

1
2
3

4
OASIS

5 **ebXML Test Framework**
6 **Committee Specification Version 1.0**

7

8 **OASIS ebXML Implementation, Interoperability and**
9 **Conformance Technical Committee**

10 **07 March, 2003**

11 **Document identifier:**
12 ebxml-iic-test-framework-10

13 **Location:**
14 http://www.oasis-open.org/committees/documents.php?wg_abbrev=ebxml-iic

15 **Authors/Editors:**
16 Steven Yung, Sun Microsystems <steven.yung@sun.com>
17 Prakash Sinha, IONA <prakash.sinha@iona.com>
18 Matthew Mackenzie, Individual <matt@mac-kenzie.net>
19 Hatem El Sebaaly, IPNet Solutions <hatem@ipnetsolutions.com>
20 Monica Martin, Sun Microsystems <monica.martin@sun.com>
21 Jacques Durand, Fujitsu <jdurand@fsw.fujitsu.com>
22 Michael Kass, NIST <michael.kass@nist.gov>
23 Eric VanLydegraf, Kinzan <ericv@kinzan.com>
24 Jeff Turpin, CycloneCommerce <jturpin@cyclonecommerce.com>
25 Serm Kulvatunyou, NIST <serm@nist.gov>

26 **Contributors:**
27
28 Christopher Frank <C.Frank@seeburger.de>

29 **Abstract:**
30 This document specifies ebXML interoperability testing specification for the eBusiness
31 community.

32 **Status:**
33 This document has been approved as a committee specification, and is updated periodically on
34 no particular schedule.
35 Committee members should send comments on this specification to the ebxml-iic@lists.oasis-open.org list. Others should subscribe to and send comments to the ebxml-iic-comment@lists.oasis-open.org list. To subscribe, send an email message to ebxml-iic-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.
36
37 For more information about this work, including any errata and related efforts by this committee,
38 please refer to our home page at <http://www.oasis-open.org/committees/ebxml-iic>.

39
40
41
42 **Errata to this version:**
43 None
44
45

46 Table of Contents

47	1	Introduction.....	5
48	1.1	Summary of Contents of this Document.....	5
49	1.2	Document Conventions	5
50	1.3	Audience	5
51	1.4	Caveats and Assumptions	6
52	1.5	Related Documents	6
53	1.6	Minimal Requirements for Conformance	6
54	2	Principles and Methodology of Operations	8
55	2.1	General Objectives	8
56	2.2	General Methodology	9
57	3	The Test Framework Components.....	11
58	3.1	The Test Driver	11
59	3.1.1	Functions	11
60	3.1.2	Using the Test Driver in Connection Mode	13
61	3.1.3	Using the Test Driver in Service Mode.....	15
62	3.2	The Test Service.....	17
63	3.2.1	Functions and Interactions	17
64	3.2.2	Modes of Operation of the Test Service.....	19
65	3.2.3	Parameters of the Test Service.....	20
66	3.2.4	The Actions of the Test Service	21
67	3.3	Executing Test Cases.....	25
68	3.3.1	Test Case as a Sequence of Test Steps.....	25
69	3.3.2	Related Message Data and Message Declarations	26
70	3.3.3	Related Configuration Data	26
71	Part II:	Test Suite Representation	28
72	4	Test Suite	29
73	4.1	Conformance vs. Interoperability Test Suite.....	29
74	4.2	The Test Suite Document.....	30
75	4.2.1	Test Suite Metadata	32
76	4.2.1	The ConfigurationGroup.....	33
77	5	Test Requirements	38
78	5.1	Purpose and Structure.....	38
79	5.2	The Test Requirements Document.....	38
80	5.3	Specification Coverage	40
81	5.4	Test Requirements Coverage (or Test Run-Time Coverage)	41
82	6	Test Profiles.....	43
83	6.1	The Test Profile Document	43
84	6.2	Relationships between Profiles, Requirements and Test Cases.....	44
85	7	Test Cases	46
86	7.1	Structure of a Test Case	46
87	7.1.1	Test Steps	47
88	7.1.2	Test Step Operations	47

89	7.1.3 The PutMessage Operation	48
90	7.1.4 The Message Declaration	50
91	7.1.5 The SetPayload Operation	69
92	7.1.6 The Dsign Operation	70
93	7.1.7 The GetMessage Operation.....	73
94	7.1.8 The TestPreCondition Operation.....	75
95	7.1.9 The TestAssertion Operation	77
96	7.1.10 The GetPayload Operation.....	79
97	7.1.11 Message Store Schema	80
98	7.1.12 Service-Specific Message Payloads	82
99	7.1.13 Test Report Schema	87
100	8 Test Material.....	89
101	8.1.1 Testing Profile Document.....	89
102	8.1.2 Test Requirements Document.....	89
103	8.1.3 Test Suite Document.....	89
104	8.1.4 Base CPA and derived CPAs.....	90
105	9 Test Material Examples.....	91
106	9.1 Example Test Requirements	91
107	9.1.1 Conformance Test Requirements	91
108	9.1.2 Interoperability Test Requirements	93
109	9.2 Example Test Profiles.....	94
110	9.2.1 Conformance Test Profile Example	94
111	9.2.2 Interoperability Test Profile Example	95
112	9.3 Example Test Suites.....	95
113	9.3.1 Conformance Test Suite.....	95
114	9.3.2 Interoperability Test Suite.....	99
115	Appendix A (Normative) The ebXML Test Profile Schema.....	104
116	Appendix B (Normative) The ebXML Test Requirements Schema	105
117	Appendix C (Normative) The ebXML Test Suite Schema and Supporting Subschemas	111
118	Appendix D (Normative) The ebXML Message Store Schema	128
119	Appendix E (Normative) The ebXML Test Report Schema	132
120	Appendix F (Normative) Service Related Message Schema.....	135
121	Appendix G Terminology.....	137
122	Appendix H References	140
123	H.1 Normative References	140
124	H.2 Non-Normative References.....	141
125	Appendix I Acknowledgments	142
126	I.1 Committee Members	142
127	Appendix J Notices	144
128		

129 1 Introduction

131 1.1 Summary of Contents of this Document

132 This specification defines a test suite for ebXML Messaging basic interoperability. The testing procedure
133 design and naming conventions follow the format specified in the Standard for Software Test
134 Documentation IEEE Std 829-1998.

135 This specification is organized around the following topics:

- 136 • Interoperability testing architecture
- 137 • Test cases for basic interoperability
- 138 • Test data materials

140 1.2 Document Conventions

141 Terms in *Italics* are defined in the ebXML Glossary of Terms in the TestFramework specification
142 [ebTestFramework]. Terms listed in **Bold Italics** represent the element and/or attribute content. Terms
143 listed in Courier font relate to test data. Notes are listed in Times New Roman font and are informative
144 (non-normative). Attribute names begin with lowercase. Element names begin with Uppercase.

145 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
146 RECOMMENDED, MAY and OPTIONAL, when they appear in this document, are to be interpreted as
147 described in [RFC2119] as quoted here:

- 148 • *MUST: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute
149 requirement of the specification.*
- 150 • *MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of
151 the specification.*
- 152 • *SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in
153 particular circumstances to ignore a particular item, but the full implications MUST be understood and
154 carefully weighed before choosing a different course.*
- 155 • *SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid
156 reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full
157 implications should be understood and the case carefully weighed before implementing any behavior
158 described with this label.*
- 159 • *MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may
160 choose to include the item because a particular marketplace requires it or because the vendor feels that it
161 enhances the product while another vendor may omit the same item. An implementation that does not
162 include a particular option MUST be prepared to interoperate with another implementation which does
163 include the option, though perhaps with reduced functionality. In the same vein an implementation that does
164 include a particular option MUST be prepared to interoperate with another implementation which does not
165 include the option (except, of course, for the feature the option provides).*

167 1.3 Audience

168 The target audience for this specification is:

- 169 • The community of software developers who implement and/or deploy the ebXML Messaging
170 Service (ebMS),

- 171 • The testing or verification authority, which will implement and deploy conformance or
172 interoperability testing for ebXML Messaging implementations.

173

174 **1.4 Caveats and Assumptions**

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

177

178 **1.5 Related Documents**

179 The following set of related specifications are developed independent of this specification as part of the
180 ebXML initiative, they can be found on the OASIS web site (<http://www.oasis-open.org>).

- 181 • **ebXML Collaboration Protocol Profile and Agreement Specification [ebCPP]** – CPP defines
182 one business partner's technical capabilities to engage in electronic business collaborations with
183 other partners by exchanging electronic messages. A CPA documents the technical agreement
184 between two (or more) partners to engage in electronic business collaboration. The MS Test
185 Requirements and Test Cases will refer to CPA documents or data as part of their material, or
186 context of verification.
- 187 • **ebXML Messaging Service Specification [ebMS]** – defines the messaging protocol and
188 service for ebXML, which provide a secure and reliable method for exchanging electronic
189 business transactions using the Internet.
- 190 • **ebXML Test Framework [ebTestFramework]** – describes the test architecture, procedures and
191 material that are used to implement the MS Interoperability Test Suite, as well as the test harness
192 for this suite.
- 193 • **ebXML MS Conformance Test Suite [ebMSConfTestSuite]** – describes the Conformance test
194 suite and material for Messaging Services.
- 195 • **ebXML Registry Specification [ebRS]** – defines how one party can discover and/or agree upon
196 the information the party needs to know about another party prior to sending them a message
197 that complies with this specification. The Test Framework is also designed to support the testing
198 of a registry implementation.
- 199 • **ebXML Business Process Specification Schema [BPSS]** – defines how two parties can
200 cooperate through message-based collaborations, which follow particular message
201 choreographies. The Test Framework is also designed to support the testing of a business
202 process implementation.

203

204 **1.6 Minimal Requirements for Conformance**

205 An implementation of the Test Framework specified in this document MUST satisfy ALL of the following
206 conditions to be considered a conforming implementation:

- 207 • It supports all the mandatory syntax, features and behavior (as identified by the [RFC2119] key words
208 MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT) defined in Part 1.1.1 – Document Conventions.
- 209 • It supports all the mandatory syntax, features and behavior defined for each of the components of the Test
210 Framework.

211 It complies with the following interpretation of the keywords OPTIONAL and MAY: When these keywords
212 apply to the behavior of the implementation, the implementation is free to support these behaviors or not,
213 as meant in [RFC2119]. When these keywords apply to data and configuration material used by an

214 implementation of the Test Framework, a conforming implementation of the Test Framework MUST be
215 capable of processing these optional materials according to the described semantics.

216

217

218

219

220

221

222

223

224

225 **2 Principles and Methodology of Operations**

226

227 **2.1 General Objectives**

228

229 The ebXML Test Framework is intended to support conformance and interoperability testing for ebXML
230 specifications. It describes a testbed architecture and its software components, how these can be
231 combined to create a test harness for each type of ebXML testing. It also describes the test material to be
232 processed by this architecture, a mark-up language and format for representing test requirements, and
233 test suites (set of Test Cases).

234

235 The Test Framework described here has been designed to achieve the following objectives:

236

- 237 • The Test Framework is a foundation for testing all ebXML architectural components such as
238 Messaging, Registry, BPSS, CPA, and Core Components.
- 239
- 240 • Test Suites and Test Cases that are related to these standards, can be defined in a formal
241 manner (including Test Steps and verification conditions). They can be automatically processed
242 by the Test Framework, and their execution can easily be reproduced.

243

244 The harnessing of an ebXML implementation (or possibly several, e.g. in case of interoperability) with the
245 Test Framework requires a moderate effort. It generally requires some interfacing work specific to an
246 implementation, in the case no standard interface (API) has been specified. For example, the Test
247 Service (a component of the Test Framework) defines Actions that will need to be called by a particular
248 MSH implementation. Besides this kind of interfacing, no application code needs to be written.

249

- 250 • Several testbed configurations - or test harnesses - can be derived from the Test Framework,
251 depending on the objectives of the testing. For example, MS conformance testing will include a
252 particular combination (architecture) of some components of the Test Framework, while
 interoperability testing will require another set-up.

253

- 254 • Operating the Test Framework - or one of the test harnesses that can be derived from it – in order
255 to execute a test suite, does not require advanced expertise in the framework internals, once the
256 test suites have been designed. The tests should be easy to operate and to repeat with moderate
257 effort or overhead, by users of the ebXML implementation(s) and IT staff responsible for
258 maintaining the B2B infrastructure, without expertise in testing activity.

259

- 260 • Users can define new Test Suites and Test Cases to be run on the framework. For this, they will
261 script their tests using the proposed test suite definition language or mark-up (XML-based) for
262 Test Cases.

263

- 264 • A Test Suite (either for conformance or for interoperability) can be run entirely and validated from
265 one component of the framework: the Test Driver. This means that all test outputs will be
266 generated - and test conditions verified - by one component, even if the test harness involves
267 several – possibly remote – components of the framework.

- 268 • The verification of each Test Case is done by the Test Driver at run-time, as soon as the Test
269 Case execution is completed. The outcome of the verification can be obtained immediately as the
270 Test Suite has completed, and a verification report be generated.

271

272 **2.2 General Methodology**

273

274 This specification only addresses the technical aspect of ebXML testing, and this section describes the
275 portion of testing methodology that relates directly to the usage of the Test Framework. A more general
276 test program for ebXML, describing a comprehensive methodology oriented toward certification, is
277 promoted by the OASIS Conformance TC and is described in [ConfCertTestFrmk] (NIST). When
278 conformance certification is the objective, the ebXML Test Framework should be used in a way that is
279 compliant with a conformance certification model as described in [ConfCertModelNIST]. More general
280 resources on Testing methodology and terminology can be found on the OASIS site (www.oasis-open.org), as well as at NIST (www.itl.nist.gov.)

282 This specification adopts the terminology and guidelines published by the OASIS Conformance
283 Committee [ConfReqOASIS].

284

285 The Test Framework is intended for the following mode of operation, when testing for conformance or for
286 interoperability. In order for a testing process (or validation process) to be conform to this specification,
287 the following phases need to be implemented:

288

- 289 • Phase 1: **Test Plan** (RECOMMENDED). An overall test plan is defined, which includes a
290 validation program and its objectives, the conditions of operations of the testing, levels or profiles
291 of conformance or of interoperability, and the requirements for Candidate Implementations to be
292 tested (context of deployment, configuration).

293

- 294 • Phase 2: **Test Requirements Design** (MANDATORY). A list of Test Requirements (also called
295 Test Assertions) is established for the tested specification, and for the profile/level of
296 conformance/interoperability that is targeted. These Test Requirements should refer to the
297 specification document. Jointly to this list, it is RECOMMENDED that Specification Coverage be
298 reported. This document shows, for each feature in the original specification, the Test
299 Requirements items that address this feature. It also estimates to which degree the feature is
300 validated by these Test Requirements items.

301

- 302 • Phase 3: **Test Harness Design** (MANDATORY). A Test Harness is defined for this particular test
303 plan. It describes an architecture built from components of the Test Framework, along with
304 operation instructions and conditions. In order to be conforming to this specification, a test
305 harness MUST be described as a system that includes a Test Driver as specified in this
306 document, and MUST be able to interpret conforming test suites.

307

- 308 • Phase 4: **Test Suite Design** (MANDATORY). Each Test Requirement from Phase 2 is translated
309 into one or more Test Cases. A Test Case is defined as a sequence of operations (Test Steps)
310 over the Test Harness. Each Test Case includes: configuration material (CPA data), message
311 material associated with each Test Step, test verification condition that defines criteria for passing
312 this test. All this material, along with particular operation directives, defines a Test Suite, as
313 specified in Part II. In order to be conforming to this specification, a test suite needs to be
314 described as a document (XML) conforming to part II of this specification.

315

- 316 • Phase 5: **Validation Conditions** (RECOMMENDED). Validation criteria are defined for the profile
317 or level being tested, and expressed as a general condition over the set of results from the
318 verification report of each Test Case of the suite. These validation criteria define the certification
319 or “badging” for this profile/level.

320

- 321 • Phase 6: **Test Suite Execution** (MANDATORY). The Test Suite is interpreted and executed by
322 the test Driver component of the Test Harness.

323

324 **3 The Test Framework Components**

325

326 The components of the framework are designed so that they can be combined in different configurations,
327 or Test Harnesses.

328

329 We describe here two components that are central to the Test Framework:

330

331 • The Test Driver, which interprets Test Case data and drives Test Case execution.

332 • The Test Service, which implements some test operations (actions) that can be triggered by
333 messages. These operations support and automate the execution of Test Cases.

334

335 These components interface with the ebXML Message Service Handler (MSH), but are not restricted to
336 testing an MSH implementation.

337

338 **3.1 The Test Driver**

339

340 The Test Driver is the component that drives the execution of each step of a Test Case. Depending on
341 the test harness, the Test Driver may drive the Test Case by interacting with other components in
342 *connection mode* or in *service mode*.

343 • In connection mode, the Test Driver directly generates ebXML messages at transport protocol
344 level – e.g. by using an appropriate transport adapter.

345 • In service mode, the Test Driver does not operate at transport level, but at application level, by
346 invoking actions in the Test Service, which is another component of the framework. These actions
347 will in turn send or receive messages to and from the MSH.

348

349 **3.1.1 Functions**

350

351 The primary function of the Test Driver is to parse and interpret the Test Case definitions that are part of a
352 Test Suite, as described in the Test Framework mark-up language. Even when these Test Cases involve
353 several components of the Test Framework, the interpretation of the Test Cases is under control of the
354 Test Driver.

355 The Test Driver component of the ebXML Test Framework MUST have the following capabilities:

356

357 • **Self-Configuration** - Based upon supplied Test Case configuration parameters specified in the
358 ebXML TestSuite.xsd schema (Appendix C), Test Driver configuration is done at startup, and
359 MAY be modified at the TestCase and TestStep levels as well.

360 • **ebXML Message Construction** – Includes MIME, SOAP and ebXML portions of the message

- **Persistence of (Received) Messages** – received messages MUST persist for the life of a Test Case. Persistent messages MUST validate to the ebXMLMessageStore.xsd schema in Appendix D.
- **Parse and query persistent messages** – Test Driver MUST use XPath query syntax to query MIME, SOAP and ebXML persistent message content
- **Parse and query message payloads** – Test Driver MUST support XPath query syntax to query XML message payloads of persistent messages.
- **Controls the execution and workflow of the steps of a Test Case.** Some steps may be executed by other components, but their initiation is under control of the Test Driver.
- **Repeat previously executed Test Steps** – Test Driver MUST be capable of repeating previously executed Test Steps for the current Test Case.
- **Send messages** - Either directly at transport layer (e.g. by opening an HTTP connection), or by using Test Service actions.
- **Receive messages** - Either directly at transport layer, or by notification from Test Service actions.
- **Perform discreet message content validation** – Test Driver MUST be capable of performing discreet validation of Time, URI, Signature and the entire XML message
- **Perform discreet payload content validation** – Test Driver MUST be capable of performing discreet validation of Time, URI, Signature and an XML payload
- **Report Conformance Test Results** – Test Driver MUST generate an XML conformance report for all executed tests in the profile. Conformance reports MUST validate to the ebXMLTestReport.xsd schema in Appendix E.

A possible design that supports these functions is illustrated in Figure 1.

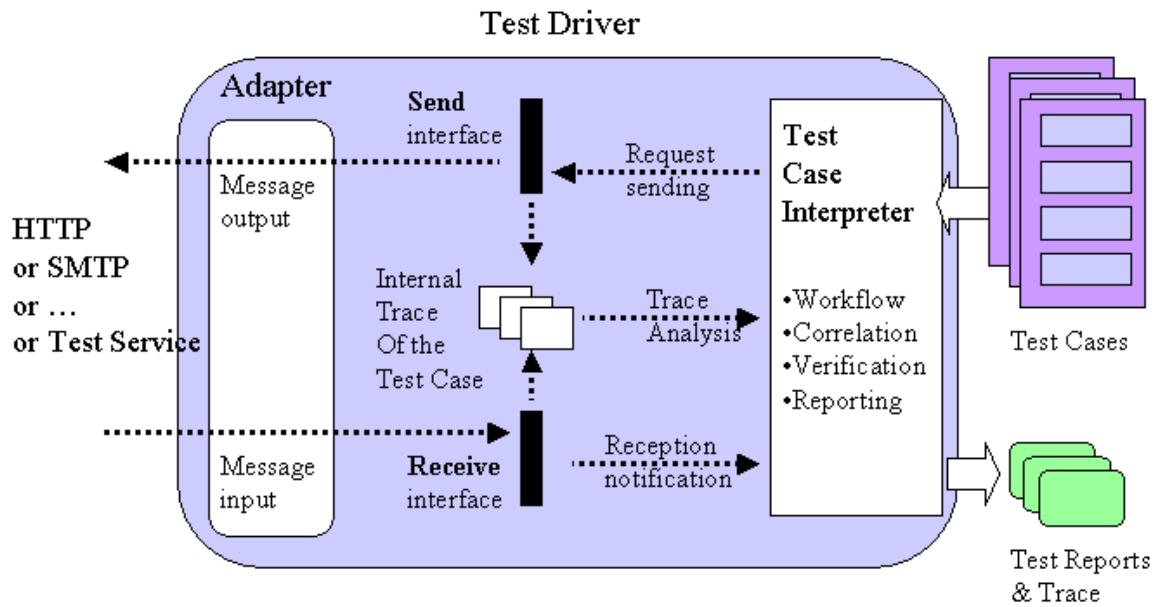


Fig 1. The Test Driver: Functions and Data Flows

1

386
387

388 3.1.2 Using the Test Driver in Connection Mode

389

390 The Test Driver MUST be able to control the inputs and outputs of an MSH at transport level. This can be
 391 achieved by using an embedded transport adapter. This adapter has transport knowledge, and can
 392 format message material into the right transport envelope. Independently from the way to achieve this,
 393 the Test Driver MUST be able to:

- Create a message envelope for the transports authorized by ebXML MS 2.0, and generate fully formed messages for this transport.
- Parse a message envelope for the transports authorized by ebXML MS 2.0, and extract header data from a message, as well as from the message payload in case it is an XML document.
- Open a message communication channel (connection) with a remote ebXML message handler.
 In that case the Test Driver is said to operate in connection mode.

400 When used in connection mode, the Test Driver is acting as a transport end-point that can receive or
 401 send messages with an envelope consistent with the transport protocol (e.g. HTTP or SMTP). The
 402 interaction between the MSH and the Test Service is of same nature as the interaction between the MSH
 403 and an application (as the Test Service simulates an application), i.e. it involves the MSH API, and/or a
 404 callback mechanism. Figure 2 illustrates how the Test Driver operates in connection mode.

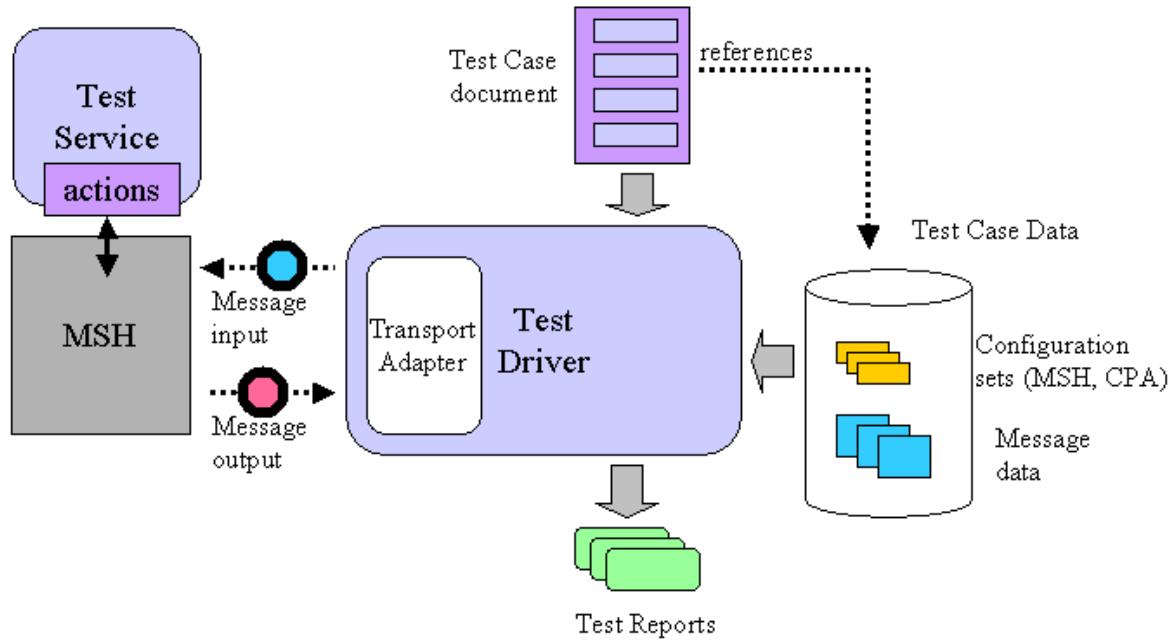


Fig 2. Test Driver used in Connection Mode

2

405

406

407 Figure 3 shows an example of conformance test harness with Test Driver used in connection mode.

408

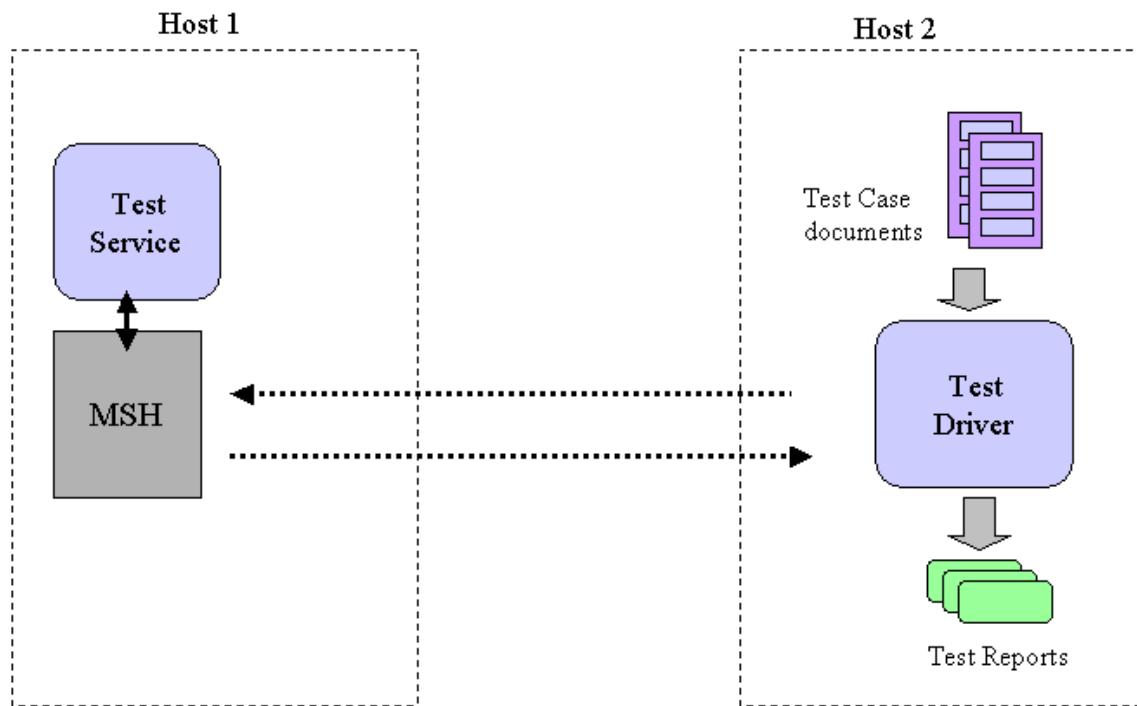


Fig 3. Example of Test Driver in Connection Mode: remote Conformance Testing of a Message Handler

3

409

410 3.1.3 Using the Test Driver in Service Mode

411

412 In this configuration, the Test Driver directly interacts with the Service/Actions of the Test Service
 413 component, without involving the transport layer, e.g. by invoking these actions via a software interface, in
 414 the same process space. This allows for controlling the Test Cases execution from the application layer
 415 (as opposed to the transport layer). Such a configuration is appropriate when doing interoperability testing
 416 - for example between two MSH implementations – and in particular, in situations where the transport
 417 layer should not be tampered with, or interfered with. The interactions with the Test Service will consist
 418 of:

419

- 420 • **Sending:** One action of the Test Service, the “Initiator”, serves as a channel to send requests to
 421 the MSH it has been interfaced with. This action – normally triggered by received messages –
 422 also MUST provide an interface at application level. When invoked by a call that contains
 423 message data, the action generates a sending request to the MSH API for this message.
- 424
- 425 • **Receiving:** As all actions of the Test Service can participate in the execution of a Test Case (i.e.
 426 of its Test Steps), the Test Driver needs to be aware of their invocation by incoming messages.
 427 Each of these actions will notify the Test Driver through its “Receive” interface, passing received
 428 message data, as well as response data. This way, the Test Driver will build an internal trace (or
 429 state) for the Test Case execution, and will be able to verify the test based on this data.

430
431
432
433
434

The Test Driver MUST support the above communication operations with the Test Service. This may be achieved by using an embedded Service Adapter to bridge the sending and receiving functions of the Test Driver, with the Service/Action calls of the Test Service. Figure 4 illustrates how the Test Driver operates with a Service Adapter.

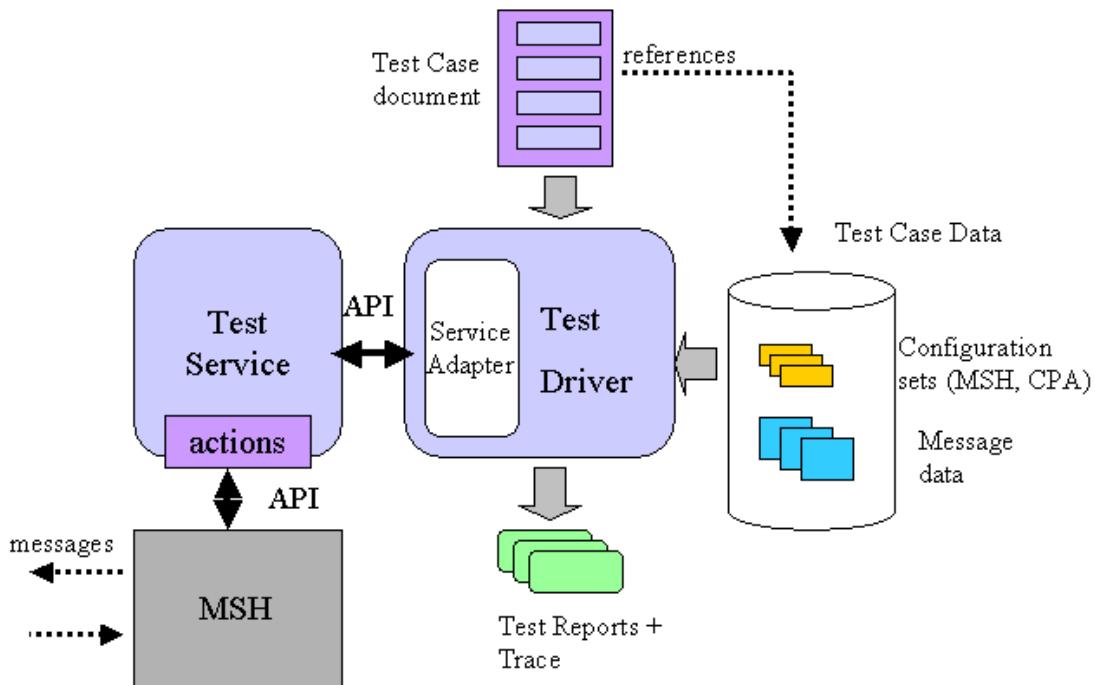


Fig 4. Test Driver used in Service Mode

4

435
436
437

438 This design allows for a minimal exposure of the MSH-specific API, to the components of the Test
439 Framework. The integration code that needs to be written for connecting the MSH implementation is then
440 restricted to an interface with the Service/Actions defined by the framework. Neither the Test Driver, nor
441 the Service Adapter, need to be aware of the MSH-specific interface. An example of test harness using
442 the Test Driver in Service Mode is shown in Figure 5.
443

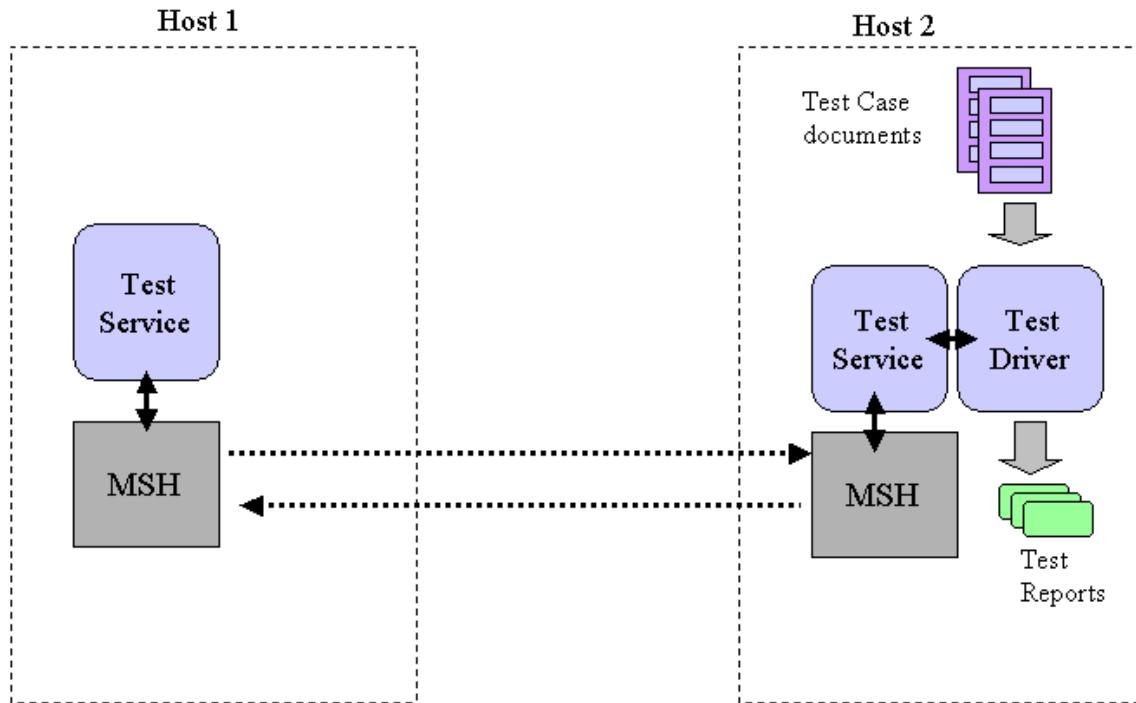


Fig 5. Example of Test Driver in Service Mode: Point-to-Point Interoperability Testing of Message Handlers

5

444

445

446 3.2 The Test Service

447

448 3.2.1 Functions and Interactions

449

450 The Test Service defines a set of Actions that are useful for executing Test Cases. The Test Service
451 represents the application layer for a message handler. It receives message content and error
452 notifications from the MSH, and also generates requests to the MSH, which normally are translated
453 into messages being sent out. The Test Actions are predefined, and are part of the Test Framework
454 (i.e. not user-written). Test Service and Actions will map to the Service and Action header attributes of
455 ebXML messages generated during the testing.

456

457 The Test Service name is: urn:ebXML:iic:test.

458

459 Figure 6 shows the details of the Test Service and its interfaces.

460

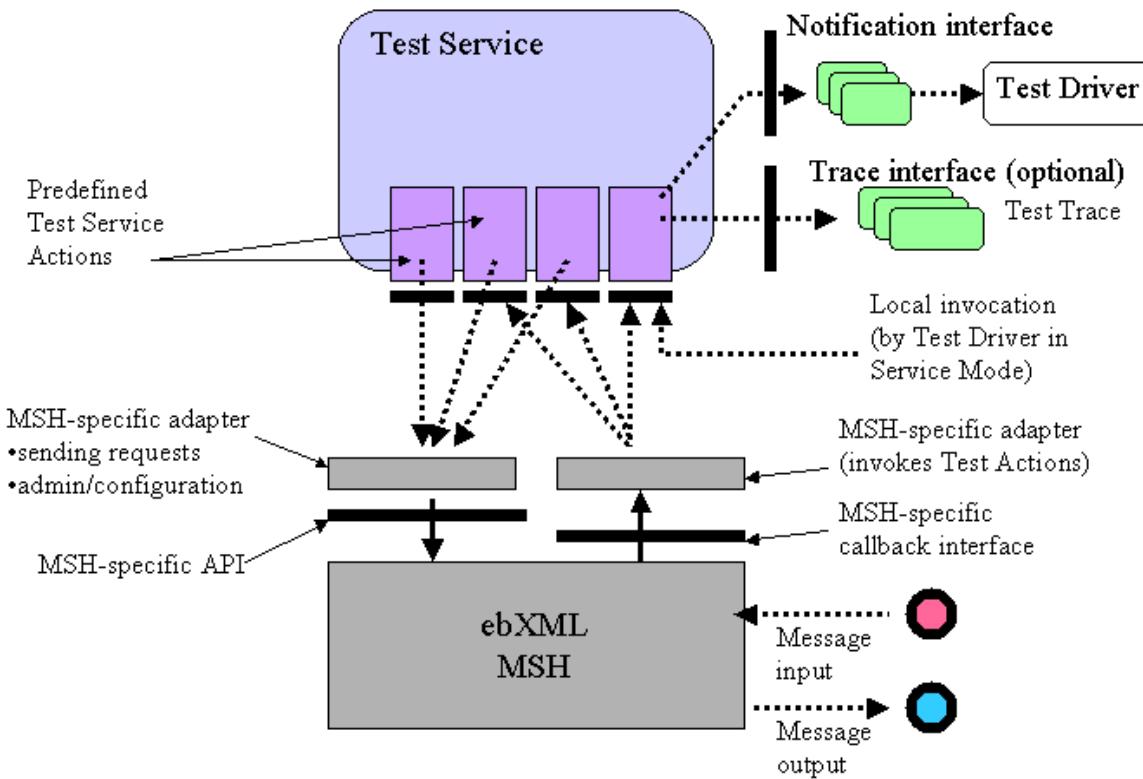


Fig 6. The Test Service and its Interfaces

6

464 The functions of the Test Service are:

- 466 • To implement the actions which map to Service / Action fields in a message header. The set of
467 test actions which are pre-defined in the Test Service will perform diverse functions, which are
468 enumerated below:
- 469 • To notify the Test Driver of incoming messages. This only occurs when the Test Service is
470 deployed in *reporting mode*, which assumes it is coupled with a Test Driver.
- 471 • To perform some message processing, e.g. compare a received message payload with a
472 reference payload (or their digests).
- 473 • To send back a response to the MSH. Depending on the action invoked, the response may
474 range from a pre-defined acknowledgment to a specific message previously specified.
- 475 • Optionally, to generate a trace of its operations, in order to help trouble-shooting, or for reporting
476 purpose.

