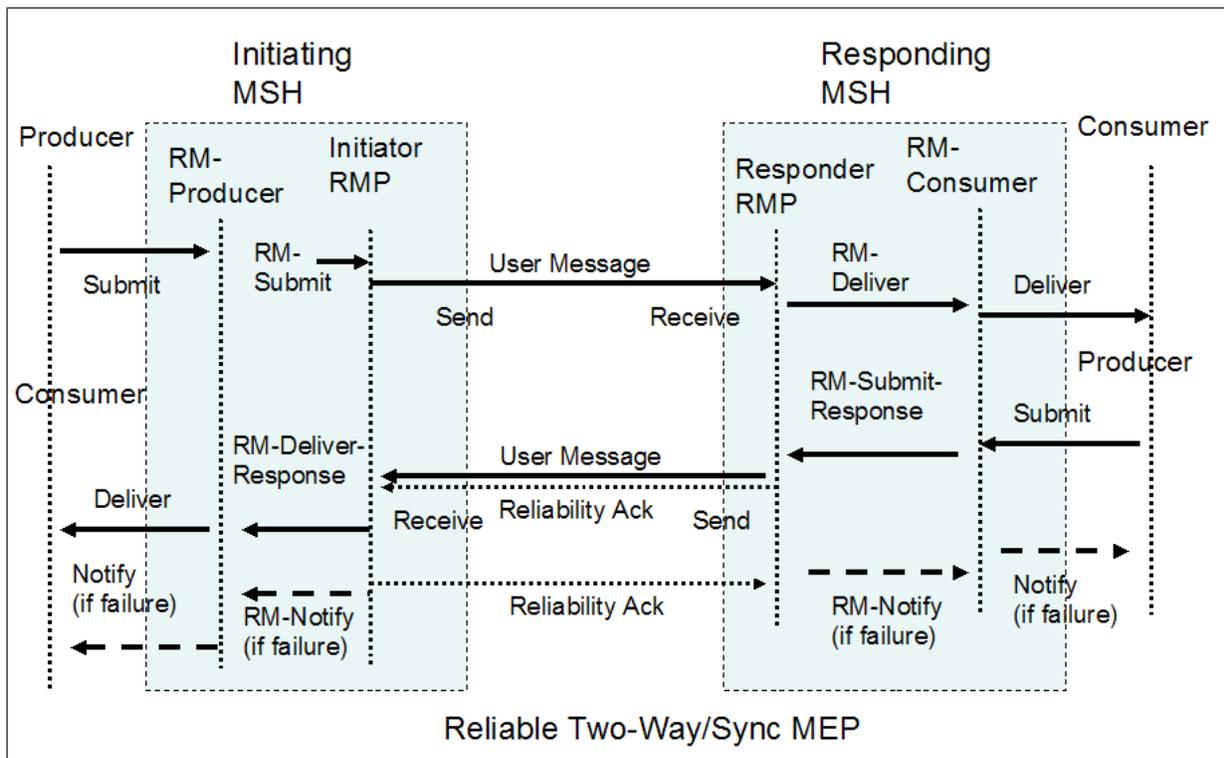
- 3254 • Step (3): **RM-Deliver**: after processing of reliability headers, delivery of the request message to
 3255 RM-Consumer.

- 3256 • Step (4): **Deliver**: delivery of the request message data to the Consumer of the MSH.
- 3257 • Step (5): **Submit**: submission of a response message data to the MSH by the Consumer of the
- 3258 request message, intended for the Producer on the Initiator side.
- 3259 • Step (6): **RM-SubmitResponse**: submission by the RM-Producer of the response message to
- 3260 the Responder RMP.

3261 **On Initiator MSH side:**

- 3262 • Step (7): **RM-DeliverResponse**: delivery of the response message to the RM-Consumer.
- 3263 • Step (8): **Deliver**: delivery of the response message data to the Consumer of the Initiator MSH.

3264 Figure 14 illustrates the message flow for this reliable MEP.



3265 *Figure 14: Reliable Two-Way/Sync MEP*

3266

3267 The way in which the Reliability Acknowledgments are bound to the underlying protocol is controlled by

3268 the P-Mode parameter Reliability.AtLeastOnce.ReplyPattern.

3269 When the MEP response is under reliability contract, the same dependencies with the reliability of the

3270 MEP request that are described for the One-Way/Pull MEP, also apply here.

3271 8.3.4. Reliability of Other Transport-Channel-Bound MEPs

3272 Each one of the MEPs defined in Section 2.2.8: Two-Way/Push-and-Push, Two-Way/Push-and-Pull, and

3273 Two-Way/Pull-and-Push, has been characterized as having a message choreography equivalent to a

3274 sequence of two of the previous MEPs (e.g. Two-Way/Push-and-Pull has a choreography equivalent to

3275 One-Way/Push + One-Way/Pull). The reliability of these more complex MEPs may be handled by

3276 composing reliable versions of these simpler exchanges, which are described in Sections 8.3.1, 8.3.2

3277 and 8.3.3. It can be noted that the reliable Two-Way/Push-and-Push MEP will not make use of the RM-

3278 SubmitResponse operation.

3279

APPENDIX A. The ebXML SOAP Extension Element Schema

3280

3281 Following is the XML schema that describes the eb:Messaging header, as described in Section 5.2. This
 3282 copy is provided for convenience only, and is non-normative. The normative version of the schema may
 3283 be found in a separate file, at [http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-](http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd)
 3284 [200704.xsd](http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/ebms-header-3_0-200704.xsd).

```

3285 <?xml version="1.0" encoding="UTF-8"?>
3286 <xsd:schema xmlns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
3287 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3288 xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
3289 xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
3290 xmlns:tns="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
3291 targetNamespace="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/"
3292 elementFormDefault="qualified"
3293 attributeFormDefault="unqualified">
3294   <xsd:annotation>
3295     <xsd:appinfo>Schema for ebMS-3 XML Infoset</xsd:appinfo>
3296     <xsd:documentation xml:lang="en">
3297       This schema defines the XML Infoset of ebMS-3 headers. These headers are
3298       placed within the SOAP Header element of either a SOAP 1.1 or SOAP 1.2
3299       message.
3300     </xsd:documentation>
3301   </xsd:annotation>
3302   <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
3303     schemaLocation="http://schemas.xmlsoap.org/soap/envelope/" />
3304   <xsd:import namespace="http://www.w3.org/2003/05/soap-envelope"
3305     schemaLocation="http://www.w3.org/2003/05/soap-envelope/" />
3306   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
3307     schemaLocation="http://www.w3.org/2001/03/xml.xsd" />
3308   <xsd:element name="Messaging" type="Messaging" />
3309   <xsd:complexType name="Messaging">
3310     <xsd:annotation>
3311       <xsd:documentation xml:lang="en">
3312         The eb:Messaging element is the top element of ebMS-3 headers, and it is
3313         placed within the SOAP Header element (either SOAP 1.1 or SOAP 1.2). The
3314         eb:Messaging element may contain several instances of eb:SignalMessage
3315         and eb:UserMessage elements. However in the core part of the ebMS-3
3316         specification, only one instance of either eb:UserMessage or eb:SignalMessage
3317         must be present. The second part of ebMS-3 specification may need to include
3318         multiple instances of either eb:SignalMessage, eb:UserMessage or both.
3319         Therefore, this schema is allowing multiple instances of eb:SignalMessage
3320         and eb:UserMessage elements for part 2 of the ebMS-3 specification. Note
3321         that the eb:Messaging element cannot be empty (at least one of
3322         eb:SignalMessage or eb:UserMessage element must present).
3323       </xsd:documentation>
3324     </xsd:annotation>
3325     <xsd:sequence>
3326       <xsd:element name="SignalMessage" type="SignalMessage" minOccurs="0"
3327         maxOccurs="unbounded" />
3328       <xsd:element name="UserMessage" type="UserMessage" minOccurs="0"
3329         maxOccurs="unbounded" />
3330       <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3331         maxOccurs="unbounded" />
3332     </xsd:sequence>
3333     <xsd:attributeGroup ref="tns:headerExtension" />
3334   </xsd:complexType>
3335   <xsd:complexType name="SignalMessage">
3336     <xsd:annotation>
3337       <xsd:documentation xml:lang="en">
3338         In the core part of ebMS-3 specification, an eb:Signal Message is allowed to
3339         contain eb:MessageInfo and at most one Receipt Signal, at most one
3340         eb:PullRequest
3341         element, and/or a series of eb:Error elements. In part 2 of the ebMS-3
3342         specification, new signals may be introduced, and for this reason,
3343         an extensibility point is added here to the eb:SignalMessage element to
3344         allow it to contain any elements.
3345       </xsd:documentation>
3346     </xsd:annotation>
3347     <xsd:sequence>
3348       <xsd:element name="MessageInfo" type="MessageInfo" />
3349       <xsd:element name="PullRequest" type="PullRequest" minOccurs="0" />
  
```

```

3350         <xsd:element name="Receipt" type="Receipt" minOccurs="0"/>
3351         <xsd:element name="Error" type="Error" minOccurs="0"
3352 maxOccurs="unbounded"/>
3353         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3354 maxOccurs="unbounded"/>
3355     </xsd:sequence>
3356 </xsd:complexType>
3357 <xsd:complexType name="Error">
3358     <xsd:sequence>
3359         <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
3360         <xsd:element name="ErrorDetail" type="xsd:token" minOccurs="0"/>
3361     </xsd:sequence>
3362     <xsd:attribute name="category" type="xsd:token" use="optional"/>
3363     <xsd:attribute name="refToMessageInError" type="xsd:token" use="optional"/>
3364     <xsd:attribute name="errorCode" type="xsd:token" use="required"/>
3365     <xsd:attribute name="origin" type="xsd:token" use="optional"/>
3366     <xsd:attribute name="severity" type="xsd:token" use="required"/>
3367     <xsd:attribute name="shortDescription" type="xsd:token" use="optional"/>
3368 </xsd:complexType>
3369 <xsd:complexType name="PullRequest">
3370     <xsd:sequence>
3371         <xsd:any namespace="##other" processContents="lax" minOccurs="0"
3372 maxOccurs="unbounded"/>
3373     </xsd:sequence>
3374     <xsd:attributeGroup ref="pullAttributes"/>
3375 </xsd:complexType>
3376 <xsd:complexType name="Receipt">
3377     <xsd:sequence>
3378         <xsd:any namespace="##other" processContents="lax"
3379 maxOccurs="unbounded"/>
3380     </xsd:sequence>
3381 </xsd:complexType>
3382 <xsd:complexType name="UserMessage">
3383     <xsd:sequence>
3384         <xsd:element name="MessageInfo" type="MessageInfo"/>
3385         <xsd:element name="PartyInfo" type="PartyInfo"/>
3386         <xsd:element name="CollaborationInfo" type="CollaborationInfo"/>
3387         <xsd:element name="MessageProperties" type="tns:MessageProperties"
3388 minOccurs="0"/>
3389         <xsd:element name="PayloadInfo" type="tns:PayloadInfo" minOccurs="0"/>
3390     </xsd:sequence>
3391     <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
3392 </xsd:complexType>
3393 <xsd:complexType name="MessageInfo">
3394     <xsd:sequence>
3395         <xsd:element name="Timestamp" type="xsd:dateTime"/>
3396         <xsd:element name="MessageId" type="tns:non-empty-string"/>
3397         <xsd:element name="RefToMessageId" type="tns:non-empty-string"
3398 minOccurs="0"/>
3399     </xsd:sequence>
3400 </xsd:complexType>
3401 <xsd:complexType name="PartyInfo">
3402     <xsd:sequence>
3403         <xsd:element name="From" type="tns:From"/>
3404         <xsd:element name="To" type="tns:To"/>
3405     </xsd:sequence>
3406 </xsd:complexType>
3407 <xsd:complexType name="PartyId">
3408     <xsd:simpleContent>
3409         <xsd:extension base="tns:non-empty-string">
3410             <xsd:attribute name="type" type="tns:non-empty-string"/>
3411         </xsd:extension>
3412     </xsd:simpleContent>
3413 </xsd:complexType>
3414 <xsd:complexType name="From">
3415     <xsd:sequence>
3416         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
3417         <xsd:element name="Role" type="tns:non-empty-string"/>
3418     </xsd:sequence>
3419 </xsd:complexType>
3420 <xsd:complexType name="To">
3421     <xsd:sequence>
3422         <xsd:element name="PartyId" type="tns:PartyId" maxOccurs="unbounded"/>
3423         <xsd:element name="Role" type="tns:non-empty-string"/>
3424     </xsd:sequence>
3425 </xsd:complexType>
3426 <xsd:complexType name="CollaborationInfo">