479 Although the Test Service simulates an application, it is part of the Test Framework, and does not vary
480 from one test harness to the other. However, in order to connect to the Test Service, a developer will
481 have to write wrapper code to the Test Service/Actions that is specific to the MSH implementation that
482 needs to be integrated. This proprietary code is expected to require a minor effort, but is necessary as the

483 API and callback interfaces of each MSH are not specified in the [ebMS] standard and is implementation-
484 dependent.

485

486

487

488 **3.2.2 Modes of Operation of the Test Service**

489

490 The Test Service can operate in two modes:

491

- 492 • **Reporting mode:** in that mode, the actions of the Test Service instance, when invoked, will send
493 a notification to the Test Driver. The Test Driver can then be aware of the workflow of the test
494 case. There are actually two cases of **reporting** mode:

- 495 ○ **Local Reporting Mode:** The Test Driver is installed on the same host as the Test
496 Service, and executes in the same process space. The notification uses the *Receive*
497 interface of the Test Driver, which is operating in service mode.
498 ○ **Remote Reporting Mode:** The Test Driver is installed on a different host than the Test
499 Service. The notification is done via messages to the Test Driver, which is generally
500 operating in connection mode. Remote message notifications are identified by the Test
501 Driver through the “Notify” Action name and “urn:ebXML:iic:test” Service name, in
502 the header of a received ebXML message.

503

- 504 • **Loop mode:** in that mode, the actions of the Test Service instance, when invoked, will NOT send
505 a notification to the Test Driver. The only interaction of the Test Service with external parties, is
506 by sending back messages via the message handler

507

508 Except for the notification, which may or may not take place, the actions operate similarly in both
509 reporting and loop modes, unless specified otherwise. In other words, the mode of operation does not
510 normally affect the logic of the action. The action may send a response message, to the requesting
511 party via the “response URL”. In general, the response URL is the same as the requestor URL.

512

513 Figure 7 shows a test harness with a Test Driver in connection mode, controlling a Test Service (Host
514 1) in remote reporting mode. The other Test Service (Host 3) is operating in loop mode. This
515 configuration is used when the test cases are controlled from a third party test center, when doing
516 interoperability testing. The test center may also act as a Hub, and be involved in monitoring the
517 traffic between the interoperating parties.

518

519

520

521

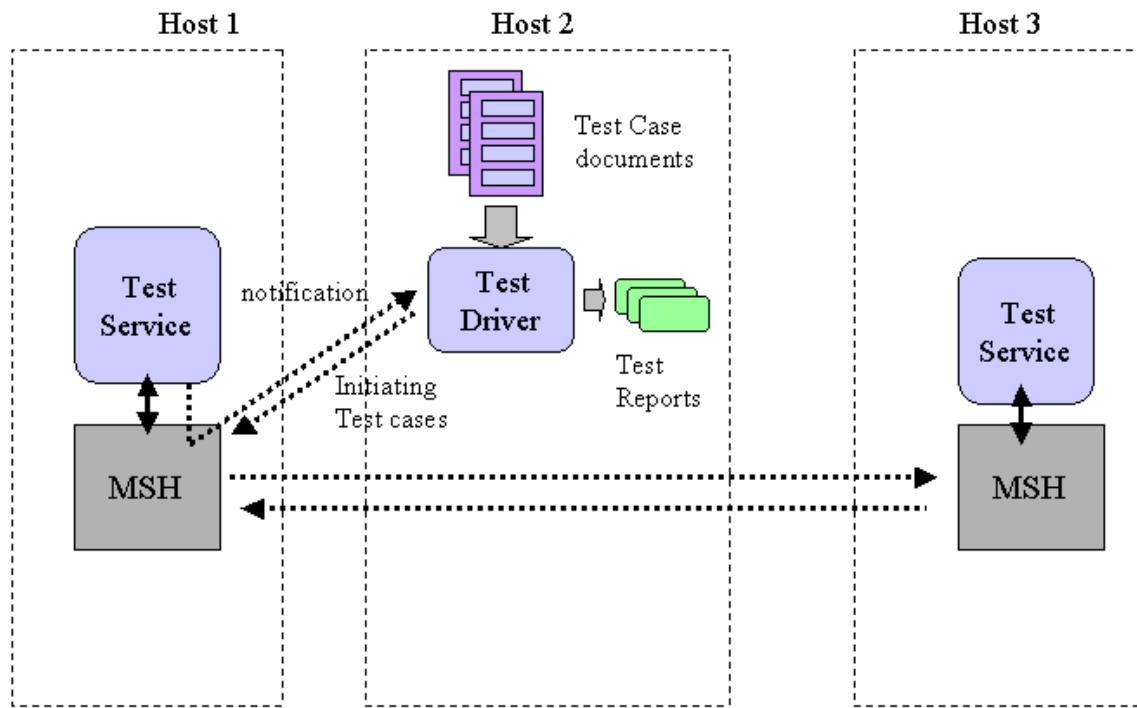


Fig 7. Example of Test Service in Remote Reporting Mode: The Interoperability Test Center model

7

522
523

524 **3.2.3 Parameters of the Test Service**

525

526 The Test Service has only two parameters that can be modified by action invocation (and therefore
527 represent its state):

528

- 529 • Operation mode (either reporting or loop)
- 530 • Response URL (destination for response messages)
- 531 • Notification URL (destination for notification messages, if applicable)

532

533 In addition, a Test Service instance is identified by an ID that will be reported in some response
534 messages. The three parameters above can be set by invoking the Configurator action described below.
535 In a test harness where an interoperability test suite involves two parties, the test suite will need to be
536 executed twice - alternatively driven from each party. In that case, each Test Service instance will
537 alternatively be set to a reporting mode, while the other will be set to loop mode. These settings can be
538 done remotely by sending messages to the Configurator action.

539

540 Except for these parameters, the Test Service is stateless.

541

542

543 3.2.4 The Actions of the Test Service

544

545 The actions described here are standard to the Test Service, and should suffice in supporting most Test
546 Cases. These actions map to the Service/Action field of an ebXML message, and will be triggered on
547 reception of such messages.

548

549 3.2.4.1 Common Functions

550

551 Some functions are common to several actions, in addition to the specific functions they fulfill. These
552 common functions are:

- 553 • **Generate a response message.** Response messages are destined to the Response URL
554 (see 3.2.3). They also specify a Service/Action, as they are usually intended for another Test
555 Service although in case the Response URL directly points to the Test Driver in connection
556 mode, Service/Action will not have the regular MSH semantics.
- 557 • **Notify the Test Driver.** This assumes the Test Service is coupled with a Test Driver. In that
558 configuration, the Test Service is in reporting mode. The reporting is done by a message
559 (sent to the Notification URL) when in remote reporting mode, or by a call to the Receive
560 interface when in local reporting mode.

561

562

563 3.2.4.2 Test Service Actions

564

565 The standard test actions are:

566

567 3.2.4.2.1 Mute action

568

569 Reporting/Loop Mode Action Description: This is a “dummy” action, which does not generate any
570 response message back. Such an action is used for messages that do not require any effect, except
571 possibly to cause some side-effect in the MSH, like generating an error.

572 Response Destination: None

573 In Reporting Mode: The action will notify the associated Test Driver. The notification containing the
574 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,
575 or with a message with Service / Action fields set to “urn:ebXML:iic:test”/ “Notify”, if in remote
576 reporting mode. The notification will report the action name (“mute”) and the instance ID of the Test
577 Service.

578

579 3.2.4.2.2 Dummy action

580

581 Reporting/Loop Mode Action Description: This is a “dummy” action, used by messages that do not need a
582 specific response. On invocation, this action will however generate a pre-canned response message back

583 (no payload, simplest header with no extra-features), with no dependency on the received message,
584 except for the previous MessageID (for correlation) in the RefToMessageId header attribute.
585 Response Destination: a message with a **Mute** action element is sent to the Test Component (Test Driver
586 or Service) associated with the Response URL. This notice serves as proof that the message has been
587 received, although no assumption can be made on the integrity of its content.
588 In Reporting Mode: The action will also notify the associated Test Driver. The notification containing the
589 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,
590 or with a message with Service / Action fields set to "urn:ebXML:iic:test"/ "**Notify**", if in remote
591 reporting mode. The notification will report the action name ("Dummy") and the instance ID of the Test
592 Service.

593

594 **3.2.4.2.3 Reflector**

595 Reporting/Loop Mode Action Description: On invocation, this action generates a
596 response to a received message, by using the same message material, with minimal
597 changes in the header:

- 598
 - 599 • Swapping of the to/from parties so that the "to" is now the initial sender.
 - 600 • Setting RefToMessageId to the ID of the received message.
 - 601 • Removing AckRequested or syncReply elements if any.
 - 602 • All other header elements (except for time stamps) are unchanged. The conversation ID remains
603 unchanged, as well as the CPAId. The payload is the same as in the received message, i.e. same
attachment(s).

604 Response Destination: a message with a **Mute** action element is sent to the Test Component (Test Driver
605 or Service) associated with the Response URL. This action acts as a *reflector* for the initial sending party

606 In Reporting Mode: The action also notifies the associated Test Driver. The notification containing the
607 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,
608 or with a message with Service / Action fields set to "urn:ebXML:iic:test"/ "**Notify**", if in remote
609 reporting mode. The notification will report the action name ("reflector") and the instance ID of the Test
610 Service.

611

612 **3.2.4.2.4 Initiator action**

613

614 Reporting/Loop Mode Action Description: On invocation, this action generates a new message, totally
615 unrelated to the header data of the enclosing ebXML message. The new message material (payload and
616 header) is provided in a pre-convened way in the payload of the received message. The header of the
617 new message can be anything that is specified. For example, this action would be used to generate a
618 "first" message of a new conversation, different from the conversation ID specified in the invoking
619 message. Note: unlike in the Reflector action, MSH-controlled header attributes will not be determined by
620 the invoking message header (messageID, RefToMessageId, timestamps...). So if the response needs to
621 refer to the previous MessageID (for correlation), the RefToMessageId must be explicitly pre-set in the
622 message material.

623 Response Destination: any service/action of the sender, specified with message material (by default: a
624 message with a **Mute** action element is sent to the Test Component (Test Driver or Service) associated
625 with the Response URL.)

626 In Reporting mode: In addition to generating the message, the action also notifies the associated Test
627 Driver. The notification containing the received header and payload(s) material, will be done via the

628 Receive interface, if in local reporting mode, or with a message with Service / Action fields set to
629 “urn:ebXML:iic:test” / “Notify”, if in remote reporting mode. The notification will report the action
630 name (“initiator”) and the instance ID of the Test Service.

631

632 3.2.4.2.5 PayloadVerify action

633

634 Reporting/Loop Mode Action Description: On invocation, this action will compare the payload(s) of the
635 received message, with the expected payload. Instead of using real payloads, to be pre-installed on the
636 site of the Test Service, it is RECOMMENDED that a digest (or signature) of the reference payloads (files)
637 be pre-installed on the Test Service host. The PayloadVerify action will then calculate the digest of each
638 received payload and compare with the reference digests. This action will test the service contract
639 between application and MSH, as errors may originate either on the wire, or at every level of message
640 processing in the MSH until message data is passed to the application. The action responds with a
641 response message to the party associated with the Response URL, reporting the outcome of the
642 comparison. The previous MessageID is reported (for correlation) in the RefToMessageId header
643 attribute of the response. The previous ConversationId is also reported. The payload message will
644 contain a verification status notification for each verified payload

645 The XML format used by the response message is described in the section 7.1.12 (“Service Messages”).

646

647 Response Destination: a message is sent with a **Mute** action element to the Test Component (Test Driver
648 or Service) associated with the Response URL.

649 In Reporting mode: Action will also notify the associated Test Driver. The notification containing the
650 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,
651 or with a message with Service / Action fields set to “urn:ebXML:iic:test” / “Notify”, if in remote
652 reporting mode.

653 3.2.4.2.6 ErrorAppNotify action

654

655 Reporting/Loop Mode Action Description: This action will capture specific error notifications from the MSH
656 to its using application. It is not triggered by reception of an error message, but it is directly triggered by
657 the internal error module of the MSH local to this Test Service. If the MSH implementation does not
658 support such direct notification of the application (e.g. instead, it writes such notifications to a log), then
659 an adapter needs to be written to read this log and invoke this action whenever such an error is notified.

660 Such errors fall into two categories:

- 661 • MSH errors that need to be directly communicated to its application – and not to any remote party, e.g.
662 failure to send a message (no Acknowledgments received after maximum retries).
- 663 • In case an MSH generates regular errors with a severity level set to “Error” – as opposed to “Warning” – the
664 MSH is supposed to (SHOULD) also notify its application. The ErrorAppNotify action is intended to support
665 both types of notifications.

666

667 Response Destination: A response message containing the verification status notification (see “Service
668 Messages” in 7.1.12) is sent only when in loop mode, with a **Mute** action element to the Test Component
669 (Test Driver or Service) associated with the Response URL.

670 In Reporting mode: Action will notify an error to the associated Test Driver. The notification containing the
671 received ErrorList document only (not the received message content), will be done via the Receive
672 interface, if in local reporting mode, or with a message with Service / Action fields set to
673 “urn:ebXML:iic:test” / “Notify”, if in remote reporting mode.

674

675 **3.2.4.2.7 ErrorURLNotify action**

676

677 Reporting/Loop Mode Action Description: This action will capture error messages, assuming that an adapter has
678 been written for invoking this action. The adapter must have same URI as the ErrorURI specified in the CPA.
679 The adapter will pass the entire message as is (in its ebXML envelope) to the action. The action extracts the
680 ErrorCode and Severity elements, and sends then an error notification message back to the originator, when
681 operating in loop mode only. The action will make such error notifications visible to the other party (generally
682 the driver party), by generating a “report” message to the Response URL.

683

684 The XML format used to both the received and response message payload for this action is described in section
685 8.1.2, the Test Message Schema.

686

687 Response Destination: When in loop mode, generates a verification status notification (see “Service
688 Messages” in 7.1.12) with a **Mute** action element to the Test Component (Test Driver or Service)
689 associated with the Response URL.

690 In Reporting mode: Action only notifies the associated Test Driver. The notification containing the
691 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,
692 or with a message with Service / Action fields set to “urn:ebXML:iic:test” / “**Notify**”, if in remote
693 reporting mode. The notification will report the action name (“ErrorURLNotify”) and the instance ID of the
694 Test Service.

695

696 **3.2.4.2.8 Configurator action**

697

698 Reporting/Loop Mode Action Description: This action is called to either dynamically (re)configure the
699 receiver party, or to verify that the receiver party has the right configuration set-up. Configuration may
700 concern:

- 701 • MSH internals assumed by a Test Case (if applicable),
- 702 • CPA set-up assumed by a Test Case,
- 703 • Test Service parameters (e.g. ID, response-URL, mode of operation). In the case of CPA, the action can
704 verify that the collaboration agreement for a conversation related to a Test Case or a set of Test Cases is
705 available. If the payload contains a CPAId, this action will verify that the corresponding CPA is accessible.
706 The previous MessageID is reported (for correlation) in the RefToMessageID header attribute of the
707 response.
- 708 • Predefined digests of payloads to be used in Test Cases. These digests or signatures will be then used as
709 references for comparing digests from received payloads. Such comparison will be done by the
710 PayloadVerify action.

711

712 The XML format used by the configuration message and its response message is described in the section 7.1.12
713 (“Service Messages”).

714

715

716 Response Destination of response: a message with a **Mute** action element is sent to the Test
717 Component (Test Driver or Service) associated with the ResponseURL.

718 In Reporting mode: Action notifies the associated Test Driver. The notification containing the received
719 header and payload(s) material, will be done via the Receive interface, if in local reporting mode, or with a

720 message with Service / Action fields set to “urn:ebXML:iic:test” / “Notify”, if in remote reporting
721 mode. The notification will report the action name (“configurator”) and the instance ID of the Test Service.

722

723 Note: The above actions are specific to the Test Service, and are not supported by the Test Driver.

724

725 **3.2.4.3 Integration with an Implementation**

726

727 As mentioned before, the actions above are predefined and part of the Test Framework, and will require
728 some integration code with the MSH implementation, in form of three adapters, to be provided by the
729 MSH development (or user) team. These adapters are:

730

731 (1) **Reception adapter**, which is specific to the MSH callback interface. This code allows for
732 invocation of the actions of the Test Service, on reception of a message.

733

734 (2) **MSH control adapter**, which will be invoked by some Test Service actions, and will invoke in turn
735 the MSH-specific Message Service Interface (or API). Examples of such invocations are for
736 sending messages (e.g. by actions which send response messages), and MSH configuration
737 changes (done by the Configurator action).

738

739 (3) **Error URL adapter**, which is actually independent from the candidate MSH. This adapter will
740 catch error messages, and invoke the **ErrorURLNotify** action of the Test Service. If the Test
741 Service is in reporting mode, the Test Driver is notified of this error message.

742

743

744 **3.3 Executing Test Cases**

745

746

747 A Test Suite contains a sequence of Test Cases. Each Test Case is intended to verify that an
748 implementation fulfills a requirement item (or a set of items) of the specification.

749

750 **3.3.1 Test Case as a Sequence of Test Steps**

751 :

752 A Test Case is a sequence of Test Steps. A Test Step is an aggregate of one or more operations
753 performed by a single component of the test harness. A Test Step usually involves a single message
754 sending or receiving operation, plus some message data processing operations, like checking a condition

755 on message header. A Test Step may also include conditional actions that are a basis for the execution
756 of the assertion within the Test Step itself.

757 A Test Case instance is an execution of a particular Test Case, identified by some specific message
758 attribute values. For example, two instances of the same Test Case will be distinguished by distinct
759 MessageID values in the generated messages. An example of a sequence of Test Steps associated with
760 an MS Conformance Test Case is:

761
762 Step 1: Test driver sends a sample message to the Reflector action of the Test Service. Message header
763 data is obtained from message header declaration, and message payload from file ABC.

764 Step 2: Test driver receives the response message and adds it to the stored sequence for this Test Case
765 instance (correlation with Step 3 is done based on the RefToMessageID attribute, which should be
766 identical to the MessageID of Step 3.)

767 Step 3: Test driver verifies the test condition on response message, for example that the SOAP envelope
768 and extensions are well-formed.

769

770 **3.3.2 Related Message Data and Message Declarations**

771

772

773 Some Test Steps will require message data. This message data MUST be specified using a Message
774 Declaration (Section 7), which is an XML-based script. Header content is scripted in the message
775 declaration by using XML and XPath expressions. The message envelope is also described in the same
776 way. Payload material is not included in the messages declaration, but referenced by it. The script that
777 describes a test step may also include operations that allow for extracting a payload from or for adding a
778 payload to a message. The Test Driver MUST be capable of interpreting these scripts in order to:

- 779 • Assemble a message from script material and referenced payloads.
780 • Analyze and select a received message based on header and envelope content (as well as
781 based on payload content if the payload is in XML).

782

783 **3.3.3 Related Configuration Data**

784

785 Test Cases will be executed under a pre-defined agreement, as defined in CPA [ebXML CPA]. This
786 agreement will configure the ebXML Candidate Implementations involved in the testing, or the
787 collaborations that execute on these implementations. Each Test Case will therefore reference a Test
788 Configuration document.

789

- 790 • **Test Configuration document:** it contains (1) a CPA (or CPA-like) document, (2) configuration
791 data for the ebXML implementation(s) involved, expressed at an abstract level and expected to
792 be general enough to most implementations, even if not specified.

793

794 Figure 8 illustrates how a Test Case references message data.

795

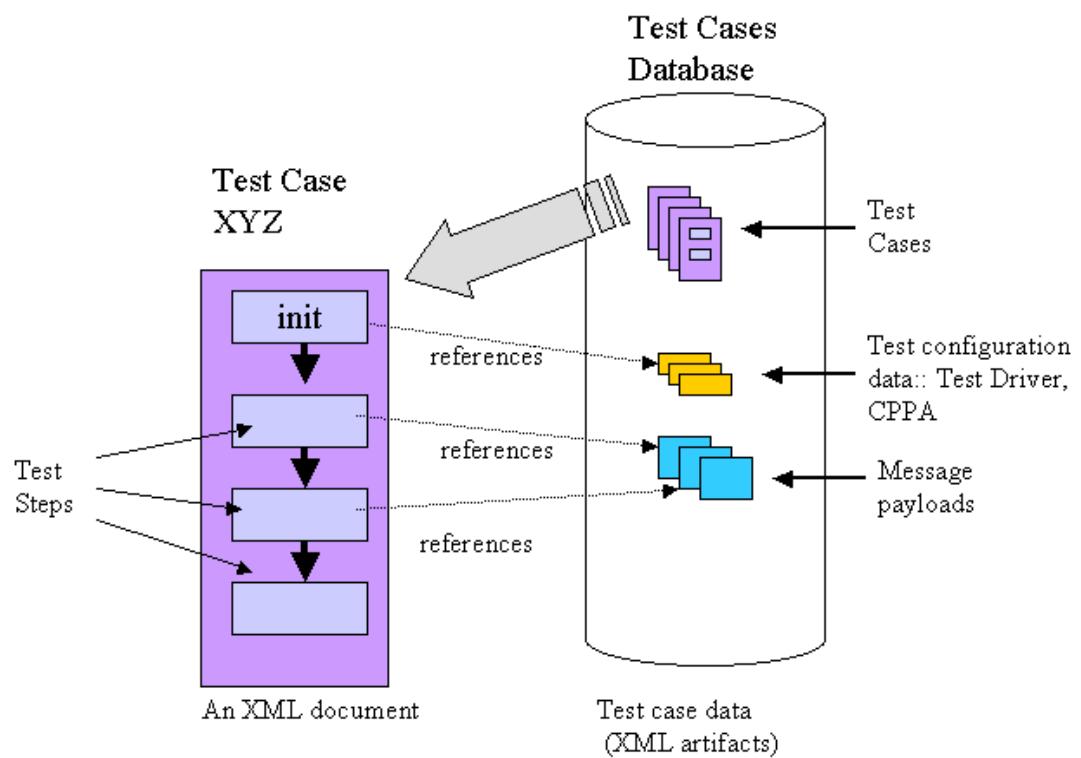


Fig 8. Test Case Document and Database

9

796
797
798
799
800

801

Part II: Test Suite Representation

802

803 4 Test Suite

804

805 4.1 Conformance vs. Interoperability Test Suite

806

807 We distinguish two types of test suites, which share similar document schemas and architecture
808 components, but serve different purposes:

809

- 810 ▪ **Conformance Test Suite.** The objective is to verify the adherence or non-adherence of a Candidate
811 Implementation to the target specification. The test harness and Test Cases will be designed around
812 a single (candidate) implementation. The suite material emphasizes the target specification, by
813 including a comprehensive set of Test Requirements, as well as a clear mapping of these to the
814 original specification (e.g. in form of an annotated version of this specification).
- 815
- 816 ▪ **Interoperability Test Suite.** The objective is to verify that two implementations (or more) of the same
817 specification, or that an implementation and its operational environment, can interoperate according
818 to an agreement or contract (which is compliant with the specification, but usually restricts further the
819 requirements). These implementations are assumed to be conforming (i.e. have passed conformance
820 tests or have achieved the level of function of such tests), so the reference to the specification is not
821 as important as in conformance. Such a test suite involves two or more Candidate Implementations of
822 the target specification. The test harness and Test Cases will be designed in order to drive and
823 monitor these implementations.

824

825 A conformance test suite is composed of:

826

- 827 • One or more **Test Profile** documents (XML). Such documents represent the level or profile of
828 conformance to the specification, as verified by this Test Suite.
- 829 • Design of a **Test Harness** for the Candidate Implementation that is based on components of the
830 ebXML IIC Test Framework.
- 831 • A **Test Requirements** document. This document contains a list of conformance test assertions
832 that are associated with the test profile to be tested.
- 833 • An **annotation** of the target specification, that indicates the degree of Specification Coverage for
834 each specification feature or section, that this set of Test Requirements provides.
- 835 • A **Test Suite** document. This document implements the Test Requirements, described using the
836 Test Framework material (XML mark-up, etc.)

837

838 An Interoperability Test Suite is composed of:

839

- 840 • One or more **Test Profile** documents (XML). Such documents represent a set of features specific
841 to a particular functionality, represented in a Test Suite through Test Cases that only test those
842 particular features, and hence, that profile.
- 843 • Design of a **Test Harness** for two or more interoperating implementations of the specification that
844 is based on components of the ebXML Test Framework.
- 845 • A **Test Requirements** document. This document contains a list of test assertions associated with
846 this profile (or level) of interoperability.
- 847 • A **Test Suite** document. This document implements the Test Requirements, described using the
848 Test Framework material (XML mark-up, etc.)

849

850

851 **4.2 The Test Suite Document**

852

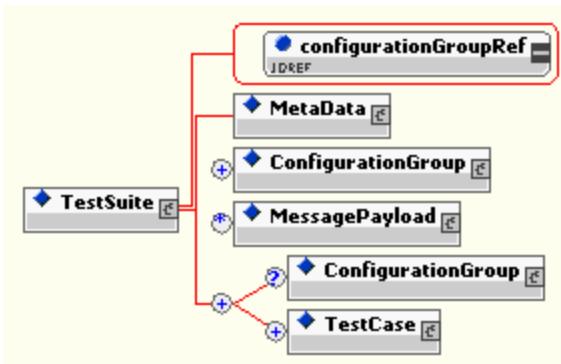
853 The Test Suite XML document is a collection of Test Driver configuration data, documentation and
854 executable Test Cases.

- 855 ▪ **Metadata** provides documentation used by the Test Driver to generate a Test Report for all executed
856 Test Cases.
- 857 ▪ **Configuration data** provide basic Test Driver parameters used to modify the configuration of the
858 Test Driver to accurately perform and evaluate test results. It also contains configuration data for the
859 candidate ebXML implementation(s).
- 860 ▪ **Message data** is a collection of pre-defined XML payload messages that can be referenced for
861 inclusion in an ebXML test message.
- 862 ▪ **Test Cases** are a collection of discrete Test Steps. Each Test Step can execute any number of test
863 Operations (including sending, receiving, and examining returned messages). An ebXML Test Suite
864 document MUST validate against the ebXMLTestSuite.xsd file in Appendix C.
- 865 ▪ **Payloads** provide XML and non-XML content for use as material for test messages, as well as
866 message data for Test Services linked to the Test Driver.

867

868

869



870
871

872 Figure 9 – Graphic representation of basic view of ebXMLTestSuite.xsd schema

873

874

875

876

877

878

879

880

881

882

883

884 **Definition of Content**

885

Name	Description	Default Value From Test Driver	Required/Optional
TestSuite	Container for all configuration, documentation and tests		Required
Metadata	Container for general documentation of the entire Test Suite		Required
ConfigurationGroup	Container for configuration of the Test		Required

	Driver and /or MSH		
MessagePayload	XML Payload message for inclusion in a Test Case		Optional
ConfigurationGroup	Container for modifications to base ConfigurationGroup		Optional
TestCase	Container for an individual Test Case		Required

886

887

888

889

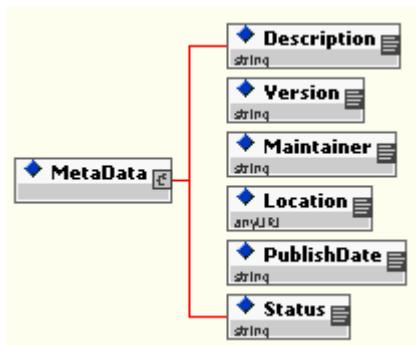
890 **Test Suite Metadata**

891

892 Documentation for the ebXML MS Test Suite is done through the Metadata element. It is a container
 893 element for general documentation.

894

895



896

897 Figure 10 – Graphic representation of expanded view of the Metadata element

898

899

900

901

902 **Definition of Content**

903

Name	Description	Default Value From Test Driver	Required/Optional
Description	General description of the Test Suite		Required
Version	Version identifier for Test Suite		Required
Maintainer	Name of person(s) maintaining the Test Suite		Required
Location	URL or filename of this test suite		Required
PublishDate	Date of publication		Required
Status	Status of this test suite		Required

904

905

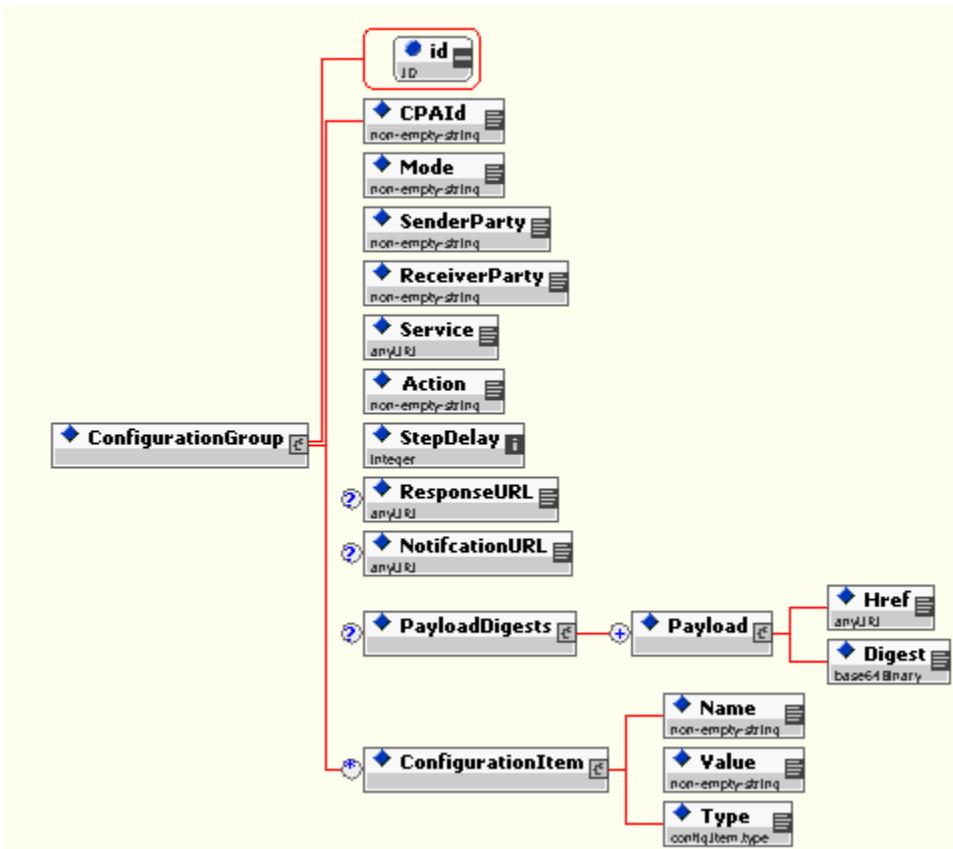
906 **4.2.1 The ConfigurationGroup**

907

908 The ConfigurationGroup contains configuration data for modifying the content of test messages sent by
909 the Test Driver (when in Connection Mode) or the MSH (when the Test Driver is in Service Mode). In
910 addition, three of the parameters have functions beyond defining message content. The "Mode"
911 parameter toggles the mode of the Test Driver between Connection Mode and Service Mode. CPAId is
912 used by the Test Driver (in Service Mode) to configure its interfaced MSH through its CPAId reference.
913 PayloadDigests provides a list of payload identifiers and their MD5 digest values for payload content
914 verification by the Test Driver or Test Service.

915

916



917

918

919

920 Figure 11 – Graphic representation of expanded view of the BaseConfigurationGroup element

921

922

923 **Definition of Content**

924

Name	Description	Default Value From Test Driver	Required/Optional
ConfigurationGroup	Container Test Driver/MSH configuration data		Required

CPAId	Unique identifier matching one of the testing CPA's in the Conformance or Interoperability Test Suite. Inserted inside outgoing messages, as content for the CPAId element in an ebXML MessageHeader. Value is also used to configure MSH when in "service" mode.		Optional
Mode	One of two types, "driver" (interfaced to MSH) or "non-driver" (standalone)	Non-driver	Required
SenderParty	Default identifier used in message header From/PartyId		Required
ReceiverParty	Default identifier used in message header To/PartyId		Required
Service	Default Service to be inserted into outgoing message Service element content		Required
Action	Default Service Action to be inserted into outgoing messages Action element content		Required
StepDelay	Milliseconds of delay between execution of the last Test Step and the current Test Step		Required
ResponseURL	Parameter defining the URL for the Test Service to send response messages to		Optional
NotificationURL	Parameter defining the location for the Test Service to send notification messages to		Optional
PayloadDigests	Container for one or more message payload identifiers with corresponding computed digest values, used by Test Driver to verify received message payload content		Optional
Payload	Container for id and digest value pair		Required
href	Identifier (CID) for message payload to be verified against Digest value		Required
Digest	Pre-computed MD5 digest value to be used by Test Driver to verify integrity of received message payload		Required
ConfigurationItem	Container for individual name/value pair used by the Test Driver for configuration or possibly for message payload content construction		Optional
Name	Name for the ConfigurationItem		Required
Value	Value of the ConfigurationItem		Required
Type	Type of ConfigurationItem (namespace or parameter)		Required

925

926

927

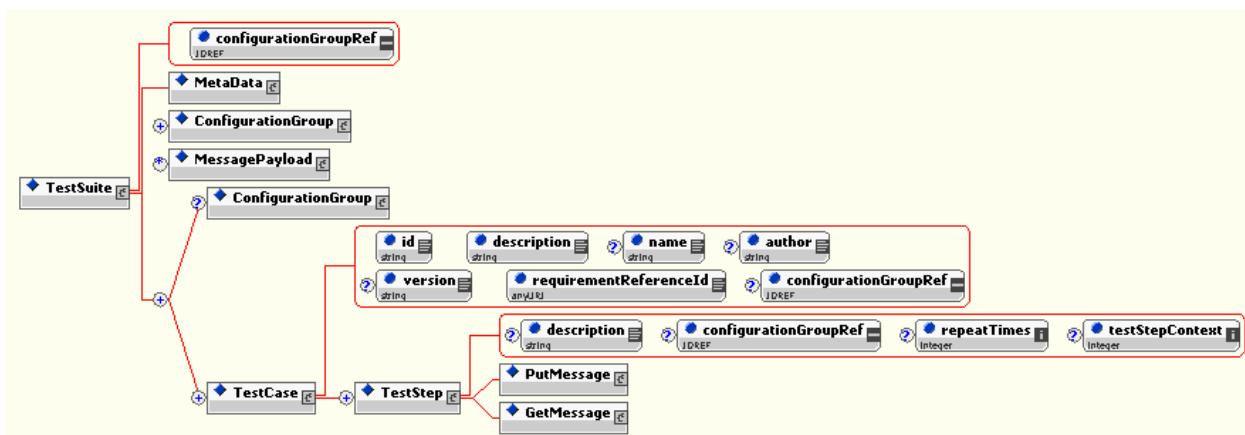
928 **4.2.1.1 Multiple Configuration Declarations within the Test Suite**

929

930 It is possible to dynamically modify the configuration of the Test Driver or MSH through the course of
931 execution of the Test Suite. After initial configuration is set through the required configurationGroupRef
932 attribute of the TestSuite element, configuration can be modified at the Test Case and Test Step levels
933 through the introduction of a new configurationGroupRef attribute value.

934 The scope of the configuration change is hierarchical, and exists only for the definition of the current Test
935 Suite, Case or Step. Upon completion, of that test component, configuration reverts to that previously
936 defined at the higher level. This hierarchy is illustrated in the figure below.

937



938

939

940 Figure 12 – Graphic representation of hierarchical use of the ConfigurationGroup via reference

941

942

943 **4.2.1.2 Precedence Rules for Test Driver/MSH configuration**

944

945 In order to generate messages correctly, the Test Driver MUST follow the precedence rules for
946 interpreting a Configuration Group declaration. The precedence rules are:

947

948 Certain portions of an ebXML message are auto-generated by the Test Driver at run-time, unless they
949 are overridden explicitly in the Message Declaration. These include:

950

- 951 • ConversationId
- 952 • MessageId
- 953 • Timestamp

954

955 In addition, other message content, such as the “soap:mustUnderstand” attribute and value, the ebXML
956 “version” attribute of message header extension elements and other content is also auto-generated by the
957 Test Driver (in Connection Mode) or interfaced MSH (in Service Mode) for every Message Declaration.
958 These values can also be overridden through explicit declaration.

959

960

961 If there is no explicit declaration of an element or attribute value in the Message Declaration content, and
962 it is not auto-generated by the Test Driver, and it is a required element/attribute value, then the
963 Configuration Group value MUST be provided in the Message Declaration.

964

965

966 5 Test Requirements

968 5.1 Purpose and Structure

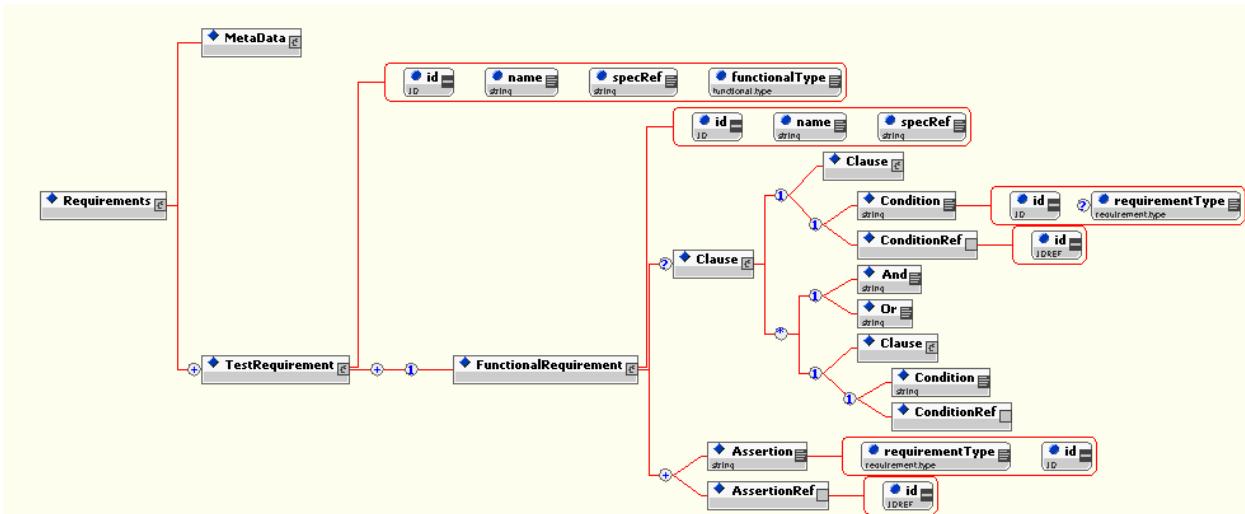
970 The next step in designing a test suite is to define Test Requirements. This material, when used in a
971 conformance testing context, is also called Test Assertions in NIST and OASIS terminology (see definition
972 in glossary in Appendix).

973 When used for conformance testing, each Test Requirement defines a test item to be performed, that
974 covers a particular requirement of the target specification. It rewords the specification element in a
975 “testable form”, closer to the final corresponding Test Case, but unlike the latter, independently from the
976 test harness specifics. In the ebXML Test Framework, a Test Requirement will be made of three parts:

- 978 • **Pre-condition** The pre-condition defines the context or situation under which this test item
979 applies. It should help a reader understand in which case the corresponding specification
980 requirement applies. In order to verify this Test Requirement, the test harness will attempt to
981 create such a situation, or at the very least to identify when it occurs. If for some reason the pre-
982 condition is not satisfied when doing testing, then it does not mean that the outcome of this test is
983 negative – only that the situation in which it applies did not occur. In that case, the corresponding
984 specification requirement could simply not be validated, and the subsequent Assertion will not be
985 tested.
- 986 • **Assertion** The assertion actually defines the specification requirement, as usually qualified by a
987 MUST or SHALL keyword. In the test harness, the verification of an assertion will be attempted
988 only if the pre-condition is itself satisfied. When doing testing, if the assertion cannot be verified
989 while the pre-condition was, then the outcome of this test item is negative.
- 990 • **Requirement Level** Qualifies the degree of requirement in the specification, as indicated by
991 such keywords as RECOMMENDED, SHOULD, MUST, and MAY. Three levels can be
992 distinguished: (1) “required” (MUST, SHALL), (2) “recommended” ([HIGHLY] RECOMMENDED,
993 SHOULD), (3) “optional” (MAY, OPTIONAL). Any level lower than “required” qualifies a Test
994 Requirement that is not mandatory for Conformance testing. Yet, lower requirement degrees may
995 be critical to interoperability tests. The test requirement level can be override by explicit
996 declaration in the Test Profile document, in case a lower or higher level is required.

1000 5.2 The Test Requirements Document

1001
1002 The Test Requirements XML document provides metadata describing the Testing Requirements, their
1003 location in the specification, and their requirement type (REQUIRED, HIGHLY RECOMMENDED,
1004 RECOMMENDED, or OPTIONAL). A Test Profile driver file MUST validate against the
1005 ebXMLTestRequirements.xsd file found in Appendix B. The ebXML MS Conformance Test Requirements
1006 instance file can be found in Appendix E.



1008

1009 Figure 13 – Graphic representation of ebXMLTestRequirements.xsd schema

1010

1011

1012 **Definition of Content**

1013

Name	Description	Default Value From Test Driver	Required/Optional
Requirements	Container for all test requirements		Required
MetaData	Container for requirements metadata, including Description, Version, Maintainer, Location, Publish Date and Status		Required
Test Requirement	Container for all components of a single test requirement		Required
description	Description of requirement		Required
id	Unique identifier for each Test Requirement		Required
name	Name of test requirement		Required
specRef	Pointer to location in specification where requirement is found		Required
functionalType	Generic classification of function to be tested		Required
FunctionalRequirement	Sub-requirement for the main Test Requirement		Required
id	Unique ID for the sub-requirement		Required
name	Short descriptor of Functional Requirement		Required

specRef	Pointer to location in specification where sub-requirement is found		Required
Clause	Grouping element for Condition expression(s)		Optional
Condition	Textual description of test precondition		Required
ConditionRef	Reference (via id attribute) to existing Condition element already defined in the Test Requirements document		Required
And/Or	Union/Intersection operators for Conditions		Optional
Assertion	Axiom expressing expected behavior of an MSH implementation under conditions specified by any Clause		Required
AssertionRef	Reference (via id attribute) to existing Assertion element already defined in the Test Requirements document		Required
requirementType	Enumerated Assertion descriptor (REQUIRED, OPTIONAL...etc.)		Required

1014

1015 **5.3 Specification Coverage**

1016

1017 A Test Requirement is a formalized way to express a requirement of the target specification. The
 1018 reference to the specification is included in each Test Requirement, and is made of one or more
 1019 section numbers. There is no one-to-one mapping between sections of a specification document and
 1020 the Test Requirement items listed in the test material for this specification:

1021

- 1022 • A specification section may map to several Test Requirements.
- 1023 • A Test Requirement item may also cover (partially or not) more than one section or sub-
 1024 section.

1025

1026 A Test Requirement item may then cover a subset of the requirements that are specified in a section.

1027 For these reasons, it is important to determine to which degree the requirements of each section of a
 1028 specification, are fully satisfied by the set of Test Requirements listed in the test suite document. This
 1029 is done by establishing the Specification Coverage by the Test Requirements.

1030

1031 The Specification Coverage document is a separate document containing a list of all sections and
 1032 subsections of a specification document, each annotated with:

1033

- 1034 • A coverage qualifier.
- 1035 • A list of Test Requirements that map to this section.

1036

1037 The coverage qualifier may have values:

- 1038
- 1039 • **Full:** The requirements included in the specification document section are fully covered by
1040 the associated set of Test Requirements. This means that if each one of these Test
1041 Requirements is satisfied by an implementation, then the requirements of the corresponding
1042 document section are fulfilled. When the tests requirements are about conformance: The
1043 associated set of test requirement(s) are a clear indicator of conformance to the specification
1044 item, i.e. if a Candidate Implementation passes a Test Case that implements this test
1045 requirement(s) in a verifiable manner, there is a strong indication that it will behave similarly
1046 in all situations identified by the spec item.
- 1047
- 1048 • **None:** This section of the specification is not covered at all. Either there is no associated set
1049 of Test Requirements, or it is known that the test requirements cannot be tested even
1050 partially, at least with the Test Framework on which the test suite is to be implemented, and
1051 under the test conditions that are defined.
- 1052
- 1053 • **Partial:** The requirements included in this document section are only partially covered by the
1054 associated (set of) Test Requirement(s). This means that if each one of these Test
1055 Requirements is satisfied by an implementation, then it cannot be asserted that all the
1056 requirements of the corresponding document section are fulfilled: only a subset of all
1057 situations identified by the specification item are addressed. Reasons may be:
1058
- 1059 ○ (1) The pre-condition(s) of the test requirement(s) ignores on purpose a subset of
1060 situations that cannot be reasonably tested under the Test Framework.
1061 ○ (2) The occurrence of situations that match the pre-condition of a Test Requirement
1062 is known to be under control of the implementation (e.g. implementation-dependent)
1063 or of external factors, and out of the control of the testbed. (See *contingent run-time*
1064 coverage definition, Section 7).

1065 When the tests requirements are about conformance: The associated set of test
1066 requirement(s) are a weak indicator of conformance to the specification item. A negative test
1067 result will indicate non-conformance of the implementation.

1069 **5.4 Test Requirements Coverage (or Test Run-Time Coverage)**

1070

1071 In a same way as Test Requirements may not be fully equivalent to the specification items they represent
1072 (see Specification Coverage, Section 5.3), the Test Cases that implement these Test Requirements may
1073 not fully verify them, for practical reasons.

1074

1075 Some Test Requirements may be difficult or impossible to verify in a satisfactory manner. The reason for
1076 this generally resides in an inability to satisfy the pre-condition. When processing a Test Case, the Test
1077 Harness will attempt to generate an operational context or situation that intends to satisfy the pre-
1078 condition, and that is supposed to be representative enough of real operational situations. The set of such
1079 real-world situations that is generally covered by the pre-condition of the Test Requirement is called the
1080 *test requirements (or test run-time) coverage* of this test Requirement. This happens in the following
1081 cases:

- 1082
- 1083 • **Partial run-time coverage:** It is in general impossible to generate all the situations that should
1084 verify a test. It is however expected that the small subset of run-time situations generated by the
1085 Test Harness, is representative enough of all real-world situations that are relevant to the pre-

1086 condition. However, it is in some cases obvious that the Test Case definition (and its processing)
1087 will not generate a representative-enough (set of) situation(s). It could be that a significant subset
1088 of situations identified by the pre-condition of a Test Requirement cannot be practically set-up
1089 and verified. For example, this is the case when some combinations of events or of configurations
1090 of the implementation will not be tested due to the impracticality to address the combinatorial
1091 nature of their aggregation. Or, some time-related situations cannot be tested under expected
1092 time constraints.

- 1093
- 1094 • **Contingent run-time coverage:** It may happen that the test harness has no complete control in
1095 producing the situation that satisfies the pre-condition of a Test Requirement. This is the case for
1096 Test Requirements that only concern optional features that an implementation may or may not
1097 decide to exhibit, depending on factors under its own control and that are not understood or not
1098 easy to control by the test developers. An example is: “ IF the implementation chooses to bundle
1099 together messages [e.g. under some stressed operation conditions left to the appreciation of this
1100 implementation] THEN the bundling must satisfy condition XYZ”.

1101

1102 When a set of Test Cases is written for a particular set of Test Requirements, the degree of coverage of
1103 these Test Requirements by these Test Cases **SHOULD** be assessed. The Test Requirements coverage
1104 – not to be confused with the Specification Coverage - is represented by a list of the Test Requirements
1105 Ids, which associates with each Test Requirement:

- 1106
- 1107 ○ The Test Case (or set of Test Cases) that cover it,
 - 1108 ○ The coverage qualifier, which indicates the degree to which the Test Requirement is covered.

1109

1110 The coverage qualifier may have values:

- 1111
- 1112 • **Full:** the Test Requirement item is fully verified by the set of Test Cases.
 - 1113 • **Contingent:** The run-time coverage is contingent (see definition).
 - 1114 • **Partial:** the Test Requirement item is only partially verified by the associated set of Test
1115 Cases. The run-time coverage is partial (see definition).
 - 1116 • **None:** the Test Requirement item is not verified at all: there is no relevant Test Case.

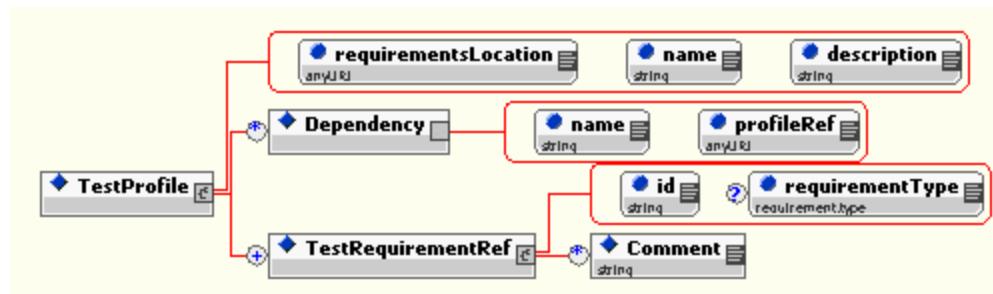
6 Test Profiles

6.1 The Test Profile Document

The Test Profile document points to a subset of Test Requirements (in the Test Requirements document), that are relevant to the profile - either conformance, or interoperability profile - to be tested.

The document will drive the Test Harness by providing the Test Driver with a list of unique reference IDs of Test Requirements for a particular Test Profile. The Test Driver reads this document, and executes all Test Cases (located in the Test Suite document) that contain a reference to each of the test requirements. A Test Profile driver file MUST validate against the ebXMLTestProfile.xsd file found in Appendix A. A Test Profile example file can be found in section 10.2.

1129



1131 Figure 14 – Graphic representation of ebXMLTestProfile.xsd schema

Definition of Content

Name	Description	Default Value From Test Driver	Required/Optional
TestProfile	Container for all references to test requirements		Required
requirementsLocation	URI of test requirements XML file		Required
name	Name of profile		Required
description	Short description of profile		Required
Dependency	Prerequisite profile reference container		Optional
name	Name of the required prerequisite profile		Required
profileRef	Identifier of prerequisite profile to be loaded by Test Driver		Required

TestRequirementRef	Test Requirement reference		Required
id	Unique Identifier of Test Requirement, as defined in the Test Requirements document		Required
requirementType	Override existing requirement type with enumerated type of (REQUIRED, OPTIONAL, STRONGLY RECOMMENDED or RECOMMENDED)		Optional
Comment	Profile author's comment for a particular requirement		Optional

1136

1137 6.2 Relationships between Profiles, Requirements and Test Cases

1138

1139 Creation of a testing profile requires selection of a group of Test Requirement references that fulfill a
 1140 particular testing profile. For example, to create a testing profile for a Core Profile would require the
 1141 creation of an XML document referencing Test Requirements 1,2,3,4,5 and 8.

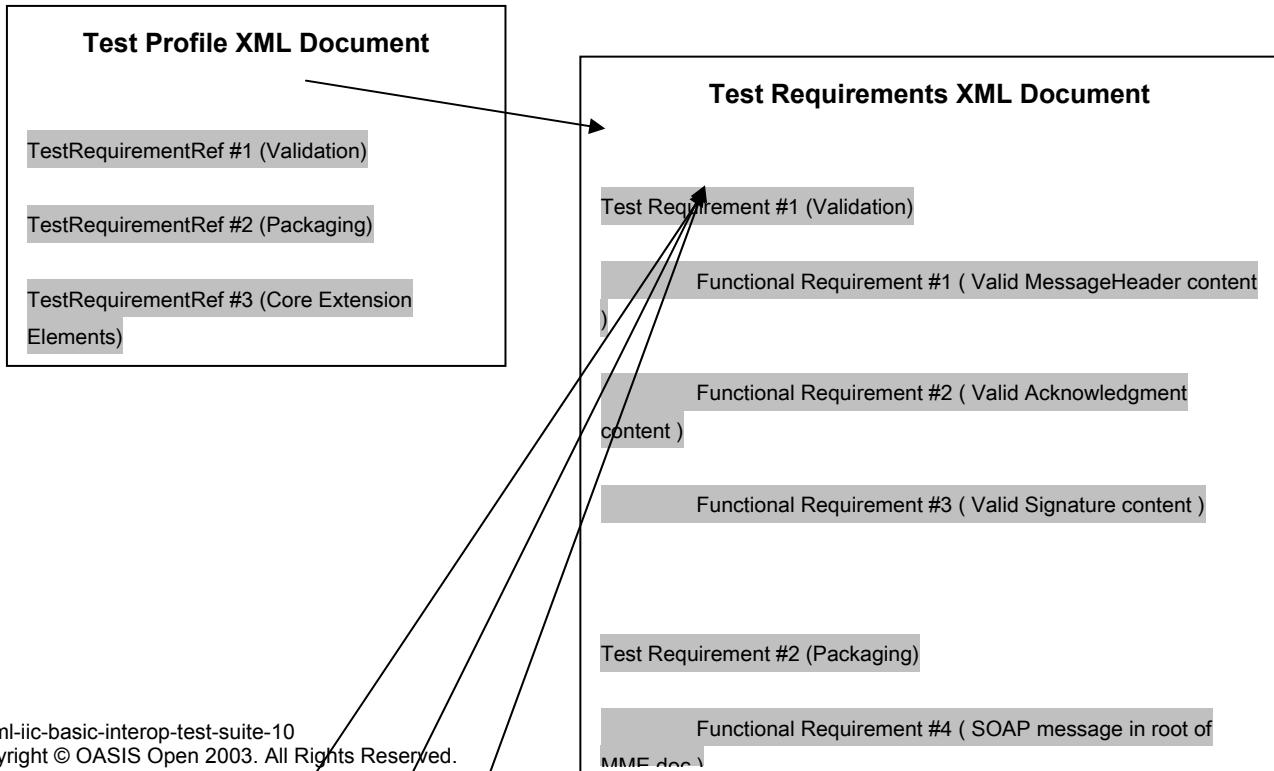
1142

1143 The Test Driver would read this list, and select (from the Test Requirements Document) the
 1144 corresponding Test Requirements (and their "sub" Functional Requirements). The Test Driver then
 1145 searches the Test Suite document to find all Test Cases that "point to" the selected Functional
 1146 Requirements. If more than one Test Case is necessary to satisfactorily test a single Functional
 1147 Requirement (as is the case for Functional Requirement #1) there may be more than one Test Case
 1148 pointing to it. The Test Driver would then execute Test Cases #1, #2 and #3 in order to fully test an
 1149 ebXML application against Functional Requirement #1.

1150

1151 The only test material outside of the three documents below that MAY require an external file reference
 1152 from within a Test Case are large, or non-XML message Payloads

1153



1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189

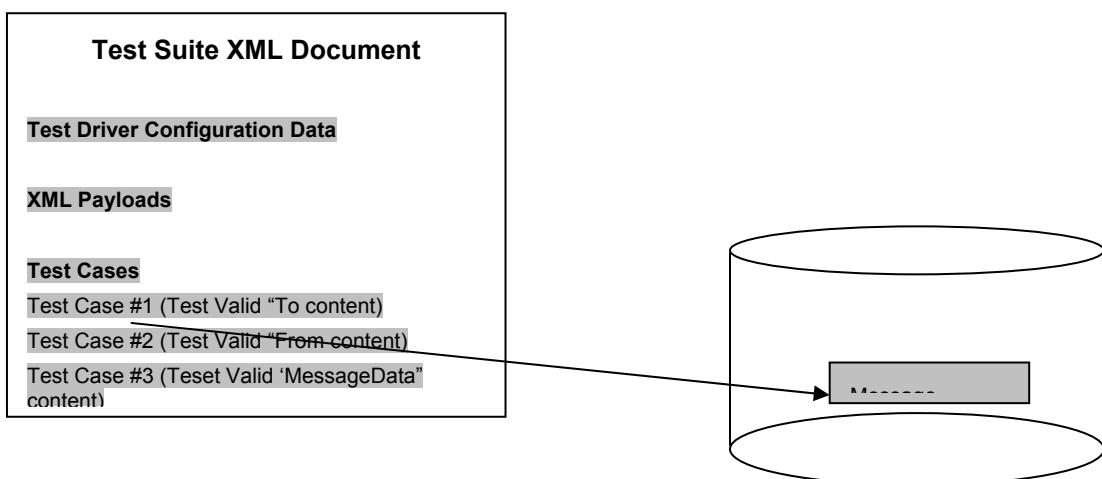


Figure 15 – Test Framework documents and their relationships

1190 7 Test Cases

1192 7.1 Structure of a Test Case

1193
1194 A Test Case is the translation of a Test Requirement (or a part of a Test Requirement), in an executable
1195 form, for a particular Test Harness. A Test Case includes the following information:

- 1196
- 1197 • Test Requirement reference.
 - 1198 • A Sequence of Test Steps.
 - 1199 • Condition(s) of success or of failure.

1200

1201 NOTE: The same Test Case may consolidate several Test Requirement items, i.e. a successful outcome
1202 of its execution will verify the associated set of Test Requirement items. This is usually the case when
1203 each of these Test Requirement items can make use of the same sequence of operations, varying only in
1204 the final test condition. When several Test Requirement items are covered by the same Test Case, the
1205 processing of the latter SHOULD produce separate verification reports.

1206

1207

1208 Test Cases MUST evaluate to a Boolean value of “true/false” or (semantically) a “pass/fail”. The
1209 aggregated results of all Test Steps in a Test Case MUST evaluate to “true” for a Test Case result to be
1210 “pass”.

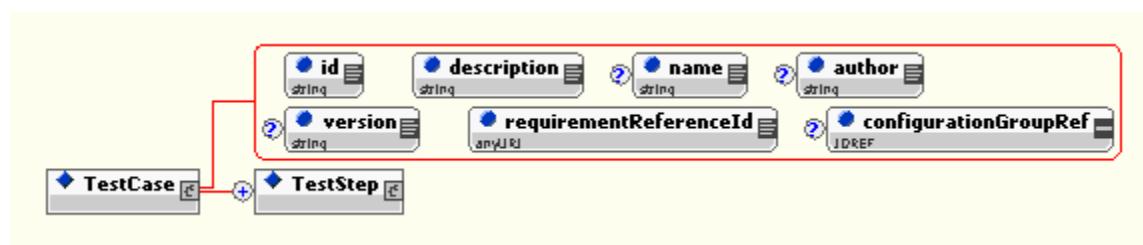
1211

1212 Prior to executing a Test Case, any configuration data necessary to modify the default configuration of the
1213 Test Driver MUST be included via a reference to a configurationGroupRef.

1214

1215

1216



1217 Figure 16 – Graphic representation of expanded view of the TestCase element

1222 Definition of Content

Name	Description	Default Value From Test Driver	Required/Optional
id	Unique identifier for this Test Case		Required
description	Short description of TestCase		Optional
name	Short name for Test Case		Required
author	Name of person(s) creating the Test Case		Optional
version	Version number of Test Case		Optional
requirementReferenceld	Pointer to the unique ID of the FunctionalRequirement		Required
configurationGroupRef	IDREF to reconfigure Test Driver using selected ConfigurationGroup		Optional
TestStep	Container for send, receive and message verification operations		Required

1224

1225 **7.1.1 Test Steps**

1226

1227 Test Steps are operations that MUST evaluate to a Boolean value of “true/false” or (semantically) a
 1228 “pass/fail”. The aggregated results of all Test Steps in a Test Case MUST evaluate to “true” for a Test
 1229 Case result to be “pass”.

1230

1231 Prior to executing a Test Step, any configuration data necessary to modify the default configuration of the
 1232 Test Driver MUST be included via a reference to a configurationGroupRef.

1233

1234 **7.1.2 Test Step Operations**

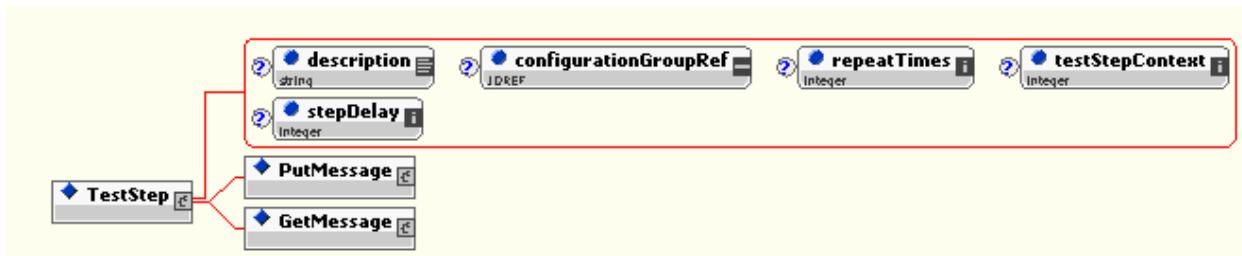
1235

1236 Within a Test Step, the Test Driver may perform one of two main operations: message construction and
 1237 transmission (PutMessage) or message retrieval and examination (GetMessage).

1238 Associated with a Test Step are optional and required attributes. Altering the configuration of the Test
 1239 Driver from its “bootstrap” or “base” configuration may be done through the addition of a
 1240 configurationGroupRef attribute and its value.

1241

1242



1243

1244 Figure 17 – Graphic representation of expanded view of the TestStep element

1245

1246

1247 **Definition of Content**

1248

Name	Description	Default Value From Test Driver	Required/Optional
description	Short description of Test Step		Optional
configurationGroupRef	Reference to existing ConfigurationGroup to change current configuration for this Test Step		Optional
repeatTimes	Integer value indicating number of times this step should be repeated	1	Optional
testStepContext	Use CPAId, ConversationId, MessageId and RefToMessageId from previous step number indicated		Optional
stepDelay	Override the default delay between execution of this Test Step and the previous Test Step	Taken from current ConfigurationGroup value	Optional
PutMessage	Directive to construct and send an ebXML Message in its entirety (MIME, SOAP, and ebXML)		Required
GetMessage	Directive to retrieve messages from Message Store in their entirety		Required

1249

1250

1251 **7.1.3 The PutMessage Operation**

1252

1253 The PutMessage Operation builds an ebXML message, along with its SOAP and MIME containers. A minimal Message Declaration is required to create a message, with default values for MIME headers and ebXML message content provided by the Test Driver. A Message Declaration described in a PutMessage Operation MUST validate against the ebXMLTestSuite.xsd schema in Appendix C.

1257

1258 The message components that can be created and modified by PutMessage include:

1259

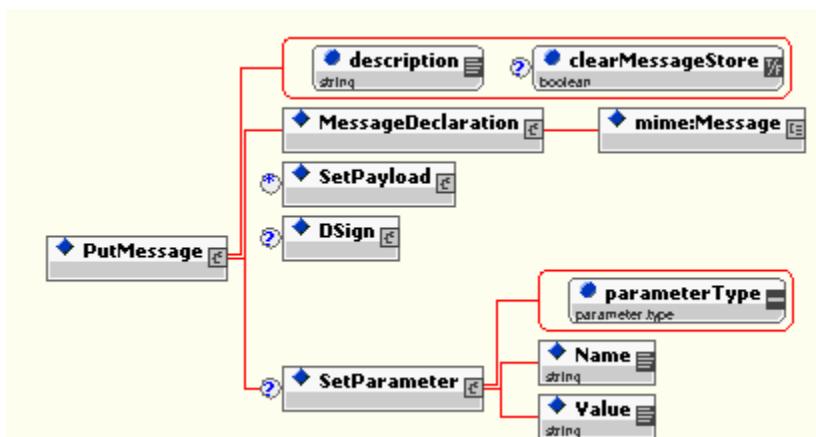
1260

- 1261 • **MIME header data:** MIME headers MUST be created or modified using the declarative syntax described in the ebXMLTestSuite.xsd schema in Appendix C. Default message MIME header data is illustrated in the message envelope template in section 7.1.6. How the MIME headers are

1264 actually constructed is implementation dependent. Test Drivers operating in “service” mode MAY
1265 ignore the MIME portion of a Message Declaration, since message MIME manipulation may be
1266 unavailable at the application level interface used for a particular ebXML MSH implementation.
1267 Test drivers in “connection” mode MUST properly interpret the MIME portion of a Message
1268 Declaration and generate the appropriate MIME header information.

- 1269
- 1270 • **SOAP header and body data:** SOAP message content MUST be created or modified using the
1271 declarative syntax described in the ebXMLTestSuite.xsd schema described in Appendix C.
1272 Default message SOAP content is illustrated in the message envelope template in section 7.1.6.
1273 How the actual SOAP message is constructed is implementation dependent. Test Drivers
1274 operating in “service” mode MAY ignore the SOAP portion of a MessageDeclaration, since
1275 message SOAP manipulation may be unavailable at the application level interface used for an
1276 MSH implementation. Test drivers in “connection” mode MUST properly interpret the SOAP
1277 portion of a Message Declaration and generate the appropriate SOAP header information.
 - 1278
 - 1279 • **ebXML Message data:** ebXML message content MUST be created or modified using the
1280 declarative syntax described in the ebXMLTestSuite.xsd schema described in Appendix C. How
1281 the actual ebXML message is constructed is implementation dependent. Test drivers operating in
1282 both “service” and “connection” modes MUST properly interpret the ebXML portion of a Message
1283 Declaration, and generate the appropriate ebXML content.
 - 1284
 - 1285 • **ebXML payload data:** ebXML message payload content, when there is a payload, MUST be
1286 created or modified using the syntax described in the ebXMLTestSuite.xsd schema described in
1287 Appendix C. ebXML payloads are created through file or through an XML ID reference inclusion
1288 into a message, and MAY be modified through any implementation-specific XML syntax.
 - 1289

1290 In addition to creating message content, the PutMessage operation can also execute sub-operations to
1291 create payloads, digitally sign any portion of the message and set global Test Case parameters that can
1292 be used by other Test Steps in the current Test Case in their XPath evaluation of message content.



1294
1295 Figure 18 – Graphic representation of expanded view of the PutMessage element

1301 **Definition of Content**

1302

Name	Description	Default Value From Test Driver	Required/Optional
description	Metadata describing the nature of the PutMessage operation		Required
clearMessageStore	Boolean attribute directive to purge the Test Driver Message Store of all previous received messages for this particular Test Case	false	Optional
MessageDeclaration	Container for XML content to construct the message		Required
SetPayload	Container element for Test Driver directives to create MIME attachments (or Payloads) to message		Optional
DSign	Container element for XML Digital Signature declaration(s) for this message		Optional
SetParameter	Container for user-defined parameter to be made available to other Test Steps (source can be this message's content retrieved via XPath statement or simply a user-supplied string value)		Optional
parameterType	Choice of Xpath or string		Required
Name	Name of new parameter		Required
Value	String value or XPath expression pointing to desired element/attribute value in message or simply a user-supplied string value		Required

1303

1304 **7.1.4 The Message Declaration**

1305

1306 The MessageDeclaration element is a container element for XML content describing the construction of
1307 MIME, SOAP and ebXML portions of a message. The XML content necessary to describe a basic
1308 message is minimal, with default parameter values supplied by the Test Driver for most message content.
1309 If the test developer wishes to "override" the default element and attribute values, they may do so by
1310 explicitly declaring those values in the XML markup.

1311

1312 Default values for element and attribute content may come from two sources. Either the Test Driver/MSH
1313 generates that value, (such as for message Timestamp), or the value is declared in the
1314 ConfigurationGroup parameters described in section 4.2.1. For example, a Test Suite
1315 ConfigurationGroup element content may be:

1316

1317

```
<ebTest:ConfigurationGroup ebTest:id="cpa_basic">  
1318     <ebTest:CPAId>cpa_basic</ebTest:CPAId>  
1319     <ebTest:Mode>connect</ebTest:Mode>  
1320     <ebTest:SenderParty>urn:oasis:iic:testdriver</ebTest:SenderParty>
```

```

1321 <ebTest:ReceiverParty>urn:oasis:iic:testsuite</ebTest:ReceiverParty>
1322 <ebTest:Service>urn:ebXML:iic:test.</ebTest:Service>
1323 <ebTest:Action>Dummy</ebTest:Action>
1324 </ebTest:ConfigurationGroup>

1325
1326
1327 A Test Driver could then read the MessageDeclaration (below) authored by the test writer and build the
1328 message, inserting element and attribute content wherever it knows default content should be, and
1329 declaring, or overriding default values where they are explicitly defined in the Message Declaration.
1330
1331

1332 <ebTest:MessageDeclaration>
1333 <mime:Message>
1334   <mime:MessageContainer>
1335     <soap:Envelope>
1336       <soap:Header>
1337         <eb:MessageHeader/>
1338       </soap:Header>
1339       <soap:Body />
1340     </soap:Envelope>
1341   </mime:MessageContainer>
1342 </mime:Message>
1343 </ebTest:MessageDeclaration>

1344
1345
1346 For illustrative purposes, the resulting message can be represented by the example message below. The
1347 Test Driver, after parsing the simple Message Declaration above, would generate the following MIME
1348 message with enclosed SOAP/ebXML content.

1349
1350

1351 Content-Type: multipart/related; type="text/xml"; boundary="boundaryText";
1352 start=messagepackage@oasis.org
1353
1354 --boundaryText
1355
1356 Content-ID: <messagepackage@oasis.org>
1357 Content-Type: text/xml; charset=UTF-8
1358
1359 <soap:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"
1360   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
1361   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
1362   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
1363 2_0.xsd" xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
1364 http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
1365 http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
1366
1367 <http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd>
1368 <soap:Header>
1369   <eb:MessageHeader soap:mustUnderstand="1" eb:version="2.0">
```

```

1372 <eb:From>
1373   <eb:PartyId>urn:oasis:iic:testdriver</eb:PartyId>
1374 </eb:From>
1375 <eb:To>
1376   <eb:PartyId>urn:oasis:iic:testserv</eb:PartyId>
1377 </eb:To>
1378 <eb:CPAId> urn:config:cpa_basic</eb:CPAId>
1379 <eb:ConversationId> 987654321</eb:ConversationId>
1380 <eb:Service>urn:ebXML:ic:test</eb:Service>
1381 <eb:Action>Dummy</eb:Action>
1382 <eb:MessageData>
1383   <eb:MessageId>0123456789</eb:MessageId>
1384   <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp> MessageData>
1385 </eb:MessageHeader>
1386 </soap:Header>
1387 </soap:Envelope>
1388
1389 --boundaryText

```

1390
1391 Certain XML message values (illustrated in red above) are supplied by the Test Driver, and are also
1392 made available to subsequent Test Step operations through global XPath parameter names. These
1393 parameters define the Test Step Context:

- 1394
- 1395 • \$CPAId
 - 1396 • \$ConversationId
 - 1397 • \$MessageId
 - 1398 • \$RefToMessageId
 - 1399 • \$Service
 - 1400 • \$Action
 - 1401 • \$SenderParty
 - 1402 • \$ReceiverParty

1403 Some of these values dynamically change with each PutMessage operation. These include MessageId.
1404 Other's change with each Test Case (\$ConversationId). Still other parameters remain "constant" within
1405 the scope of the current ConfigurationGroup definition (\$CPAId, \$Service, \$Action, \$SenderParty and
1406 \$ReceiverParty). ALL can be "overridden" through explicit declaration in the Message Declaration.

1407 The ebXMLTestSuite.xsd schema in Appendix C defines the format for element and attribute content
1408 declaration. However, the schema alone DOES NOT define default element content, since this is
1409 beyond the capability of schemas. Therefore, Test Driver implementers MUST consult the "Definition of
1410 Content" tables for this section of the specification to determine what default XML content must be
1411 generated by the Test Driver or MSH to create a valid ebXML message.

1412 The following sections description of how a Test Driver or MSH MUST interpret the MessageDeclaration
1413 content in order to be conformant to this specification.

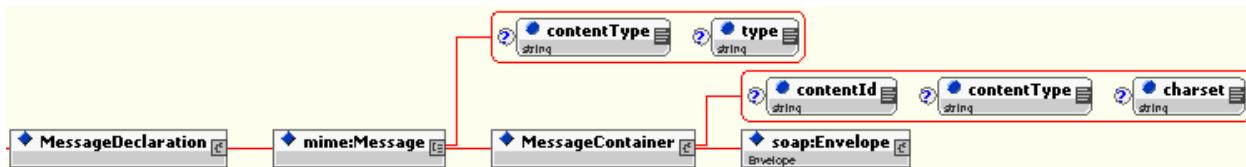
1414

1415 7.1.4.1 Interpreting the MIME portion of the Message Declaration

1416

1421 The XML syntax used by the Test Driver to construct the MIME message content consists of the
1422 declaration of a main MIME container for the entire message, followed by a MIME container for the SOAP
1423 message envelope. Default values for MIME headers MAY be "overridden" by explicit declaration of their
1424 values in the MessageDeclaration content; otherwise, default values are used by the Test Driver to
1425 construct the MIME headers.

1426



1427
1428 Figure 19 – Graphic representation of expanded view of the MessageDeclaration element

1429

1430

1431 Definition of Content

1432

Name	Declaration Description	Default Value From Test Driver	Required/Optional
mime:Message	Generate container for MIME, SOAP and ebXML message content		Required
contentType	Generate a MIME message 'Content-Type' header	multipart/related	Optional
type	Generate a MIME message 'type' header	text/xml	Optional
MessageContainer	Generate a MIME container in message		Required
contentId	Generate a 'Content-ID' MIME header for the container	messagepackage @oasis.org	Optional
contentType	Generate a MIME message package 'Content-Type' header	text/xml	Optional
charset	Generate a MIME message package character set	UTF-8	Optional
soap:Envelope	Generates a MIME container for SOAP message		Required

1433

1434

1435 An Example of Minimal MIME Declaration Content

1436

1437 The following XML represents all the information necessary to permit a Test Driver to construct a MIME
1438 message that may contain a SOAP envelope in its first MIME container. The XML document below
1439 validates against the ebXMLTestSuite.xsd schema in Appendix C.

1440

1441

```
<mime:Message xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/testing/mime">
  <mime:MessageContainer/>
```

1443 </mime:Message>

1444

1445

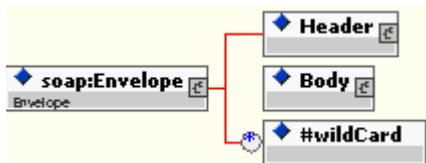
1446 **7.1.4.2 Interpreting the SOAP portion of the Message Declaration**

1447

1448 The XML syntax interpreted by the Test Driver to construct the SOAP message content consists of the
1449 declaration of a SOAP Envelope element, which in turn is a container for the SOAP Header, Body and
1450 non-SOAP XML content. Construction of the SOAP Header and Body content is simple for the Test
1451 Driver, requiring only the creation of the two container elements with their namespace properly declared,
1452 and valid according to the [SOAP]. The SOAP Body element, or any “wildcard” XML content is only
1453 constructed by the Test Driver if it is explicitly declared in the content.

1454

1455



1456

1457 Figure 20 – Graphic representation of expanded view of the soap:Envelope element declaration

1458

1459

1460

1461

1462 **Definition of Content**

1463

Name	Declaration Description	Default Value From Test Driver	Required/Optional
soap:Envelope	Generate container element with its proper namespace for SOAP Header and Body elements and their content		Required
soap:Header	Generate SOAP Header extension element		Required
soap:Body	Modify the default Body element	Element is auto-generated by Test Driver at run time	Optional
#wildCard	Generate “inline” wildcard content inside SOAP Envelope		Optional

1464

1465

1466

1467 **An Example of Minimal SOAP Declaration Content**

1468

1469 The following XML represents all the information necessary to permit a Test Driver to construct a minimal
1470 SOAP message.

1471

```
1472 <soap:Envelope>
1473   <soap:Header/>
1474 </soap:Envelope>
```

1475

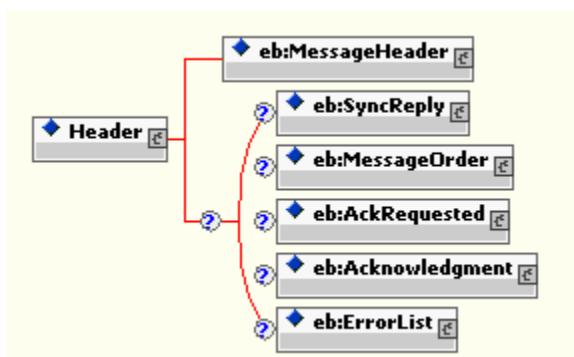
1476 **7.1.4.3 Interpreting the SOAP Header Extension Element Declaration**

1477

1478 The declarative syntax interpreted by the Test Driver to construct the ebXML Header extension message
1479 content consists of the declaration of a SOAP Header element, which in turn is a container for the ebXML
1480 Header extension elements and their content. The only extension element that is required in the container
1481 is the eb:MessageHeader element, which directs the Test Driver to construct an ebXML MessageHeader
1482 element, along with its proper namespace declaration, as defined in [EBMS]. The Test Driver does not
1483 construct any other Header extension elements unless they are explicitly declared as content in the
1484 SOAP Header Declaration.