```

```

3427     <xsd:sequence>
3428         <xsd:element name="AgreementRef" type="tns:AgreementRef" minOccurs="0"/>
3429         <xsd:element name="Service" type="tns:Service"/>
3430         <xsd:element name="Action" type="xsd:token"/>
3431         <xsd:element name="ConversationId" type="xsd:token"/>
3432     </xsd:sequence>
3433 </xsd:complexType>
3434 <xsd:complexType name="Service">
3435     <xsd:simpleContent>
3436         <xsd:extension base="tns:non-empty-string">
3437             <xsd:attribute name="type" type="tns:non-empty-string"
3438 use="optional"/>
3439         </xsd:extension>
3440     </xsd:simpleContent>
3441 </xsd:complexType>
3442 <xsd:complexType name="AgreementRef">
3443     <xsd:simpleContent>
3444         <xsd:extension base="tns:non-empty-string">
3445             <xsd:attribute name="type" type="tns:non-empty-string"
3446 use="optional"/>
3447             <xsd:attribute name="pmode" type="tns:non-empty-string"
3448 use="optional"/>
3449         </xsd:extension>
3450     </xsd:simpleContent>
3451 </xsd:complexType>
3452 <xsd:complexType name="PayloadInfo">
3453     <xsd:sequence>
3454         <xsd:element name="PartInfo" type="tns:PartInfo" maxOccurs="unbounded"/>
3455     </xsd:sequence>
3456 </xsd:complexType>
3457 <xsd:complexType name="PartInfo">
3458     <xsd:sequence>
3459         <xsd:element name="Schema" type="tns:Schema" minOccurs="0"/>
3460         <xsd:element name="Description" type="tns:Description" minOccurs="0"/>
3461         <xsd:element name="PartProperties" type="tns:PartProperties"
3462 minOccurs="0"/>
3463     </xsd:sequence>
3464     <xsd:attribute name="href" type="xsd:token"/>
3465 </xsd:complexType>
3466 <xsd:complexType name="Schema">
3467     <xsd:attribute name="location" type="xsd:anyURI" use="required"/>
3468     <xsd:attribute name="version" type="tns:non-empty-string" use="optional"/>
3469     <xsd:attribute name="namespace" type="tns:non-empty-string" use="optional"/>
3470 </xsd:complexType>
3471 <xsd:complexType name="Property">
3472     <xsd:simpleContent>
3473         <xsd:extension base="tns:non-empty-string">
3474             <xsd:attribute name="name" type="tns:non-empty-string"
3475 use="required"/>
3476         </xsd:extension>
3477     </xsd:simpleContent>
3478 </xsd:complexType>
3479 <xsd:complexType name="PartProperties">
3480     <xsd:sequence>
3481         <xsd:element name="Property" type="tns:Property" maxOccurs="unbounded"/>
3482     </xsd:sequence>
3483 </xsd:complexType>
3484 <xsd:complexType name="MessageProperties">
3485     <xsd:sequence>
3486         <xsd:element name="Property" type="Property" maxOccurs="unbounded"/>
3487     </xsd:sequence>
3488 </xsd:complexType>
3489 <xsd:attributeGroup name="headerExtension">
3490     <xsd:attribute name="id" type="xsd:ID" use="optional"/>
3491     <xsd:attribute ref="S11:mustUnderstand" use="optional">
3492         <xsd:annotation>
3493             <xsd:documentation>
3494                 if SOAP 1.1 is being used, this attribute is required
3495             </xsd:documentation>
3496         </xsd:annotation>
3497     </xsd:attribute>
3498     <xsd:attribute ref="S12:mustUnderstand" use="optional">
3499         <xsd:annotation>
3500             <xsd:documentation>
3501                 if SOAP 1.2 is being used, this attribute is required
3502             </xsd:documentation>
3503         </xsd:annotation>

```

```
3504     </xsd:attribute>
3505     <xsd:anyAttribute namespace="##other" processContents="lax"/>
3506 </xsd:attributeGroup>
3507 <xsd:attributeGroup name="pullAttributes">
3508     <xsd:attribute name="mpc" type="xsd:anyURI" use="optional"/>
3509     <xsd:anyAttribute namespace="##other" processContents="lax"/>
3510 </xsd:attributeGroup>
3511 <xsd:complexType name="Description">
3512     <xsd:simpleContent>
3513         <xsd:extension base="tns:non-empty-string">
3514             <xsd:attribute ref="xml:lang" use="required"/>
3515         </xsd:extension>
3516     </xsd:simpleContent>
3517 </xsd:complexType>
3518 <xsd:simpleType name="non-empty-string">
3519     <xsd:restriction base="xsd:string">
3520         <xsd:minLength value="1"/>
3521     </xsd:restriction>
3522 </xsd:simpleType>
3523 </xsd:schema>
```

3524

3525 APPENDIX B. Reliable Messaging Bindings

3526 The reliability contracts defined in Section 8 may be implemented by profiling different reliability
3527 specifications. Either one of two OASIS reliability specifications may be used by an MSH implementation:
3528 WS-Reliability 1.1 [WS-R11], or WS-ReliableMessaging 1.1 [WSRM11].

3529 Although either one of the above OASIS reliability specifications is sufficient, each one has strong
3530 arguments in favor of its use. In the same way as two MSH implementations must support the same
3531 transfer protocol or cryptographic algorithms in order to interoperate, two MSHs must also implement the
3532 same reliability specification in order to have interoperable reliability features. The reliability specification
3533 being used in an implementation is a parameter of the conformance profiles for ebMS (see Section G).

3534 B.1. WS-Reliability Binding

3535 B.1.1. Operations and Contracts Binding

3536 The Reliable Messaging Processor (RMP) in ebMS is instantiated by the RMP as defined in WS-
3537 Reliability 1.1. To avoid confusion, we will call the RMP as defined in WS-Reliability 1.1 the WSR-RMP.

3538 The RMP abstract operations RM-Submit, RM-Deliver, RM-SubmitResponse, RM-DeliverResponse and
3539 RM-Notify, map respectively to Submit, Deliver, Respond, Notify and Notify in WS-Reliability 1.1. Note
3540 that a single operation in WS-Reliability (Notify) is used to carry both notification of failure, and response
3541 message. In order to avoid confusion with WS-Reliability operations, the MSH operations Submit,
3542 Deliver, Notify, are respectively renamed in this section: MSH-Submit, MSH-Deliver, MSH-Notify.

3543 The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery
3544 respectively map to the RM agreement items: GuaranteedDelivery, NoDuplicateDelivery,
3545 OrderedDelivery in WS-Reliability.

- 3546 • Message processing faults such as FeatureNotSupported, PermanentProcessingFailure, or
3547 GroupAborted faults, when received by an RMP must be communicated to the MSH. The MSH
3548 must escalate such faults as DysfunctionalReliability ebMS errors (EBMS:0201).
- 3549 • Message format faults, if they result in non-delivery, must be escalated as DeliveryFailure ebMS
3550 errors (EBMS:0202).

3551 B.1.2. Complement to the Reliability of the One-Way/Push MEP

3552 When At-Least-Once delivery is required, it is RECOMMENDED that an Initiator MSH be made aware of
3553 a delivery failure from the Responder MSH to its Consumer. Such a failure is notified to the Producer
3554 party via MSH-Notify. In order to achieve this awareness, the RM-Deliver operation should be
3555 implemented so that it will fail if the MSH-Deliver invocation fails. In such a case the Responder WSR-
3556 RMP generates a **MessageProcessingFailure** fault, and will not acknowledge the reliable message that
3557 has not been successfully delivered by the Responder MSH to its Consumer.

3558 The RM-Agreement associated with the message, as defined in WS-Reliability, is restricted as follows:

- 3559 • In case ReplyPattern has value "Poll" in a message sent reliably, the PollRequest sent later by
3560 the sending RMP for this message must be synchronous (the ReplyTo element MUST NOT be
3561 present).

3562 B.1.3. Complement to the Reliability of the One-Way/Pull MEP

3563 When At-Least-Once delivery is required, it is RECOMMENDED that a Responder MSH be made aware
3564 of a delivery failure from the Initiator MSH to its Consumer. Such a failure is notified to the Producer
3565 party (Responder side) via MSH-Notify. In order to achieve this awareness, the RM-DeliverResponse
3566 operation should be implemented so that it will fail if the MSH-Deliver invocation fails (Initiator side). In
3567 such a case the Initiator WSR-RMP generates a **MessageProcessingFailure** fault, and will not
3568 acknowledge the reliable message that has not been successfully delivered by the Initiator MSH to its
3569 Consumer.

3570 The RM-Agreement associated with the pulled message MUST comply with the following restrictions:
 3571

Name	Allowed Values	Additional Requirements
GuaranteedDelivery	"enabled", "disabled"	<p>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:</p> <ol style="list-style-type: none"> 1. The ReplyPattern value associated with the PullRequest signal is "Response". 2. The NoDuplicateDelivery agreement item is also enabled for the PullRequest signal. <p>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).</p>
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"	

3572

3573 **Note**
 3574 WS-Reliability 1.1 is silent about the reliability of messages submitted as responses to
 3575 other messages, over the same SOAP MEP instance. Such messages would be
 3576 submitted using the abstract operation RM-Respond, which requires an WSR-RMP to
 3577 correlate the response message with the related request. This specification requires that
 3578 the reliability of these responses, in the case of pulled messages, be also supported. by
 3579 the Responder MSH. This means that the implementation of WSR-RMP used in an MSH
 3580 should also support RM agreements that cover such responses.

3581 **B.1.4. Complement to the Reliability of the Two-Way/Sync MEP**

3582 As already mentioned for the One-Way/Push MEP and the One-Way/Pull MEP when At-Least-Once
 3583 delivery is required, it is RECOMMENDED that the Initiator MSH be made aware of a request delivery
 3584 failure from the Responder MSH to its Consumer, and also that the Responder MSH be made aware of a
 3585 response delivery failure from the Initiator MSH to its Consumer.