1485

1486



1487

1488 Figure 21 – Graphic representation of expanded view of the soap:Header element declaration

1489

1490

1491

1492

1493 **Definition of Content**

1494

1495

Name	Declaration Description	Default Value From Test Driver	Required/Optional
Header	SOAP Header declaration and container for ebXML ebXML Header Extension Element declarations		Required
eb:MessageHeader	Create an ebXML MessageHeader element with namespace declaration		Required

eb:ErrorList	Create an ebXML ErrorList element		Optional
eb:SyncReply	Create an ebXML SyncReply element		Optional
eb:MessageOrder	Create an ebXML MessageOrder element		Optional
eb:AckRequested	Create an ebXML AckRequested element		Optional
eb:Acknowledgment	Create an ebXML Acknowledgment element		Optional

1496

1497

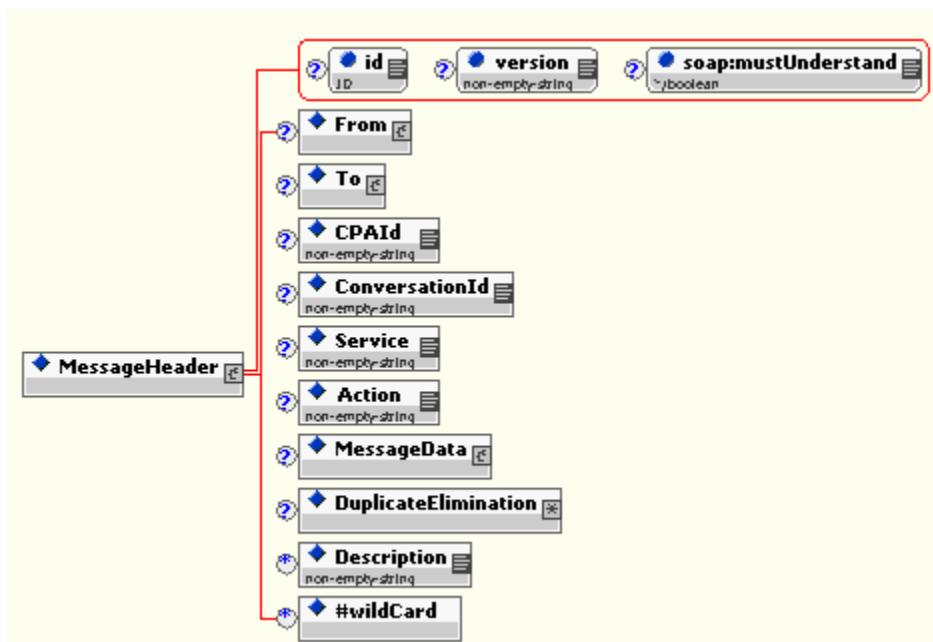
1498 7.1.4.4 Interpreting the ebXML MessageHeader Element Declaration

1499

1500 The XML syntax interpreted by the Test Driver to construct the ebXML MessageHeader extension content
 1501 consists of the declaration of a MessageHeader element, and a required declaration of CPAId and Action
 1502 elements within it. This is the "minimum" declaration a Test Driver needs to generate an ebXML Message
 1503 Header. All other required content, as defined in the schema in the ebXML MS v2.0 Specification, is
 1504 provided by the Test Driver through either default parameters defined in the ebXMLTestSuite.xsd schema
 1505 in Appendix C, or directly generated by the Test Driver (e.g. to generate necessary message container
 1506 elements) or by explicit declaration of content in the Message Declaration. The figure below illustrates
 1507 the schema for an ebXML Message Header declaration to be interpreted by the Test Driver.

1508

1509



1510

1511 Figure 22 – Graphic representation of expanded view of the ebXML MessageHeader element declaration

1512

1513 Definition of Content

1514

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:MessageHeader	Generate MessageHeader element and all of its default element/attribute content		Required
id	Generate attribute with declared value		Optional
version	Modify default attribute value	2.0	Optional
soap:mustUnderstand	Modify default attribute value	true	Optional
From	Modify default From message element generated by Test Driver	Generated by Test Driver/MSH at run time	Optional
PartyId	Replace default element value with new value	Generated by Test Driver/MSH at run time, using config value	Required
type	Generate a type attribute with value		Optional
Role	Generates a Role element with its value		Optional
To	Modify default To message element generated by Test Driver	Generated by Test Driver at run time	Optional
PartyId	Replace default element value with new value	Generated by Test Driver/MSH at run time, using config value	Required
type	Generate type attribute with value		Optional
Role	Generates a Role element with its value		Optional
CPAId	Generate element with its value	Generated by Test Driver/MSH at run time, using config value	Optional
ConversationId	Modify default value provided by Test Driver	Generated by Test Driver at run time	Optional
Service	Modify default value generated by Test Driver	Generated by Test Driver/MSH at run time, using config value	Optional
Action	Replace default value with	Generated by Test	

	specified Action name	Driver/MSH at run time, using config value	Optional
MessageData	Modify default container generated by Test Driver	Generated by Test Driver at run time	Optional
MessageId	Modify default value generated by Test Driver	Generated by Test Driver at run time	Optional
Timestamp	Modify default value generated by Test Driver	Generated by Test Driver at run time	Optional
RefToMessageId	Generate element and its value		Optional
TimeToLive	Generate element and its value	Generated by Test Driver at run time	Optional
DuplicateElimination	Generate element		Optional
Description	Generate element with value		Optional
#wildcard	Generate content inline		Optional

1515

1516

1517 **An Example of a Minimal ebXML MessageHeader Content Declaration**

1518

1519 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
 1520 MessageHeader element with all necessary content to validate against the ebXML MS V2.0 schema. All
 1521 declared content must validate the ebXMLTestSuite.xsd schema in Appendix C.

1522

1523

`<eb:MessageHeader/>`

1524

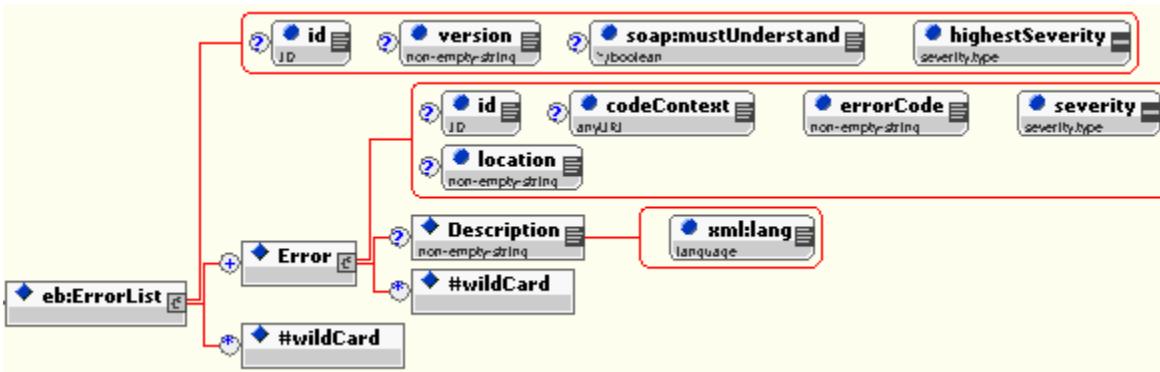
1525 **7.1.4.5 Interpreting the ebXML ErrorList Element Declaration**

1526

1527 The XML syntax interpreted by the Test Driver to construct the ebXML ErrorList extension content
 1528 consists of the declaration of an ErrorList element, and a required declaration of one or more Error
 1529 elements within it. All required content, as defined in the schema in the ebXML MS V2.0 Specification, is
 1530 provided through either default parameters defined in the ebXMLTestSuite.xsd schema and included by
 1531 the Test Driver, or by explicit declaration.

1532

1533



1534

1535 Figure 23 – Graphic representation of expanded view of the ebXML ErrorList element declaration

1536

1537

1538 **Definition of Content**

1539

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:ErrorList	Generate container element		Optional
id	Generate attribute and its value		Optional
version	Modify default value	2.0	Optional
soap:mustUnderstand	Modify default value	true	Optional
highestSeverity	Generate required attribute and value		Required
Error	Generate new Error container		Required
id	Generate attribute with declared value		Optional
codeContext	Generate element with declared value		Optional
errorCode	Generate required attribute and value		Required
severity	Generate required attribute and value		Required
location	Generate attribute with declared value		Optional
Description	Generate element with declared value		Optional
#wildCard	Generate content “inline” into message		Optional

1540

1541

1542

1543 An Example of a Minimal ebXML ErrorList Content Declaration

1544

1545 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
1546 ErrorList element with all necessary content to validate against the ebXML MS v2.0 schema. All required
1547 content not visible in the example would be generated by the Test Driver.

1548

```
1549 <eb:ErrorList eb:highestSeverity=Error>  
1550   <eb:Error eb:errorCode="Inconsistent" eb:severity="Error"/>  
1551 </eb:ErrorList>
```

1552

1553

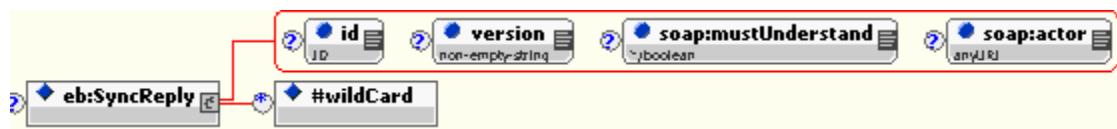
1554 7.1.4.6 Interpreting the ebXML SyncReply Element Declaration

1555

1556 The XML syntax interpreted by the Test Driver to construct the ebXML SyncReply extension content
1557 consists of the declaration of a SyncReply element. All required content, as defined in the schema in
1558 [EBMS], is provided through either default parameters provided by the Test Driver or through explicit
1559 declaration.

1560

1561



1562

1563 Figure 24 – Graphic representation of expanded view of the ebXML SyncReply element declaration

1564

1565

1566

1567

1568

1569 Definition of Content

1570

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:SyncReply	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default attribute value	2.0	Optional
soap:mustUnderstand	Modify default attribute value	true	Optional
soap:actor	Modify default attribute value	http://schemas.xmlsoap.org/soap/actor/next	Optional
#wildCard	Generate content "inline"		Optional

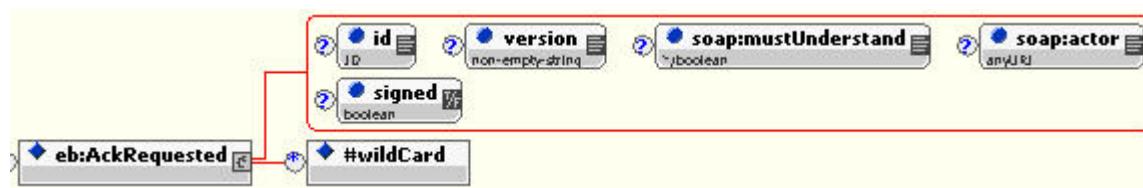
1571
1572
1573
1574
1575
1576 **An Example of a Minimal ebXML SyncReply Content Declaration**
1577

1578 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
1579 AckRequested element with all necessary content to validate against the [EBMS] schema schema.
1580

1581 `<eb:SyncReply/>`

1582 **7.1.4.7 Interpreting the ebXML AckRequested Element Declaration**

1583
1584 The XML syntax interpreted by the Test Driver to construct the ebXML AckRequested extension content
1585 consists of the declaration of an AckRequested element. All required content as defined in the [EBMS]
1586 schema, is provided by the Test Driver or by explicit declaration.
1587
1588



1589
1590 Figure 25 – Graphic representation of expanded view of the ebXML AckRequested element declaration
1591
1592

1593 **Definition of Content**

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:AckRequested	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default value	2.0	Optional
soap:mustUnderstand	Modify default value	true	Optional
soap:actor	Modify default attribute value with new value	urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH	Optional
signed	Modify default attribute value	false	Optional
#wildCard	Generate content “inline”		Optional

1595
1596
1597

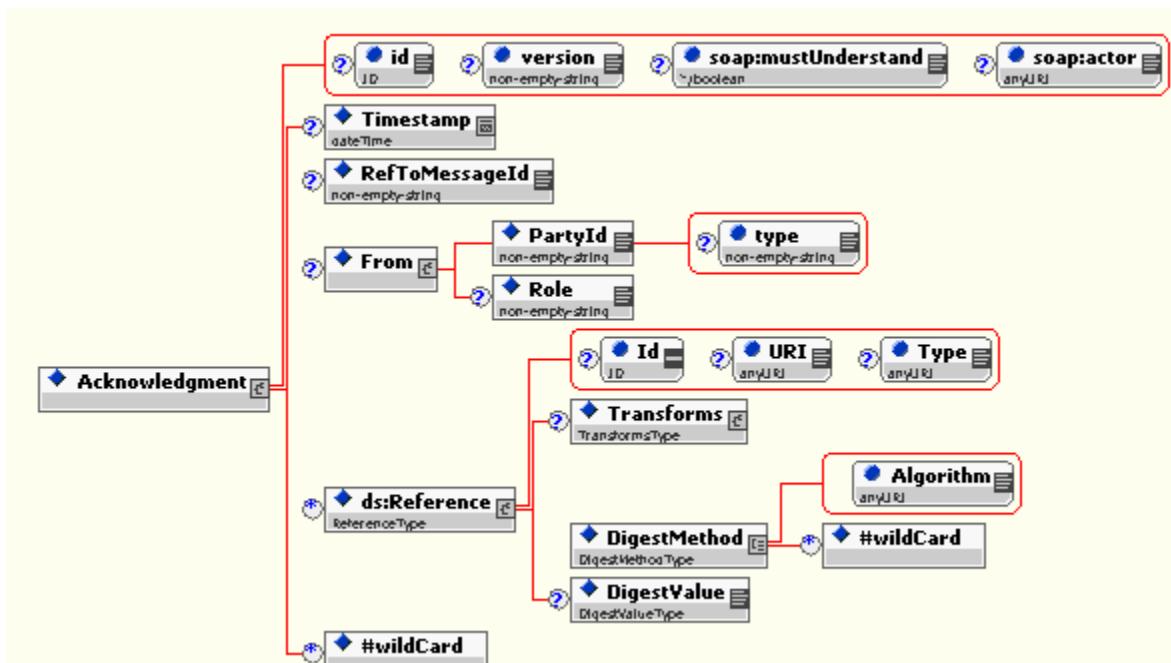
1598 An Example of a Minimal ebXML AckRequested Content Declaration

1599
1600 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
1601 AckRequested element with all necessary content to validate against the [EBMS] schema.

1602
1603 <eb:AckRequested/>
1604

1605 7.1.4.8 Interpreting the ebXML Acknowledgment Element Declaration

1606
1607 The XML syntax interpreted by the Test Driver to construct the ebXML Acknowledgment extension
1608 content consists of the declaration of an Acknowledgment element. All required content, as defined in the
1609 [EBMS] schema, is provided by the Test Driver or through explicit declaration.
1610
1611



1612
1613 Figure 26 – Graphic representation of expanded view of the ebXML Acknowledgment element declaration
1614
1615

1616 Definition of Content

Name	Declaration Description	Default Value From Test Driver	Required/Optional
------	-------------------------	--------------------------------	-------------------

eb:Acknowledgment	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default attribute value	2.0	Optional
soap:mustUnderstand	Modify default attribute value	true	Optional
soap:actor	Modify default attribute value	urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH	Optional
Timestamp	Modify default element value	Generated by Test Driver at run time	Optional
RefToMessageId	Modify default element value	Generated by Test Driver at run time	Optional
From	Modify default container	Generated by Test Driver at run time	Optional
PartyId	Modify default value	urn:ebxml:iic:testdriver	Required
type	Generate type attribute with value		Optional
Role	Generates a Role element with its value		Optional
ds:Reference	Generate container element and all default content		Optional
Id	Generate attribute and its value		Optional
URI	Modify default attribute value	""	Required
type	Generate attribute and its value		Optional
Transforms	Generate container relement		Optional
Transform	Generate element with its value		Optional
Algorithm	Modify default attribute value	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	Required
#wildCard	Generate content "inline"		Optional
XPath	Generate element with its value		Optional
DigestMethod	Generate element with its value		Required
Algorithm	Modify default attribute value	Generated by Test Driver at run time, based upon CPA	Required
#wildCard	Generate content "inline"		Optional

DigestValue	Generate element with its value	Computed by Test Driver at run time	Required
#wildCard	Generate content "inline"		Optional

1618

1619

1620

1621 An Example of a Minimal "unsigned" ebXML Acknowledgment Content Declaration

1622

1623 The following XML represents the minimum information necessary to permit a Test Driver to construct an
1624 ebXML Acknowledgment element.

1625

```
1626 <eb:Acknowledgment/>
```

1627

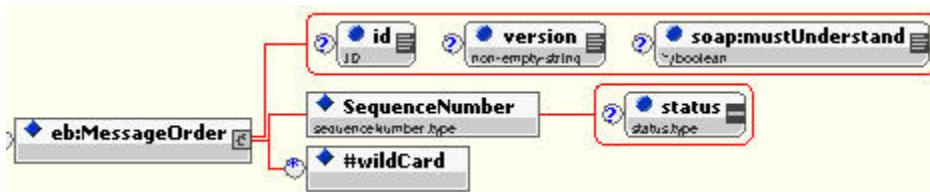
1628

1629 7.1.4.9 Interpreting the ebXML MessageOrder Element Declaration

1630

1631 The XML syntax interpreted by the Test Driver to construct the ebXML MessageOrder extension content
1632 consists of the declaration of a MessageOrder element. All required content, as defined in the [EBMS]
1633 schema, is provided by the Test Driver or through explicit declaration.

1634



1635

1636 Figure 27 – Graphic representation of expanded view of the ebXML MessageOrder element declaration

1637

1638

1639 Definition of Content

1640

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:MessageOrder	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default attribute value	2.0	Optional
soap:mustUnderstand	Modify default attribute value	true	Optional
SequenceNumber	Generate element with declared value		Required

status	Generate attribute with declared value		Optional
#wildCard	Generate content "inline"		Optional

1641

1642

1643 **An Example of a Minimal ebXML MessageOrder Content Declaration**

1644

1645 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
1646 MessageOrder element.

1647

```
1648 <eb:MessageOrder>
1649 <eb:SequenceNumber>1</eb:SequenceNumber>
1650 </eb:MessageOrder>
```

1651

1652 **7.1.4.10 Interpreting the SOAP Body Extension Element Declaration**

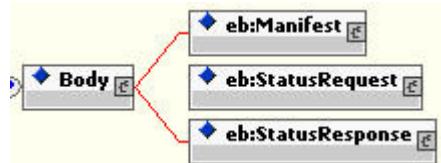
1653

1654 The XML syntax used by the Test Driver to construct the ebXML Body extension message content
1655 consists of the declaration of a SOAP Body element, which in turn is a container for the ebXML Manifest,
1656 StatusRequest or StatusResponse elements.

1657 The Test Driver does not construct any of these SOAP Body extension elements unless they are explicitly
1658 declared as content in the SOAP Body Declaration.

1659

1660



1661

1662 Figure 28 – Graphic representation of expanded view of the soap:Body element declaration

1663

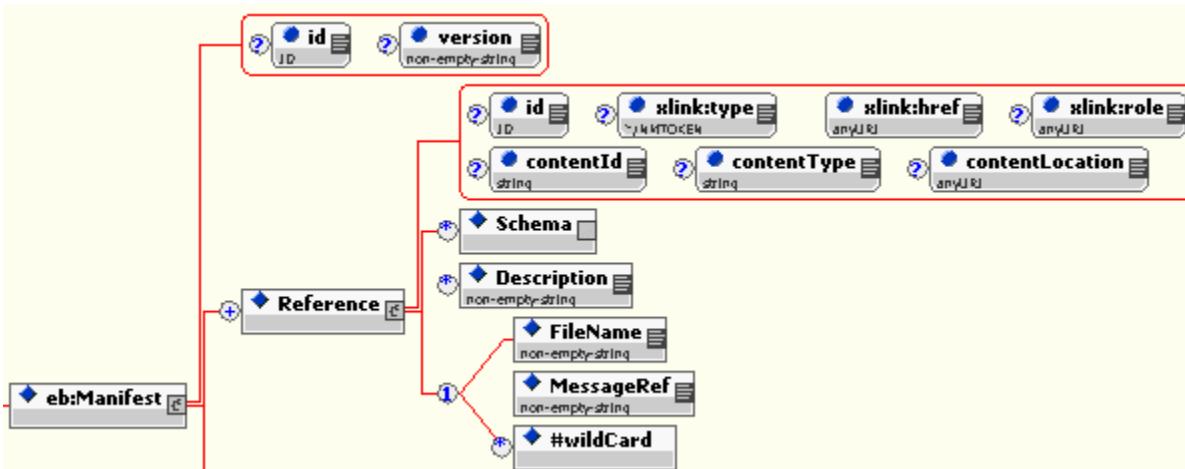
1664 **7.1.4.11 Interpreting the ebXML Manifest Element Declaration**

1665

1666 The XML syntax interpreted by the Test Driver to construct the ebXML Manifest extension content
1667 consists of the declaration of a Manifest element. All required content, as defined in the [EBMS] schema,
1668 is provided by the Test Driver or through explicit declaration

1669

1670



1671
1672 Figure 29 – Graphic representation of expanded view of the ebXML Manifest element declaration
1673
1674

1675 **Definition of Content**

1676

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:Manifest	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default attribute value	2.0	Optional
id	Modify default attribute value	true	Optional
xlink:type	Generate element with declared value		Optional
xlink:href	Generate attribute with declared value		Required
xlink:role	Generate attribute with declared value		Optional
contentId	Modify the Content-ID MIME header of the payload		Optional
contentType	Set the Content-Type MIME header of the payload		Optional
contentLocation	Set the Content-Location MIME header of the payload		Optional
Schema	Generate schema container element		Optional
location	Generate URI attribute and value of schema location		Required
version	Generate schema version attribute and value		Optional
Description	Generate description element and value		Optional

xml:lang	Generate description language attribute and value		Required
PayloadLocation	Load specified file as a MIME attachment to message		Required
MessageRef	Load designated XML document via IDREF as a MIME attachment to message		Required
PayloadDeclaration	“Inline” the XML content of this element as a MIME message attachment		Required

1677

1678

1679 **An Example of a Minimal ebXML Manifest Content Declaration**

1680

1681 The following XML represents the minimum information necessary to permit a Test Driver to construct an
1682 ebXML Manifest element with all necessary content to validate against the ebXML MS v2.0 schema.

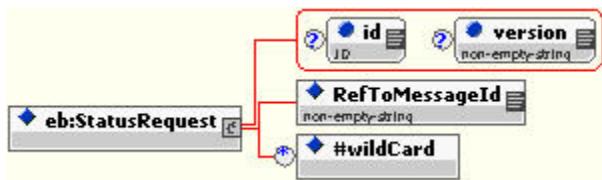
1683

1684 <eb:Manifest>
1685 <eb:Reference xlink:href="cid:payload 1"/>
1686 </eb:Manifest>1687 **7.1.4.12 Interpreting the ebXML StatusRequest Element Declaration**

1688

1689 The XML syntax interpreted by the Test Driver to construct the ebXML StatusRequest extension content
1690 consists of the declaration of a StatusRequest element. All required content, as defined in the [EBMX]
1691 schema. All required content, as defined in the [EBMS] schema, is provided by the Test Driver or through
1692 explicit declaration

1693



1694

1695 Figure 30 – Graphic representation of expanded view of the ebXML StatusRequest element declaration

1696

1697

1698 **Definition of Content**

1699

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:StatusRequest	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional

version	Modify default value	2.0	Optional
RefToMessageId	Generate element and its value		Required
#wildCard	Generate content "inline"		Optional

1700

1701

1702 An Example of a Minimal ebXML StatusRequest Content Declaration

1703

1704 The following XML represents all the minimum information necessary to permit a Test Driver to construct
 1705 an ebXML StatusRequest element with all necessary content to validate against the [EBMS] schema.

1706

```
1707 <eb>StatusRequest>
1708 <eb:RefToMessageId>20001209-133003-28571@example.com</eb:RefToMessageId>
1709 </eb>StatusRequest>
```

1710

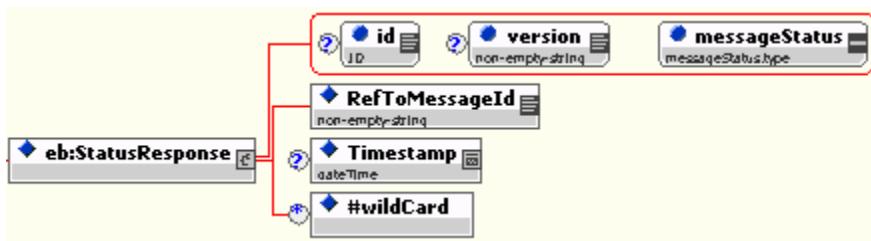
1711

1712 7.1.4.13 Interpreting the ebXML StatusResponse Element Declaration

1713

1714 The XML syntax used by the Test Driver to construct the ebXML StatusResponse extension content
 1715 consists of the declaration of a StatusResponse element with required and optional element/attribute
 1716 content.

1717



1718

1719 Figure 31 – Graphic representation of expanded view of the ebXML StatusResponse element declaration

1720

1721

1722 Definition of Content

1723

Name	Declaration Description	Default Value From Test Driver	Required/Optional
eb:StatusResponse	Generate container element and all default content		Optional
id	Generate attribute and its value		Optional
version	Modify default attribute value	2.0	Optional
messageStatus	Generate attribute and its		Optional

	value		
RefToMessageId	Generate element and its value		Required
Timestamp	Modify default value	Generated by Test Driver at run time	Optional
#wildCard	Generate content "inline"		Optional

1724

1725

1726 An Example of a Minimal ebXML StatusResponse Content Declaration

1727

1728 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML
 1729 StatusResponse element with all necessary content to validate against the [EBMX] schema.

1730

```
1731 <eb>StatusResponse messageStatus="Processed"/>
```

1732

1733

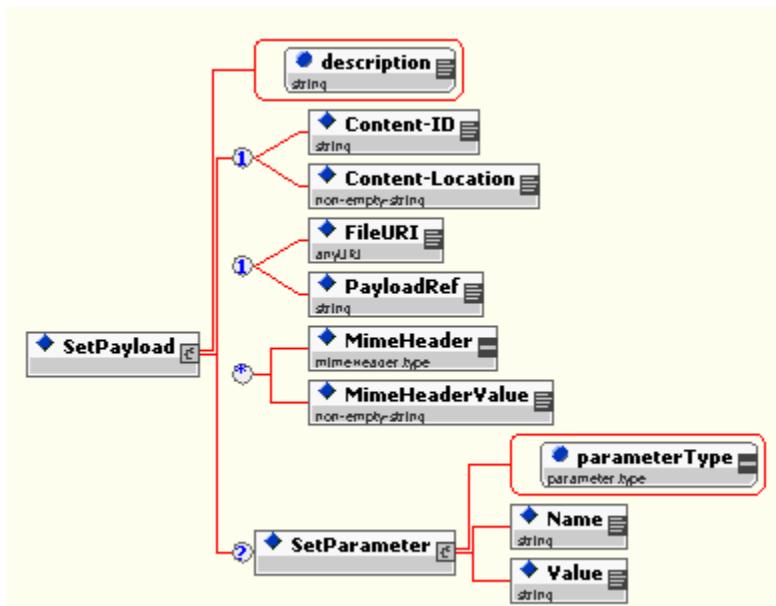
1734 7.1.5 The SetPayload Operation

1735

1736 The SetPayload Operation is a sub-operation for the PutMessage Operation. It provides the Test Driver
 1737 with the necessary information to append a message payload. Payloads can be provided to the driver
 1738 through a file name reference, an in-memory message document reference, or can be constructed "on-
 1739 the-fly" through any declarative syntax specific to an ebXML application.

1740

1741



1742

1743 Figure 32 – Graphic representation of expanded view of the SetPayload element

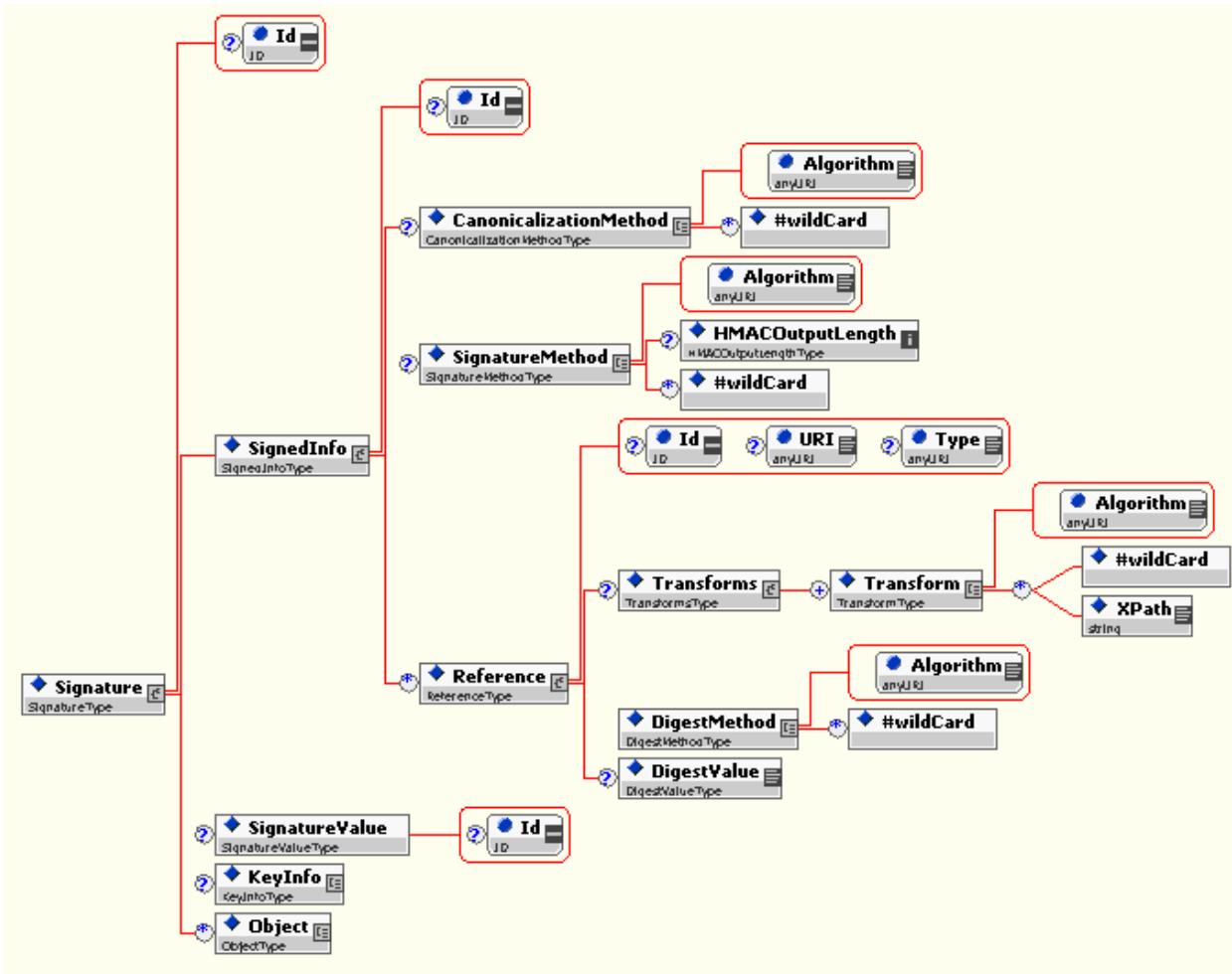
1744
1745
1746 **Definition of Content**
1747

Name	Description	Default Value	Required/Optional
description	Metadata describing the nature of the SetPayload operation		Required
Content-ID	Set the Content-Id MIME header of the payload		Required
Content-Location	Set the the MIME Content-Location header of the payload		Required
FileURI	URI of the file to be loaded as a payload		Required
PayloadRef	Unique ID of the in memory XML document to be loaded as the payload		Required
MimeHeader	Set any type of MIME header name		Optional
MimeHeaderValue	Set corresponding MIME header value		Optional
SetParameter	Container for user-defined parameter to be made available to other Test Steps (source can be this message's content retrieved via XPath statement or simply a user-supplied string value)		Optional
parameterType	Choice of xpath or string		Required
Name	Name of new parameter		Required
Value	String value or XPath expression pointing to desired element/attribute value in payload, or simply a user-supplied string value		Required

1748
1749

1750 **7.1.6 The Dsign Operation**

1751
1752 The DSign Operation instructs the Test Driver to digitally sign the portion of the message defined in its
1753 Reference element content.
1754



1755

1756 Figure 33 – Graphic representation of expanded view of the DSign element

1757

1758

1759 Definition of Content

1760

Name	Description	Default Value From Test Driver	Required/Optional
DSign	Container for Signature declaration content		Optional
ds:Signature	Signature root element, as defined in [XMLDSIG]		Required
Id	Unique identifier for Signature		Optional
SignedInfo	Create container for Canonicalizatoin and Signature algorithms and		Required

	References		
CanonicalizationMethod	Modify default container element	Container auto-generated by Test Driver	Optional
Algorithm	Modify default attribute and value	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	Required
#wildCard	Generate content "inline"		Optional
SignatureMethod	Create container element		Required
Algorithm	Create attribute and value		Required
HMACOutputLength	Generate Element and its value		Optional
#wildcard	Generate content "inline"		Optional
ds:Reference	Generate container element and all default content		Optional
Id	Generate attribute and its value		Optional
URI	Modify default attribute value	“”	Optional
type	Generate attribute and its value		Optional
Transforms	Generate container relement		Optional
Transform	Generate element with its value		Optional
Algorithm	Modify default attribute value	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	Required
#wildCard	Generate content "inline"		Optional
XPath	Generate element with its value		Optional
DigestMethod	Generate element with its value		Required
Algorithm	Generate attribute and value		Required
#wildCard	Generate content "inline"		Optional
DigestValue	Generate element	Set by Test Driver, based upon URI	

	with its value	value	Optional
#wildCard	Generate content "inline"		Optional
SignatureValue	Generate element and its value	Set by Test Driver at run time	Optional
Id	Generate attribute and its value		Optional
KeyInfo	Generate container Element	All required and optional content, as described in [XMLDSIG] MUST be explicitly declared (no auto-generation by Test Driver)	Optional
Object	Generate container element		Optional

1761

1762

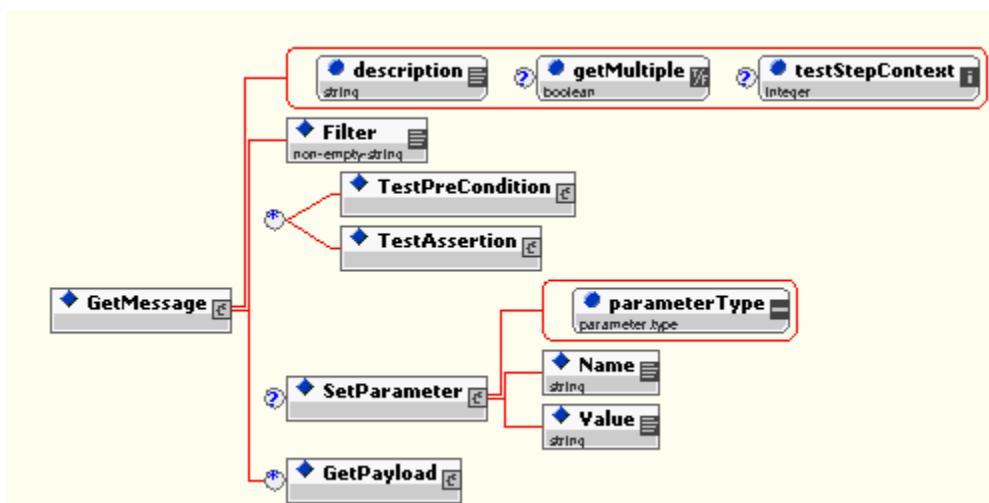
1763 7.1.7 The GetMessage Operation

1764

1765 The GetMessage Operation, using its child XPath Filter query retrieves a node-list of Messages received
 1766 from the Message Store of the Test Driver. The content of the node-list is dependent upon the Filter
 1767 provided. The resulting node-list MAY be queried for Precondition or Test Condition tests. Any Payload
 1768 associated with a message MAY be queried through the GetPayload operation.

1769

1770



1771

1772 Figure 34 – Graphic representation of expanded view of the GetMessage element

1773

1774

1775

1776 Definition of Content

1777

Name	Description	Default Value from Test Driver	Required/Optional
GetMessage	Container element for filtering, verifying and validating message and payload content		Optional
description	Description the nature of the GetMessage operation		Required
getMultiple	By default, getMultiple is “false”, indicating that only one message should be present in the node-list , even if multiple messages are in the Message Store	false	Optional
testStepContext	Integer value corresponding to previous Test Step number whose ConversationId, CPAId, MessageId and RefToMessageId is re-used		Optional
Filter	XPath query to select message(s) from Message Store		Required
TestPreCondition	Container for verification or validation operation to be performed on message as pre-condition to testing the Assertion		Optional
TestAssertion	Container for verification or validation operation to be performed to test a conformance or interoperability assertion		Optional
SetParameter	Container for user-defined parameter to be made available to other Test Steps (source can be this message’s XML content retrieved via XPath statement or simply a user-supplied string value)		Optional
parameterType	Choice of xpath or string		Required
Name	Name of new parameter		Required
Value	String value or XPath expression pointing to desired element/attribute value in message, or simply a user-supplied string value		Required

1778

1779

1780

1781 **Semantics of the GetMessage operation**

1782

1783

1784 The Message Store is an XML document object that contains an XML representation of all synchronous
 1785 and asynchronously received ebXML messages for a Test Case. The received messages for a particular
 1786 Test Case MUST exist in the Message Store only for the life of the Test Case. Messages in the Message
 1787 Store contain all MIME, SOAP and ebXML content, represented as an XML document. The XML format
 1788 of the Message Store document MUST validate against the ebXMLMessageStore.xsd schema in
 1789 appendix D

1790

1791 The GetMessage operation queries the Message Store document object, and retrieves (by default) the
1792 earliest (determined by Timestamp) Message element and its XML content that satisfies the XPath
1793 expression in its Filter child element. If multiple Messages satisfy the XPath Filter expression, they MAY
1794 be included in the resulting node-set by setting the “getMultiple” attribute of the GetMessage element to
1795 “true”. This is useful when a test writer wishes to count the number of “duplicate” messages received by
1796 the Test Driver for example.

1797

1798 A TestPreCondition or TestAssertion operation MAY query the resulting node-list generated by a
1799 GetMessage operation

1800

1801 In addition, new parameters that can be used by other Test Steps in this Test Case may be created using
1802 the SetParameter sub-operation. The new parameter can be used in other XPath expressions as an
1803 XPath parameter value. The SetParameter operation can select a discreet element or attribute value
1804 using the XPath expression supplied in the “Value” sub-element. Or the SetParameter operation can
1805 define a new parameter through a simple user-supplied string value assignment to the parameter.

1806

1807

1808 7.1.8 The TestPreCondition Operation

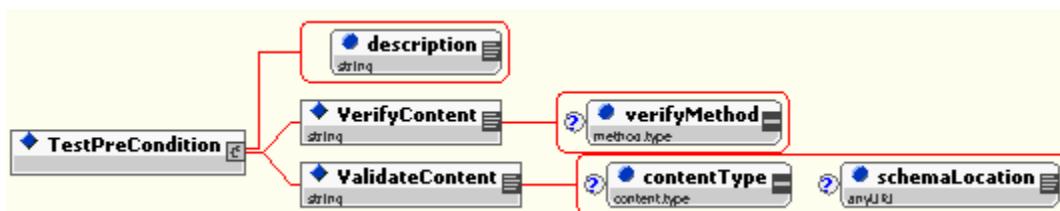
1809

1810 The TestPreCondition Operation examines a message or messages in a GetMessage node-list by testing
1811 the content of the node-list against the VerifyContent (content value comparison) or ValidateContent
1812 (content integrity evaluation) operation.