3586 The RM-Agreement associated with the request message MUST comply with the same restrictions as for
 3587 the One-Way/Push MEP, and also with those entailed by the RM-Agreement options used for the
 3588 response message (see below.)

3589 The RM-Agreement associated with the Response message MUST comply with the following restrictions:

3590

Name	Allowed Values	Additional Requirements
GuaranteedDelivery	"enabled", "disabled"	<p>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:</p> <ol style="list-style-type: none"> 1. The ReplyPattern value associated with the PullRequest signal is "Response". 2. The NoDuplicateDelivery agreement item is also enabled for the Request. <p>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).</p>
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"	

3591

3592

Note

3593

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

3594

3595

3596 **B.2. WS-ReliableMessaging Binding**

3597 Note

3598 This section is based on the Committee Specification (11 April 2007) of the WS-
3599 ReliableMessaging Version 1.1 specification [WSRM11]. It is possible that updates will
3600 be required in order to conform with the final release of WS-ReliableMessaging as
3601 OASIS Standard. However, it is expected that such updates, if any, will be minor and can
3602 be handled via the errata process.

3603 **B.2.1. Operations and Contracts Binding**

3604 The Reliable Messaging Processor (RMP) in ebMS is mapping to the following notions in WS-RM [WS-
3605 ReliableMessaging]: the Sending RMP maps to RMS (Reliable Messaging Source), the Receiving RMP
3606 maps to RMD (Reliable Messaging Destination).

3607 The RMP abstract operations RM-Submit, RM-Deliver, map respectively to Send, Deliver in WSRM. So
3608 do RM-SubmitResponse, RM-DeliverResponse, as there is no distinction in applying reliability features
3609 to a SOAP request and to a SOAP response in WS-RM. RM-Notify must be implemented so that failures
3610 detected by RMS are escalated to the MSH as follows:

- 3611 • CreateSequenceRefused, SequenceTerminated, SequenceClosed, MessageNumberRollover or
3612 UnknownSequence faults, when received by an RMS and when the RMS cannot establish a
3613 substitute sequence that would support reliable transmission of messages in the same
3614 conditions as the failed sequence would have, must be communicated to the MSH on the Source
3615 side. The MSH must escalate such faults as DysfunctionalReliability ebMS errors (EBMS:0201).
- 3616 • WSRM-Required fault, when received by an RMS, must be communicated to the MSH on Source
3617 side. The MSH must escalate such faults as ProcessingModeMismatch (EBMS:0010). It is
3618 recommended to report the RM Error code in the ErrorDetail element of EBMS:0010.
- 3619 • InvalidAcknowledgment and UnknownSequence, when received by the RMD, must be
3620 communicated to the MSH on Destination side. The MSH must escalate such faults as
3621 DysfunctionalReliability ebMS errors (EBMS:0201).

3622 The reliability contracts At-Least-Once Delivery, At-Most-Once Delivery and In-Order Delivery map to
3623 equivalent delivery assurance definitions in the WS-RM specification. Although WS-RM does not
3624 mandate support for these delivery assurances (DAs), and only specifies the protocol aspect, a
3625 conformance profile supporting reliable messaging requires the use of a WS-RM implementation (RMD)
3626 that supports at least some of these DAs as extensions.

3627 It is RECOMMENDED that all messages transmitted over a same sequence use the same MPC. This
3628 becomes a requirement for the In-Order reliability contract.

3629 Note: the WS-RM protocol always assumes acknowledgment of messages. Although acknowledgments
3630 are unnecessary for the At-Most-Once reliability contract, the use of sequence numbers allows for an
3631 efficient duplicate detection. It is then RECOMMENDED to use the WS-RM protocol for At-Most-Once.

3632 Parameters of the WS-RM protocol such as acknowledgment interval, timeouts, resending frequency,
3633 etc. MAY be specified in the Processing Mode, as extensions to the PMode.Reliability group (see
3634 Appendix D).

3635 Sequence acknowledgments and sequence operations (such as CreateSequence,
3636 CreateSequenceResponse) MUST use MEPs of the underlying protocol in a way that is compatible with
3637 the conformance profile of the MSH which defines the ebMS MEPs that must be supported, along with
3638 the underlying protocol binding. For example, if the ebMS conformance profile for an MSH only requires
3639 ebMS messages to be reliably pulled by this MSH over HTTP, then their sequence must either be
3640 created by a CreateSequence message carried over an HTTP response, the HTTP request being
3641 initiated by this MSH, or be offered (using wsm:Offer) by the CreateSequence used for opening a
3642 sequence for sending Pull signals reliably.

3643 Either one of the two following options MUST be used, in order to enable MSH interoperability based on
3644 WS-ReliableMessaging, regarding the reliability contracts for messages exchanged between two MSHs:

- 3645 1. The reliability contract and parameters apply equally to all messages sent between two MSHs.

3646 All messages exchanged in the same direction between two MSHs are subject to the same
3647 reliability quality of service. In such a case, the P-Modes.Reliability parameters associated with
3648 each one of these messages must not conflict with this common quality of service.

3649 2. The reliability contract and parameters MAY vary from one message to the other. In that case,
3650 the scope of application of a reliability contract MUST be the sequence, meaning all messages
3651 within the same sequence are subject to the same reliability contract.

3652 When support for case (2) above is required, the source of a sequence (RMS) must be able to indicate
3653 which delivery assurance is associated with this sequence, so that the RMD implements the expected
3654 DA. Indeed, although both MSHs share knowledge of the reliability contracts associated with each
3655 message (P-Mode.reliability), the RMD has no access to the ebMS header, and can only rely on the
3656 sequence number. In order to avoid the constraint of using predefined sequence numbers, the
3657 association DA-sequence must be dynamically supported by an RMS. Consequently, an implementation
3658 of WS-ReliableMessaging that supports case (2) MUST also support the extension of the
3659 wsrmp:CreateSequence element with a child element structured as a policy assertion as defined in
3660 [WSRMP11], i.e. either one of the following:

```
3661 (a) <wsrmp:AtLeastOnceDelivery wsrmp:InOrder='true|false' />  
3662 (b) <wsrmp:AtMostOnceDelivery wsrmp:InOrder='true|false' />  
3663 (c) <wsrmp:ExactlyOnceDelivery wsrmp:InOrder='true|false' />
```

3664 The above extensions MUST also be supported in wsrmp:Accept/{any} and understood, in case of a
3665 conformance profile that requires support for reliable One-Way/Pull or reliable Two-Way/Sync. It is also
3666 RECOMMENDED that the above extensions be supported in wsrmp:Offer/{any} and understood.

3667 The above DA assertion (a) must match a P-Mode.Reliability with parameters AtMostOnce.Contract =
3668 "false", AtLeastOnce.Contract = "true"; and its attribute @wsrmp:InOrder must match the
3669 InOrder.Contract value.

3670 The above DA assertion (b) must match a P-Mode.Reliability with parameters AtMostOnce.Contract =
3671 "true", AtLeastOnce.Contract = "false"; and its attribute @wsrmp:InOrder must match the
3672 InOrder.Contract value.

3673 The above DA assertion (c) must match a P-Mode.Reliability with parameters AtMostOnce.Contract =
3674 "true", AtLeastOnce.Contract = "true"; and its attribute @wsrmp:InOrder must match the InOrder.Contract
3675 value.

3676 Additional reliability parameters – if any, e.g. resending frequency, etc. - associated with each one of the
3677 reliability contracts (At-Least-Once, At-Most-Once, In-Order) are to be defined in P-Mode.Reliability
3678 extensions and known from both parties prior to the exchange with no need to be transmitted via the RM
3679 protocol. When receiving a CreateSequence message with the above extension specifying a reliability
3680 contract, the RMD MUST be able to resolve it to a single set of additional parameters governing this
3681 mode of reliability. For example, the P-Modes of all messages sent under At-Least-Once should have
3682 same values for the set of PMode.Reliability parameters related to this contract (AcksTo,
3683 AcksOnDelivery, ReplyPattern and any other custom parameters such as those controlling message
3684 resending, if any), as well as for the NotifyProducerDeliveryFailures parameter about failure reporting.

3685 Because acknowledgments in WS-ReliableMessaging are on receipt, the Reliability.AckOnDelivery
3686 parameter in the P-Mode of messages sent reliably MUST be "false".

3687 **B.2.2. Complement to the Reliability of the One-Way/Push MEP**

3688 When At-Least-Once delivery is required for the ebMS User message carried by this MEP, the RMP on
3689 Initiator side is acting as an RMS, and the RMP on Responder side is acting as an RMD.

3690 It is RECOMMENDED that the sequence be initiated by the RMS sending a wsrmp:CreateSequence
3691 message, as opposed to responding to an wsrmp:Offer.

3692 In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Response", then the
3693 CreateSequence/AcksTo element MUST contain an WS-Addressing anonymous IRI.

3694 In case the P-Mode.Reliability.AtLeastOnce.ReplyPattern has value "Callback", then the
3695 CreateSequence/AcksTo element MUST contain an URI specified in an additional P-Mode.Reliability
3696 parameter.

3697 The P-Mode.Reliability.AtLeastOnce.ReplyPattern MUST NOT have value "Poll",
3698 When an underlying two-way protocol is used, any pair of sequence lifecycle message
3699 (CreateSequence/CreateSequenceResponse, CloseSequence/CloseSequenceResponse,
3700 TerminateSequence/ TerminateSequenceResponse) SHOULD be exchanged over a single request-
3701 response MEP of the protocol.
3702 It is RECOMMENDED that the Initiator MSH be made aware of a delivery failure from the Responder
3703 MSH to its Consumer (NotifyProducerDeliveryFailures = "true"). Such a failure is notified to the Producer
3704 party via Notify.
3705 • A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment for a sent
3706 message, must be communicated to the Initiator MSH. The MSH must escalate such a fault as
3707 DeliveryFailure ebMS errors (EBMS:0202).
3708 • A failure to deliver that is detected by the RMD (Responder side), e.g. failure to deliver
3709 (operation Deliver) after the message has been received and acknowledged by the RMD, must
3710 be communicated to the Responder MSH. The MSH must escalate such a fault as
3711 DeliveryFailure ebMS errors (EBMS:0202). It is RECOMMENDED that this ebMS error be
3712 reported to the Initiator MSH.

3713 **B.2.3. Complement to the Reliability of the One-Way/Pull MEP**

3714 When At-Least-Once delivery is required for the ebMS User message carried by this MEP, the RMP on
3715 Responder side is acting as an RMS, and the RMP on Initiator side (which sent the PullRequest) is
3716 acting as an RMD.

3717 When initiating an instance of the One-Way/Pull MEP, and if it is expected – based on P-Modes
3718 deployed - that pulled message may be sent reliably, then the PullRequest signal itself MUST be sent
3719 under At-Least-Once delivery (see Section 8). Acknowledgments for Pull signals should be sent over
3720 the second leg of the One-Way/Pull MEP (PMode.Reliability.AtLeastOnce.ReplyPattern ="Response"),
3721 bundled with the pulled ebMS user message. However the frequency of acknowledgments may not need
3722 be on a per message basis.