1813

1814

1815



1816

1817 Figure 35 – Graphic representation of expanded view of the TestPreCondition element

1818

1819

1820

1821 Definition of Content

1822

Name	Description	Default Value From Test Driver	Required/Optional
description	Metadata describing the nature of the TestPreCondition operation		Required
VerifyContent	Use XPath expression to evaluate content of message(s)		Optional

verifyMethod	Either XPath or mda-5 (for message or payload integrity test)		Optional
ValidateContent	Empty if entire XML document is to be validated or XPath expression to “point to” content to be validated for correct format if type is URI, dateTIme or Signature		Optional
contentType	An enumerated list of XML, URI, time, or Signature validation descriptors		Optional
schemaLocation	URI pointing to location of schema used to validate a content type of XML		Optional

1823

1824

1825

1826 Semantics of the TestPreCondition operation

1827

1828 The TestPreCondition operation MUST return either a true or false result (or semantically a pass/fail
1829 result) to the Test Driver.

1830

1831 If TestPreCondition includes a VerifyContent sub-operation, the VerifyContent operation MUST yield a
1832 boolean value of true/false, regardless of the resulting XPath object yielded by the XPath expression. The
1833 VerifyContent XPath expression may yield a node-set, boolean, number or string object. All of these
1834 resulting objects MUST be evaluated using the “boolean” function described in [XPath]. Those evaluation
1835 rules are:

1836

- 1837 • a returned node-set object evaluates to true if and only if it is non-empty
- 1838 • a returned boolean object evaluates to true if it evaluates to “true” and false if it evaluates to
1839 “false”
- 1840 • a returned number object evaluates to true if and only if it is neither positive or negative zero nor
1841 NaN
- 1842 • a returned string object evaluates to true if and only if its length is non-zero

1843

1844

1845 If TestPreCondition includes a ValidateContent sub-operation, the ValidateContent operation MUST yield
1846 a boolean value of true/false. Rules for determining the resulting Boolean value are:

1847

- 1848 • if the contentType attribute value is XML, as defined in [XML] , the operation evaluates to true if
1849 the content at the specified XPath validates according to the schema defined in the
1850 “schemaLocation” attribute

- 1851 • if the contentType is URI, as defined in [XMLSCHEMA], the operation evaluates to true if the
 1852 content at the specified XPath is a valid URI
 1853 • if the contentType is dateTime, as defined in [XMLSCHEMA], the operation evaluates to true if
 1854 the content at the specified XPath is a valid dateTime
 1855 • if the contentType is signature, as defined in [XMLDSIG], the operation evaluates to true if the
 1856 content at the specified XPath is a validly signed element.

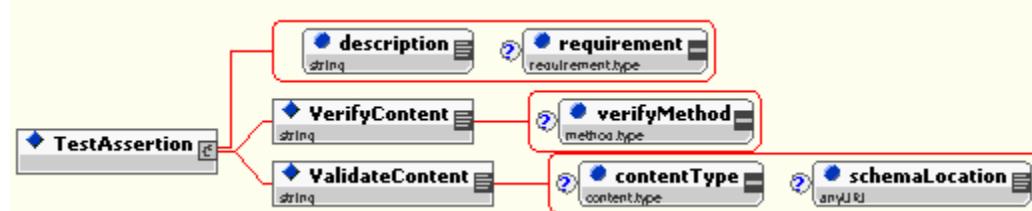
1857

1858

1859 **7.1.9 The TestAssertion Operation**

1860

1861 The TestAssertion Operation examines a message or messages in a node-list by testing the content
 1862 against an XPath expression in the TestPreCondition text. If the XPath expression returns a node-list
 1863 with one or more nodes, the ConformanceCondition is "true", else it is "false". Within a TestAssertion
 1864 Operation, content of the node-list can be further examined through the VerifyContent (content
 1865 evaluation) or ValidateContent (content format evaluation).



1869 Figure 36 – Graphic representation of expanded view of the TestAssertion element

1870

1871

1872 **Definition of Content**

1873

Name	Description	Default Value From Test Driver	Required/Optional
description	Metadata describing the nature of the TestPreCondition operation		Required
requirement	Metadata describing whether this Conformance Condition MUST or MAY exist (for use in test reporting)		Optional
VerifyContent	Use XPath expression to evaluate content of message(s)		Optional
verifyMethod	Either XPath or md5-5 (for message or payload integrity test)		

ValidateContent	Empty if entire XML document is to be validated or XPath expression to “point to” content to be validated for correct format if type is URI, dateTIme or Signature		Optional
contentType	An enumerated list of XML, URI, dateTIme, or signature validation descriptors		Optional
schemaLocation	URI describing location of validating XML schema, as defined in [XMLSCHEMA]		Optional

1875

1876 **Semantics of the TestAssertion operation**

1877

1878 The TestAssertion operation MUST return either a true or false result (or semantically a pass/fail result)
 1879 to the Test Driver.

1880

1881 If TestAssertion includes a VerifyContent sub-operation, the VerifyContent operation MUST yield a
 1882 boolean value of true/false, regardless of the resulting XPath object yielded by the XPath expression. The
 1883 VerifyContent XPath expression may yield a node-set, boolean, number or string object. All of these
 1884 resulting objects MUST be evaluated using the “boolean” function described in [XPath]. Those evaluation
 1885 rules are:

1886

- 1887 • a returned node-set object evaluates to true if and only if it is non-empty
- 1888 • a returned boolean object evaluates to true if it evaluates to “true” and false if it evaluates to
 1889 “false”
- 1890 • a returned number object evaluates to true if and only if it is neither positive or negative zero nor
 1891 NaN
- 1892 • a returned string object evaluates to true if and only if its length is non-zero

1893

1894

1895

1896 If TestAssertion includes a ValidateContent sub-operation, the ValidateContent operation MUST yield a
 1897 boolean value of true/false. Rules for determining the resulting Boolean value are:
 1898

- 1899 • if the contentType attribute value is XML, as defined in [XML], the operation evaluates to true if
 1900 the content at the specified XPath validates according to the schema defined in the
 1901 “schemaLocation” attribute
- 1902 • if the contentType is URI, as defined in [XMLSCHEMA], the operation evaluates to true if the
 1903 content at the specified XPath is a valid URI
- 1904 • if the contentType is dateTIme, as defined in [XMLSCHEMA], the operation evaluates to true if
 1905 the content at the specified XPath is a valid dateTIme
- 1906 • if the contentType is signature, as defined in [XMLDSIG], the operation evaluates to true if the
 1907 content at the specified XPath is a validly signed

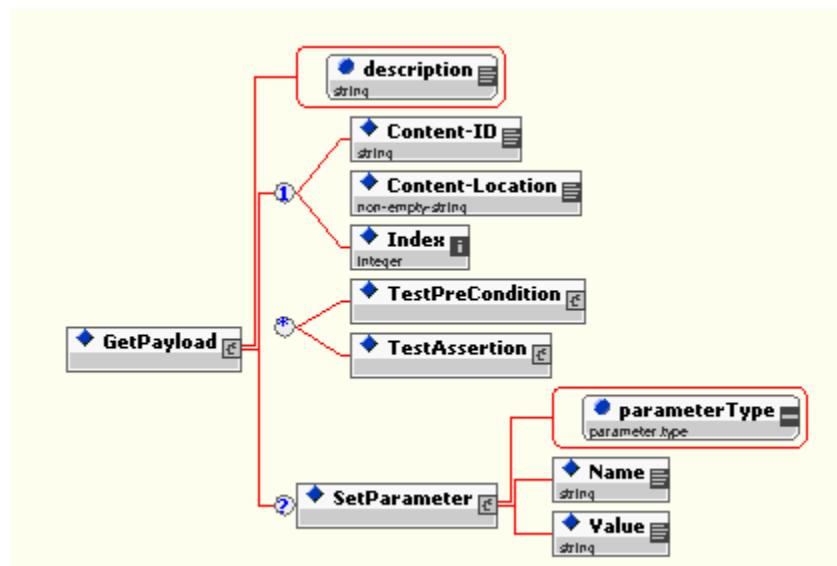
1908

7.1.10 The GetPayload Operation

1910

1911 The GetPayload operation fetches the message payload from the current message retrieved with the
1912 GetMessage operation. The message payload is retrieved based upon the required Content-ID, Content-
1913 Location or Index child element value. As with the MessageHeader, both PreCondition and TestAssertion
1914 operations can be performed on the message payload. Payload content can be verified (using the
1915 VerifyContent operation described above) using an XPath expression or by comparing its digest value
1916 against the value stored for its matching Content-Id in the current ConfigurationGroup. If the payload is
1917 an XML document, the entire document can be validated (using the ValidateContent operation described
1918 above) against the provided XML schema, or a discreet element or attribute value can be validated as a
1919 URI or dateTIme.

1920



1921

1922 Figure 37 – Graphic representation of expanded view of the GetPayload element

1923

1924 Definition of Content

1925

Name	Description	Default Value from Test Driver	Required/Optional
GetPayload	Container element for operations to verify and validate message content		Optional
description	Data describing the nature of the GetPayload operation		Required
ContentId	Retrieve the payload using the MIME Content-ID header value	false	Optional
ContentLocation	Retrieve the payload using the MIME Content-Location value		Optional
Index	Retrieve the payload as the Nth attachment after the message envelope		Required

TestPreCondition	Container for verification or validation operation to be performed on message as pre-condition to testing the Assertion		Optional
TestAssertion	Container for verification or validation operation to be performed on message as a test of the Assertion		Optional
SetParameter	Container for user-defined parameter to be made available to other Test Steps (source can be this payload's XML content retrieved via XPath statement or simply a user-supplied string value)		Optional
parameterType	Choice of xpath or string		Required
Name	Name of new parameter		Required
Value	String value or XPath expression pointing to desired element/attribute value in payload, or simply a user-supplied string value		Required

1926

1927

1928 **Semantics of the GetPayload operation**

1929

1930 Although message payloads are not stored in the MessageStore, the Test Driver MUST be able to
 1931 retrieve them for verification or validation through their corresponding Content-ID, Content-Type or Index.
 1932 How the message payloads are stored by the Test Driver is implementation dependent. Unlike the
 1933 GetMessage operation, which can evaluate multiple messages, the GetPayload operation can only
 1934 perform a TestPreCondition or TestAssertion operation on a single payload, in a single message. That
 1935 message is the "current" message retrieved by the parent GetMessage operation. If more than one
 1936 message is retrieved using the GetMessage operation, the GetPayload operation will generate an
 1937 exception, and the GetPayload operation will return a "fail" result. The Test Step will return a "fail" result
 1938 to its parent TestCase. The failed TestCase result will be logged in the Test Report, with a failure cause
 1939 of "undetermined".

1940

1941 All other rules regarding the TestPreCondition or TestAssertion operations previously described apply to
 1942 the GetPayload operation as well.

1943

1944 In addition, new parameters that can be used by other Test Steps in this Test Case may be created using
 1945 the SetParameter sub-operation. The new parameter can be used in other XPath expressions as an
 1946 XPath parameter value. The SetParameter operation can select a discreet element or attribute value of
 1947 the current payload, using the XPath expression supplied in the "Value" sub-element. Or the
 1948 SetParameter operation can define a new parameter through a simple user-supplied string value
 1949 assignment to the parameter.

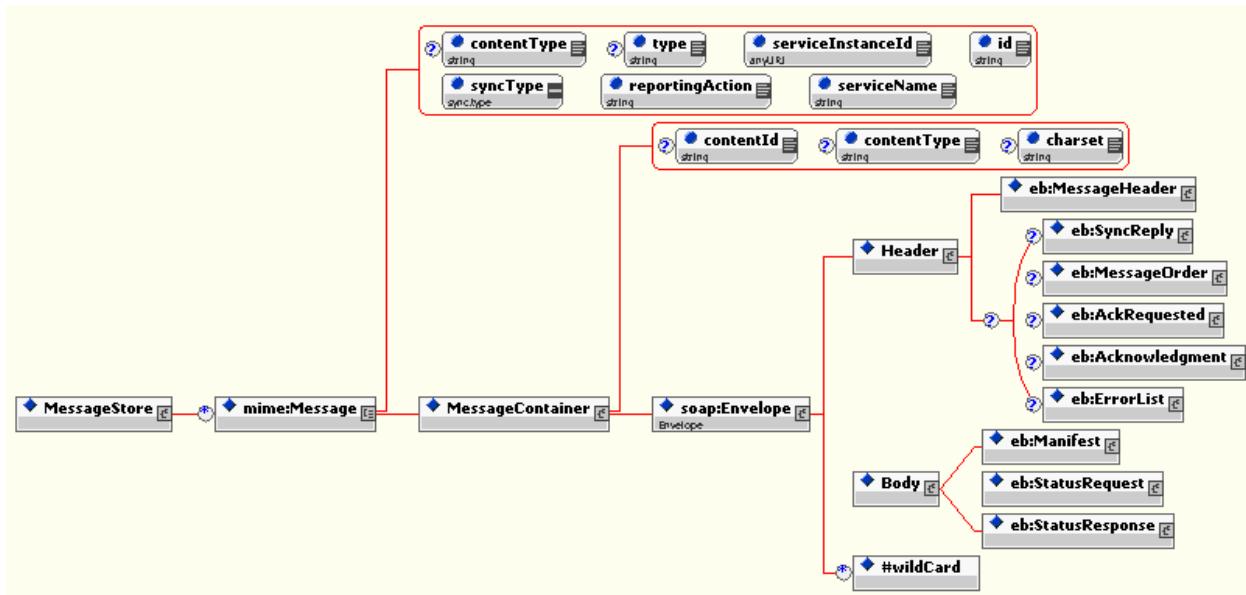
1950

1951 **7.1.11 Message Store Schema**

1952

1953 The Message Store schema (Appendix D) describes the XML document format required for a Test Driver
 1954 implementation. The schema facilitates a standard XPath query syntax to be used for retrieval and
 1955 evaluation of received messages by the Test Driver

1956



1957

1958 172-2

1959

1960 Figure 38 – Graphic representation of expanded view of the Test Driver MessageStore schema

1961

1962

1963 Definition of Content

1964

Name	Declaration Description	Default Value From Test Driver	Required/Optional
MessageStore	Container for XML representation of all messages received by Test Driver for a given Test Case		Required
mime:Message	Container for MIME, SOAP and ebXML message content		Optional
contentType	MIME message 'Content-Type' header		Optional
type	MIME message 'type' header		Optional
serviceInstanceId	Unique identifier for instance of Test Service that reports the message		Required
reportingAction	Action name that received the message on reporting service		Required
id	Unique identifier for this message		Required
syncType	Classifier of "synchronous" or "asynchronous"		Required
serviceName	Name of service that received the message		Required

MessageContainer	MIME SOAP messagecontainer		Required
contentId	SOAP container ‘Content-ID’ header		Optional
contentType	SOAP message package ‘Content-Type’ header		Optional
charset	SOAP message package character set		Optional
soap:Envelope	Generates container for SOAP message		Required

1965

1966 NOTE: All ebXML MessageStore content contained in the SOAP envelope MUST validate to the [ebMS]
1967 schema definition.

1968

1969 7.1.12 Service-Specific Message Payloads

1970

1971 The Message Payload schema (Appendix E) facilitates a standard syntax for messages sent and
1972 received by Test Service Actions that facilitates unambiguous interpretation of directives for the
1973 appropriate Action. Each possible request or response is an ebXML message payload that directs the
1974 appropriate Action defined in the ebXML MessageHeader to perform a particular function. All of the Test
1975 Framework Messages below MUST be the first payload encountered after the SOAP envelope. These
1976 Test Framework Service-Specific message payloads include:

1977

- InitiatorRequest – XML payload content to be interpreted by the “Initiator” action to construct an ebXML Message

1980

- PayloadVerifyResponse – XML payload content to be interpreted by the “mute” action of a Test Service. It consists of a list of Manifest “href” values, and the corresponding Boolean value indicating whether the payload(s) received by the PayloadVerify operation passed (“true”) or failed (“false”).

1985

1986

- ConfigurationRequest – XML payload content to be interpreted by the “Configurator” action of a Test Service. It consists of a ConfigurationRequest “root” element, with a configurationAction option of “query” or “replace”. The default value is “replace” and indicates that configuration content for this request MUST replace the existing configuration. A configurationAction value of “query” will result in a ConfigurationResponse that only echoes back the current configuration properties of the Test Service.

1993

- ConfigurationResponse – XML payload content to be interpreted by the “mute” action of a Test Service. It consists of a ConfigurationResponse “root” element. Current CPAId, Test Service Mode, Response URL and payload digest values are returned in the XML payload.

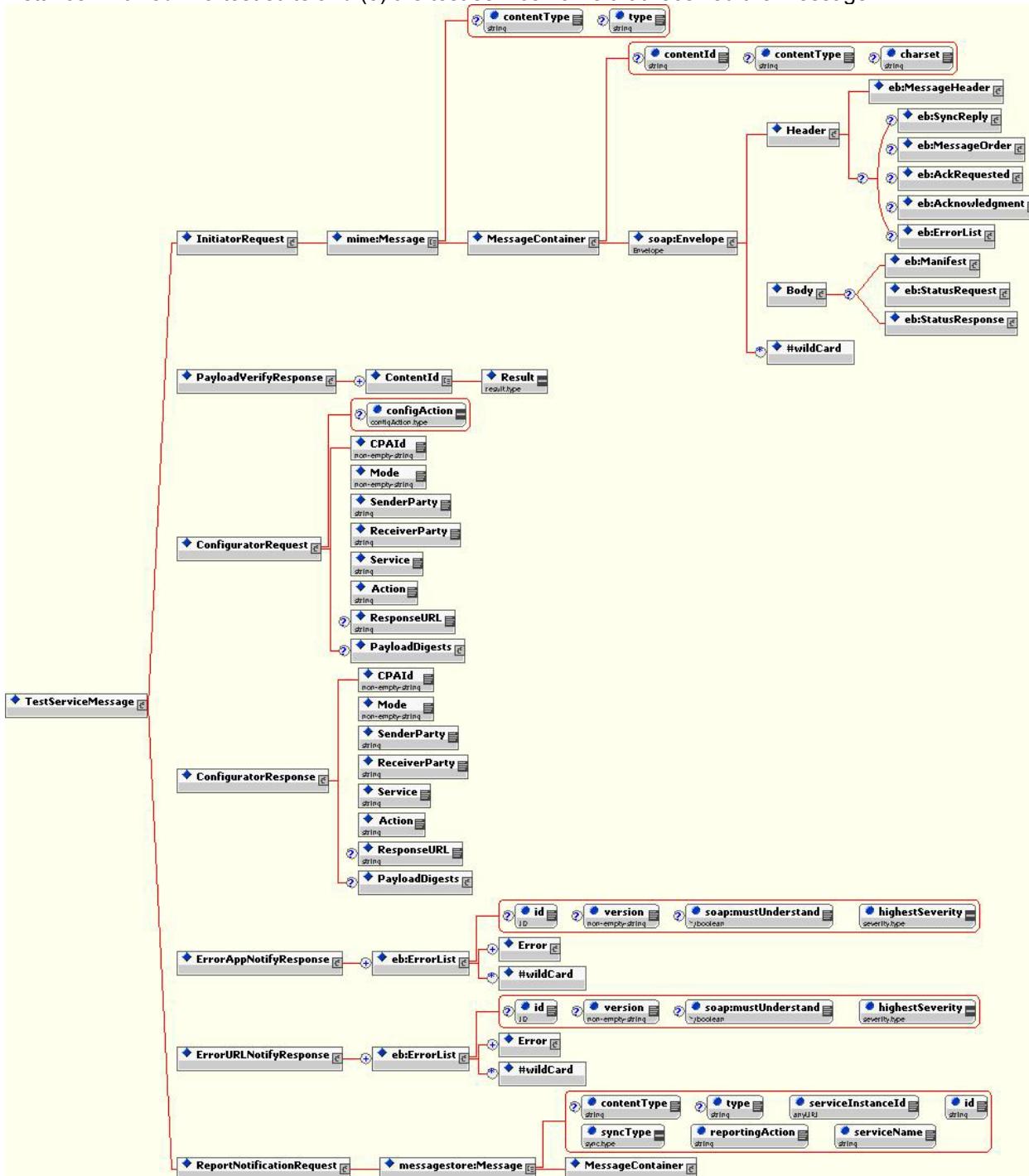
1997

- ErrorAppNotifyResponse – XML payload describing (using ebXML Errorlist format as defined in [EBMS]) application-level errors generated by the ErrorAppNotify action of the Test Service.

2000

- 2001 • ErrorURLNotifyResponse – XML payload describing (using ebXML Errorlist format as defined in
2002 [EBMS]) application errors generated by the ErrorURLNotify action of the Test Service.
2003
- 2004 ReportNotificationRequest – XML payload describing (using the ebXML MessageStore format defined in
2005 section 7.1.11 above) a received message. Such a message payload is followed by any additional
2006 message payloads that are part of the originally received message. Because MIME message information
2007 is not available to a Test Driver in Service Mode, only Manifest reference information (with payload
2008 Content-ID or Content-Location) will be present in the notification message.
- 2009 NOTE: The ReportNotificationRequest contains three additional values besides each message item, that
2010 are obtained by Test Driver when coupled with a Test Service in reporting mode: (1) the name of the
2011 action which passed the message to test driver, (2) the test service "instancId", which will identify each

2012 instance involved in a test suite and (3) the test service name that received the message.



2013

2014 Figure 39 – Graphic representation of expanded view of the Test Driver MessagePayload.xsd schema

2015

2016 Definition of Content

2017

Name	Description	Default Value	Required/Optional
------	-------------	---------------	-------------------

TestServiceMessage	Container element for all possible test messages		Required
instanceId	Unique identifier for Test Service instance		Required
InitiatorRequest	Root element containing minimal content required for a candidate MSH to initiate a new message conversation with the Test Driver		Optional
eb:MessageHeader	ebXML Message Header declaration, as defined in section 7.1.4		Required
eb:SyncReply, eb:MessageOrder, eb:AckRequested, eb:Acknowledgment, eb:Manifest, eb>StatusRequest, eb>StatusResponse	SOAP header extension element declaration, as defined in section 7.1.4		Optional
PayloadVerifyResponse	Root element containing the boolean result of a payload verification test		Optional
contentId	MIME Content-ID of message verified message payload		Required
Result	Boolean result of payload verification operation		Required
ConfiguratorRequest	Root element containing minimal content required for a candidate MSH to reconfigure itself		Optional
configAction	Modify attribute to toggle between “replace” and “query” action	replace	Optional
CPAId	Reference to CPA to be used by candidate MSH		Optional
Mode	Mode of behavior for candidate MSH		Optional
ResponseURL	URL for the Test Service to send any response messages to.		Optional
NotificationURL	URL for the Test Service to send any notification messages to.		Optional
PayloadDigests	Container for Payload descriptors and verification digests		Optional
Payload	Container for individual payload descriptor content		Required

Href	Identifier for this payload		Required
Digest	Precomputed digest value for this payload		Required
ConfiguratorResponse	Root element containing minimal content required for a candidate MSH to reconfigure itself		Optional
CPAId	Reference to CPA to be used by candidate MSH		Required
Mode	Mode of behavior for candidate MSH		Required
ResponseURL	URL for the Test Service to send any response messages to		Optional
NotificationURL	URL for the Test Service to send any notification messages to		Optional
PayloadDigests	Container for Payload descriptors and verification digests		Optional
Payload	Container for individual payload descriptor content		Required
Href	Identifier for this payload		Required
Result	Boolean value for verification result for this payload: "true" (pass) or "false" (fail)		Required
ErrorAppNotifyResponse	Container for result of ErrorAppNotify action		Optional
eb:ErrorList	Errorlist content corresponding to [EBMS] specification syntax and semantics for an ebXML error		Required
ErrorURLNotifyResponse	Container for result of ErrorAppNotify action		Optional
eb:ErrorList	Errorlist content corresponding to [EBMS] specification syntax and semantics for an ebXML error		Required
ReportNotificationREquest	Container for a persistent message to be placed in the Test Driver Message Store		Optional

2018

2019

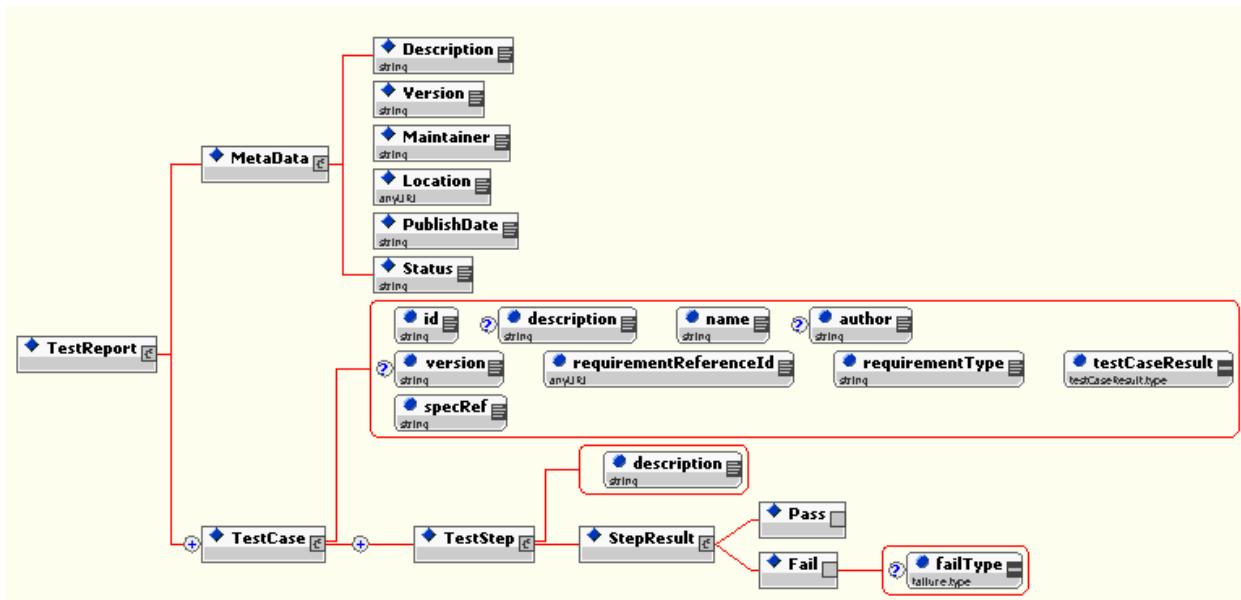
2020

2021 7.1.13 Test Report Schema

2022

2023 The Test Report schema (Appendix F) describes the XML report document format required for Test Driver
2024 implementations. The schema facilitates a standard XML syntax for reporting results of Test Cases and
2025 their Test Steps.

2026



2027

2028 Figure 40 – Graphic representation of expanded view of the Test Driver TestReport schema

2029

2030

2031 Definition of Content

2032

Name	Description	Default Value From Test Driver	Required/Optional
TestReport	Container for all Test Case results		Required
MetaData	Container for Test Suite metadata, including Description, Version, Maintainer, Location, Publish Date and Status		Required
TestCase	Container for all result data of a single Test Case		Required
id	Unique identifier for this Test Case		Required
description	Short description of TestCase		Optional
name	Short name for Test Case		Required

author	Name of person(s) creating the Test Case		Optional
version	Version number of Test Case		Optional
requirementReferenceld	Pointer to the unique ID of the Semantic Test Requirement (in Appendix E)		Required
requirementType	Type of requirement for this Test Case (REQUIRED, OPTIONAL, RECOMMENDED, STRONGLY RECOMMENDED)		Required
testCaseResult	Enumerated Pass/Fail result for entire Test Case (pass, fail or untested)		Required
specRef	Identifier to location in specification from which this Test Case is derived		Required
TestStep	Discrete step in Test Case that MUST be evaluated in a pass/fail manner		Required
description	Short description of the function of this Test Step		Required
StepResult	Container of pass/fail result data for this Test Step		Required
Pass	Indicator that this Test Step passed		Required
Fail	Indicator that this Test Step failed		Required
failType	Enumerated type indicating type of failure (preConditionTest, assertionTest or undetermined)		Required

2033
2034
2035
2036

2037 8 Test Material

2039 Test material necessary to support the ebXML Testing Framework includes:

- 2041 • A Testing Profile XML document
2042 • A Test Requirements XML document
2043 • A Test Suite XML document
2044 • A “Basic CPA” from which variants are derived for particular tests

2046 8.1.1 Testing Profile Document

2048 Both conformance and interoperability testing require the creation of a Testing Profile XML document,
2049 which lists the Test Requirements against which Test Cases will be executed. A Test Profile document
2050 MUST be included in an interoperability or conformance test suite. The Testing Profile document MUST
2051 validate against the ebXMLTestProfile.xsd schema in Appendix A.

2053 8.1.2 Test Requirements Document

2055 Both conformance and interoperability testing require the existence of a Test Requirements document.
2056 While Test Requirements for conformance testing are specific and detailed against an ebXML
2057 specification, interoperability Test Requirements may be more generic, and less rigorous in their
2058 description and in their reference to a particular portion of an ebXML specification. However, both types
2059 of testing MUST provide a Test Requirements XML document that validates against the
2060 ebXMLTestRequirements.xsd schema in Appendix B.

2062 8.1.3 Test Suite Document

2064 Both conformance and interoperability testing require the existence of a Test Suite XML document that
2065 validates against the ebXMLTestSuite.xsd schema in Appendix C. It is important to note that test case
2066 scripting inside the Test Suite document MUST take into account the test harness architecture. Although
2067 MIME and SOAP message content can be manipulated by a Test Driver in Connection Mode, such
2068 content may not be accessible by a Test Driver in Service Mode, as the MSH does not communicate this
2069 data to the application layer. Therefore, the following test scripting rules SHOULD be followed when
2070 designing Test Cases:

- 2073 • When the message material is to be sent or analyzed at by a Test Driver in Service Mode (i.e. the
2074 Test Driver acts as an application component), MIME header and SOAP content SHOULD NOT
2075 be declared (in a PutMessage operation) or queried (in a GetMessage operation). However, for
2076 the sake of uniform scripting, a Test Driver that conforms to this specification MUST accept MIME
2077 message envelope and SOAP header material defined in the declaration of the PutMessage

2078 operation (it will then ignore superfluous elements when passing the message to the Test
2079 Service). “Accepting” means: (1) when sending (e.g. via PutMessage), MIME and SOAP
2080 envelope material will be ignored when invoking the Initiator service action, (2) when receiving,
2081 any filtering condition or reference to MIME and SOAP material will be ignored, e.g. removed
2082 from the set of conditions used in a GetMessage step. In addition, a Test Driver that conforms to
2083 this specification MUST accept XPath query expressions that reference MIME and SOAP
2084 message content, even though such content MAY not be included in the MessageStore
2085 representation of a message.

- 2086
- 2087 • When the message material is to be sent or analyzed in Connection Mode (i.e. the Test Driver
2088 acts as an MSH component), MIME header and SOAP content MAY be declared (in a
2089 PutMessage operation) or queried (in a GetMessage operation). At this messaging level, all
2090 message data is accessible by the Test Driver.

2091

2092

2093 In the graphical example of the above scenarios, an [ebMS] interoperability test suite will generally require
2094 messages to be generated and received at application level (Service Mode) directly from Test Driver to
2095 Test Service as illustrated in Figure 5. In contrast, an ebMS conformance test suite which will require
2096 messages to be generated and received at transport level (Connection Mode), as illustrated in Figures 2
2097 and 3.

2098

2099 **8.1.4 Base CPA and derived CPAs**

2100

2101 Both conformance and interoperability testing require the existence of a “base CPA” configuration that
2102 describes both the Test Driver and Test Service Collaboration Protocol Profile Agreement. This is the
2103 “bootstrap” configuration for all messaging between the testing and candidate ebXML applications. How
2104 the base CPA is represented to the applications is implementation specific, however the base CPA
2105 configuration MUST be semantically equivalent to the CPA defined in Conformance or Interoperability
2106 Test Suite Specification.

2107

2108 Modified (or derived) versions of the base CPA MUST have unique CPAIds identifying them as
2109 derivations of the base CPA to both the Test Driver and Test Service. A Test Harness implementation
2110 MAY reference CPAs that are derived from the base CPA in order to perform a particular type of
2111 conformance or interoperability test. CPA’s derived from the base CPA and used in a Test Suite MUST
2112 be documented in the appropriate ebXML Conformance Test Suite or ebXML Interoperability Test Suite
2113 Specification. The unique CPAId, and the list of discreet variations from the base CPA MUST are
2114 included in the Conformance or Interoperability Test Suite document.

2117 9 Test Material Examples

- 2118
- 2119 This section includes example test material to illustrate
- 2120
- 2121 • A Test Requirements Document – Listing all Test Requirements for an ebXML implementation
 - 2122 • A Test Profile Document – Listing all selected Test Requirements to be exercised
 - 2123 • A Test Suite Document – Listing all Executable Test Cases for an ebXML implementation
- 2124

2125 9.1 Example Test Requirements

2126

2127 Below are two XML documents illustrating how Test Requirements are constructed, in this case for an
2128 ebXML MS 2.0 implementation. In this particular case, the two documents represent Conformance and
2129 Interoperability Test Requirements for an ebXML Messaging Services V2.0 implementation. The
2130 example XML documents below include a subset of testing requirements defined for implementations of
2131 the ebXML Messaging Services v2.0 Specification. Each Test Requirement may have one or more
2132 Functional Requirements that together must be satisfied in order for an implementation to fully meet that
2133 Test Requirement.

2134

2135

2136 9.1.1 Conformance Test Requirements

2137

2138 In the example below, a “packaging” TestRequirement element contains two FunctionalRequirement
2139 elements. The first Functional Requirement states that the primary SOAP message MUST be the first
2140 MIME part of the message. The second packaging Functional Requirement states that the Content-Type
2141 MIME header of the Message Package MUST be “text/xml”. If all Test Cases having a requirement
2142 reference to these two Functional Requirements “pass”, then an ebXML MS v2.0 implementation would
2143 be deemed “conformant” to the specification for the “Packaging” of ebXML messages. Of course, this is a
2144 limited set of Test Requirements for illustrative purposes only.

2145

```
2146 <?xml version="1.0" encoding="UTF-8" ?>
2147 <Requirements xmlns="http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs"
2148   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2149   xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs/
2150     ebXMLTestRequirements.xsd">
2151   <MetaData>
2152     <Description>Master Requirements File: ebXML Messaging Services 2.0</Description>
2153     <Version>1.0</Version>
2154     <Maintainer>Michael Kass<Michael.kass@nist.gov></Maintainer>
2155     <Location>http://www.oasis-open.org/committees/ebxml-
2156       iic/ebmsg/requirements1.0.xml</Location>
2157     <PublishDate>20 Feb 2003</PublishDate>
2158     <Status>DRAFT</Status>
2159     </MetaData>
2160   <!--Main Test Requirement, for message packaging-->
2161   <TestRequirement id="req_id_2" name="PackagingSpecification" specRef="ebMS-2#2.1"
2162     functionalType="packaging">
2163     <!--Define first sub-requirement to fulfill packaging testing-->
```

```

2164 <FunctionalRequirement id="funreq_id_2"
2165   name="GenerateConformantSOAPWithAttachMIMEHeaders" specRef="ebMS-2#2.1.2">
2166   <Clause>
2167     <!--Set first condition of the message is of type "multipart-mime" →
2168       <Condition id="condition_id_2" requirementType="required">For each generated message,
2169       if it is multipart MIME</Condition>
2170       <Or />
2171     <!--Set alternate condition that the message is not "text/xml" →
2172       <Condition id="condition_id_305" requirementType="required">if it is not
2173       text/xml</Condition>
2174     </Clause>
2175     <!--Define the Assertion that the first part of message is a SOAP message →
2176       <Assertion id="assert_id_2" requirementType="required">The primary SOAP message is
2177       carried in the root body part of the message.</Assertion>
2178     </FunctionalRequirement>
2179     <!--Define a second sub-requirement to fulfill packaging testing→
2180     <FunctionalRequirement id="funreq_id_4" name="GenerateCorrectMessagePackageContent-Type"
2181       specRef="ebMS-2#2.1.2">
2182     <Clause>
2183       <!--Define condition that the candidate MSH generates a message →
2184         <Condition id="condition_id_4" requirementType="required">For each generated
2185         message</Condition>
2186       </Clause>
2187       <!--Define the Assertion that the Content-Type of MIME header of that message is
2188         "text/xml" →
2189         <Assertion id="assert_id_4" requirementType="required">The Content-Type MIME header in
2190         the Message Package contains a type attribute of "text/xml".</Assertion>
2191       </FunctionalRequirement>
2192     </TestRequirement>
2193     <!--Define a new Test Requirement, for the Core Extension Elements of messaging→
2194     <TestRequirement id="req_id_3" name="CoreExtensionElements" specRef="ebMS-2#3.1.1"
2195       functionalType="packaging">
2196       <!--Define a sub-requirement to test the CPAId extension element→
2197       <FunctionalRequirement id="funreq_id_35" name="ReportFailedCPAIDResolution"
2198         specRef="ebMS-2#3.1.2">
2199       <Clause>
2200         <!--First , set condition of a candidate MSH receiving a message with an unresolvable
2201           CPAId→
2202           <Condition id="condition_id_40" requirementType="required">For each received message,
2203           if value of the CPAId element on an inbound message cannot be resolved</Condition>
2204         </Clause>
2205         <!--Next , define the Assertion that the candidate MSH MUST ( since requirementType is
2206           "required") respond with an Error→
2207           <Assertion id="assert_id_35" requirementType="required">The MSH responds with an error
2208           (ValueNotRecognized/Error).</Assertion>
2209         </FunctionalRequirement>
2210         <!--Define a sub-requirement to test continuity in message ConversationId→
2211         <FunctionalRequirement id="funreq_id_36" name="ProvideConversationIdIntegrity"
2212           specRef="ebMS-2#3.1.3">
2213         <Clause>
2214           <!--First , set condition of all messages generated by a Candidate Implementation
2215             pertaining to a single CPAId→
2216             <Condition id="condition_id_41" requirementType="required">For each generated message
2217             within the context of the specified CPAId</Condition>
2218           </Clause>
2219           <!--Next , define the Assertion that a ConversationId element is always present→
2220             <Assertion id="assert_id_36" requirementType="required">The generated ConversationId
2221             will be present in all messages pertaining to the given conversation.</Assertion>
2222           </FunctionalRequirement>
2223
2224     </TestRequirement>
2225   </Requirements>

```

2226
2227

2228 9.1.2 Interoperability Test Requirements

2229

2230 In the example below, a “basic interoperability profile” TestRequirement element contains two
2231 FunctionalRequirement elements. The first Functional Requirement states that ebXML MS
2232 implementation MUST be able to receive and send a basic ebXML message without a payload. The
2233 second packaging Functional Requirement states that an ebXML MS implementation MUST be able to
2234 process and return a simple ebXML message with one payload. If all Test Cases having a requirement
2235 reference to these two Functional Requirements “pass”, then an ebXML MS v2.0 implementation would
2236 be deemed “interoperable” to the Basic Interoperability Profile Specification for ebXML Messaging. Of
2237 course, this is a limited set of Test Requirements for illustrative purposes only.