3723 In case pulled messages must be sent reliably, the following requirements apply:

- 3724 • When a sequence is initiated (CreateSequence) to be associated with PullRequest signals
3725 intended for the same MPC, then the wsrn:Offer MUST be present in the CreateSequence
3726 element. The offered sequence SHOULD be used for sending back pulled messages reliably.
- 3727 • When no more messages have to be pulled reliably from an MPC, it is RECOMMENDED that the
3728 Sending MSH closes and terminate the associated sequences. When the Sending MSH decides
3729 to terminate a reliable sequence of pulled messages, a CloseSequence message or a
3730 TerminateSequence SHOULD be sent over a pulled message, e.g. piggybacked over the
3731 EmptyMessagePartitionChannel warning (EBMS:0006).

3732 It is RECOMMENDED that the Responder MSH be made aware of a delivery failure from the Initiator
3733 MSH to its Consumer. Such a failure is notified to the Producer party (Responder side) via Notify.

- 3734 • A failure to deliver that is detected by the RMS, e.g. failure to get an acknowledgment on the
3735 Responder side for a sent message, must be communicated to the Responder MSH. The MSH
3736 must escalate such a fault as DeliveryFailure ebMS errors (EBMS:0202).
- 3737 • A failure to deliver that is detected by the RMD (Initiator side), e.g. failure to deliver (operations
3738 Deliver) after the message has been received and acknowledged by the RMD must be
3739 communicated to the Initiator MSH. The MSH must escalate such a fault as DeliveryFailure
3740 ebMS errors (EBMS:0202). It is RECOMMENDED that this ebMS error be reported to the
3741 Responder MSH.

3742 **B.2.4. Complement to the Reliability of the Two-Way/Sync MEP**

3743 In the reliable Two-Way/Sync MEP, either:

- 3744 • The request message alone is sent reliably, in which case the requirements and
3745 recommendations for the One-Way/Push also apply here.

3746 ▪ Or both the request and the reply are sent reliably. The response alone SHALL NOT be sent
3747 reliably.

3748 In case both request and reply are sent reliably, it is RECOMMENDED that both sequences are
3749 established and discarded in a coordinated way. The same rules apply as for the reliability of the One-
3750 way Pull MEP. The in-bound sequence termination SHOULD be terminated on the initiative of the MEP
3751 Initiator, after the out-bound sequence is terminated.

APPENDIX C. SOAP Format and Bindings

3752

3753 This appendix specifies the SOAP format (SOAP versions, packaging of attachments and/or binary data)
3754 used in ebMS-3, as well as how this SOAP format is transported over HTTP [HTTP11]and SMTP
3755 [SMTP].

3756 ebMS-3 does not require the usage of SOAP-1.1 and/or SwA (SOAP-1.1 With Attachments). We
3757 consider the attachments specification of SwA as being orthogonal to the SOAP version. In other words,
3758 attachments could well be used for SOAP 1.2 in the same way they are used for SOAP 1.1. Similarly, we
3759 also consider MTOM being orthogonal to the SOAP version (however, MTOM will not be addressed in
3760 this core specification).

3761 A conformant implementation of ebMS-3 may well choose to use SOAP-1.2 instead of SOAP-1.1. Since
3762 SwA is orthogonal to the SOAP version, there are two possibilities:

- 3763 (1) An implementation of ebMS-3 may choose SOAP-1.1 with Attachments
3764 (2) An implementation of ebMS-3 may choose SOAP-1.2 with Attachments

3765 Although a SOAP 1.2 version of SwA has not been formally submitted to W3C, it appears that most
3766 SOAP products have anticipated that usage, and after investigation, it appears that they have done so in
3767 a consistent, interoperable way. This specification is acknowledging these *de facto* upgrades of SwA,
3768 which are summarized below.

3769 SwA uses the multipart/related MIME encapsulation. This encapsulation is independent of the version of
3770 SOAP being used (in fact it can encapsulate any XML document, not just SOAP), and also independent
3771 of the transport protocol (the encapsulation could be transported via HTTP, SMTP, etc.).

C.1. Using SwA with SOAP-1.1

3772

3773 The following example shows an ebMS-3 message using SOAP 1.1 with attachments. The ebMS-3
3774 message in this example contains two payloads:

- 3775 • The first payload is the picture of a car. This picture is in binary form as an attachment with a
3776 Content-ID equal to "car-photo@cars.example.com".
3777 • The second payload is an XML fragment within the SOAP body. This XML fragment has id
3778 attribute equal to "carData"

3779 The XML fragment in the SOAP body contains a reference to another binary data, namely the picture of
3780 the car owner):

```
3781 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;  
3782 start="<car-data@cars.example.com>"  
3783  
3784 --MIME_boundary  
3785 Content-Type: text/xml; charset=UTF-8  
3786 Content-Transfer-Encoding: 8bit  
3787 Content-ID: <car-data@cars.example.com>  
3788  
3789 <?xml version='1.0' ?>  
3790 <S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"  
3791 xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">  
3792 <S11:Header>  
3793 <eb:Messaging S11:mustUnderstand="1">  
3794 ...  
3795 <eb:PayloadInfo>  
3796 <eb:PartInfo href="cid:car-photo@cars.example.com" />  
3797 <eb:PartInfo href="#carData" />  
3798 </eb:PayloadInfo>  
3799 </eb:Messaging>  
3800 </S11:Header>  
3801  
3802 <S11:Body>  
3803 <t:Data id="carData" xmlns:t="http://cars.example.com">  
3804 <t:Mileage>20000</t:Mileage>  
3805 <t:OwnerPicture href="cid:picture-of-owner@cars.example.com"/>  
3806 </t:Data>  
3807 </S11:Body>
```

```

3808 </S11:Envelope>
3809
3810 --MIME_boundary
3811 Content-Type: image/tiff
3812 Content-Transfer-Encoding: binary
3813 Content-ID: <car-photo@cars.example.com>
3814
3815 ...binary TIFF image of the car...
3816
3817 --MIME_boundary-
3818 Content-Type: image/tiff
3819 Content-Transfer-Encoding: binary
3820 Content-ID: <picture-of-owner@cars.example.com>
3821
3822 ...binary TIFF image of the car's owner...
3823 --MIME_boundary-
3824

```

3825 **Example 1: SOAP-1.1 with Attachment**

3826 **C.2. Using SwA with SOAP-1.2**

3827 The following (Example 2) shows the same message given in Example 1 above, except that SOAP-1.2 is
3828 being used instead of SOAP-1.1:

```

3829 Content-Type: Multipart/Related; boundary=MIME_boundary;
3830 type=application/soap+xml;
3831 start="<car-data@cars.example.com>"
3832
3833 --MIME_boundary
3834 Content-Type: application/soap+xml; charset=UTF-8
3835 Content-Transfer-Encoding: 8bit
3836 Content-ID: <car-data@cars.example.com>
3837
3838 <?xml version='1.0' ?>
3839 <S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
3840 xmlns:eb="http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/">
3841 <S12:Header>
3842 <eb:Messaging S12:mustUnderstand="true">
3843 ...
3844 <eb:PayloadInfo>
3845 <eb:PartInfo href="cid:car-photo" />
3846 <eb:PartInfo href="#carData" />
3847 </eb:PayloadInfo>
3848 </eb:Messaging>
3849 </S12:Header>
3850
3851 <S12:Body>
3852 <t:Data id="carData" xmlns:t="http://car.example.com">
3853 <t:Mileage>20000</t:Mileage>
3854 <t:OwnerPicture href="cid:picture-of-owner"/>
3855 </t:Data>
3856 </S12:Body>
3857 </S12:Envelope>
3858
3859 --MIME_boundary
3860 Content-Type: image/tiff
3861 Content-Transfer-Encoding: binary
3862 Content-ID: <car-photo@cars.example.com>
3863
3864 ...binary TIFF image of the car...
3865
3866 --MIME_boundary
3867 Content-Type: image/tiff
3868 Content-Transfer-Encoding: binary
3869 Content-ID: <picture-of-owner@cars.example.com>
3870
3871 ...binary TIFF image of the car's owner...
3872 --MIME_boundary--
3873

```

3874 **Example 2: SOAP-1.2 with Attachments**

3875 What were the differences between Example 1 and Example 2 (SOAP 1.1/SOAP 1.2 with attachments)?
3876 The differences are the following:

- 3877 • In SOAP 1.1, the namespace of the SOAP elements (Envelope, Header, and Body) is
3878 <http://schemas.xmlsoap.org/soap/envelope/> versus the namespace
3879 <http://www.w3.org/2003/05/soap-envelope> for SOAP 1.2
- 3880 • In SOAP 1.1, the attribute mustUnderstand takes 0 or 1 as values, whereas in SOAP 1.2, the
3881 values for the attribute mustUnderstand are true and false.

3882 Another difference between SOAP 1.1 and SOAP 1.2 would be in the SOAPAction header. When using
3883 HTTP as the transport protocol, there will be an HTTP header called SOAPAction if SOAP 1.1 is being
3884 used. If SOAP 1.2 is used, instead of the SOAPAction header there will be an action parameter, as
3885 illustrated in the following listings:

```
3886 SOAPAction: leasing
3887 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3888 start="<car-data@cars.example.com>"
```

3889 HTTP headers when using SOAP 1.1 with attachments

3890

```
3891 Content-Type: Multipart/Related; boundary=MIME_boundary;
3892 type=application/soap+xml;
3893 start="<car-data@cars.example.com>"; action=leasing
```

3894 HTTP headers when using SOAP 1.2 with attachments

3895 C.3. SMTP Binding

3896 When using SMTP transport, the Mime-Version header MUST be present (among other SMTP-related
3897 headers such as To, From, Date, etc.). The following listings show the headers for both SOAP 1.1 and
3898 SOAP 1.2 over SMTP:

```
3899 From: user@customer.example.com
3900 To: leasing-office@cars.example.com
3901 Date: Mon, 23 Jan 2006 17:33:00 CST
3902 Mime-Version: 1.0
3903 SOAPAction: leasing
3904 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3905 start="<car-data@cars.example.com>"
```

3906 SMTP headers when using SOAP 1.1 with attachments

3907

```
3908 From: user@customer.example.com
3909 To: leasing-office@cars.example.com
3910 Date: Mon, 23 Jan 2006 17:33:00 CST
3911 Mime-Version: 1.0
3912 Content-Type: Multipart/Related; boundary=MIME_boundary;
3913 type=application/soap+xml;
3914 start="<car-data@cars.example.com>"; action=leasing
```

3915 SMTP headers when using SOAP 1.2 with attachments

3916 The remaining portions of the messages in the two examples above are respectively the same as the first
3917 two HTTP binding examples of Section C.

3918 **Note:**

3919 This binding applies only to the ebMS One-Way/Push or Two-Way/Push-and-Push
3920 MEPs. An SMTP binding for the other ebMS MEPs involving the Pull or Synchronous
3921 transfer features would require an SMTP binding of the SOAP Request-Response MEP;
3922 for example, [SOAPEMAIL]. Use of such bindings are out of scope of this specification,
3923 and may be detailed in a forthcoming Part 2 of this specification.

3924

APPENDIX D. Processing Modes

3925

D.1. Objectives and Usage

3926

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.

3927

3928

3929

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

3930

- 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.

3931

3932

3933

3934

- It is configuration data for a Sending MSH, as well as for a Receiving MSH.

3935

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.

3936

3937

3938

3939

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.

3940

3941

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:

3942

3943

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.

3944

3945

3946

3947

3948

3949

3950

3951

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.

3952

3953

3954

3955

3956

3957

3958

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.

3959

3960

3961

3962

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.

3963

3964

3965

3966

3967

3968

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]).

3969

3970

3971

3972

A P-Mode, or set of P-Modes, may also be represented as parts of a CPA document, the details of which

3973 are out of scope of this Appendix.

3974 In order to promote the portability of P-Mode representations across MSH implementations, a
3975 conformance profile may require support for a particular P-Mode representation.

3976 An implementation may decide to extend the P-Mode data model specified here, with additional
3977 parameters. Conversely, depending on its conformance profile an implementation may only need to
3978 support a subset of the P-Mode parameters described here.

3979 **D.2. Model for Processing Modes**

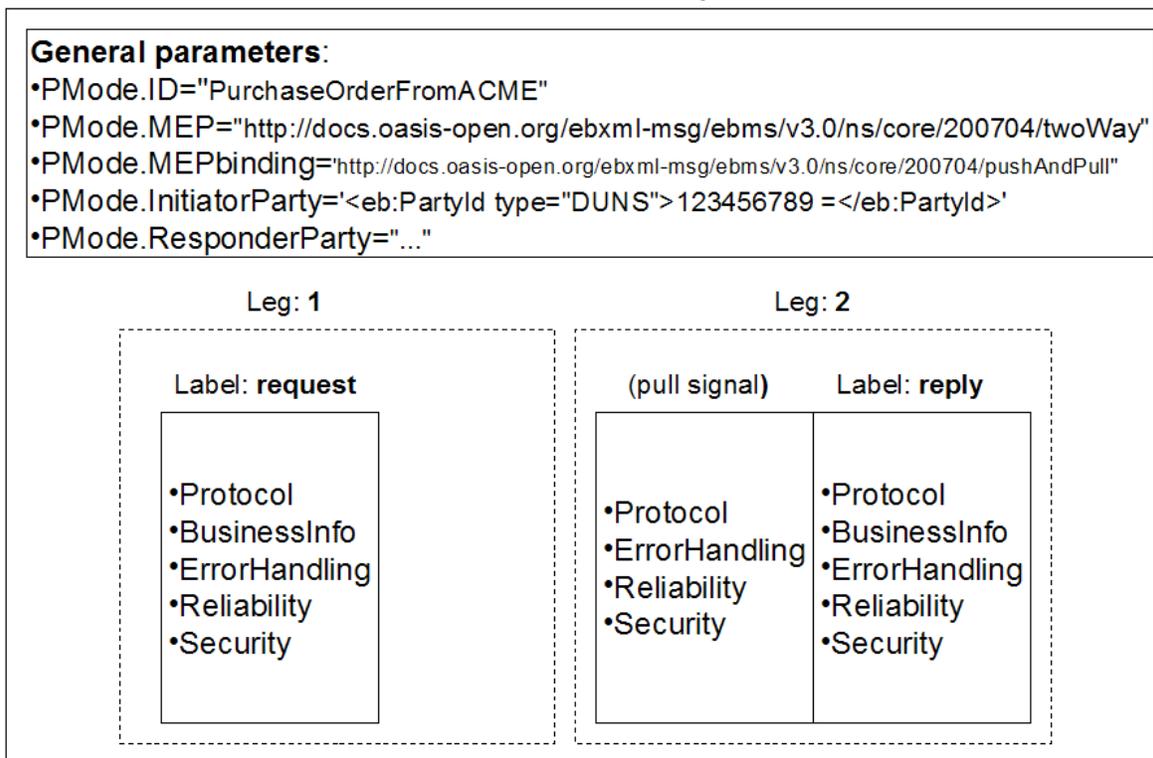
3980 A P-Mode actually governs the transmission of all the messages involved in an ebMS MEP between two
3981 MSHs. P-Mode parameters are grouped into six functional categories, also called P-Mode features (see
3982 Section 4):

- 3983 • **General Parameters:** as a P-Mode concerns all messages in an ebMS MEP, these parameters
3984 are not associated with any particular message in the MEP, but are attributes of the entire MEP.
- 3985 • **Protocol:** defines protocol-related parameters necessary for interoperating, that are associated
3986 with a particular message of the MEP.
- 3987 • **BusinessInfo:** defines the business profile of a user message in terms of business header
3988 elements and their values (e.g. Service, Action) or other items with business significance
3989 (payload profile, MPC).
- 3990 • **ErrorHandling:** defines the mode of handling and of reporting of errors associated with the
3991 message in this leg.
- 3992 • **Reliability:** defines the reliability contracts and their parameters, applying to the message in this
3993 leg.
- 3994 • **Security:** defines the security level expected for the message in the exchange, and provides
3995 related security context data.

3996 Because messages in the same MEP may be subject to different requirements - e.g. the reliability,
3997 security and error reporting of a response may not be the same as for a request – the P-Mode will be
3998 divided into "legs". Each user message label in an ebMS MEP is associated with a P-Mode leg. Each P-
3999 Mode leg has a full set of parameters of the six categories above (except for General Parameters), even
4000 though in many cases parameters will have same value across the MEP legs. Signal messages that
4001 implement transport channel bindings (such as PullRequest) are also controlled by the same categories
4002 of parameters, except for BusinessInfo group.

4003 The following figure illustrates the general structure of a P-Mode for a Two-Way/Push-and-Pull MEP; for
4004 example, a PurchaseOrder business transaction that includes the pair PurchaseOrderRequest +
4005 PurchaseOrderConfirm. Its binding channel is "Push-and-Pull" e.g. because the buyer cannot receive
4006 incoming requests.

Overall P-Mode Structure for a Two-Way/Push-and-Pull MEP



4007 *Figure 15: P-Mode Structure for Two-Way/Push-and-Pull MEP*

4008 In the above illustration, each leg of the MEP may have different P-Mode parameters, although in many
 4009 cases these parameters will be identical from one leg to the other. Because the P-Mode specifies the
 4010 MEP transport channel binding, it may also specify a set of parameters for the Pull signal, which may be
 4011 subject to specific requirements (reliability, security/authorization).

4012 **Note:**

4013 In general, a Pull signal cannot be precisely targeted to a particular MEP, but instead to
 4014 an MPC. For this reason, all Pull signals for a particular MPC will usually share similar P-
 4015 Mode parameters.

4016 **D.2.1. Notation**

4017 Consider a PurchaseOrder business transaction as defined above.

- 4018 • The P-Mode associated with this type of transaction between two partners, may be called:
 4019 **PurchaseOrder.PMode.**
- 4020 • An index notation is used to identify the legs of an MEP. The part of the P-Mode that relates to
 4021 Leg 1 of the PurchaseOrder MEP ("request" label), will be called
 4022 **PurchaseOrder.PMode[request]**. A number representing the occurrence order may be used
 4023 instead of the leg label, e.g. **PurchaseOrder.PMode[1]**. This is appropriate for a MEP in which
 4024 the legs are strictly serialized over time.
- 4025 • In case there are two sets of P-Mode parameters associated with a leg, as for the pulled "reply",
 4026 the part of the P-Mode that concerns the user message in leg 2 is noted:
 4027 **PurchaseOrder.PMode[2][u]**, while the part of the P-Mode that concerns the (pull) signal
 4028 message in leg 2 is noted: **PurchaseOrder.PMode[2][s]**.

4029 D.3. Processing Mode Parameters

4030 P-Mode parameters define how a message should be processed. These parameters either define
4031 elements that are expected to be found in the message, or processing behavior expected for this
4032 message (e.g. level of reliability, error reporting). Every parameter in this section does not need to be
4033 given a value when defining a P-Mode. In such a case, either the corresponding header element can
4034 take any value for a message processed under this P-Mode, or the MSH behavior this parameter
4035 controls is not constrained by the P-Mode. It is also possible to associate multiple authorized values (or
4036 a range of values) with a parameter in a P-Mode (e.g. multiple MPC values).

4037 D.3.1. General P-Mode Parameters

4038 The general P-Mode parameters (i.e. not specific to any single message in the MEP) are:

- 4039 • **PMode.ID:** (optional) The identifier for the P-Mode, e.g. the name of the business transaction:
4040 PurchaseOrderFromACME. This identifier is user-defined and optional, for the convenience of P-
4041 Mode management. It must uniquely identify the P-Mode among all P-Modes deployed on the
4042 same MSH, and may be absent if the P-Mode is identified by other means, e.g. embedded in a
4043 larger structure that is itself identified, or has parameter values distinct from other P-Modes used
4044 on the same MSH. If the ID is specified, the AgreementRef/@pmode attribute value is also
4045 expected to be set in associated messages.
- 4046 • **PMode.Agreement:** The reference to the agreement governing this message exchange (maps to
4047 eb:AgreementRef in message header).
- 4048 • **PMode.MEP:** The type of ebMS MEP associated with this P-Mode. The value must be a URI,
4049 e.g: <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/oneWay>.
- 4050 • **PMode.MEPbinding:** The transport channel binding assigned to the MEP (push, pull, sync,
4051 push-and-push, push-and-pull, pull-and-push, pull-and-pull, ...). The value must be a URI, e.g:
4052 <http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/ns/core/200704/push>.
- 4053 • **PMode.Initiator.Party:** 1.(PMode.Initiator and its subelements are optional if PMode.Responder
4054 is present.) Qualifies the party initiating the MEP (see Section 2.2.3). A user message initiating
4055 an MEP instance under this P-Mode must have its
4056 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From element contain the same PartyId
4057 elements as the PartyId elements defined in this parameter. Any user message sent to the
4058 initiator must have its eb:PartyInfo/eb:To map to or be compatible with this parameter.
- 4059 • **PMode.Initiator.Role:** Name of the role assumed by the party sending the first message of this
4060 MEP. Either the message element
4061 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role or the element
4062 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To/eb:Role of each message in
4063 this MEP must have this value, depending on the direction of message transfer.
- 4064 • **PMode.Initiator.Authorization.username and PMode.Initiator.Authorization.password:**
4065 Describe authorization information for messages sent by Initiator. These parameters need to be
4066 matched by a wsse:UsernameToken element in a message (in a security header only intended
4067 for authorization) for this message to be processed successfully on receiver side – here by
4068 Responder MSH.
- 4069 • **PMode.Responder.Party:** (PMode.Responder and its subelements are optional if
4070 PMode.Initiator is present.) Qualifies the party responding to the initiator party in this MEP. Any
4071 user message sent to the responder must have its
4072 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To element contain the same PartyId elements
4073 as the PartyId elements defined in this parameter.
- 4074 • **PMode.Responder.Role:** Name of the role assumed by the party receiving the first message of
4075 this MEP. Either the message element
4076 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:From/eb:Role or the element
4077 eb:Messaging/eb:UserMessage/eb:PartyInfo/eb:To/eb:Role of each message in
4078 this MEP must have this value, depending on the direction of message transfer.
- 4079 • **PMode.Responder.Authorization.username and**
4080 **PMode.Responder.Authorization.password:** Describe authorization information for messages
4081 sent by Responder. These parameters need to be matched by a wsse:UsernameToken element

4082 in a message (in a security header only intended for authorization) for this message to be
4083 processed successfully on receiver side – here by Initiator MSH.