2238

```
2239 <?xml version="1.0" encoding="UTF-8" ?>
2240 <Requirements xmlns="http://www.oasis-open.org/tc/ebxml-iic/interop/reqs"
2241   xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2242   xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/interop/reqs
2243   ebXMLTestRequirements.xsd">
2244   <MetaData>
2245     <Description>Interoperability Requirements File: ebXML Messaging Services
2246 2.0</Description>
2247     <Version>1.0</Version>
2248     <Maintainer>Michael Kass <michael.kass@nist.gov></Maintainer>
2249     <Location>http://www.oasis-open.org/committees/ebxml-
2250 iic/ebmsg/ms_2.0_interop_requirements1.0.xml</Location>
2251     <PublishDate>11 Feb 2003</PublishDate>
2252     <Status>DRAFT</Status>
2253   </MetaData>
2254   <!--Main Test Requirement, for basic interoperability testing-->
2255   <TestRequirement id="req_id_1" name="Basic Interoperability Profile" specRef="MS 2.0 BIP
2256 0.8" functionalType="basic interoperability">
2257     <!--Define first sub-requirement to fulfill basic testing, sending a "no payload"
2258 message-->
2259     <FunctionalRequirement id="funreq_id_1" name="BasicExchangeNoPayload" specRef="ebMS 2.0
2260 BIP#3.2.1">
2261       <Clause>
2262         <!--First , set condition of a candidate MSH receiving a message with no payload-->
2263         <Condition id="condition_id_1" requirementType="required">For each received ebXML
2264 message with no payload, received by the "Dummy" action</Condition>
2265       </Clause>
2266       <!--Next , define the Assertion of expected behavior for the Dummy Action-->
2267         <Assertion id="assert_id_1" requirementType="required">The message is received and
2268 processed, and a simple response message is returned</Assertion>
2269       </FunctionalRequirement>
2270     <!--Define second sub-requirement to fulfill basic testing, sending a "one payload"
2271 message-->
2272
2273     <FunctionalRequirement id="funreq_id_2" name="BasicExchangeOnePayload" specRef="ebMS 2.0
2274 BIP#3.2.2">
2275       <Clause>
2276         <!--Set condition of a candidate MSH receiving a message with one payload-->
2277         <Condition id="condition_id_2" requirementType="required">For each received ebXML
2278 message with one payload, received by the "Reflector" action </Condition>
2279       </Clause>
2280       <!--Define the Assertion of expected behavior for the Reflector Action-->
2281         <Assertion id="assert_id_2" requirementType="required">The message is received and
2282 processed, and a simple response message with the identical payload is
2283 returned</Assertion>
2284       </FunctionalRequirement>
2285     <!--Define third sub-requirement to fulfill basic testing, sending a "three payload"
2286 message-->
2287     <FunctionalRequirement id="funreq_id_3" name="BasicExchangeThreePayloads" specRef="ebMS
2288 2.0 BIP#3.2.3">
2289       <Clause>
2290         <!--Set condition of a candidate MSH receiving a message with three payloads-->
2291         <Condition id="condition_id_3" requirementType="required">For each received ebXML
2292 message with three payloads, received by the "Reflector" action</Condition>
```

```

2293   </Clause>
2294   <!--Define the Assertion of expected behavior for the Reflector Action-->
2295   <Assertion id="assert_id_3" requirementType="required">The message is received and
2296   processed, and a simple response message with the identical three payloads are
2297   returned</Assertion>
2298   </FunctionalRequirement>
2299   <!--Define third sub-requirement to fulfill basic testing, generating Error messages-->
2300   <FunctionalRequirement id="funreq id 4" name="BasicExchangeGenerateError" specRef="ebMS
2301   2.0 BIP#3.2.4">
2302   <Clause>
2303   <!--Set condition of a candidate MSH receiving an erroneous message-->
2304   <Condition id="condition_id_4" requirementType="required">For each received basic
2305   ebXML message that should generate an Error </Condition>
2306   </Clause>
2307   <!--Define the Assertion of expected behavior for the candidate MSH -->
2308   <Assertion id="assert_id_4" requirementType="required">The message is received and,
2309   the MSH returns a message to the originating party with an ErrorList and appropriate
2310   Error message </Assertion>
2311   </FunctionalRequirement>
2312   </TestRequirement>
2313 </Requirements>
```

2314
2315

2316 9.2 Example Test Profiles

2317
2318 Below are two XML documents illustrating how a Test Profile document is constructed, in this case for an
2319 ebXML MS v2.0 implementation. The example XML documents below represent a subset of test
2320 requirements to be exercised. The Test Profile document provides a list of ID references (pointers) to
2321 Test Requirements or Functional Requirements in an external Test Requirements document (see above).
2322 A Test Harness would read this document, resolve the location of the Test Requirements document, and
2323 then execute all Test Cases in the Test Suite document that point to (via ID reference) the Test
2324 Requirements listed below. Note that a Test Driver can execute Test Cases pointing to a Functional
2325 Requirement (discreet requirement) or a Test Requirement (a container of a group of Functional
2326 Requirements). If the TestRequirementRef id attribute value points to a Test Requirement, then all Test
2327 Cases for all child Functional Requirements will be executed by the Test Harness (This is a way to
2328 conveniently execute a cluster of Test Cases by specifying a single Test Requirement.). This method is
2329 used for both conformance and interoperability testing.
2330

2331 9.2.1 Conformance Test Profile Example

2332
2333 The Test Profile document below would be used to drive a Test Harness, by executing all Test Cases that
2334 point (via ID) to the listed Test Requirement references (including individual Functional Requirements and
2335 a single Test Requirement listed in the above example Conformance Test Requirements document.
2336

```

2337 <?xml version="1.0" encoding="UTF-8" ?>
2338 <TestProfile xmlns="http://www.oasis-open.org/tc/ebxml-iic/test-profile"
2339   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
2340   open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-
2341   profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml"
2342   name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing
2343   profile for ebXML MS v2.0 implementations">
2344   <TestRequirementRef id="funreq_id_2"/> <!--Execute all Test Cases that reference the
2345   Basic SOAP message structure Functional Requirement-->
```

```
2346 <TestRequirementRef id="funreq_id_4" /> <!--Execute all Test Cases that reference Message  
2347 Packaeg Content Type Functional Requirement-->  
2348 <TestRequirementRef id="req_id_2" /> <!--Execut all Test Cases that reference all  
2349 Functional Requirements within the Core Extension Elements Test Requirement-->  
2350 </TestProfile>
```

2351

2352 **9.2.2 Interoperability Test Profile Example**

2353

2354

2355 The Test Profile document below would be used to drive a Test Harness, by executing all Test Cases that
2356 point (via ID) to the listed Test Requirement references (including individual Functional Requirements
2357 and a single Test Requirement listed in the above example Interoperability Test Requirements document.

2358

```
2359 <?xml version="1.0" encoding="UTF-8" ?>  
2360 <TestProfile xmlns="http://www.oasis-open.org/tc/ebxml-iic/test-profile"  
2361 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-  
2362 open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-  
2363 profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml"  
2364 name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing  
2365 profile for ebXML MS v2.0 implementations">  
2366 <TestRequirementRef id="funreq_id_1.1" /> <!--Execute all Test Casses that reference the  
2367 "Basic Exchange, No Payload" Functional Requirement-->  
2368 <TestRequirementRef id="funreq_id_1.2" /> <!--Execute all Test Casses that reference the  
2369 "Basic Exchange, One Payload" Functional Requirement-->  
2370 </TestProfile>
```

2371

2372

2373

2374 **9.3 Example Test Suites**

2375

2376 Below are two XML documents illustrating how Test Cases are constructed, in this case for testing an
2377 ebXML MS v2.0 implementation. Each Test Case has a required "requirementReferenceld" attribute,
2378 pointing to a Functional Requirement in the Test Requirements document. A Test Driver executes all
2379 Test Cases in this document that have a requirementReferenceld value matching the particular Semantic
2380 Test Requirement being exercised.

2381

2382 **9.3.1 Conformance Test Suite**

2383

2384

2385 In the example below, a series of four Test Cases make up a Test Suite. A Test Driver executing
2386 conformance Test Cases operates in "connection" mode, meaning it is not interfaced to any MSH, and is
2387 acting on its own. Each Test Case exercises a Functional Requirement listed in section 10.1 The Test
2388 Cases below do the following:

2389

- 2390 • Send a message and elicit a response message that is verified as a SOAP message

- 2391 • Verifies that an elicited response message content type is “text/xml”
 2392 • Verifies that an ebXML Error is returned in a response message when an unresolvable CPAId is
 2393 received
 2394 • Verifies that the ConversationId element is present in a simple response message
- 2395
- 2396

```

2397 <?xml version="1.0" encoding="UTF-8" ?>
2398 <!--
2399   EbXML Messaging v2 Conformance Test Suite Sample Instance File.
2400   Michael Kass <michael.kass@nist.gov>.
2401   Date: 02/20/03
2402   --
2403
2404   <!-Define Test Suite, with configuration reference to "bootstrap" Test Driver→
2405   <ebTest:TestSuite ebTest:configurationGroupRef="cpa_basic" xmlns:ebTest="http://www.oasis-
2406   open.org/tc/ebxml-iic/tests" xmlns>xpath="http://www.oasis-open.org/tc/ebxml-iic/xpath"
2407   xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
2408   xmlns:soap="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
2409   xmlns:eb="http://www.oasis-open.org/tc/ebxml-iic/tests/eb" xmlns:tns="http://www.oasis-
2410   open.org/tc/ebxml-iic/tests/tns" xmlns:xlink="http://www.oasis-open.org/tc/ebxml-
2411   iic/tests/xlink" xmlns:cfg="http://www.oasis-open.org/tc/ebxml-iic/tests/config"
2412   xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
2413   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
2414   open.org/tc/ebxml-iic/tests ebXMLTestSuite.xsd">
2415   <!-Document Test Suite for later use in Test Reporting→
2416   <ebTest:MetaData>
2417     <ebTest:Description>Conformance Test Suite File: ebXML Messaging Services
2418     2.0</ebTest:Description>
2419     <ebTest:Version>1.0</ebTest:Version>
2420     <ebTest:Maintainer>Michael Kass <michael.kass@nist.gov></ebTest:Maintainer>
2421     <ebTest:Location>http://www.oasis-open.org/committees/ebxml-
2422     iic/ebmsg/conf\_testsuite1.0.xml</ebTest:Location>
2423     <ebTest:PublishDate>12 February 2003</ebTest:PublishDate>
2424     <ebTest:Status>DRAFT</ebTest:Status>
2425   </ebTest:MetaData>
2426   <!-Define basic "bootstrap" configuration data for this Test Suite →
2427   <ebTest:ConfigurationGroup ebTest:id="cpa_basic">
2428     <ebTest:CPAId>cpa_basic</ebTest:CPAId>
2429     <ebTest:Mode>connection</ebTest:Mode>
2430     <ebTest:SenderParty>urn:oasis:iic:testdriver</ebTest:SenderParty>
2431     <ebTest:ReceiverParty>urn:oasis:iic:testservic</ebTest:ReceiverParty>
2432     <ebTest:Service>urn:ebXML:iic:test.</ebTest:Service>
2433     <ebTest:Action>Dummy</ebTest:Action>
2434   </ebTest:ConfigurationGroup>
2435   <!-Define Test Case, referencing corresponding Functional Requirement in Conformance Test
2436   Requirements XML document →
2437   <ebTest:TestCase ebTest:requirementReferenceId="urn:Funreq:2" ebTest:id="urn:TestCase:id:2"
2438   ebTest:description="SOAP message must be in root part of MIME message">
2439     <ebTest:TestStep ebTest:mode="connection">
2440       <ebTest:PutMessage ebTest:description="Send basic message header">
2441         <ebTest:MessageDeclaration> <!—Declare a basic ebXML message →
2442           <mime:Message>
2443             <mime:MessageContainer>
2444               <soap:Envelope>
2445                 <soap:Header>
2446                   <eb:MessageHeader> <!—Declare MessageHeader, using default ConfigurationGroup parameter values
2447                   for Action and CPAId ( Dummy Action, basic CPA )→
2448
2449                     </eb:MessageHeader>
2450                   </soap:Header>

```

```

2451     </soap:Envelope>
2452     </mime:MessageContainer>
2453     </mime:Message>
2454     </ebTest:MessageDeclaration>
2455     </ebTest:PutMessage>
2456     </ebTest:TestStep>
2457     <ebTest:TestStep >
2458     <ebTest:GetMessage ebTest:description="Correlate returned message">
2459     <!--Filter returned messages from MessageStore, using RefToMessageId to find the desired message →
2460     <ebTest:Filter>eb:CPAId='cpa_basic' and eb:ConversationebTest:id=$ConversationId and
2461 eb:MessageData/eb:RefToMessageebTest:id=$RefToMessageId</ebTest:Filter>
2462     <ebTest:TestAssertion ebTest:description="Verify that an SOAP Message is found in the root
2463 part of the MIME message">
2464     <!--Use XPath to verify correct message structure-->
2465     →<ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Envelope]</ebTest:
2466 VerifyContent>
2467     </ebTest:TestAssertion>
2468     </ebTest:GetMessage>
2469     </ebTest:TestStep>
2470     </ebTest:TestCase>
2471     <!--Define Test Case, referencing corresponding Functional Requirement in Conformance
2472 Test Requirements XML document -->
2473     <ebTest:TestCase ebTest:requirementReferenceId="urn:funreq:4" ebTest:id="urn:TestCase:id:4"
2474 ebTest:description="Message package Content-Type is text/xml">
2475     <ebTest:TestStep >
2476     <ebTest:PutMessage ebTest:description="Send basic message header">
2477     <ebTest:MessageDeclaration> <!--Declare a basic ebXML message →
2478     <mime:Message>
2479     <mime:MessageContainer
2480     <soap:Envelope>
2481     <soap:Header>
2482     <eb:MessageHeader><!--Declare MessageHeader, using default ConfigurationGroup parameter values
2483 for Action and CPAId ( Dummy Action, basic CPA )→
2484
2485     </eb:MessageHeader>
2486     </soap:Header>
2487     </soap:Envelope>
2488     </mime:MessageContainer>
2489     </mime:Message>
2490     </ebTest:MessageDeclaration>
2491     </ebTest:PutMessage>
2492     </ebTest:TestStep>
2493     <ebTest:TestStep >
2494     <ebTest:GetMessage ebTest:description="Correlate returned message">
2495     <!--Filter returned messages from MessageStore, using RefToMessageId to find the desired message →
2496     <ebTest:Filter>eb:CPAId='cpa_basic' and eb:ConversationebTest:id=$ConversationId and
2497 eb:MessageData/eb:RefToMessageebTest:id=$RefToMessageId</ebTest:Filter>
2498     <ebTest:TestAssertion ebTest:description="Verify message package Content-type">
2499     <!--Use XPath to verify that MIME Content-Type is "text/xml" →
2500     <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[0] and (@Content-Type =
2501 'text/xml')]</ebTest:VerifyContent>
2502     </ebTest:TestAssertion>
2503     </ebTest:GetMessage>
2504     </ebTest:TestStep>
2505     </ebTest:TestCase>
2506     <!--Define Test Case, referencing corresponding Functional Requirement in Conformance Test
2507 Requirements XML document →
2508     <ebTest:TestCase ebTest:requirementReferenceId="Funreq:35" ebTest:id="urn:TestCase:id:35"
2509 ebTest:description="If CPAId cannot be resolved, respond with ValueNotRecognized Error">
2510     <ebTest:TestStep >
2511     <ebTest:PutMessage ebTest:description="Declare message with and CPAId set to 'null'">
2512     <ebTest:MessageDeclaration>
2513     <mime:Message>
```

```

2514 <mime:MessageContainer>
2515   <soap:Envelope>
2516     <soap:Header>
2517       <!--Override default configuration value for CPAId in ebXML MessageHeader element and override the
2518       default CPAId, using instead an unresolvable CPAId -->
2519     <eb:MessageHeader>
2520
2521       <eb:CPAId>null</eb:CPAId>
2522     </eb:MessageHeader>
2523     </soap:Header>
2524   </soap:Envelope>
2525   </mime:MessageContainer>
2526   </mime:Message>
2527 </ebTest:MessageDeclaration>
2528 </ebTest:PutMessage>
2529 </ebTest:TestStep>
2530 <ebTest:TestStep>
2531   <ebTest:GetMessage ebTest:description="Correlate returned messages">
2532     <!--Filter returned messages from MessageStore, using RefToMessageId to find the desired message -->
2533     <ebTest:Filter>eb:CPAId='cpa_basic' and eb:ConversationebTest:id=$ConversationId and
2534       eb:MessageData/RefToMessageebTest:id=$RefToMessageId</ebTest:Filter>
2535     <!--Use XPath to verify that an Error element with the correct errorCode attribute value
2536       is present in the message -->
2537     <ebTest:TestAssertion ebTest:description="Verify that Error is returned">
2538     <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Envelope/soap:Hea
2539       der/eb:ErrorList/eb:Error[@eb:errorCode = 'ValueNotRecognized' and @eb:severity =
2540         'Error']]</ebTest:VerifyContent>
2541     </ebTest:TestAssertion>
2542     </ebTest:GetMessage>
2543     </ebTest:TestStep>
2544     </ebTest:TestCase>
2545     <!--Define Test Case, referencing corresponding Functional Requirement in Conformance Test
2546       Requirements XML document -->
2547     <ebTest:TestCase ebTest:requirementReferenceId="Funreq:36" ebTest:id="urn:TestCase:id:36"
2548       ebTest:description="ConversationId is always present">
2549     <ebTest:TestStep>
2550     <ebTest:PutMessage ebTest:description="Send basic Dummy message header">
2551     <ebTest:MessageDeclaration> <!--Declare a basic ebXML message -->
2552       <mime:Message>
2553         <mime:MessageContainer>
2554           <soap:Envelope>
2555             <soap:Header>
2556               <eb:MessageHeader>
2557
2558               </eb:MessageHeader>
2559             </soap:Header>
2560           </soap:Envelope>
2561           </mime:MessageContainer>
2562           </mime:Message>
2563         </ebTest:MessageDeclaration>
2564         </ebTest:PutMessage>
2565         </ebTest:TestStep>
2566       <ebTest:TestStep>
2567         <ebTest:GetMessage ebTest:description="Correlate returned messages">
2568           <ebTest:Filter>eb:CPAId='cpa_basic' and eb:ConversationebTest:id=$ConversationId and
2569             eb:MessageData/RefToMessageebTest:id=$RefToMessageId</ebTest:Filter>
2570           <ebTest:TestAssertion ebTest:description="Verify that Conversation Id is not present">
2571             <!--Use XPath to verify the XML content, and therefore the Assertion -->
2572           <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Envelope/soap:Head
2573             er/eb:MessageHeader/eb:ConversationId]</ebTest:VerifyContent>
2574           </ebTest:TestAssertion>
2575           </ebTest:GetMessage>
2576         </ebTest:TestStep>

```

```
2577 </ebTest:TestCase>
2578 </ebTest:TestSuite>
2579
2580
```

2581 9.3.2 Interoperability Test Suite

2582

2583 In the example below, a series of four Test Cases make up an Interoperability Test Suite. A Test Driver
2584 executing conformance Test Cases operates in “service” mode, meaning it is interfaced to a MSH. Each
2585 Test Case exercises a Functional Requirement listed in section 10.2 The Test Cases below do the
2586 following:

2587

- 2588 • Perform a basic message exchange with no message payload
- 2589 • Verify integrity of 1 payload in round-trip message transmission
- 2590 • Verify integrity of 3 payloads in round-trip message transmission
- 2591 • Perform a basic message exchange with a returned Error message

2592

```
2593 <?xml version="1.0" encoding="UTF-8" ?>
2594 <!--
2595   EbXML Messaging v2 Interop Test Suite Sample Instance File.
2596   Michael Kass <michael.kass@nist.gov>.
2597   Date: 02/20/03
2598   -->
2599   <!--Define Test Suite, with configuration reference to "bootstrap" Test Driver→
2600     <ebTest:TestSuite ebTest:configurationGroupRef="cpa basic"
2601       xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests"
2602       xmlns>xpath="http://www.oasis-open.org/tc/ebxml-iic>xpath"
2603       xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/mime"
2604       xmlns:soap="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
2605       xmlns:eb="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
2606       xmlns:tns="http://www.oasis-open.org/tc/ebxml-iic/tests/tns"
2607       xmlns:xlink="http://www.oasis-open.org/tc/ebxml-iic/tests/xlink"
2608       xmlns:cfg="http://www.oasis-open.org/tc/ebxml-iic/tests/config"
2609       xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
2610       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2611       xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/tests/ebXMLTestSuite.xsd">
2612     <!--Document Test Suite for later use in Test Reporting→
2613     <ebTest:MetaData>
2614       <ebTest:Description>Interoperability Test Suite File: ebXML Messaging Services
2615       2.0</ebTest:Description>
2616       <ebTest:Version>1.0</ebTest:Version>
2617       <ebTest:Maintainer>Michael Kass <michael.kass@nist.gov></ebTest:Maintainer>
2618       <ebTest:Location>http://www.oasis-open.org/committees/ebxml-
2619       iic/ebmsg/interop_testsuite1.0.xml</ebTest:Location>
2620       <ebTest:PublishDate>12 February 2003</ebTest:PublishDate>
2621       <ebTest:Status>DRAFT</ebTest:Status>
2622       </ebTest:MetaData>
2623     <!--Define basic "bootstrap" configuration data for this Test Suite →
2624     <ebTest:ConfigurationGroup ebTest:id="cpa_basic">
2625       <ebTest:CPAId>cpa_basic</ebTest:CPAId>
2626       <ebTest:Mode>connect</ebTest:Mode>
2627       <ebTest:SenderParty>urn:oasis:iic:testdriver</ebTest:SenderParty>
2628       <ebTest:ReceiverParty>urn:oasis:iic:testservice</ebTest:ReceiverParty>
2629       <ebTest:Service>urn:ebXML:iic:test.</ebTest:Service>
2630       <ebTest:Action>Dummy</ebTest:Action>
2631     <!--Predefine message payload MDA-5 digest values for use in payload verification Test
2632     Cases →
2633       <ebTest:PayloadDigests>
2634         <ebTest:Payload>
2635           <ebTest:Href>cid:payload_1</ebTest:Href>
2636           <ebTest:Digest>abc</ebTest:Digest>
2637         </ebTest:Payload>
```

```

2637 <ebTest:Payload>
2638   <ebTest:Href>cid:payload_2</ebTest:Href>
2639   <ebTest:Digest>def</ebTest:Digest>
2640   </ebTest:Payload>
2641 <ebTest:Payload>
2642   <ebTest:Href>cid:payload_3</ebTest:Href>
2643   <ebTest:Digest>ghi</ebTest:Digest>
2644   </ebTest:Payload>
2645   </ebTest:PayloadDigests>
2646   </ebTest:ConfigurationGroup>
2647 <!--Define "alternate" configuration data for use in particular Test Cases in this Test
2648 Suite →
2649 <ebTest:ConfigurationGroup ebTest:id="cpa_basic_no_key_info">
2650   <ebTest:CPAId>cpa_basic</ebTest:CPAId>
2651   <ebTest:Mode>connect</ebTest:Mode>
2652   <ebTest:SenderParty>urn:</ebTest:SenderParty>
2653   <ebTest:ReceiverParty>urn:</ebTest:ReceiverParty>
2654   <ebTest:Service>whatever</ebTest:Service>
2655   <ebTest:Action>whatever</ebTest:Action>
2656   </ebTest:ConfigurationGroup>
2657 <!--Declare XML Message Payload content for use in particular Test Cases in this Test
2658 Suite→
2659 <ebTest:Payload ebTest:id="payload_1">
2660   <Message name="payload_1" />
2661   </ebTest:Payload>
2662 <ebTest:Payload ebTest:id="payload_2">
2663   <Message name="payload_2" />
2664   </ebTest:Payload>
2665 <ebTest:Payload ebTest:id="payload_3">
2666   <Message name="payload_3" />
2667   </ebTest:Payload>
2668 <!--Define Test Case, referencing corresponding Functional Requirement in Interoperability
2669 Test Requirements XML document →
2670 <ebTest:TestCase ebTest:requirementReferenceId="funreq_id_1.1"
2671 ebTest:id="urn:TestCase:id:1.1" ebTest:description="Basic exchange, no payload">
2672   <ebTest:TestStep>
2673     <ebTest:PutMessage ebTest:description="Send basic message header">
2674       <ebTest:MessageDeclaration>
2675         <mime:Message>
2676           <mime:MessageContainer>
2677             <soap:Envelope>
2678               <soap:Header>
2679                 <!--Declare MessageHeader, using default ConfigurationGroup parameter values for Action
2680 and CPAId ( Dummy Action, basic CPA )→
2681               <eb:MessageHeader>
2682
2683                 </eb:MessageHeader>
2684                 </soap:Header>
2685                 </soap:Envelope>
2686                 </mime:MessageContainer>
2687                 </mime:Message>
2688               </ebTest:MessageDeclaration>
2689             </ebTest:PutMessage>
2690           </ebTest:TestStep>
2691         <ebTest:TestStep>
2692           <ebTest:GetMessage ebTest:description="Correlate returned message">
2693             <!--Filter returned messages from MessageStore, using RefToMessageId to find the desired
2694 message →
2695             <ebTest:Filter>eb:CPAId=$CPAId and eb:Conversationid=$ConversationId and
2696 eb:Action='Mute'</ebTest:Filter>
2697             <ebTest:TestAssertion ebTest:description="Verify that an ebXML message is returned">
2698               <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Envelope/soap:Header/
2699 eb:MessageHeader]</ebTest:VerifyContent>
2700             </ebTest:TestAssertion>
2701             </ebTest:GetMessage>
2702             </ebTest:TestStep>
2703           </ebTest:TestCase>
2704 <!--Define Test Case, referencing corresponding Functional Requirement in Interoperability
2705 Test Requirements XML document →

```

```

2706 <ebTest:TestCase ebTest:requirementReferenceId="funreq_id_1.2"
2707 ebTest:id="urn:TestCase:id:1.2" ebTest:description="Basic asynchronous exchange with one
2708 payload">
2709   <ebTest:TestStep >
2710     <ebTest:PutMessage ebTest:description="Send basic message header">
2711       <ebTest:MessageDeclaration>
2712         <mime:Message>
2713           <mime:MessageContainer>
2714             <soap:Envelope>
2715               <soap:Header>
2716                 <!--Override "default" Action in MessageHeader declaration -->
2717                 <eb:MessageHeader>
2718                   <eb:Action>Reflector</eb:Action>
2719                 </eb:MessageHeader>
2720               </soap:Header>
2721             <soap:Body>
2722               <!--Provide a Manifest declaration to include a reference to the payload -->
2723               <eb:Manifest>
2724                 <eb:Reference xlink:href="cid:payload_1" />
2725               </eb:Manifest>
2726             </soap:Body>
2727           </soap:Envelope>
2728           </mime:MessageContainer>
2729           </mime:Message>
2730         </ebTest:MessageDeclaration>
2731       <ebTest:SetPayload ebTest:description="Add content-id and payload to mime message">
2732         <ebTest:Content-ID>cid:payload_1</ebTest:Content-ID>
2733         <ebTest:MessageRef>payload_1</ebTest:MessageRef>
2734       </ebTest:SetPayload>
2735     </ebTest:PutMessage>
2736   </ebTest:TestStep>
2737   <ebTest:TestStep >
2738     <ebTest:GetMessage ebTest:description="Correlate returned messages">
2739       <!--Filter returned messages from MessageStore, using RefToMessageId and Action name to
2740       find the desired message -->
2741       <ebTest:Filter>eb:CPAID='cpa_basic' and eb:Conversationid=$ConversationId and
2742         eb:Action='Mute'</ebTest:Filter>
2743       <ebTest:TestAssertion ebTest:description="Check for returned payload">
2744         <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Body/eb:Manifest/eb:R
2745           eference[@xlink:href='cid:payload_1']]</ebTest:VerifyContent>
2746       </ebTest:TestAssertion>
2747       <ebTest:GetPayload ebTest:description="Find payload in message">
2748         <ebTest:Content-ID>cid:payload_1</ebTest:Content-ID>
2749       <ebTest:TestAssertion ebTest:description="Verify returned payload contents">
2750         <ebTest:VerifyContent ebTest:verifyMethod="xpath" />
2751       </ebTest:TestAssertion>
2752     </ebTest:GetPayload>
2753     </ebTest:GetMessage>
2754   </ebTest:TestStep>
2755 </ebTest:TestCase>
2756   <!--Define Test Case, referencing corresponding Functional Requirement in Interoperability
2757   Test Requirements XML document -->
2758   <ebTest:TestCase ebTest:requirementReferenceId="funreq_id_1.3"
2759 ebTest:id="urn:TestCase:id:1.3" ebTest:description="Basic exchange with three payloads">
2760     <ebTest:TestStep >
2761       <ebTest:PutMessage ebTest:description="Send basic message header">
2762         <ebTest:MessageDeclaration>
2763           <mime:Message>
2764             <mime:MessageContainer>
2765               <soap:Envelope>
2766                 <soap:Header>
2767                   <!--Override default value of Action in MessageHeader -->
2768                   <eb:MessageHeader>
2769                     <eb:Action>Reflector</eb:Action>
2770                   </eb:MessageHeader>
2771                 </soap:Header>
2772               <soap:Body>
2773               <!--Declare Manifest element with 3 payload references -->
2774               <eb:Manifest>
2775                 <eb:Reference xlink:href="cid:payload_1" />

```

```

2776 <eb:Reference xlink:href="cid:payload_2" />
2777 <eb:Reference xlink:href="cid:payload_3" />
2778 </eb:Manifest>
2779 </soap:Body>
2780 </soap:Envelope>
2781 </mime:MessageContainer>
2782 </mime:Message>
2783 </ebTest:MessageDeclaration>
2784 <ebTest:SetPayload ebTest:description="Add content-id and payload to mime message">
2785 <ebTest:Content-ID>cid:payload_1</ebTest:Content-ID>
2786 <ebTest:MessageRef>payload_1</ebTest:MessageRef>
2787 </ebTest:SetPayload>
2788 <ebTest:SetPayload ebTest:description="Add content-id and payload to mime message">
2789 <ebTest:Content-ID>cid:payload_2</ebTest:Content-ID>
2790 <ebTest:MessageRef>payload_2</ebTest:MessageRef>
2791 </ebTest:SetPayload>
2792 <ebTest:SetPayload ebTest:description="Add content-id and payload to mime message">
2793 <ebTest:Content-ID>payload_3</ebTest:Content-ID>
2794 <ebTest:MessageRef>payload_3</ebTest:MessageRef>
2795 </ebTest:SetPayload>
2796 </ebTest:PutMessage>
2797 </ebTest:TestStep>
2798 <ebTest:TestStep >
2799 <ebTest:GetMessage ebTest:description="Correlate returned messages">
2800 <ebTest:Filter>eb:CPAId='cpa_basic' and eb:Conversationid=$ConversationId and
2801 eb:Action='Mute'</ebTest:Filter>
2802 <ebTest:TestAssertion ebTest:description="Check for returned payloads reference in
2803 Manifest">
2804 <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Body/eb:Manifest[eb:R
2805 eference[@xlink:href='cid:payload_1']] and eb:Reference[@xlink:href='cid:payload_2']] and
2806 eb:Reference[@xlink:href='cid:payload_3']]</ebTest:VerifyContent>
2807 </ebTest:TestAssertion>
2808 <ebTest:GetPayload ebTest:description="Find payload in message">
2809 <ebTest:Content-ID>cid:payload_1</ebTest:Content-ID>
2810 <ebTest:TestAssertion ebTest:description="Verify returned payload contents">
2811 <ebTest:VerifyContent />
2812 </ebTest:TestAssertion>
2813 </ebTest:GetPayload>
2814 <ebTest:GetPayload ebTest:description="Find payload in message">
2815 <ebTest:Content-ID>cid:payload_2</ebTest:Content-ID>
2816 <ebTest:TestAssertion ebTest:description="Verify returned payload contents">
2817 <ebTest:VerifyContent />
2818 </ebTest:TestAssertion>
2819 </ebTest:GetPayload>
2820 <ebTest:GetPayload ebTest:description="Find payload in message">
2821 <ebTest:Content-ID>cid:payload_3</ebTest:Content-ID>
2822 <ebTest:TestAssertion ebTest:description="Verify returned payload contents">
2823 <ebTest:VerifyContent />
2824 </ebTest:TestAssertion>
2825 </ebTest:GetPayload>
2826 </ebTest:GetMessage>
2827 </ebTest:TestStep>
2828 </ebTest:TestCase>
2829 <!--Define Test Case, referencing corresponding Functional Requirement in Interoperability
2830 Test Requirements XML document -->
2831 <ebTest:TestCase ebTest:requirementReferenceId="funreq_id_1.4"
2832 ebTest:id="urn:TestCase:id:1.4" ebTest:description="Basic exchange with Error Message">
2833 <ebTest:TestStep >
2834 <ebTest:PutMessage ebTest:description="MessageHeader mustUnderstand set to 'true'">
2835 <ebTest:MessageDeclaration>
2836 <mime:Message>
2837 <mime:MessageContainer>
2838 <soap:Envelope>
2839 <soap:Header>
2840 <eb:MessageHeader>
2841 <eb:Action>Dummy</eb:Action>
2842 <eb:ExtensionElement soap:mustUnderstand="true" />
2843 </eb:MessageHeader>
2844 </soap:Header>
2845 </soap:Envelope>
2846 </mime:MessageContainer>
```

```
2847      </mime:Message>
2848    </ebTest:MessageDeclaration>
2849    </ebTest:PutMessage>
2850    </ebTest:TestStep>
2851    <ebTest:TestStep >
2852      <ebTest:GetMessage ebTest:getMultiple="true" ebTest:description="Correlate returned
2853 messages">
2854        <ebTest:Filter>eb:CPAId='cpa_basic' and eb:Conversationid=$ConversationId and
2855 eb:ErrorList</ebTest:Filter>
2856        <ebTest:TestAssertion ebTest:description="Test if Error is generated">
2857        <ebTest:VerifyContent>mime:Message[mime:MessageContainer[1]/soap:Envelope/soap:Body/soa
2858 p:Fault/soap:Code[soap:Value='MustUnderstand']]</ebTest:VerifyContent>
2859        </ebTest:TestAssertion>
2860        </ebTest:GetMessage>
2861        </ebTest:TestStep>
2862        </ebTest:TestCase>
2863      </ebTest:TestSuite>
```

2864

2865 Appendix A (Normative) The ebXML Test Profile 2866 Schema

2867 The OASIS ebXML Implementation and Interoperability Committee has provided a version of the ebXML
2868 Test Profile schema using the schema vocabulary that conforms to the W3C XML Schema
2869 Recommendation specification [XMLSchema].

```
2870 <?xml version="1.0" encoding="UTF-8" ?>
2871 <!--
2872 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
2873 -->
2874 <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.oasis-
2875 open.org/tc/ebxml-iic/test-profile" xmlns:tns="http://www.oasis-open.org/tc/ebxml-
2876 iic/test-profile">
2877 <!--
2878 $Id: TestProfile.xsd,v 1.2 2002/07/02 15:28:27 matt Exp $
2879 -->
2880 <element name="TestProfile">
2881 <complexType>
2882 <sequence>
2883   <element ref="tns:Dependency" minOccurs="0" maxOccurs="unbounded" />
2884   <element ref="tns:TestRequirementRef" maxOccurs="unbounded" />
2885 </sequence>
2886   <attribute name="requirementsLocation" use="required" type="anyURI" />
2887   <attribute name="name" use="required" type="string" />
2888   <attribute name="description" use="required" type="string" />
2889 </complexType>
2890 </element>
2891 <element name="Dependency">
2892 <complexType>
2893   <attribute name="name" use="required" type="string" />
2894   <attribute name="profileRef" use="required" type="anyURI" />
2895 </complexType>
2896 </element>
2897 <element name="TestRequirementRef">
2898   <!--
2899 To overide the conformance type of the underlying requirement ...
2900 -->
2901 <complexType>
2902 <sequence>
2903   <element name="Comment" type="string" minOccurs="0" maxOccurs="unbounded" />
2904 </sequence>
2905   <attribute name="id" use="required" type="string" />
2906   <attribute name="requirementType" use="optional" type="tns:requirement.type" />
2907 </complexType>
2908 </element>
2909 <simpleType name="requirement.type">
2910 <restriction base="string">
2911   <enumeration value="required" />
2912   <enumeration value="strongly recommended" />
2913   <enumeration value="recommended" />
2914   <enumeration value="optional" />
2915 </restriction>
2916 </simpleType>
2917 </schema>
```

2918

2919 Appendix B (Normative) The ebXML Test 2920 Requirements Schema

2921 The OASIS ebXML Implementation and Interoperability Committee has provided a version of the ebXML
2922 Test Requirements schema using the schema vocabulary that conforms to the W3C XML Schema
2923 Recommendation specification [XMLSchema].