4084 The P-Mode parameters that are specific to a P-Mode leg (here, associated with leg 1 of an MEP) are
4085 grouped into five categories: Protocol, BusinessInfo, ErrorHandling, Reliability, and Security:

4086 **D.3.2. PMode[1].Protocol**

- 4087 • **PMode[1].Protocol.Address**: the value of this parameter represents the address (endpoint
4088 URL) of the Receiver MSH (or Receiver Party) to which Messages under this P-Mode leg are to
4089 be sent. Note that a URL generally determines the transport protocol (for example, if the
4090 endpoint is an email address, then the transport protocol must be SMTP; if the address scheme
4091 is "http", then the transport protocol must be HTTP).
- 4092 • **PMode[1].Protocol.SOAPVersion**: this parameter indicates the SOAP version to be used (1.1
4093 or 1.2). In some implementations, this parameter may be constrained by the implementation, and
4094 not set by users.

4095 **D.3.3. PMode[1].BusinessInfo**

4096 Note:
4097 This set of parameters only applies to user messages.

- 4098 • **PMode[1].BusinessInfo.Service**: Name of the service to which the User message is intended to
4099 be delivered. Its content should map to the element
4100 `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Service`.
- 4101 • **PMode[1].BusinessInfo.Action**: Name of the action the User message is intended to invoke. Its
4102 content should map to the element
4103 `eb:Messaging/eb:UserMessage/eb:CollaborationInfo/eb:Action`
- 4104 • **PMode[1].BusinessInfo.Properties[]**: The value of this parameter is a list of properties. A
4105 property is a data structure that consists of four values: the property name, which can be used as
4106 an identifier of the property (e.g. a required property named "messagetype" can be noted as:
4107 `Properties[messagetype].required="true"`); the property description; the property data type; and a
4108 Boolean value, indicating whether the property is expected or optional, within the User message.
4109 This parameter controls the contents of the element
4110 `eb:Messaging/eb:UserMessage/eb:MessageProperties`.
- 4111 • **PMode[1].BusinessInfo.PayloadProfile[]**: This parameter allows for specifying some constraint
4112 or profile on the payload. It specifies a list of payload parts. A payload part is a data structure
4113 that consists of five properties: name (or Content-ID) that is the part identifier, and can be used
4114 as an index in the notation `PayloadProfile[]`; MIME data type (text/xml, application/pdf, etc.);
4115 name of the applicable XML Schema file if the MIME data type is text/xml; maximum size in
4116 kilobytes; and a Boolean value indicating whether the part is expected or optional, within the
4117 User message. The message payload(s) must match this profile.
- 4118 • **PMode[1].BusinessInfo.PayloadProfile.maxSize**:: This parameter allows for specifying a
4119 maximum size in kilobytes for the entire payload, i.e. for the total of all payload parts.
- 4120 • **PMode[1].BusinessInfo.MPC**: The value of this parameter is the identifier of the MPC (Message
4121 Partition Channel) to which the message is assigned. It maps to the attribute
4122 `eb:Messaging/eb:UserMessage/@mpc`.

4123 **D.3.4. PMode[1].ErrorHandling**

4124 Note:
4125 This P-Mode group concerns errors generated by the reception of the message (for
4126 either a User message or a Signal message, unless indicated otherwise) sent over leg 1
4127 of the MEP.

- 4128 • **PMode[1].ErrorHandling.Report.SenderErrorsTo**: This parameter indicates the address, or
4129 comma-separated list of addresses, to which to send ebMS errors generated by the MSH that
4130 was trying to send the message in error.

- 4131 • **PMode[1].ErrorHandling.Report.ReceiverErrorsTo**: This parameter indicates the address, or
4132 comma-separated list of addresses, to which to send ebMS errors generated by the MSH that
4133 receives the message in error; e.g. this may be the address of the MSH sending the message in
4134 error.
- 4135 • **PMode[1].ErrorHandling.Report.AsResponse**: This Boolean parameter indicates whether (if
4136 "true") errors generated from receiving a message in error are sent over the back-channel of the
4137 underlying protocol associated with the message in error, or not.
- 4138 • **PMode[1].ErrorHandling.Report.ProcessErrorNotifyConsumer**: This Boolean parameter
4139 indicates whether (if "true") the Consumer (application/party) of a User Message matching this P-
4140 Mode should be notified when an error occurs in the Receiving MSH, during processing of the
4141 received User message.
- 4142 • **PMode[1].ErrorHandling.Report.ProcessErrorNotifyProducer**: This Boolean parameter
4143 indicates whether (if "true") the Producer (application/party) of a User Message matching this P-
4144 Mode should be notified when an error occurs in the Sending MSH, during processing of the
4145 User Message to be sent.
- 4146 • **PMode[1].ErrorHandling.Report.DeliveryFailuresNotifyProducer**: This Boolean parameter
4147 indicates whether (if "true") the Producer (application/party) of a User Message matching this P-
4148 Mode must always be notified when the delivery to Consumer failed, or whether (if "false"), in
4149 some cases, it is sufficient to notify the Consumer only
4150 (Report.ProcessErrorNotifyConsumer="true"). This assumes that
4151 Reliability.AtLeastOnce.Contract is "true". This also assumes that the Sending MSH
4152 implementation has the ability to determine or to be made aware of all cases of non-delivery that
4153 occur after the message has been received by the Receiving MSH.

4154 **D.3.5. PMode[1].Reliability**

- 4155 • **PMode[1].Reliability.AtLeastOnce.Contract**: If "true", this Boolean parameter indicates that the
4156 "At-Least-Once" reliability contract (see Section 8.2.2) is to be used between MSH and
4157 Consumer (Guaranteed Delivery). It also indicates that this contract applies to ebMS signals
4158 (see Section 8.2.1) – e.g. PullRequest – between the receiving reliability module and the next
4159 MSH component (e.g. RM-Consumer).
- 4160 • **PMode[1].Reliability.AtLeastOnce.Contract.AckOnDelivery**: This Boolean parameter
4161 indicates the semantics of acknowledgments that are generated by the reliability module. It is
4162 usually constrained by the implementation and not set by users. For User messages: if "true", the
4163 acknowledgment is only sent after the message has been delivered by the MSH to the Consumer
4164 entity (see Case 2 in Section 8.2.4). If "false", the only guarantee for the sender when receiving
4165 an acknowledgment is that the User message has been well received (see Case 1 or 3 in
4166 Section 8.2.4), and made available for further processing within the MSH. For Signal messages
4167 – e.g. PullRequest: if "true", indicates that Signal messages are acknowledged only if delivered
4168 (see Section 8.2.1) from the receiving reliability module to the next MSH component (Case 3 in
4169 Section 8.2.4), i.e. to the RM-Consumer (see 8.1.2). If "false", the message acknowledgment
4170 only guarantees receipt of the signal (Case 1 in Section 8.2.4).
- 4171 • **PMode[1].Reliability.AtLeastOnce.Contract.AcksTo**: This parameter is a URI that specifies
4172 where acknowledgments are to be sent. It may contain an anonymous URI (defined in WS-
4173 Addressing). If absent, acknowledgments are to be sent to the same URI associated with the
4174 MSH sending messages reliably.
- 4175 • **PMode[1].Reliability.AtLeastOnce.Contract.AckResponse**: This Boolean is true when an
4176 Acknowledgment must be sent, for a response that is sent reliably.
- 4177 • **PMode[1].Reliability.AtLeastOnce.ReplyPattern**: This parameter indicates whether a reliability
4178 acknowledgment is to be sent as a callback, synchronously in the response (back-channel of
4179 underlying protocol), or as response of separate ack pulling. Three values are possible for this
4180 parameter, when using WS-Reliability: "Response", "Callback", or "Poll".
- 4181 • **PMode[1].Reliability.AtMostOnce.Contract**: If "true", this Boolean parameter indicates that "At-
4182 Most-Once" (or duplicate elimination) should be enforced when receiving a message. The
4183 contract is for delivery between MSH and Consumer for User messages (see Section 8.2.2), and
4184 between reliability module and next MSH component for Signal messages (see Section 8.2.1).

- 4185 • **PMode[1].Reliability.InOrder.Contract:** If "true", this Boolean parameter indicates that this
4186 message is part of an ordered sequence. It only concerns User messages (delivery contract
4187 between MSH and Consumer application, see Section 8.2.2).
- 4188 • **PMode[1].Reliability.StartGroup:** This parameter is a Boolean that may be used to indicate if
4189 messages matching this P-Mode must be associated with a new reliability group or sequence.
4190 For example, a particular Service and Action may have the application semantics of initiating a
4191 new ordered sequence of messages.
- 4192 • **PMode[1].Reliability.Correlation:** This parameter tells how to correlate a message matching
4193 this P-Mode with an existing reliability group or sequence. It is a comma-separated list of XPath
4194 elements relative to the eb:Messaging header. Each one of these XPaths identifies an element
4195 or attribute inside eb:UserMessage or eb:SignalMessage, and may include predicates. For
4196 example, "eb:UserMessage/eb:CollaborationInfo/eb:ConversationId,
4197 eb:UserMessage/eb:MessageProperties/eb:Property[@name=\"ProcessInstance\"] will correlate
4198 all messages that share the same ConversationId and have the same value for the message
4199 property named "ProcessInstance". In case there is no ongoing group or sequence associated
4200 with the values in Reliability.Correlation for a message under this P-Mode, then a new
4201 group/sequence is started.
- 4202 • **PMode[1].Reliability.TerminateGroup:** This parameter is a Boolean value that may be used to
4203 indicate if messages matching this P-Mode must cause the closure of the reliability group or
4204 sequence with which they correlate.