2924

```
2925 <?xml version="1.0" encoding="UTF-8" ?>
2926     <!--
2927     Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
2928     -->
2929     <schema targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/tests"
2930         xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
2931         xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
2932         xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests"
2933         xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
2934         attributeFormDefault="unqualified" version="1.0">
2935         <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
2936             schemaLocation="xmldsig.xsd" />
2937         <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
2938             schemaLocation="mime.xsd" />
2939         <!--
2940         edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST)
2941         -->
2942         <!--
2943         <!--
2944         <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
2945             schemaLocation="xmldsig.xsd"/>
2946             -->
2947             <!--
2948             <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
2949                 schemaLocation="soap.xsd"/>
2950                 -->
2951             <element name="TestSuite">
2952                 <complexType>
2953                     <sequence>
2954                         <element ref="ebTest:MetaData" />
2955                         <element ref="ebTest:ConfigurationGroup" maxOccurs="unbounded" />
2956                         <element ref="ebTest:MessagePayload" minOccurs="0" maxOccurs="unbounded" />
2957                     <choice maxOccurs="unbounded">
2958                         <element ref="ebTest:ConfigurationGroup" minOccurs="0" />
2959                         <element ref="ebTest:TestCase" maxOccurs="unbounded" />
2960                     </choice>
2961                     </sequence>
2962                     <attribute name="configurationGroupRef" type="IDREF" use="required" />
2963                 </complexType>
2964             </element>
2965             <element name="MetaData">
2966                 <complexType>
2967                     <sequence>
2968                         <element ref="ebTest:Description" />
2969                         <element ref="ebTest:Version" />
2970                         <element ref="ebTest:Maintainer" />
2971                         <element ref="ebTest:Location" />
2972                         <element ref="ebTest:PublishDate" />
2973                         <element ref="ebTest>Status" />
2974                     </sequence>
2975                 </complexType>
2976             </element>
2977             <element name="Description" type="ebTest:non-empty-string" />
2978             <element name="Version" type="ebTest:non-empty-string" />
2979             <element name="Maintainer" type="ebTest:non-empty-string" />
2980             <element name="Location" type="anyURI" />
```

```

2981 <element name="PublishDate" type="ebTest:non-empty-string" />
2982 <element name="Status" type="ebTest:non-empty-string" />
2983 <element name="TestCase">
2984 <complexType>
2985 <sequence>
2986   <element ref="ebTest:TestStep" maxOccurs="unbounded" />
2987 </sequence>
2988   <attribute name="id" type="string" use="required" />
2989   <attribute name="description" type="string" use="required" />
2990   <attribute name="name" type="string" use="optional" />
2991   <attribute name="author" type="string" use="optional" />
2992   <attribute name="version" type="string" use="optional" />
2993   <attribute name="requirementReferenceId" type="anyURI" use="required" />
2994   <attribute name="configurationGroupRef" type="IDREF" use="optional" />
2995 </complexType>
2996 </element>
2997 <element name="TestStep">
2998 <complexType>
2999 <choice>
3000   <element ref="ebTest:PutMessage" />
3001   <element ref="ebTest:GetMessage" />
3002 </choice>
3003   <attribute name="description" type="string" use="optional" />
3004   <attribute name="configurationGroupRef" type="IDREF" use="optional" />
3005   <attribute name="repeatTimes" type="integer" default="1" />
3006   <attribute name="testStepContext" type="integer" use="optional" />
3007   <attribute name="stepDelay" type="integer" use="optional" />
3008 </complexType>
3009 </element>
3010 <element name="MessageExpression">
3011 <complexType>
3012 <sequence>
3013   <element ref="ebTest:ErrorMessage" />
3014 </sequence>
3015 </complexType>
3016 </element>
3017   <element name="ErrorMessage" type="ebTest:non-empty-string" />
3018 <element name="PutMessage">
3019 <complexType>
3020 <sequence>
3021   <element ref="ebTest:MessageDeclaration" />
3022   <element ref="ebTest:SetPayload" minOccurs="0" maxOccurs="unbounded" />
3023 <element name="DSign" minOccurs="0">
3024 <complexType>
3025 <sequence>
3026   <element ref="ds:Signature" maxOccurs="unbounded" />
3027 </sequence>
3028 </complexType>
3029 </element>
3030   <element ref="ebTest:SetParameter" minOccurs="0" />
3031 </sequence>
3032   <attribute name="description" type="string" use="required" />
3033   <attribute name="clearMessageStore" type="boolean" default="false" />
3034 </complexType>
3035 </element>
3036 <element name="GetPayload">
3037 <complexType>
3038 <sequence>
3039 <choice>
3040   <element ref="ebTest:Content-ID" />
3041   <element ref="ebTest:Content-Location" />
3042   <element ref="ebTest:Index" />
3043 </choice>
3044 <choice minOccurs="0" maxOccurs="unbounded">
3045   <element ref="ebTest:TestPreCondition" />
3046   <element ref="ebTest:TestAssertion" />
3047 </choice>
3048   <element ref="ebTest:SetParameter" minOccurs="0" />
3049 </sequence>
3050   <attribute name="description" type="string" use="required" />
3051 </complexType>
```

```

3052     </element>
3053 <element name="GetMessage">
3054 <complexType>
3055 <sequence>
3056     <element ref="ebTest:Filter" />
3057     <element ref="ebTest:TestPreCondition" minOccurs="0" maxOccurs="unbounded" />
3058     <element ref="ebTest:TestAssertion" minOccurs="0" maxOccurs="unbounded" />
3059     <element ref="ebTest:SetParameter" minOccurs="0" />
3060     <element ref="ebTest:GetPayload" minOccurs="0" maxOccurs="unbounded" />
3061 </sequence>
3062     <attribute name="description" type="string" use="required" />
3063     <attribute name="getMultiple" type="boolean" default="false" />
3064     <attribute name="testStepContext" type="integer" use="optional" />
3065 </complexType>
3066 </element>
3067     <element name="Filter" type="ebTest:non-empty-string" />
3068 <element name="SetPayload">
3069 <complexType>
3070 <sequence>
3071 <choice>
3072     <element ref="ebTest:Content-ID" />
3073     <element ref="ebTest:Content-Location" />
3074 </choice>
3075 <choice>
3076     <element ref="ebTest:FileURI" />
3077     <element ref="ebTest:PayloadRef" />
3078 </choice>
3079 <sequence minOccurs="0" maxOccurs="unbounded">
3080     <element ref="ebTest:MimeHeader" />
3081     <element ref="ebTest:MimeHeaderValue" />
3082 </sequence>
3083     <element ref="ebTest:SetParameter" minOccurs="0" />
3084 </sequence>
3085     <attribute name="description" type="string" use="required" />
3086 </complexType>
3087 </element>
3088 <element name="TestPreCondition">
3089 <complexType>
3090 <choice>
3091     <element ref="ebTest:VerifyContent" />
3092     <element ref="ebTest:ValidateContent" />
3093 </choice>
3094     <attribute name="description" type="string" use="required" />
3095 </complexType>
3096 </element>
3097 <element name="TestAssertion">
3098 <complexType>
3099 <choice>
3100     <element ref="ebTest:VerifyContent" />
3101     <element ref="ebTest:ValidateContent" />
3102 </choice>
3103     <attribute name="description" type="string" use="required" />
3104     <attribute name="requirement" type="ebTest:requirement.type" default="required" />
3105 </complexType>
3106 </element>
3107 <simpleType name="mimeHeader.type">
3108 <restriction base="NMOKEN">
3109     <enumeration value="MIMEMessageContent-Type" />
3110     <enumeration value="MIMEMessageStart" />
3111     <enumeration value="Content-Type" />
3112     <enumeration value="start" />
3113     <enumeration value="charset" />
3114     <enumeration value="type" />
3115     <enumeration value="wildcard" />
3116 </restriction>
3117 </simpleType>
3118 <simpleType name="content.type">
3119 <restriction base="NMOKEN">
3120     <enumeration value="XML" />
3121     <enumeration value="date" />
3122     <enumeration value="URI" />

```

```

3123    <enumeration value="signature" />
3124    <enumeration value="signedAck" />
3125    </restriction>
3126    </simpleType>
3127  <simpleType name="method.type">
3128    <restriction base="NMTOKEN">
3129      <enumeration value="xpath" />
3130      <enumeration value="sha-1" />
3131    </restriction>
3132  </simpleType>
3133  <simpleType name="mode.type">
3134    <restriction base="NMTOKEN">
3135      <enumeration value="service" />
3136      <enumeration value="connection" />
3137    </restriction>
3138  </simpleType>
3139  <simpleType name="messageContext.type">
3140    <restriction base="NMTOKEN">
3141      <enumeration value="true" />
3142      <enumeration value="false" />
3143    </restriction>
3144  </simpleType>
3145  <simpleType name="requirement.type">
3146    <restriction base="NMTOKEN">
3147      <enumeration value="required" />
3148      <enumeration value="stronglyrecommended" />
3149      <enumeration value="recommended" />
3150      <enumeration value="optional" />
3151    </restriction>
3152  </simpleType>
3153  <simpleType name="non-empty-string">
3154    <restriction base="string">
3155      <minLength value="1" />
3156    </restriction>
3157  </simpleType>
3158  <simpleType name="configAction.type">
3159    <restriction base="NMTOKEN">
3160      <enumeration value="query" />
3161      <enumeration value="replace" />
3162    </restriction>
3163  </simpleType>
3164  <simpleType name="action.type">
3165    <restriction base="NMTOKEN">
3166      <enumeration value="reset" />
3167      <enumeration value="modify" />
3168    </restriction>
3169  </simpleType>
3170  <simpleType name="configItem.type">
3171    <restriction base="NOTATION">
3172      <enumeration value="parameter" />
3173      <enumeration value="namespace" />
3174    </restriction>
3175  </simpleType>
3176  <simpleType name="parameter.type">
3177    <restriction base="NMTOKEN">
3178      <enumeration value="xpath" />
3179      <enumeration value="string" />
3180    </restriction>
3181  </simpleType>
3182  <element name="MimeType" type="ebTest:mimeType" />
3183  <element name="MimeTypeValue" type="ebTest:non-empty-string" />
3184  <element name="Content-Location" type="ebTest:non-empty-string" />
3185  <element name="Index" type="integer" />
3186  <element name="FileURI" type="anyURI" />
3187  <element name="PayloadRef" type="string" />
3188  <element name="Signature" type="base64Binary" />
3189  <element name="Content-ID" type="string" />
3190  <element name="MessageDeclaration">
3191    <complexType>
3192      <sequence>
3193        <element ref="mime:Message" />

```

```

3194      </sequence>
3195    </complexType>
3196  </element>
3197  <element name="ValidateContent">
3198    <complexType>
3199    <simpleContent>
3200      <extension base="ebTest:non-empty-string">
3201        <attribute name="contentType" type="ebTest:content.type" default="XML" />
3202        <attribute name="schemaLocation" type="anyURI" use="optional" />
3203      </extension>
3204    </simpleContent>
3205  </complexType>
3206  </element>
3207  <element name="VerifyContent">
3208    <complexType>
3209    <simpleContent>
3210      <extension base="string">
3211        <attribute name="verifyMethod" type="ebTest:method.type" default="xpath" />
3212      </extension>
3213    </simpleContent>
3214  </complexType>
3215  </element>
3216  <element name="MessagePayload">
3217    <complexType>
3218    <sequence>
3219      <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3220    </sequence>
3221    <attribute name="id" type="ID" use="required" />
3222  </complexType>
3223  </element>
3224  <element name="ConfigurationGroup">
3225    <complexType>
3226    <sequence>
3227      <element ref="ebTest:CPAId" />
3228      <element ref="ebTest:Mode" />
3229      <element ref="ebTest:SenderParty" />
3230      <element ref="ebTest:ReceiverParty" />
3231      <element ref="ebTest:Service" />
3232      <element ref="ebTest:Action" />
3233      <element ref="ebTest:StepDelay" />
3234      <element ref="ebTest:ResponseURL" minOccurs="0" />
3235      <element ref="ebTest:NotifcationURL" minOccurs="0" />
3236      <element name="PayloadDigests" minOccurs="0">
3237        <complexType>
3238        <sequence>
3239          <element name="Payload" maxOccurs="unbounded">
3240            <complexType>
3241              <sequence>
3242                <element name="Href" type="anyURI" />
3243                <element name="Digest" type="base64Binary" />
3244              </sequence>
3245            </complexType>
3246          </element>
3247        </sequence>
3248      </complexType>
3249    </element>
3250    <element name="ConfigurationItem" minOccurs="0" maxOccurs="unbounded">
3251      <complexType>
3252      <sequence>
3253        <element name="Name" type="ebTest:non-empty-string" />
3254        <element name="Value" type="ebTest:non-empty-string" />
3255        <element name="Type" type="ebTest:configItem.type" />
3256      </sequence>
3257    </complexType>
3258  </element>
3259  </sequence>
3260  <attribute name="id" type="ID" use="required" />
3261  </complexType>
3262  </element>
3263  <element name="SenderParty" type="ebTest:non-empty-string" />
3264  <element name="ReceiverParty" type="ebTest:non-empty-string" />

```

```
3265 <element name="Service" type="anyURI" />
3266 <element name="Action" type="ebTest:non-empty-string" />
3267 <element name="StepDelay" type="integer" />
3268 <element name="ResponseURL" type="anyURI" />
3269 <element name="NotifcationURL" type="anyURI" />
3270 <element name="CPAId" type="ebTest:non-empty-string" />
3271 <element name="Mode" type="ebTest:non-empty-string" />
3272 <element name="SetParameter">
3273 <complexType>
3274 <sequence>
3275 <element name="Name" type="string" />
3276 <element name="Value" type="string" />
3277 </sequence>
3278 <attribute name="parameterType" type="ebTest:parameter.type" use="required" />
3279 </complexType>
3280 </element>
3281 </schema>
3282
```

3283

3284 Appendix C (Normative) The ebXML Test Suite 3285 Schema and Supporting Subschemas

3286 The OASIS ebXML Implementation and Interoperability Committee has provided a version of the ebXML
3287 Test Requirements schema using the schema vocabulary that conforms to the W3C XML Schema
3288 Recommendation specification [XMLSchema].

3289

```
3290 <?xml version="1.0" encoding="UTF-8" ?>
3291 <!--
3292 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
3293 -->
3294 <schema targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/tests"
3295 xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
3296 xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3297 xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests"
3298 xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
3299 attributeFormDefault="unqualified" version="1.0">
3300   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3301   schemaLocation="xmldsig.xsd" />
3302   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
3303   schemaLocation="mime.xsd" />
3304   <!--
3305   edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST)
3306   -->
3307   <!--
3308   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3309   schemaLocation="xmldsig.xsd"/>
3310   -->
3311   <!--
3312   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3313   schemaLocation="soap.xsd"/>
3314   -->
3315   <element name="TestSuite">
3316     <complexType>
3317       <sequence>
3318         <element ref="ebTest:MetaData" />
3319         <element ref="ebTest:ConfigurationGroup" maxOccurs="unbounded" />
3320         <element ref="ebTest:MessagePayload" minOccurs="0" maxOccurs="unbounded" />
3321       <choice maxOccurs="unbounded">
3322         <element ref="ebTest:ConfigurationGroup" minOccurs="0" />
3323         <element ref="ebTest:TestCase" maxOccurs="unbounded" />
3324       </choice>
3325       </sequence>
3326       <attribute name="configurationGroupRef" type="IDREF" use="required" />
3327     </complexType>
3328   </element>
3329   <element name="MetaData">
3330     <complexType>
3331       <sequence>
3332         <element ref="ebTest:Description" />
3333         <element ref="ebTest:Version" />
3334         <element ref="ebTest:Maintainer" />
3335         <element ref="ebTest:Location" />
3336         <element ref="ebTest:PublishDate" />
3337         <element ref="ebTest>Status" />
3338       </sequence>
3339     </complexType>
3340   </element>
3341   <element name="Description" type="ebTest:non-empty-string" />
3342   <element name="Version" type="ebTest:non-empty-string" />
3343   <element name="Maintainer" type="ebTest:non-empty-string" />
3344   <element name="Location" type="anyURI" />
3345   <element name="PublishDate" type="ebTest:non-empty-string" />
3346   <element name="Status" type="ebTest:non-empty-string" />
```

```

3347 <element name="TestCase">
3348   <complexType>
3349     <sequence>
3350       <element ref="ebTest:TestStep" maxOccurs="unbounded" />
3351     </sequence>
3352     <attribute name="id" type="string" use="required" />
3353     <attribute name="description" type="string" use="required" />
3354     <attribute name="name" type="string" use="optional" />
3355     <attribute name="author" type="string" use="optional" />
3356     <attribute name="version" type="string" use="optional" />
3357     <attribute name="requirementReferenceId" type="anyURI" use="required" />
3358     <attribute name="configurationGroupRef" type="IDREF" use="optional" />
3359   </complexType>
3360 </element>
3361 <element name="TestStep">
3362   <complexType>
3363     <choice>
3364       <element ref="ebTest:PutMessage" />
3365       <element ref="ebTest:GetMessage" />
3366     </choice>
3367     <attribute name="description" type="string" use="optional" />
3368     <attribute name="configurationGroupRef" type="IDREF" use="optional" />
3369     <attribute name="repeatTimes" type="integer" default="1" />
3370     <attribute name="testStepContext" type="integer" use="optional" />
3371     <attribute name="stepDelay" type="integer" use="optional" />
3372   </complexType>
3373 </element>
3374 <element name="MessageExpression">
3375   <complexType>
3376     <sequence>
3377       <element ref="ebTest:ErrorMessage" />
3378     </sequence>
3379   </complexType>
3380 </element>
3381 <element name="ErrorMessage" type="ebTest:non-empty-string" />
3382 <element name="PutMessage">
3383   <complexType>
3384     <sequence>
3385       <element ref="ebTest:MessageDeclaration" />
3386       <element ref="ebTest:SetPayload" minOccurs="0" maxOccurs="unbounded" />
3387     <element name="DSign" minOccurs="0">
3388       <complexType>
3389         <sequence>
3390           <element ref="ds:Signature" maxOccurs="unbounded" />
3391         </sequence>
3392       </complexType>
3393     </element>
3394     <element ref="ebTest:SetParameter" minOccurs="0" />
3395     <sequence>
3396       <attribute name="description" type="string" use="required" />
3397       <attribute name="clearMessageStore" type="boolean" default="false" />
3398     </complexType>
3399   </element>
3400 <element name="GetPayload">
3401   <complexType>
3402     <sequence>
3403       <choice>
3404         <element ref="ebTest:Content-ID" />
3405         <element ref="ebTest:Content-Location" />
3406         <element ref="ebTest:Index" />
3407       </choice>
3408     <choice minOccurs="0" maxOccurs="unbounded">
3409       <element ref="ebTest:TestPreCondition" />
3410       <element ref="ebTest:TestAssertion" />
3411     </choice>
3412     <element ref="ebTest:SetParameter" minOccurs="0" />
3413     <sequence>
3414       <attribute name="description" type="string" use="required" />
3415     </complexType>
3416   </element>
3417 <element name="GetMessage">
```

```

3418 <complexType>
3419   <sequence>
3420     <element ref="ebTest:Filter" />
3421     <element ref="ebTest:TestPreCondition" minOccurs="0" maxOccurs="unbounded" />
3422     <element ref="ebTest:TestAssertion" minOccurs="0" maxOccurs="unbounded" />
3423     <element ref="ebTest:SetParameter" minOccurs="0" />
3424     <element ref="ebTest:GetPayload" minOccurs="0" maxOccurs="unbounded" />
3425   </sequence>
3426   <attribute name="description" type="string" use="required" />
3427   <attribute name="getMultiple" type="boolean" default="false" />
3428   <attribute name="testStepContext" type="integer" use="optional" />
3429 </complexType>
3430 </element>
3431   <element name="Filter" type="ebTest:non-empty-string" />
3432 <element name="SetPayload">
3433   <complexType>
3434     <sequence>
3435       <choice>
3436         <element ref="ebTest:Content-ID" />
3437         <element ref="ebTest:Content-Location" />
3438       </choice>
3439       <choice>
3440         <element ref="ebTest:FileURI" />
3441         <element ref="ebTest:PayloadRef" />
3442       </choice>
3443     <sequence minOccurs="0" maxOccurs="unbounded">
3444       <element ref="ebTest:MimeHeader" />
3445       <element ref="ebTest:MimeHeaderValue" />
3446     </sequence>
3447     <element ref="ebTest:SetParameter" minOccurs="0" />
3448   </sequence>
3449   <attribute name="description" type="string" use="required" />
3450 </complexType>
3451 </element>
3452 <element name="TestPreCondition">
3453   <complexType>
3454     <choice>
3455       <element ref="ebTest:VerifyContent" />
3456       <element ref="ebTest:ValidateContent" />
3457     </choice>
3458     <attribute name="description" type="string" use="required" />
3459   </complexType>
3460 </element>
3461 <element name="TestAssertion">
3462   <complexType>
3463     <choice>
3464       <element ref="ebTest:VerifyContent" />
3465       <element ref="ebTest:ValidateContent" />
3466     </choice>
3467     <attribute name="description" type="string" use="required" />
3468     <attribute name="requirement" type="ebTest:requirement.type" default="required" />
3469   </complexType>
3470 </element>
3471 <simpleType name="mimeHeader.type">
3472   <restriction base="NMTOKEN">
3473     <enumeration value="MIMEMessageContent-Type" />
3474     <enumeration value="MIMEMessageStart" />
3475     <enumeration value="Content-Type" />
3476     <enumeration value="start" />
3477     <enumeration value="charset" />
3478     <enumeration value="type" />
3479     <enumeration value="wildcard" />
3480   </restriction>
3481 </simpleType>
3482 <simpleType name="content.type">
3483   <restriction base="NMTOKEN">
3484     <enumeration value="XML" />
3485     <enumeration value="date" />
3486     <enumeration value="URI" />
3487     <enumeration value="signature" />
3488     <enumeration value="signedAck" />

```

```

3489      </restriction>
3490    </simpleType>
3491    <simpleType name="method.type">
3492      <restriction base="NMOKEN">
3493        <enumeration value="xpath" />
3494        <enumeration value="sha-1" />
3495      </restriction>
3496    </simpleType>
3497    <simpleType name="mode.type">
3498      <restriction base="NMOKEN">
3499        <enumeration value="service" />
3500        <enumeration value="connection" />
3501      </restriction>
3502    </simpleType>
3503    <simpleType name="messageContext.type">
3504      <restriction base="NMOKEN">
3505        <enumeration value="true" />
3506        <enumeration value="false" />
3507      </restriction>
3508    </simpleType>
3509    <simpleType name="requirement.type">
3510      <restriction base="NMOKEN">
3511        <enumeration value="required" />
3512        <enumeration value="stronglyrecommended" />
3513        <enumeration value="recommended" />
3514        <enumeration value="optional" />
3515      </restriction>
3516    </simpleType>
3517    <simpleType name="non-empty-string">
3518      <restriction base="string">
3519        <minLength value="1" />
3520      </restriction>
3521    </simpleType>
3522    <simpleType name="configAction.type">
3523      <restriction base="NMOKEN">
3524        <enumeration value="query" />
3525        <enumeration value="replace" />
3526      </restriction>
3527    </simpleType>
3528    <simpleType name="action.type">
3529      <restriction base="NMOKEN">
3530        <enumeration value="reset" />
3531        <enumeration value="modify" />
3532      </restriction>
3533    </simpleType>
3534    <simpleType name="configItem.type">
3535      <restriction base="NOTATION">
3536        <enumeration value="parameter" />
3537        <enumeration value="namespace" />
3538      </restriction>
3539    </simpleType>
3540    <simpleType name="parameter.type">
3541      <restriction base="NMOKEN">
3542        <enumeration value="xpath" />
3543        <enumeration value="string" />
3544      </restriction>
3545    </simpleType>
3546    <element name="MimeHeader" type="ebTest:mimeHeader.type" />
3547    <element name="MimeHeaderValue" type="ebTest:non-empty-string" />
3548    <element name="Content-Location" type="ebTest:non-empty-string" />
3549    <element name="Index" type="integer" />
3550    <element name="FileURI" type="anyURI" />
3551    <element name="PayloadRef" type="string" />
3552    <element name="Signature" type="base64Binary" />
3553    <element name="Content-ID" type="string" />
3554    <element name="MessageDeclaration">
3555      <complexType>
3556        <sequence>
3557          <element ref="mime:Message" />
3558        </sequence>
3559      </complexType>

```

```

3560      </element>
3561  <element name="ValidateContent">
3562    <complexType>
3563      <simpleContent>
3564        <extension base="ebTest:non-empty-string">
3565          <attribute name="contentType" type="ebTest:content.type" default="XML" />
3566          <attribute name="schemaLocation" type="anyURI" use="optional" />
3567        </extension>
3568      </simpleContent>
3569    </complexType>
3570  </element>
3571  <element name="VerifyContent">
3572    <complexType>
3573      <simpleContent>
3574        <extension base="string">
3575          <attribute name="verifyMethod" type="ebTest:method.type" default="xpath" />
3576        </extension>
3577      </simpleContent>
3578    </complexType>
3579  </element>
3580  <element name="MessagePayload">
3581    <complexType>
3582      <sequence>
3583        <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3584      </sequence>
3585        <attribute name="id" type="ID" use="required" />
3586      </complexType>
3587  </element>
3588  <element name="ConfigurationGroup">
3589    <complexType>
3590      <sequence>
3591        <element ref="ebTest:CPAId" />
3592        <element ref="ebTest:Mode" />
3593        <element ref="ebTest:SenderParty" />
3594        <element ref="ebTest:ReceiverParty" />
3595        <element ref="ebTest:Service" />
3596        <element ref="ebTest:Action" />
3597        <element ref="ebTest:StepDelay" />
3598        <element ref="ebTest:ResponseURL" minOccurs="0" />
3599        <element ref="ebTest:NotifcationURL" minOccurs="0" />
3600      <element name="PayloadDigests" minOccurs="0">
3601        <complexType>
3602          <sequence>
3603            <element name="Payload" maxOccurs="unbounded">
3604              <complexType>
3605                <sequence>
3606                  <element name="Href" type="anyURI" />
3607                  <element name="Digest" type="base64Binary" />
3608                </sequence>
3609              </complexType>
3610            </sequence>
3611          </complexType>
3612        </element>
3613      </element>
3614      <element name="ConfigurationItem" minOccurs="0" maxOccurs="unbounded">
3615        <complexType>
3616          <sequence>
3617            <element name="Name" type="ebTest:non-empty-string" />
3618            <element name="Value" type="ebTest:non-empty-string" />
3619            <element name="Type" type="ebTest:configItem.type" />
3620          </sequence>
3621        </complexType>
3622      </element>
3623      </sequence>
3624      <attribute name="id" type="ID" use="required" />
3625    </complexType>
3626  </element>
3627  <element name="SenderParty" type="ebTest:non-empty-string" />
3628  <element name="ReceiverParty" type="ebTest:non-empty-string" />
3629  <element name="Service" type="anyURI" />
3630  <element name="Action" type="ebTest:non-empty-string" />

```

```
3631 <element name="StepDelay" type="integer" />
3632 <element name="ResponseURL" type="anyURI" />
3633 <element name="NotifcationURL" type="anyURI" />
3634 <element name="CPAId" type="ebTest:non-empty-string" />
3635 <element name="Mode" type="ebTest:non-empty-string" />
3636 <element name="SetParameter">
3637 <complexType>
3638 <sequence>
3639 <element name="Name" type="string" />
3640 <element name="Value" type="string" />
3641 </sequence>
3642 <attribute name="parameterType" type="ebTest:parameter.type" use="required" />
3643 </complexType>
3644 </element>
3645 </schema>
```

```
3646
3647
3648
```

SOAP Message Declaration Schema

```
3649
```

```
3650
3651 <?xml version="1.0" encoding="UTF-8" ?>
3652 <!--
3653 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
3654 -->
3655 <schema targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3656 xmlns:eb="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
3657 xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.oasis-
3658 open.org/tc/ebxml-iic/tests/soap" xmlns="http://www.w3.org/2001/XMLSchema">
3659 <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
3660 schemaLocation="eb.xsd" />
3661 <attributeGroup name="encodingStyle">
3662 <attribute name="encodingStyle" type="tns:encodingStyle" />
3663 </attributeGroup>
3664 <!--
3665 Schema for the SOAP/1.1 envelope
3666
3667 This schema has been produced using W3C's SOAP Version 1.2 schema
3668 found at:
3669
3670 http://www.w3.org/2001/06/soap-envelope
3671
3672 Copyright 2001 Martin Gudgin, Developmentor.
3673
3674 Changes made are the following:
3675 - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
3676 - reverted mustUnderstand to only allow 0 and 1 as lexical values
3677
3678
3679 Copyright 2003 OASIS
3680
3681 Changes made are the following:
3682 - SOAP Header and Body element content models constrained to include ebXML content
3683
3684
3685 Original copyright:
3686
3687 Copyright 2001 W3C (Massachusetts Institute of Technology,
3688 Institut National de Recherche en Informatique et en Automatique,
3689 Keio University). All Rights Reserved.
3690 http://www.w3.org/Consortium/Legal/
3691
3692 This document is governed by the W3C Software License [1] as
3693 described in the FAQ [2].
3694
3695 [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
```

```

3696 [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
3697 -->
3698 <!--
3699 Envelope, header and body
3700 -->
3701 <element name="Envelope" type="tns:Envelope" />
3702 <complexType name="Envelope">
3703 <sequence>
3704 <element ref="tns:Header" />
3705 <element ref="tns:Body" />
3706 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3707 </sequence>
3708 <anyAttribute namespace="##other" processContents="lax" />
3709 </complexType>
3710 <group name="optionElements">
3711 <all minOccurs="0">
3712 <element ref="eb:SyncReply" minOccurs="0"/>
3713 <element ref="eb:MessageOrder" minOccurs="0"/>
3714 <element ref="eb:AckRequested" minOccurs="0"/>
3715 <element ref="eb:Acknowledgment" minOccurs="0"/>
3716 <element ref="eb:ErrorList" minOccurs="0"/>
3717 </all>
3718 </group>
3719 <element name="Header">
3720 <complexType>
3721 <sequence>
3722 <element ref="eb:MessageHeader" />
3723 <group ref="tns:optionElements" />
3724 </sequence>
3725 </complexType>
3726 </element>
3727 <complexType name="Header">
3728 <sequence>
3729 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3730 </sequence>
3731 <anyAttribute namespace="##other" processContents="lax" />
3732 </complexType>
3733 <element name="Body">
3734 <complexType>
3735 <choice minOccurs="0">
3736 <element ref="eb:Manifest" />
3737 <element ref="eb>StatusRequest" />
3738 <element ref="eb>StatusResponse" />
3739 </choice>
3740 </complexType>
3741 </element>
3742 <complexType name="Body">
3743 <annotation>
3744 <documentation>Prose in the spec does not specify that attributes are allowed on the
3745 Body element</documentation>
3746 </annotation>
3747 <sequence>
3748 <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3749 </sequence>
3750 <anyAttribute namespace="##any" processContents="lax" />
3751 </complexType>
3752 <!--
3753 Global Attributes. The following attributes are intended to be usable via qualified
3754 attribute names on any complex type referencing them.
3755 -->
3756 <attribute name="mustUnderstand" default="0">
3757 <simpleType>
3758 <restriction base="boolean">
3759 <pattern value="0|1" />
3760 </restriction>
3761 </simpleType>
3762 </attribute>
3763 <attribute name="actor" type="anyURI" />
3764 <simpleType name="encodingStyle">
3765 <annotation>

```

```

3766   <documentation>'encodingStyle' indicates any canonicalization conventions followed in
3767   the contents of the containing element. For example, the value
3768   'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern described in SOAP
3769   specification</documentation>
3770   </annotation>
3771   <list itemType="anyURI" />
3772   </simpleType>
3773   <complexType name="Fault" final="extension">
3774   <annotation>
3775     <documentation>Fault reporting structure</documentation>
3776   </annotation>
3777   <sequence>
3778     <element name="faultcode" type="QName" />
3779     <element name="faultstring" type="string" />
3780     <element name="faultactor" type="anyURI" minOccurs="0" />
3781     <element name="detail" type="tns:detail" minOccurs="0" />
3782   </sequence>
3783   </complexType>
3784   <complexType name="detail">
3785     <sequence>
3786       <any namespace="#any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3787     </sequence>
3788     <anyAttribute namespace="#any" processContents="lax" />
3789   </complexType>
3790 </schema>
```

3791
3792
3793

3794 ebXML Message Declaration Schema

```

3795
3796
3797 <?xml version="1.0" encoding="UTF-8" ?>
3798 <!--
3799 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
3800 -->
3801 <schema targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
3802 xmlns:soap="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3803 xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3804 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:tns="http://www.oasis-
3805 open.org/tc/ebxml-iic/tests/eb" xmlns="http://www.w3.org/2001/XMLSchema"
3806 elementFormDefault="qualified" attributeFormDefault="qualified" version="1.0">
3807   <import namespace="http://www.w3.org/1999/xlink" schemaLocation="http://www.oasis-
3808 open.org/committees/ebxml-msg/schema/xlink.xsd" />
3809   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3810 schemaLocation="xmldsig.xsd" />
3811   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3812 schemaLocation="soap.xsd" />
3813   <import namespace="http://www.w3.org/XML/1998/namespace"
3814 schemaLocation="http://www.oasis-open.org/committees/ebxml-msg/schema/xml_lang.xsd" />
3815   <attributeGroup name="bodyExtension.grp">
3816     <attribute ref="tns:id" />
3817     <attribute ref="tns:version" use="optional" />
3818   </attributeGroup>
3819   <attributeGroup name="headerExtension.grp">
3820     <attribute ref="tns:id" />
3821     <attribute ref="tns:version" use="optional" />
3822     <attribute ref="soap:mustUnderstand" use="optional" />
3823   </attributeGroup>
3824 <!--
3825 MANIFEST, for use in soap:Body element
3826 -->
3827   <element name="Manifest">
3828     <complexType>
3829       <sequence>
3830         <element ref="tns:Reference" maxOccurs="unbounded" />
```

```

3831 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3832 </sequence>
3833 <attributeGroup ref="tns:bodyExtension.grp" />
3834 </complexType>
3835 </element>
3836 <element name="Reference">
3837 <complexType>
3838 <sequence>
3839 <element ref="tns:Schema" minOccurs="0" maxOccurs="unbounded" />
3840 <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded" />
3841 <choice>
3842 <element ref="tns:FileName" />
3843 <element ref="tns:MessageRef" />
3844 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3845 </choice>
3846 </sequence>
3847 <attribute ref="tns:id" />
3848 <attribute ref="xlink:type" fixed="simple" />
3849 <attribute ref="xlink:href" use="required" />
3850 <attribute ref="xlink:role" />
3851 <attribute name="contentId" type="string" use="optional" />
3852 <attribute name="contentType" type="string" use="optional" />
3853 <attribute name="contentLocation" type="anyURI" use="optional" />
3854 </complexType>
3855 </element>
3856 <element name="Schema">
3857 <complexType>
3858 <attribute name="location" type="anyURI" use="required" />
3859 <attribute name="version" type="tns:non-empty-string" />
3860 </complexType>
3861 </element>
3862 <!--
3863 MESSAGEHEADER, for use in soap:Header element
3864 -->
3865 <element name="MessageHeader">
3866 <complexType>
3867 <sequence>
3868 <element ref="tns:From" minOccurs="0" />
3869 <element ref="tns:To" minOccurs="0" />
3870 <element ref="tns:CPAID" minOccurs="0" />
3871 <element ref="tns:ConversationId" minOccurs="0" />
3872 <element ref="tns:Service" minOccurs="0" />
3873 <element ref="tns:Action" minOccurs="0" />
3874 <element ref="tns:MessageData" minOccurs="0" />
3875 <element ref="tns:DuplicateElimination" minOccurs="0" />
3876 <element ref="tns:Description" minOccurs="0" maxOccurs="unbounded" />
3877 <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3878 </sequence>
3879 <attributeGroup ref="tns:headerExtension.grp" />
3880 </complexType>
3881 </element>
3882 <element name="CPAID" type="tns:non-empty-string" />
3883 <element name="ConversationId" type="tns:non-empty-string" />
3884 <element name="Service">
3885 <complexType>
3886 <simpleContent>
3887 <extension base="tns:non-empty-string">
3888 <attribute name="type" type="tns:non-empty-string" />
3889 </extension>
3890 </simpleContent>
3891 </complexType>
3892 </element>
3893 <element name="Action" type="tns:non-empty-string" />
3894 <element name="MessageData">
3895 <complexType>
3896 <sequence>
3897 <element ref="tns:MessageId" minOccurs="0" />
3898 <element ref="tns:Timestamp" minOccurs="0" />
3899 <element ref="tns:RefToMessageId" minOccurs="0" />
3900 <element ref="tns:TimeToLive" minOccurs="0" />
3901 </sequence>

```

```

3902    </complexType>
3903  </element>
3904  <element name="MessageId" type="tns:non-empty-string" />
3905  <element name="TimeToLive" type="dateTime" />
3906  <element name="DuplicateElimination" />
3907  <!--
3908  SYNC REPLY, for use in soap:Header element
3909  -->
3910  <element name="SyncReply">
3911  <complexType>
3912  <sequence>
3913  <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3914  </sequence>
3915  <attributeGroup ref="tns:headerExtension.grp" />
3916  <attribute ref="soap:actor" default="urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH" />
3917  </complexType>
3918  </element>
3919  <!--
3920  MESSAGE ORDER, for use in soap:Header element
3921  -->
3922  <element name="MessageOrder">
3923  <complexType>
3924  <sequence>
3925  <element ref="tns:SequenceNumber" />
3926  <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3927  </sequence>
3928  <attributeGroup ref="tns:headerExtension.grp" />
3929  </complexType>
3930  </element>
3931  <element name="SequenceNumber" type="tns:sequenceNumber.type" />
3932  <!--
3933  ACK REQUESTED, for use in soap:Header element
3934  -->
3935  <element name="AckRequested">
3936  <complexType>
3937  <sequence>
3938  <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3939  </sequence>
3940  <attributeGroup ref="tns:headerExtension.grp" />
3941  <attribute ref="soap:actor" />
3942  <attribute name="signed" type="boolean" use="optional" />
3943  </complexType>
3944  </element>
3945  <!--
3946  ACKNOWLEDGMENT, for use in soap:Header element
3947  -->
3948  <element name="Acknowledgment">
3949  <complexType>
3950  <sequence>
3951  <element ref="tns:Timestamp" minOccurs="0" />
3952  <element ref="tns:RefToMessageId" minOccurs="0" />
3953  <element ref="tns:From" minOccurs="0" />
3954  <element name="Reference" minOccurs="0" maxOccurs="unbounded" />
3955  <any namespace="##other" processContents="lax" minOccurs="0" />
3956  <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded" />
3957  </sequence>
3958  <attributeGroup ref="tns:headerExtension.grp" />
3959  <attribute ref="soap:actor" default="urn:oasis:names:tc:ebxml-msg:actor:toPartyMSH" />
3960  </complexType>
3961  </element>
3962  <!--
3963  ERROR LIST, for use in soap:Header element
3964  -->
3965  <element name="ErrorList">
3966  <complexType>
3967  <sequence>
3968  <element ref="tns:Error" maxOccurs="unbounded" />
3969  <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3970  </sequence>
3971  <attributeGroup ref="tns:headerExtension.grp" />
3972  <attribute name="highestSeverity" type="tns:severity.type" use="required" />