4205 D.3.6. PMode[1].Security

- 4206 • **PMode[1].Security.WSSVersion:** This parameter has two possible values, 1.0 and 1.1. The
4207 value of this parameter represents the version of WS-Security to be used.
- 4208 • **PMode[1].Security.X509.Sign:** The value of this parameter is a list of the names of XML
4209 elements (inside the SOAP envelope) that should be signed, as well as whether or not
4210 attachments should also be signed. The list is represented in two sublists that extend this
4211 parameter: **Sign.Element[]** and **Sign.Attachment[]**. An element within the Element[] list could
4212 be specified either by its XML name or by its qualified name (its XML name and the namespace
4213 to which it belongs). An element within the Attachment[] list is identified by the Content-Id.
- 4214 • **PMode[1].Security.X509.Signature.Certificate:** The value of this parameter identifies the public
4215 certificate to use when verifying signed data.
- 4216 • **PMode[1].Security.X509.Signature.HashFunction:** The value of this parameter identifies the
4217 algorithm that is used to compute the digest of the message being signed. The definitions for
4218 these values are in the [XMLDSIG] specification.
- 4219 • **PMode[1].Security.X509.Signature.Algorithm:** The value of this parameter identifies the
4220 algorithm that is used to compute the value of the digital signature. The definitions for these
4221 values are found in the [XMLDSIG] or [XMLENC] specifications.
- 4222 • **PMode[1].Security.X509.Encryption.Encrypt:** The value of this parameter lists the names of
4223 XML elements(inside the SOAP envelope) that should be encrypted, as well as whether or not
4224 attachments should also be encrypted. The list is represented in two sublists that extend this
4225 parameter: **Encrypt.Element[]** and **Encrypt.Attachment[]**. An element within these lists is
4226 identified as in **Security.X509.Sign** lists.
- 4227 • **PMode[1].Security.X509.Encryption.Certificate:** The value of this parameter identifies the
4228 public certificate to use when encrypting data.
- 4229 • **PMode[1].Security.X509.Encryption.Algorithm:** The value of this parameter identifies the
4230 encryption algorithm to be used. The definitions for these values are found in the [XMLENC]
4231 specification.
- 4232 • **PMode[1].Security.X509.Encryption.MinimumStrength:** The integer value of this parameter
4233 describes the effective strength the encryption algorithm MUST provide in terms of "effective" or
4234 random bits. The value is less than the key length in bits when check bits are used in the key.
4235 So, for example the 8 check bits of a 64-bit DES key would not be included in the count, and to
4236 require a minimum strength the same as supplied by DES would be reported by setting
4237 MinimumStrength to 56.
- 4238 • **PMode[1].Security.UsernameToken.username:** The value of this parameter is the username to

- 4239 include in a WSS Username Token.
- 4240 • **PMode[1].Security.UsernameToken.password**: The value of this parameter is the password to
- 4241 use inside a WSS Username Token.
- 4242 • **PMode[1].Security.UsernameToken.Digest**: The Boolean value of this parameter indicates
- 4243 whether a password digest should be included in the WSS UsernameToken element.
- 4244 • **PMode[1].Security.UsernameToken.Nonce**: The Boolean value of this parameter indicates
- 4245 whether the WSS UsernameToken element should contain a Nonce element.
- 4246 • **PMode[1].Security.UsernameToken.Created**: The Boolean value of this parameter indicates
- 4247 whether the WSS UsernameToken element should have a Created timestamp element.
- 4248 • **PMode[1].Security.PModeAuthorize**: The Boolean value of this parameter indicates whether
- 4249 messages on this MEP leg must be authorized for processing under this P-Mode. If the
- 4250 parameter is "true" this implies that either
- 4251 PMode.Responder.Authorization.{username/password}, if the message is sent by Responder, or
- 4252 PMode.Initiator.Authorization if the message is sent by Initiator, must be used for this purpose,
- 4253 as specified in Section 7.10. For example, when set to "true" for a PullRequest message sent by
- 4254 the Initiator, the pulling will only be authorized over the MPC indicated by this Pull signal if (a)
- 4255 the MPC is the same as specified in the P-Mode leg for the pulled message, and (b) the signal
- 4256 contains the right credentials (e.g. username/password).
- 4257 • **PMode[1].Security.SendReceipt**: The Boolean value of this parameter indicates whether a
- 4258 signed receipt (Receipt ebMS signal) containing a digest of the message must be sent back.
- 4259 • **PMode[1].Security.SendReceipt.ReplyPattern**: This parameter indicates whether the Receipt
- 4260 signal is to be sent as a callback (value "callback"), or synchronously in the back-channel
- 4261 response (value "response"). If not present, any pattern may be used.

4262

APPENDIX E. P-Mode Values and ebMS MEP Bindings

4263

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

4264

4265

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

4266

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.

4267

4268

No combination of P-Mode values other than those listed below are expected to be used. Valid

4269

combinations not explicitly represented in the table below are mentioned in "notes" as variants of the

4270

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.

4271

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

4272 The following table illustrates how the One-Way/Pull MEP binds to HTTP, depending on the values of P-
4273 Mode parameters that affect message content.

4274 No combination of P-Mode values other than those listed below are expected to be used. Valid
4275 combinations not explicitly represented in the table below are mentioned in "notes" as variants of the
4276 most common ones.

4277

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.

4278

[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.

4279

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

4280 The following table illustrates how the Two-Way/Sync MEP binds to HTTP, depending on the values of
4281 P-Mode parameters that affect message content.

4282 No combination of P-Mode values other than those listed below are expected to be used. Valid
4283 combinations not explicitly represented in the table below are mentioned in "notes" as variants of the
4284 most common ones.

4285

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

4286

[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.

4287

APPENDIX F. Compatibility Mapping to ebMS 2.0

4288

F.1. Objectives and Approach

4289

4290

4291

4292

4293

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*.

4294

4295

4296

4297

4298

4299

4300

4301

4302

4303

4304

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.

4305

4306

4307

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:

4308

4309

4310

4311

- 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).

4312

F.2. Compatibility Mapping Rules

4313

4314

4315

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 :

4316

CM rules:

4317

4318

4319

4320

4321

4322

- 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

4323

4324

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

4325

F.2.1. (CM1) Header Mapping Rules

4326

4327

4328

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.

4329 **F.2.1.1. Rule CM1-a: Mapping General Message Information**

4330 eb2:MessageHeader/eb2:MessageData element maps to eb3/Messaging/eb3:MessageInfo, along with
4331 their contained elements (Timestamp, MessageId, RefToMessageId).

4332 Depending on its usage, the optional eb2:TimeToLive would map differently to an eb3 header. In case it
4333 has some application semantics (e.g. validity period of the enclosed business document), such a value
4334 can be added in V3 as eb3:Messageproperties/eb3:property/@name="timetolive". However, it has no
4335 MSH semantics in V3, unlike in V2 where it controls delivery. Implementing similar semantics would be
4336 done as an extension to V3. In case eb2:TimeToLive is used as a reliability feature (e.g. expected
4337 maximum time during which reliability mechanisms are expected to operate on the message before
4338 declaring failure) then it should map to message ExpiryTime (see rule CM3-c).

4339 **F.2.1.2. Rule CM1-b: Mapping Party Information**

4340 eb2:MessageHeader/eb2:From maps to eb3:PartyInfo/eb3:From, along with their sub-elements.

4341 Similarly, eb2:MessageHeader/eb2:To maps to eb3:PartyInfo/eb3:To, along with their sub-elements.

4342 **F.2.1.3. Rule CM1-c: Mapping Collaboration Information**

4343 eb2:ConversationId, eb2:Service, eb2:Action respectively map to
4344 eb3:CollaborationInfo/eb3:ConversationId, eb3:CollaborationInfo/eb3:Service and
4345 eb3:CollaborationInfo/eb3:Action.

4346 **F.2.1.4. Rule CM1-d: Mapping Agreement Reference**

4347 eb2:MessageHeader/eb2:CPAId maps to eb3:CollaborationInfo/eb3:AgreementRef.

4348 **F.2.2. (CM2) Payload Mapping Rules**

4349 **F.2.2.1. Rule CM2-a: Mapping Attachments**

4350 Every attachment (MIME part) in V2 maps to a similar attachment in V3. The SOAP Body should not be
4351 used in V3. If a V3 message that must map to a V2 message has a non-empty SOAP Body, the child
4352 XML document must be mapped to a separate MIME part in V2.

4353 **F.2.3. (CM3) Reliability Mapping Rules**

4354 These rules define how some V2 header elements map to a separate reliability header in V3, and vice-
4355 versa. When the reliability quality of service is not apparent in the V3 reliability header (e.g. in case V3
4356 uses WS-ReliableMessaging protocol), these rules rely on the P-Mode.Reliability parameters to
4357 determine the reliability elements in ebMS2 header.

4358 **F.2.3.1. Rule CM3-a: Acknowledgments**

4359 V2: AckRequested element maps to: in V3, wsrn:Request/AckRequested (if using WS-Reliability),
4360 optional wsrn:AckRequested header if using WS-ReliableMessaging (not necessary to get
4361 acknowledgments).

4362 V2: Acknowledgment element maps to: in V3, wsrn:Response/SequenceReplies (if using WS-
4363 Reliability), wsrn:SequenceAcknowledgment if using WS-ReliableMessaging.

4364 **Note:**

4365 The meaning of acknowledgments may be different in V2 and V3. See Section 8.2.4 for
4366 the options in acknowledgment semantics, depending on which reliability module is
4367 used. In V2, the baseline semantics is "on receipt": the message has been safely stored
4368 in persistent storage or delivered to the application interface. In V3, the recommended
4369 semantics is: the message has been delivered to the application. It may however be
4370 similar to V2 semantics depending on the implementation (e.g. when using WS-

4371 ReliableMessaging). In V3 the P-Mode parameter
4372 Reliability.AtLeastOnce.AckOnDelivery specifies this semantics which in general
4373 depends on the implementation: when "false", it is similar to V2 (on receipt).

4374 **F.2.3.2. Rule CM3-b: Reliability Contracts**

4375 The reliability contracts At-Least-Once delivery, At-Most-Once delivery, In-Order delivery, that in V3 are
4376 specified in the P-Mode, and also in the message header in case WS-Reliability is used, respectively
4377 map to V2 header elements: eb2:AckRequested, eb2:DuplicateElimination, eb2:MessageOrder.

4378 Any of the above reliability contracts requires the use of a reliable messaging module in V3, e.g. an
4379 implementation of WS-Reliability or of WS-ReliableMessaging.

4380 The delivery failure notification in V2 (always required for non-acknowledged messages) is supported by
4381 WS-Reliability and therefore by V3 using WS-Reliability. Such failure notification is not explicitly
4382 mandated by WS-ReliableMessaging, or could take place on either side. In order to achieve the same
4383 notification policy as in V2, when used in V3 an implementation of WS-ReliableMessaging must be
4384 extended with the same notification capability.

4385 **Note:**

4386 The conditions under which delivery failure is notified to the From Party (in eb2) or
4387 message Producer (in eb3) may be different.

4388 **F.2.3.3. Rule CM3-c: Duplicate Elimination**

4389 eb2:MessageHeader/eb2:DuplicateElimination maps to wsrn:Request/wsrn:DuplicateElimination in WS-
4390 Reliability. It maps to the AtMostOnce delivery assurance definition in WS-ReliableMessaging, assuming
4391 an implementation of WS-ReliableMessaging that supports this delivery assurance.

4392 **F.2.3.4. Rule CM3-d: Use of Sequences and Sequence Numbers**

4393 An eb2 message that contains either AckRequested or DuplicateElimination or both, and no
4394 eb2:MessageOrder, may map to a V3 message (when using WS-Reliability) with no wsrn:SequenceNum
4395 – only a wsrn:MessageId/@groupId value, which is unique for every such message.

4396 **Note:**

4397 The elements that identify a message sent reliably in V3 (wsrn:SequenceNum,
4398 wsrn:MessageId/@groupId in WS-Reliability, or
4399 /wsrn:Sequence/wsrn:MessageNumber in WS-ReliableMessaging) do NOT map to the
4400 ebMS message ID element (i.e. eb2:MessageData/eb2:MessageId in V2, and
4401 eb3:MessageInfo/eb3:MessageId in V3).

4402 **F.2.3.5. Rule CM3-e: Message Ordering**

4403 In case message ordering is required:

4404 eb2:MessageOrder maps to wsrn:Request/wsrn:MessageOrder.

4405 eb2:SequenceNumber maps to wsrn:Request/wsrn:SequenceNum (with WS-Reliability).

4406 The scope of a message sequence (and of the message ordering contract) is determined by
4407 eb2:ConversationId in V2, and by MessageId/@groupId in V3; i.e. sequence numbers must be unique
4408 within this scope.

4409 The feature maps to the InOrder delivery assurance definition in WS-ReliableMessaging, assuming an
4410 implementation of WS-ReliableMessaging that supports this delivery assurance.

4411 **F.2.3.6. Rule CM3-f: Expiration Timeout**

4412 In case eb2:MessageHeader/eb2:MessageData/eb2:TimeToLive is used for expressing the maximum
4413 time during which reliability mechanisms are required to handle the message, it maps to
4414 wsrn:Request/wsrn:ExpiryTime.

4415 **F.2.4. (CM4) MEP Mapping Rules**

4416 Defines how V2 header elements that control the MEP in use and its mapping to the underlying protocol,
4417 map into V3 and vice versa. Also defines how CPA elements that control ebMS V2 MEPs map to P-Mode
4418 parameter and vice-versa.

4419 **F.2.4.1. Rule CM4-a: One-Way/Push With No Signals**

4420 In V3, this MEP, with no ebMS signal and no reliability acknowledgments on the response (back-
4421 channel), will map to a V2 message with no SyncReply element in eb2 header. RefToMessageId must
4422 not be used in the V3 message (it has a strict MEP semantics). The agreements map as follows:

4423 V2 (CPA): syncReplyMode=none.

4424 V3 (P-Mode): PMode.MEP="One-way", PMode.MEPbinding="push",
4425 PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern must NOT be
4426 "Response".

4427 **F.2.4.2. Rule CM4-b: One-Way/Push With Signals**

4428 One-Way / Push in V3, with ebMS signal and reliability acknowledgments on the response (back-
4429 channel), will map to a V2 message with SyncReply element in eb2 header. RefToMessageId must not
4430 be used in the V3 message (it has a strict MEP semantics). The agreements map as follows:

4431 V2 (CPA): syncReplyMode= mshSignalsOnly.

4432 V3 (P-Mode): PMode.MEP="One-way", PMode.MEPbinding="push",
4433 PMode.ErrorHandling.Report.AsResponse="true", PMode.Reliability.ReplyPattern="Response".

4434 **F.2.4.3. Rule CM4-c: Two-Way/Sync With No Signals**

4435 In V3, this MEP, with no ebMS signal and no reliability acknowledgments on the response (back-
4436 channel), will map to a V2 message (1st leg) with SyncReply element in eb2 header. In both versions,
4437 the response message refers to the request (leg 1) using RefToMessageId. The agreements map as
4438 follows:

4439 V2 (CPA): (leg 1) syncReplyMode= responseOnly.

4440 V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="sync",
4441 PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern may NOT be
4442 "Response".

4443 **F.2.4.4. Rule CM4-d: Two-Way/Sync With Signals**

4444 In V3, this MEP, with ebMS signal and reliability acknowledgments on the response (back-channel), will
4445 map to a V2 message (1st leg) with SyncReply element in eb2 header. In both versions, the response
4446 message refers to the request (leg 1) using RefToMessageId. The agreements map as follows:

4447 V2 (CPA): (leg 1) syncReplyMode= signalsAndResponse

4448 V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="sync",
4449 PMode.ErrorHandling.Report.AsResponse="true". PMode.Reliability.ReplyPattern ="Response".

4450 **F.2.4.5. Rule CM4-e: Two-Way/Push-and-Push**

4451 In V3, this MEP will map to an exchange of two messages in V2, where the second message refers to the
4452 first one using RefToMessageId (as in V3). The agreements map as follows:

4453 Option 1: (signals may be sent back on underlying response)

4454 V2 (CPA): (leg 1 and leg 2) syncReplyMode= mshSignalsOnly.

4455 V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="Push-and-Push".

4456 PMode.ErrorHandling.Report.AsResponse="true". PMode.Reliability.ReplyPattern="Response".

4457 Option 2: (signals may NOT be sent back on underlying response)

4458 V2 (CPA): (leg 1 and leg 2) syncReplyMode= none.
4459 V3 (P-Mode): PMode.MEP="Two-way", PMode.MEPbinding="Push-and-Push".
4460 PMode.ErrorHandling.Report.AsResponse="false". PMode.Reliability.ReplyPattern different from
4461 "Response".

4462 **F.2.5. (CM5) Signal Mapping Rules**

4463 **F.2.5.1. Rule CM5-a: Error Metadata Mapping**

4464 The metadata mapping of the Error elements in V2 and V3 is as follows. In some cases the semantics is
4465 close though not exactly same.

- 4466 (a) Cases where a straight mapping exist from V2 to V3:
- 4467 1. V2: Error/@severity (warning, error) maps to V3: eb:Error/@severity (respectively:
4468 warning, failure)
 - 4469 2. V2: Error/@codeContext maps to V3: eb:Error/@origin
 - 4470 3. V2: Error/@errorCode maps to V3: eb:Error/shortDescription
 - 4471 4. V2: Error/@location maps to V3: eb:Error/ErrorDetail
 - 4472 5. V2: Error/Description maps to V3: eb:Error/Description
 - 4473 6. V2: MessageData/RefToMessageId maps to V3: eb:Error/@refToMessageInError
- 4474 (b) Cases where error element in V2 has no specified counterpart in V3:
- 4475 1. V2: Error/@id. In V3 would map to: XML Id attribute.
- 4476 (c) Cases where error element in V3 has no specified counterpart in V2:
- 4477 1. V3: eb:Error/@errorCode
 - 4478 2. V3: eb:Error/@category

4479 **F.2.5.2. Rule CM5-b: Error Value Mapping**

4480 The value-equivalence between Errors in V2 and V3 is as follows, based on the semantics of these
4481 errors:

4482 Note: the severity levels may not map in some cases, meaning that processing may continue in V3 while
4483 aborting in V2.

- 4484 (a) Cases where a straight mapping exist from V2 to V3:
- 4485 1. V2: ValueNotRecognized maps to V3: ValueNotRecognized
 - 4486 2. V2: NotSupported maps to V3: FeatureNotSupported
 - 4487 3. V2: DeliveryFailure maps to V3: DeliveryFailure
 - 4488 4. V2: MimeProblem maps to V3: MimeInconsistency
- 4489 (b) Cases where a case by case mapping exist from V2 to V3:
- 4490 1. V2: Inconsistent may map to V3: ValueInconsistent, in some cases InvalidHeader
 - 4491 2. V2: SecurityFailure maps to V3: FailedAuthentication or FailedDecryption
 - 4492 3. V2: OtherXML may map to V3: Other
 - 4493 4. V2: Unknown maps to (in most cases) V3: Other
- 4494 (c) Cases where error value in V2 has no counterpart in V3:
- 4495 1. V2: TimeToLiveExpired: no counterpart (not relevant).
- 4496 (d) Cases where error value in V3 has no counterpart in V2:
- 4497 1. V3: ConnectionFailure,
 - 4498 2. V3: EmptyMessagePartitionChannel

- 4499 3. V3: ProcessingModeMismatch
4500 4. V3: DysfunctionalReliability

4501 **F.2.5.3. Rule CM5-c: Ping and Pong Services**

4502 (a) Ping Service:

- 4503 1. V2: Service element: urn:oasis:names:tc:ebxml-msg:service, and Action element
4504 containing: Ping.
4505 2. V3: Service element: http://docs.oasis-open.org/ebxml-
4506 msg/ebms/v3.0/ns/core/200704/service, and Action element: http://docs.oasis-
4507 open.org/ebxml-msg/ebms/v3.0/ns/core/200704/test

4508 (b) Pong Service:

4509 No corresponding Pong service in V3 Core specification. This feature may be defined in
4510 a forthcoming Part 2 (Advanced Features).

4511 **F.2.6. (CM6) Processing Mode Mapping Rules**

4512 These mapping rules, to be specified in a separate white paper, will define how the messaging subset of
4513 an existing CPA instance in V2 maps to a V3 P-Mode. They also provide guidance on how to represent
4514 a P-Mode with a CPA and related extensions.
4515

4516 APPENDIX G. Conformance

4517 This section introduces the notion of conformance profiles for MSH implementations. The expression
4518 "conformance profile" is to be understood in the sense of [QAFW]. A conformance profile in ebMS will
4519 define a class of implementations that may implement only a subset of this specification, and/or a
4520 particular set of options (e.g. transport protocol binding, SOAP version). This specification does not
4521 define nor recommend any specific conformance profile. Such conformance profiles will be defined
4522 separately from the ebMS standard, in an adjunct document. A particular conformance profile will be
4523 distinguished as the baseline for achieving interoperability between most implementations dedicated to
4524 e-Business or e-Government.

4525 The section defines a common structure and syntax for defining conformance profiles.

4526 Note: "Conformance profile" should not be confused with "usage profile":

- 4527 • *Conformance profile*: defines a set of capabilities that an MSH implementation must have. This is
4528 determined at development time regardless of the way the MSH is being used later.
- 4529 • *Usage profile*: defines a way of using an MSH implementation, that a community of users has
4530 agreed upon. This may in turn require a particular conformance profile.

4531 For example, a conformance profile may require that an MSH support the optional MessageProperties
4532 header element, meaning it is able to extract it from a received message or to add it to a message to be
4533 sent. In contrast, a usage profile will additionally require that some specific property name be present in
4534 the MessageProperty element of each message.

4535 The interpretation of normative material follows the general rule below, as a complement to RFC2119:

- 4536 • When the keywords OPTIONAL, SHOULD and MAY apply to the behavior of the implementation,
4537 the implementation is free to support these behaviors or not, as meant in [RFC2119].
- 4538 • When the keywords OPTIONAL, SHOULD and MAY apply to message contents that relate to a
4539 more general feature, an implementation that conforms to a profile requiring support for this
4540 feature MUST be capable of processing these optional message contents according to the
4541 described ebXML semantics.
- 4542 • The keywords REQUIRED, SHALL or MUST indicate features that an MSH must support or
4543 implement, but only within the context of a conformance profile requiring support for this feature
4544 or module containing this feature.
- 4545 • When an MSH receives a message that exhibits some content feature that is either
4546 recommended or required by the specification, and if this MSH implements a conformance profile
4547 that does not require support for that content feature, then it MUST generate a
4548 FeatureNotSupported error (see Section 6).

4549

4550

APPENDIX H. Acknowledgments

4551

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

4552

4553

Hamid Ben Malek, Fujitsu Software <hbenmalek@us.fujitsu.com>

4554

Jacques Durand, Fujitsu Software <jdurand@us.fujitsu.com>

4555

Ric Emery, Axway Inc. <remery@us.axway.com>

4556

Kazunori Iwasa, Fujitsu Limited <kiwasa@jp.fujitsu.com>

4557

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

4558

Rajashekar Kailar, Centers for Disease Control and Prevention <kailar@bnetal.com>

4559

Dale Moberg, Axway Inc. <dmoberg@us.axway.com>

4560

Sacha Schlegel, Individual <sacha@schlegel.li>

4561

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

4562

4563

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

4564

Doug Bunting, Sun Microsystems <doug.bunting@sun.com>

4565

Matthew MacKenzie, Adobe Systems Incorporated <mattm@adobe.com>

4566

Jeff Turpin, Axway Inc. <jturpin@us.axway.com>