```

```

3973     </complexType>
3974   </element>
3975   <element name="Error">
3976     <complexType>
3977       <sequence>
3978         <element ref="tns:Description" minOccurs="0" />
3979         <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3980       </sequence>
3981       <attribute ref="tns:id" />
3982       <attribute name="codeContext" type="anyURI" default="urn:oasis:names:tc:ebxml-
3983 msg:service:errors" />
3984       <attribute name="errorCode" type="tns:non-empty-string" use="required" />
3985       <attribute name="severity" type="tns:severity.type" use="required" />
3986       <attribute name="location" type="tns:non-empty-string" />
3987     </complexType>
3988   </element>
3989   <!--
3990     STATUS RESPONSE, for use in soap:Body element
3991   -->
3992   <element name="StatusResponse">
3993     <complexType>
3994       <sequence>
3995         <element ref="tns:RefToMessageId" />
3996         <element ref="tns:Timestamp" minOccurs="0" />
3997         <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
3998       </sequence>
3999       <attributeGroup ref="tns:bodyExtension.grp" />
4000       <attribute name="messageStatus" type="tns:messageStatus.type" use="required" />
4001     </complexType>
4002   </element>
4003   <!--
4004     STATUS REQUEST, for use in soap:Body element
4005   -->
4006   <element name="StatusRequest">
4007     <complexType>
4008       <sequence>
4009         <element ref="tns:RefToMessageId" />
4010         <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4011       </sequence>
4012       <attributeGroup ref="tns:bodyExtension.grp" />
4013     </complexType>
4014   </element>
4015   <!--
4016     COMMON TYPES
4017   -->
4018   <complexType name="sequenceNumber.type">
4019     <simpleContent>
4020       <extension base="positiveInteger">
4021         <attribute name="status" type="tns:status.type" default="Continue" />
4022       </extension>
4023     </simpleContent>
4024   </complexType>
4025   <simpleType name="status.type">
4026     <restriction base="NMTOKEN">
4027       <enumeration value="Reset" />
4028       <enumeration value="Continue" />
4029     </restriction>
4030   </simpleType>
4031   <simpleType name="messageStatus.type">
4032     <restriction base="NMTOKEN">
4033       <enumeration value="UnAuthorized" />
4034       <enumeration value="NotRecognized" />
4035       <enumeration value="Received" />
4036       <enumeration value="Processed" />
4037       <enumeration value="Forwarded" />
4038     </restriction>
4039   </simpleType>
4040   <simpleType name="non-empty-string">
4041     <restriction base="string">
4042       <minLength value="1" />
4043     </restriction>

```

```

4044    </simpleType>
4045    <simpleType name="severity.type">
4046      <restriction base="NMOKEN">
4047        <enumeration value="Warning" />
4048        <enumeration value="Error" />
4049      </restriction>
4050    </simpleType>
4051    <!--
4052    COMMON ATTRIBUTES and ATTRIBUTE GROUPS
4053    -->
4054    <attribute name="id" type="ID" />
4055    <attribute name="version" type="tns:non-empty-string" />
4056    <!--
4057    COMMON ELEMENTS
4058    -->
4059    <element name="PartyId">
4060      <complexType>
4061        <simpleContent>
4062          <extension base="tns:non-empty-string">
4063            <attribute name="type" type="tns:non-empty-string" />
4064          </extension>
4065        </simpleContent>
4066      </complexType>
4067    </element>
4068    <element name="To">
4069      <complexType>
4070        <sequence>
4071          <element ref="tns:PartyId" />
4072          <element name="Role" type="tns:non-empty-string" minOccurs="0" />
4073        </sequence>
4074      </complexType>
4075    </element>
4076    <element name="From">
4077      <complexType>
4078        <sequence>
4079          <element ref="tns:PartyId" />
4080          <element name="Role" type="tns:non-empty-string" minOccurs="0" />
4081        </sequence>
4082      </complexType>
4083    </element>
4084    <element name="Description">
4085      <complexType>
4086        <simpleContent>
4087          <extension base="tns:non-empty-string">
4088            <attribute ref="xml:lang" use="required" />
4089          </extension>
4090        </simpleContent>
4091      </complexType>
4092    </element>
4093    <element name="RefToMessageId" type="tns:non-empty-string" />
4094    <element name="Timestamp" type="dateTime" />
4095    <element name="FileName" type="tns:non-empty-string" />
4096    <element name="MessageRef" type="tns:non-empty-string" />
4097  </schema>

```

4098

4099 **XMLDSIG Declaration Schema**

4100

```

4101  <?xml version="1.0" encoding="UTF-8" ?>
4102  <!--
4103  Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4104  -->
4105  <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.oasis-
4106  open.org/tc/ebxml-iic/tests/xmldsig" xmlns:ds="http://www.oasis-open.org/tc/ebxml-
4107  iic/tests/xmldsig" version="0.1" elementFormDefault="qualified"
4108  attributeFormDefault="unqualified">

```

```

4109      <!--
4110      Basic Types Defined for Signatures
4111      -->
4112      <simpleType name="CryptoBinary">
4113          <restriction base="base64Binary" />
4114      </simpleType>
4115      <simpleType name="DigestValueType">
4116          <restriction base="base64Binary" />
4117      </simpleType>
4118      <simpleType name="HMACOutputLengthType">
4119          <restriction base="integer" />
4120      </simpleType>
4121      <!--
4122      Start Signature
4123      -->
4124          <element name="Signature" type="ds:SignatureType" />
4125      <complexType name="SignatureType">
4126          <sequence>
4127              <element ref="ds:SignedInfo" />
4128              <element ref="ds:SignatureValue" minOccurs="0" />
4129              <element ref="ds:KeyInfo" minOccurs="0" />
4130              <element ref="ds:Object" minOccurs="0" maxOccurs="unbounded" />
4131          </sequence>
4132          <attribute name="Id" use="optional" type="ID" />
4133      </complexType>
4134          <element name="SignatureValue" type="ds:SignatureValueType" />
4135      <complexType name="SignatureValueType">
4136          <simpleContent>
4137              <extension base="base64Binary">
4138                  <attribute name="Id" use="optional" type="ID" />
4139              </extension>
4140          </simpleContent>
4141      </complexType>
4142      <!--
4143      Start SignedInfo
4144      -->
4145          <element name="SignedInfo" type="ds:SignedInfoType" />
4146      <complexType name="SignedInfoType">
4147          <sequence>
4148              <element ref="ds:CanonicalizationMethod" minOccurs="0" />
4149              <element ref="ds:SignatureMethod" minOccurs="0" />
4150              <element ref="ds:Reference" minOccurs="0" maxOccurs="unbounded" />
4151          </sequence>
4152          <attribute name="Id" use="optional" type="ID" />
4153      </complexType>
4154          <element name="CanonicalizationMethod" type="ds:CanonicalizationMethodType" />
4155      <complexType name="CanonicalizationMethodType" mixed="true">
4156          <!--
4157          (0,unbounded) elements from (1,1) namespace
4158          -->
4159          <sequence>
4160              <any namespace="##any" processContents="strict" minOccurs="0" maxOccurs="unbounded" />
4161          </sequence>
4162          <attribute name="Algorithm" use="required" type="anyURI" />
4163      </complexType>
4164          <element name="SignatureMethod" type="ds:SignatureMethodType" />
4165      <complexType name="SignatureMethodType" mixed="true">
4166          <!--
4167          (0,unbounded) elements from (1,1) external namespace
4168          -->
4169          <sequence>
4170              <element name="HMACOutputLength" type="ds:HMACOutputLengthType" minOccurs="0" />
4171              <any namespace="##other" processContents="strict" minOccurs="0" maxOccurs="unbounded" />
4172          </sequence>
4173          <attribute name="Algorithm" use="required" type="anyURI" />
4174      </complexType>
4175          <!--
4176          Start Reference
4177          -->
4178          <element name="Reference" type="ds:ReferenceType" />

```

```

4180 <complexType name="ReferenceType">
4181   <sequence>
4182     <element ref="ds:Transforms" minOccurs="0" />
4183     <element ref="ds:DigestMethod" />
4184     <element ref="ds:DigestValue" minOccurs="0" />
4185   </sequence>
4186   <attribute name="Id" use="optional" type="ID" />
4187   <attribute name="URI" use="optional" type="anyURI" />
4188   <attribute name="Type" use="optional" type="anyURI" />
4189 </complexType>
4190   <element name="Transforms" type="ds:TransformsType" />
4191 <complexType name="TransformsType">
4192   <sequence>
4193     <element ref="ds:Transform" maxOccurs="unbounded" />
4194   </sequence>
4195 </complexType>
4196   <element name="Transform" type="ds:TransformType" />
4197 <complexType name="TransformType" mixed="true">
4198   <!--
4199   (1,1) elements from (0,unbounded) namespaces
4200   -->
4201   <choice minOccurs="0" maxOccurs="unbounded">
4202     <any namespace="#other" processContents="lax" />
4203     <element name="XPath" type="string" />
4204   </choice>
4205   <attribute name="Algorithm" use="required" type="anyURI" />
4206 </complexType>
4207   <!--
4208   End Reference
4209   -->
4210   <element name="DigestMethod" type="ds:DigestMethodType" />
4211 <complexType name="DigestMethodType" mixed="true">
4212   <sequence>
4213     <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4214   </sequence>
4215   <attribute name="Algorithm" use="required" type="anyURI" />
4216 </complexType>
4217   <element name="DigestValue" type="ds:DigestValueType" />
4218   <!--
4219   End SignedInfo
4220   -->
4221   <!--
4222   Start KeyInfo
4223   -->
4224   <element name="KeyInfo" type="ds:KeyInfoType" />
4225 <complexType name="KeyInfoType" mixed="true">
4226   <!--
4227   (1,1) elements from (0,unbounded) namespaces
4228   -->
4229   <choice maxOccurs="unbounded">
4230     <element ref="ds:KeyName" />
4231     <element ref="ds:KeyValue" />
4232     <element ref="ds:RetrievalMethod" />
4233     <element ref="ds:X509Data" />
4234     <element ref="ds:PGPData" />
4235     <element ref="ds:SPKIData" />
4236     <element ref="ds:MgmtData" />
4237     <any namespace="#other" processContents="lax" />
4238   </choice>
4239   <attribute name="Id" use="optional" type="ID" />
4240 </complexType>
4241   <element name="KeyName" type="string" />
4242   <element name="MgmtData" type="string" />
4243   <element name="KeyValue" type="ds:KeyValueType" />
4244 <complexType name="KeyValueType" mixed="true">
4245   <choice>
4246     <element ref="ds:DSAKeyValue" />
4247     <element ref="ds:RSAKeyValue" />
4248     <any namespace="#other" processContents="lax" />
4249   </choice>
4250 </complexType>
```

```

4251   <element name="RetrievalMethod" type="ds:RetrievalMethodType" />
4252 <complexType name="RetrievalMethodType">
4253   <sequence>
4254     <element ref="ds:Transforms" minOccurs="0" />
4255   </sequence>
4256   <attribute name="URI" type="anyURI" />
4257   <attribute name="Type" use="optional" type="anyURI" />
4258 </complexType>
4259   <!--
4260   Start X509Data
4261   -->
4262   <element name="X509Data" type="ds:X509DataType" />
4263 <complexType name="X509DataType">
4264   <sequence maxOccurs="unbounded">
4265     <choice>
4266       <element name="X509IssuerSerial" type="ds:X509IssuerSerialType" />
4267       <element name="X509SKI" type="base64Binary" />
4268       <element name="X509SubjectName" type="string" />
4269       <element name="X509Certificate" type="base64Binary" />
4270       <element name="X509CRL" type="base64Binary" />
4271       <any namespace="#other" processContents="lax" />
4272     </choice>
4273     </sequence>
4274   </complexType>
4275   <complexType name="X509IssuerSerialType">
4276     <sequence>
4277       <element name="X509IssuerName" type="string" />
4278       <element name="X509SerialNumber" type="integer" />
4279     </sequence>
4280   </complexType>
4281   <!--
4282   End X509Data
4283   -->
4284   <!--
4285   Begin PGPData
4286   -->
4287   <element name="PGPData" type="ds:PGPDataType" />
4288 <complexType name="PGPDataType">
4289   <choice>
4290     <sequence>
4291       <element name="PGPKeyID" type="base64Binary" />
4292       <element name="PGPKeyPacket" type="base64Binary" minOccurs="0" />
4293       <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4294     </sequence>
4295     <sequence>
4296       <element name="PGPKeyPacket" type="base64Binary" />
4297       <any namespace="#other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4298     </sequence>
4299   </choice>
4300   </complexType>
4301   <!--
4302   End PGPData
4303   -->
4304   <!--
4305   Begin SPKIData
4306   -->
4307   <element name="SPKIData" type="ds:SPKIDataType" />
4308 <complexType name="SPKIDataType">
4309   <sequence maxOccurs="unbounded">
4310     <element name="SPKISexp" type="base64Binary" />
4311     <any namespace="#other" processContents="lax" minOccurs="0" />
4312   </sequence>
4313   </complexType>
4314   <!--
4315   End SPKIData
4316   -->
4317   <!--
4318   End KeyInfo
4319   -->
4320   <!--
4321   Start Object (Manifest, SignatureProperty)

```

```

4322      -->
4323      <element name="Object" type="ds:ObjectType" />
4324  <complexType name="ObjectType" mixed="true">
4325      <!--
4326      add a grep facet
4327      -->
4328      <sequence minOccurs="0" maxOccurs="unbounded">
4329          <any namespace="#any" processContents="lax" />
4330      </sequence>
4331      <attribute name="Id" use="optional" type="ID" />
4332      <attribute name="MimeType" use="optional" type="string" />
4333      <attribute name="Encoding" use="optional" type="anyURI" />
4334  </complexType>
4335      <element name="Manifest" type="ds:ManifestType" />
4336  <complexType name="ManifestType">
4337      <sequence>
4338          <element ref="ds:Reference" maxOccurs="unbounded" />
4339      </sequence>
4340      <attribute name="Id" use="optional" type="ID" />
4341  </complexType>
4342      <element name="SignatureProperties" type="ds:SignaturePropertiesType" />
4343  <complexType name="SignaturePropertiesType">
4344      <sequence>
4345          <element ref="ds:SignatureProperty" maxOccurs="unbounded" />
4346      </sequence>
4347      <attribute name="Id" use="optional" type="ID" />
4348  </complexType>
4349      <element name="SignatureProperty" type="ds:SignaturePropertyType" />
4350  <complexType name="SignaturePropertyType" mixed="true">
4351      <!--
4352      (1,1) elements from (1,unbounded) namespaces
4353      -->
4354      <choice maxOccurs="unbounded">
4355          <any namespace="#other" processContents="lax" />
4356      </choice>
4357      <attribute name="Target" use="required" type="anyURI" />
4358      <attribute name="Id" use="optional" type="ID" />
4359  </complexType>
4360      <!--
4361      End Object (Manifest, SignatureProperty)
4362      -->
4363      <!--
4364      Start Algorithm Parameters
4365      -->
4366      <!--
4367      Start KeyValue Element-types
4368      -->
4369      <element name="DSAKeyValue" type="ds:DSAKeyValue" />
4370  <complexType name="DSAKeyValue">
4371      <sequence>
4372          <sequence minOccurs="0">
4373              <element name="P" type="ds:CryptoBinary" />
4374              <element name="Q" type="ds:CryptoBinary" />
4375          </sequence>
4376          <element name="G" type="ds:CryptoBinary" minOccurs="0" />
4377          <element name="Y" type="ds:CryptoBinary" />
4378          <element name="J" type="ds:CryptoBinary" minOccurs="0" />
4379      <sequence minOccurs="0">
4380          <element name="Seed" type="ds:CryptoBinary" />
4381          <element name="PgenCounter" type="ds:CryptoBinary" />
4382      </sequence>
4383      </sequence>
4384  </complexType>
4385      <element name="RSAKeyValue" type="ds:RSAKeyValue" />
4386  <complexType name="RSAKeyValue">
4387      <sequence>
4388          <element name="Modulus" type="ds:CryptoBinary" />
4389          <element name="Exponent" type="ds:CryptoBinary" />
4390      </sequence>
4391  </complexType>
4392      <!--

```

```
4393     End KeyValue Element-types
4394     -->
4395     <!--
4396     End Signature
4397     -->
4398     </schema>
```

4399

4400

4401

4402

4403

4404

4405

4406 **Appendix D (Normative) The ebXML Message Store**
4407 **Schema**

```
4408  <?xml version="1.0" encoding="UTF-8" ?>
4409  <!--
4410  Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4411  -->
4412  <xsd:schema targetNamespace="http://www.oasis-open.org/tc/ebxml-
4413  iic/testing/messageStore" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4414  xmlns:mime="http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
4415  xmlns="http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore">
4416  <xsd:import namespace="http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
4417  schemaLocation="messagestore_mime.xsd" />
4418  <xsd:element name="MessageStore">
4419  <xsd:complexType>
4420  <xsd:sequence>
4421  <xsd:element ref="mime:Message" minOccurs="0" maxOccurs="unbounded" />
4422  </xsd:sequence>
4423  </xsd:complexType>
4424  </xsd:element>
4425  </xsd:schema>
```

4426
4427
4428

4429 **Message Store MIME Message Schema**

4430

```
4431  <?xml version="1.0" encoding="UTF-8" ?>
4432  <!--
4433  Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4434  -->
4435  <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.oasis-
4436  open.org/tc/ebxml-iic/testing/mime" xmlns:tns="http://www.oasis-open.org/tc/ebxml-
4437  iic/testing/mime" xmlns:xs="http://www.w3.org/2001/XMLSchema"
4438  xmlns:soap="http://www.oasis-open.org/tc/ebxml-iic/testing/soap">
4439  <import namespace="http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
4440  schemaLocation="messagestore_soap.xsd" />
4441  <element name="Message">
4442  <complexType mixed="true">
4443  <choice>
4444  <element ref="tns:MessageContainer" />
4445  </choice>
4446  <attribute name="contentType" default="multipart/related" type="string" />
4447  <attribute name="type" use="optional" type="string" />
4448  <attribute name="serviceInstanceId" use="required" type="anyURI" />
4449  <attribute name="id" use="required" type="string" />
4450  <attribute name="syncType" use="required" type="tns:sync.type" />
4451  <attribute name="reportingAction" use="required" type="string" />
4452  <attribute name="serviceName" use="required" type="string" />
4453  </complexType>
4454  </element>
4455  <element name="MessageContainer">
4456  <complexType>
4457  <sequence>
4458  <element ref="soap:Envelope" />
4459  </sequence>
4460  <attribute name="contentId" use="optional" type="string" />
4461  <attribute name="contentType" use="optional" type="string" />
4462  <attribute name="charset" use="optional" type="string" />
4463  </complexType>
4464  </element>
```

```

4465 <element name="PayloadContainer">
4466   <complexType>
4467     <attribute name="contentId" use="optional" type="string" />
4468     <attribute name="contentType" use="optional" type="string" />
4469     <attribute name="charset" use="optional" type="string" />
4470   </complexType>
4471 </element>
4472 <simpleType name="sync.type">
4473   <restriction base="NMTOKEN">
4474     <enumeration value="synchronous" />
4475     <enumeration value="asynchronous" />
4476   </restriction>
4477 </simpleType>
4478 </schema>

```

```

4479
4480
4481
4482
4483
4484

```

Message Store SOAP Message Schema

```
4486
```

```

4487 <?xml version="1.0" encoding="UTF-8" ?>
4488 <!--
4489 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4490 -->
4491 <xsschema targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
4492 xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.oasis-open.org/tc/ebxml-
4493 iic/testing/soap" xmlns:xs="http://www.w3.org/2001/XMLSchema"
4494 xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
4495   <xssimport namespace="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
4496 header-2_0.xsd" schemaLocation="http://www.oasis-open.org/committees/ebxml-
4497 msg/schema/msg-header-2_0.xsd" />
4498   - <xssattributeGroup name="encodingStyle">
4499     <xssattribute name="encodingStyle" type="tns:encodingStyle" />
4500   </xssattributeGroup>
4501 - <!--
4502 Schema for the SOAP/1.1 envelope
4503
4504   This schema has been produced using W3C's SOAP Version 1.2 schema
4505   found at:
4506
4507   http://www.w3.org/2001/06/soap-envelope
4508
4509   Copyright 2001 Martin Gudgin, Developmentor.
4510
4511   Changes made are the following:
4512   - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
4513   - reverted mustUnderstand to only allow 0 and 1 as lexical values
4514
4515   Original copyright:
4516
4517   Copyright 2001 W3C (Massachusetts Institute of Technology,
4518   Institut National de Recherche en Informatique et en Automatique,
4519   Keio University). All Rights Reserved.
4520   http://www.w3.org/Consortium/Legal/
4521
4522   This document is governed by the W3C Software License [1] as
4523   described in the FAQ [2].
4524
4525   [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
4526   [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD

```

```

4527    -->
4528    <!--
4529      Envelope, header and body
4530      -->
4531      <xs:element name="Envelope" type="tns:Envelope" />
4532    <xs:complexType name="Envelope">
4533      <xs:sequence>
4534        <xs:element ref="tns:Header" />
4535        <xs:element ref="tns:Body" />
4536        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"
4537      />
4538      </xs:sequence>
4539      <xs:anyAttribute namespace="##other" processContents="lax" />
4540    </xs:complexType>
4541    <xs:group name="optionElements">
4542      <xs:all minOccurs="0">
4543        <xs:element ref="eb:SyncReply" minOccurs="0" />
4544        <xs:element ref="eb:MessageOrder" minOccurs="0" />
4545        <xs:element ref="eb:AckRequested" minOccurs="0" />
4546        <xs:element ref="eb:Acknowledgment" minOccurs="0" />
4547        <xs:element ref="eb:ErrorList" minOccurs="0" />
4548      </xs:all>
4549    </xs:group>
4550    <xs:element name="Header">
4551      <xs:complexType>
4552        <xs:sequence>
4553          <xs:element ref="eb:MessageHeader" />
4554          <xs:group ref="tns:optionElements" />
4555        </xs:sequence>
4556      </xs:complexType>
4557    </xs:element>
4558    <xs:complexType name="Header">
4559      <xs:sequence>
4560        <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"
4561      />
4562      </xs:sequence>
4563      <xs:anyAttribute namespace="##other" processContents="lax" />
4564    </xs:complexType>
4565    <xs:element name="Body">
4566      <xs:complexType>
4567        <xs:choice>
4568          <xs:element ref="eb:Manifest" />
4569          <xs:element ref="eb:StatusRequest" />
4570          <xs:element ref="eb:StatusResponse" />
4571        </xs:choice>
4572      </xs:complexType>
4573    </xs:element>
4574    <xs:complexType name="Body">
4575      <xs:annotation>
4576        <xs:documentation>Prose in the spec does not specify that attributes are allowed on
4577        the Body element</xs:documentation>
4578      </xs:annotation>
4579      <xs:sequence>
4580        <xs:any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4581      </xs:sequence>
4582      <xs:anyAttribute namespace="##any" processContents="lax" />
4583    </xs:complexType>
4584    <!--
4585      Global Attributes. The following attributes are intended to be usable via qualified
4586      attribute names on any complex type referencing them.
4587      -->
4588      <xs:attribute name="mustUnderstand" default="0">
4589        <xs:simpleType>
4590          <xs:restriction base="boolean">
4591            <xs:pattern value="0|1" />
4592          </xs:restriction>
4593        </xs:simpleType>
4594      </xs:attribute>
4595      <xs:attribute name="actor" type="anyURI" />
4596      <xs:simpleType name="encodingStyle">
4597        <xs:annotation>
```

```
4598     <xs:documentation>'encodingStyle' indicates any canonicalization conventions followed  
4599     in the contents of the containing element. For example, the value  
4600     'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern described in SOAP  
4601     specification</xs:documentation>  
4602     </xs:annotation>  
4603     <xs:list itemType="anyURI" />  
4604     </xs:simpleType>  
4605     <xs:complexType name="Fault" final="extension">  
4606     <xs:annotation>  
4607         <xs:documentation>Fault reporting structure</xs:documentation>  
4608     </xs:annotation>  
4609     <xs:sequence>  
4610         <xs:element name="faultcode" type="QName" />  
4611         <xs:element name="faultstring" type="string" />  
4612         <xs:element name="faultactor" type="anyURI" minOccurs="0" />  
4613         <xs:element name="detail" type="tns:detail" minOccurs="0" />  
4614     </xs:sequence>  
4615     </xs:complexType>  
4616     <xs:complexType name="detail">  
4617         <xs:sequence>  
4618             <xs:any namespace="#any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />  
4619         </xs:sequence>  
4620         <xs:anyAttribute namespace="#any" processContents="lax" />  
4621     </xs:complexType>  
4622 </xs:schema>
```

4623

4624 **Appendix E (Normative) The ebXML Test Report**
4625 **Schema**

4626

```
4627 <?xml version="1.0" encoding="UTF-8" ?>
4628 <!--
4629 Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4630 -->
4631 <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.oasis-
4632 open.org/tc/ebxml-iic/testReport" xmlns:ebReport="http://www.oasis-open.org/tc/ebxml-
4633 iic/testReport" version="1.0" elementFormDefault="unqualified"
4634 attributeFormDefault="unqualified">
4635 <!--
4636 edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST)
4637 -->
4638 <element name="TestReport">
4639 <complexType>
4640 <sequence>
4641   <element ref="ebReport:MetaData" />
4642   <element ref="ebReport:TestCase" maxOccurs="unbounded" />
4643 </sequence>
4644 </complexType>
4645 </element>
4646 <element name="MetaData">
4647 <complexType>
4648 <sequence>
4649   <element ref="ebReport:Description" />
4650   <element ref="ebReport:Version" />
4651   <element ref="ebReport:Maintainer" />
4652   <element ref="ebReport:Location" />
4653   <element ref="ebReport:PublishDate" />
4654   <element ref="ebReport:Status" />
4655 </sequence>
4656 </complexType>
4657 </element>
4658 <element name="Description" type="string" />
4659 <element name="Version" type="string" />
4660 <element name="Maintainer" type="string" />
4661 <element name="Location" type="anyURI" />
4662 <element name="PublishDate" type="string" />
4663 <element name="Status" type="string" />
4664 <element name="TestCase">
4665 <complexType>
4666 <sequence>
4667 <sequence maxOccurs="unbounded">
4668   <element ref="ebReport:TestStep" />
4669 </sequence>
4670 </sequence>
4671   <attribute name="id" use="required" type="string" />
4672   <attribute name="description" use="optional" type="string" />
4673   <attribute name="name" use="required" type="string" />
4674   <attribute name="author" use="optional" type="string" />
4675   <attribute name="version" use="optional" type="string" />
4676   <attribute name="requirementReferenceId" use="required" type="anyURI" />
4677   <attribute name="requirementType" use="required" type="string" />
4678   <attribute name="testCaseResult" use="required" type="ebReport:testCaseResult.type" />
4679   <attribute name="specRef" use="required" type="string" />
4680 </complexType>
4681 </element>
4682 <element name="TestStep">
4683 <complexType>
4684 <sequence>
4685   <element name="StepResult">
4686     <complexType>
4687       <choice>
```

```

4688 <element ref="ebReport:Pass" />
4689 <element ref="ebReport:Fail" />
4690 </choice>
4691 </complexType>
4692 </element>
4693 </sequence>
4694 <attribute name="description" use="required" type="string" />
4695 </complexType>
4696 </element>
4697 <element name="ErrorMessage" type="string" />
4698 <simpleType name="mimeHeader.type">
4699 <restriction base="NMTOKEN">
4700   <enumeration value="MIMEMessageContent-Type" />
4701   <enumeration value="MIMEMessageStart" />
4702   <enumeration value="Content-Type" />
4703   <enumeration value="start" />
4704   <enumeration value="charset" />
4705   <enumeration value="type" />
4706   <enumeration value="wildcard" />
4707 </restriction>
4708 </simpleType>
4709 <simpleType name="content.type">
4710 <restriction base="NMTOKEN">
4711   <enumeration value="XML" />
4712   <enumeration value="date" />
4713   <enumeration value="URI" />
4714   <enumeration value="signature" />
4715 </restriction>
4716 </simpleType>
4717 <simpleType name="method.type">
4718 <restriction base="NMTOKEN">
4719   <enumeration value="xpath" />
4720   <enumeration value="sha-1" />
4721 </restriction>
4722 </simpleType>
4723 <simpleType name="messageContext.type">
4724 <restriction base="NMTOKEN">
4725   <enumeration value="true" />
4726   <enumeration value="false" />
4727 </restriction>
4728 </simpleType>
4729 <simpleType name="requirement.type">
4730 <restriction base="NMTOKEN">
4731   <enumeration value="required" />
4732   <enumeration value="stronglyrecommended" />
4733   <enumeration value="recommended" />
4734   <enumeration value="optional" />
4735 </restriction>
4736 </simpleType>
4737 <simpleType name="testCaseResult.type">
4738 <restriction base="NMTOKEN">
4739   <enumeration value="pass" />
4740   <enumeration value="fail" />
4741   <enumeration value="untested" />
4742 </restriction>
4743 </simpleType>
4744 <simpleType name="failure.type">
4745 <restriction base="NMTOKEN">
4746   <enumeration value="preConditionTest" />
4747   <enumeration value="assertionTest" />
4748   <enumeration value="undetermined" />
4749 </restriction>
4750 </simpleType>
4751 <element name="Container">
4752 <complexType>
4753 <simpleContent>
4754 <extension base="string">
4755   <attribute name="content-id" use="optional" type="string" />
4756   <attribute name="content-location" use="optional" type="string" />
4757 </extension>
4758 </simpleContent>

```

```
4759      </complexType>
4760    </element>
4761  <element name="Message">
4762    <complexType>
4763      <sequence>
4764        <any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4765      </sequence>
4766        <attribute name="id" use="required" type="ID" />
4767      </complexType>
4768    </element>
4769  <element name="Pass">
4770    <complexType />
4771  </element>
4772  <element name="Fail">
4773    <complexType>
4774      <attribute name="failType" use="optional" type="ebReport:failure.type" />
4775    </complexType>
4776  </element>
4777  <element name="SoapFault">
4778    <complexType>
4779      <sequence>
4780        <element name="faultcode" type="QName" />
4781        <element name="faultstring" type="string" />
4782        <element name="faultactor" type="anyURI" minOccurs="0" />
4783        <element name="detail" type="NCName" minOccurs="0" />
4784      </sequence>
4785    </complexType>
4786  </element>
4787  <complexType name="detail">
4788    <sequence>
4789      <any namespace="##any" processContents="lax" minOccurs="0" maxOccurs="unbounded" />
4790    </sequence>
4791    <anyAttribute namespace="##any" processContents="lax" />
4792  </complexType>
4793  <element name="ebXMLError" type="string" />
4794</schema>
```

4795

4796 **Appendix F (Normative) Service Related Message**
4797 **Schema**

4798

```
4799 <?xml version="1.0" encoding="UTF-8" ?>
4800   <!--
4801     Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
4802       -->
4803   <xsd:schema xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
4804     2_0.xsd" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
4805     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4806     <xsd:import namespace="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
4807       header-2_0.xsd" schemaLocation="http://www.oasis-open.org/committees/ebxml-
4808       msg/schema/msg-header-2_0.xsd" />
4809     <xsd:import namespace="http://schemas.xmlsoap.org/soap/envelope/"
4810       schemaLocation="http://schemas.xmlsoap.org/soap/envelope/" />
4811     <xsd:element name="InitiatorRequest">
4812       <xsd:complexType>
4813         <xsd:sequence>
4814           <xsd:element name="CPAId" />
4815           <xsd:element name="ConversationId" />
4816           <xsd:element name="MessageId" />
4817         </xsd:sequence>
4818       </xsd:complexType>
4819     </xsd:element>
4820     <xsd:element name="PayloadVerifyResponse">
4821       <xsd:complexType>
4822         <xsd:sequence>
4823           <xsd:element name="Result" type="xsd:boolean" />
4824         </xsd:sequence>
4825       </xsd:complexType>
4826     </xsd:element>
4827     <xsd:element name="ErrorAppNotifyResponse">
4828       <xsd:complexType>
4829         <xsd:sequence>
4830           <xsd:element ref="eb:ErrorList" maxOccurs="unbounded" />
4831         </xsd:sequence>
4832       </xsd:complexType>
4833     </xsd:element>
4834     <xsd:element name="ErrorURLNotifyResponse">
4835       <xsd:complexType>
4836         <xsd:sequence>
4837           <xsd:element ref="eb:ErrorList" maxOccurs="unbounded" />
4838         </xsd:sequence>
4839       </xsd:complexType>
4840     </xsd:element>
4841     <xsd:element name="ConfiguratorRequest">
4842       <xsd:complexType>
4843         <xsd:sequence>
4844           <xsd:choice minOccurs="0">
4845             <xsd:element ref="CPAId" />
4846             <xsd:element ref="CPA" />
4847           </xsd:choice>
4848             <xsd:element ref="Mode" minOccurs="0" />
4849             <xsd:element name="ResponseURL" minOccurs="0" />
4850           </xsd:sequence>
4851         </xsd:complexType>
4852       </xsd:element>
4853     <xsd:element name="ConfiguratorResponse">
4854       <xsd:complexType>
4855         <xsd:sequence>
4856           <xsd:element ref="Status" minOccurs="0" />
4857         </xsd:sequence>
4858       </xsd:complexType>
4859     </xsd:element>
```

```
4860 <xsd:element name="CPA">
4861   <xsd:complexType>
4862     <xsd:sequence>
4863       <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"
4864     />
4865   </xsd:sequence>
4866   </xsd:complexType>
4867 </xsd:element>
4868   <xsd:element name="Status" type="xsd:boolean" />
4869   <xsd:element name="CPAId" type="non-empty-string" />
4870   <xsd:element name="Mode" type="non-empty-string" />
4871   <xsd:element name="ConversationId" type="non-empty-string" />
4872   <xsd:element name="MessageId" type="non-empty-string" />
4873 <xsd:simpleType name="non-empty-string">
4874   <xsd:restriction base="xsd:string">
4875     <xsd:minLength value="1" />
4876   </xsd:restriction>
4877 </xsd:simpleType>
4878 <xsd:element name="TestMessage">
4879   <xsd:complexType>
4880     <xsd:choice>
4881       <xsd:element ref="InitiatorRequest" />
4882       <xsd:element ref="PayloadVerifyResponse" />
4883       <xsd:element ref="ConfiguratorRequest" />
4884       <xsd:element ref="ConfiguratorResponse" />
4885     </xsd:choice>
4886   </xsd:complexType>
4887 </xsd:element>
4888 </xsd:schema>
```

```
4889
4890
```

4891

Appendix G Terminology

4892 Several terms used in this specification are borrowed from the Conformance Glossary (OASIS,
 4893 [ConfGlossary]) and also from the Standards and Conformance Testing Group at NIST.
 4894 [ConfCertModelNIST]. They are not reported in this glossary, which only reflects (1) terms that are
 4895 believed to be specific to – and introduced by - the ebXML Test Framework, or (2) terms that have a well
 4896 understood meaning in testing literature (see above references) and may have additional properties in the
 4897 context of the Test Framework that is worth mentioning.

4898

<u>Term</u>	<u>Definition</u>
Asymmetric testing	Interoperability testing where all parties are not equally tested for the same features. An asymmetric interoperability test suite is typically driven from one party, and will need to be executed from every other party in order to evenly test for all interoperability features between candidate parties.
Base CPA	Required by both the conformance and interoperability test suites that describe both the Test Driver and Test Service Collaboration Protocol Profile Agreement. This is the “bootstrap” configuration for all messaging between the testing and candidate ebXML applications. Each test suite will define additional CPAs. How the base CPA is represented to applications is implementation specific.
Candidate Implementation	(or Implementation Under test): The implementation (realization of a specification) used as a target of the testing (e.g. <u>conformance testing</u>).
Conformance	Fulfillment of an implementation of all requirements specified; adherence of an implementation to the requirements of one or more specific standards or specifications.
Connection mode (Test Driver in)	In connection mode and depending on the test harness, the test driver will interact with other components by directly generating ebXML messages at transport level (e.g. generates HTTP envelopes).
Interoperability profile	A set of test requirements for interoperability which is a subset of all possible interoperability requirements, and which usually exercises features that correspond to specific user needs.
Interoperability Testing	Process of verifying that two implementations of the same specification, or that an implementation and its operational environment, can interoperate according to the requirements of an assumed agreement or contract. This contract does not belong necessarily to the specification, but its terms and elements should be defined in it with enough detail, so that such a contract, combined with the specification, will be sufficient to determine precisely the expected behavior of an implementation, and to test it.
Local Reporting mode (Test Service in)	In this mode (a sub-mode of Reporting), the Test Service is installed on the same host as the Test Driver it reports to, and executes in the same process space. The notification uses the <i>Receive</i> interface of the Test Driver, which must be operating in service mode.

Loop mode (Test Service in)	When a test service is in loop mode, it does not generate notifications to the test driver. The test service only communicates with external parties via the message handler.
MSH	Message Service Handler, an implementation of ebXML Messaging Services
Reporting mode (Test Service in)	A test service is deployed in reporting mode, when it notifies the test driver of invoked actions. This notification usually contains material from received messages.
Profile	A profile is used as a method for defining subsets of a specification by identifying the functionality, parameters, options, and/or implementation requirements necessary to satisfy the requirements of a particular community of users. Specifications that explicitly recognize profiles should provide rules for profile creation, maintenance, registration, and applicability.
Remote Reporting mode (Test Service in)	In this mode (a sub-mode of Reporting), the Test Service is deployed on a different host than the Test Driver it reports to. The notification is done via messages to the Test Driver, which is operating in connection mode.
Service mode (Test Driver in)	The Test Driver invokes actions in the test service via a programmatic interface (as opposed to via messages). The Test Service must be in local reporting mode.
Specification coverage	Specifies the degree that the specification requirements are satisfied by the set of test requirements included in the test suite document. Coverage can be full, partial or none.
Test actions	(Or Test Service actions). Standard functions available in the test service to support most test cases.
Test case	In the TestFramework, a test case is a sequence of discrete test steps, aimed at verifying a test requirement.
Test Requirements coverage	Specifies the degree that the test requirements are satisfied by the set of test cases listed in the test suite document. Coverage can be full, contingent, partial or none.

4899

4900

4901

4902

4903

4904

4905

4906

4907

4908

4909

4910

4911

4913 Appendix H References

4914

4915 H.1 Normative References

4916

- 4917 [ConfCertModelNIST] Conformance Testing and Certification Model for Software Specifications. L. Carnahan, L. Rosenthal, M. Skall. ISACC '98 Conference. March 1998
- 4919 [ConfCertTestFrmk] Conformance Testing and Certification Framework. L. Rosenthal, M. Skall, L. Carnahan. April 2001
- 4921 [ConfReqOASIS] Conformance Requirements for Specifications. OASIS Conformance Technical Committee. March 2002.
- 4923 [ConfGlossary] Conformance Glossary. OASIS Conformance TC, L. Rosenthal. September 2000.
- 4924 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task Force, March 1997
- 4926 [RFC2045] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, N Freed & N Borenstein, Published November 1996
- 4928 [RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N. Borenstein. November 1996.
- 4930 [RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.
- 4931 [RFC2392] Content-ID and Message-ID Uniform Resource Locators. E. Levinson, August 1998
- 4932 [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, August 1998
- 4933 [RFC2821] Simple Mail Transfer Protocol, J. Klensin, Editor, April 2001 Obsoletes RFC 821
- 4934 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol, HTTP/1.1", June 1999.
- 4936 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08 May 2000,
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
- 4941 [SOAPAttach] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000
<http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211>
- 4944 [XLINK] W3C XML Linking Recommendation, <http://www.w3.org/TR/2001/REC-xlink-20010627/>
- 4945 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition), October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>
- 4947 [XMLEC14N] W3C Recommendation Canonical XML 1.0,
<http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- 4949 [XMLNS] W3C Recommendation for Namespaces in XML, World Wide Web Consortium, 14 January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>
- 4951 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,
<http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/>.

4953 [XPointer] XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11
4954 September 2001, <http://www.w3.org/TR/2001/CR-xptr-20010911/>

4955

4956 **H.2 Non-Normative References**

4957 [ebTestFramework] ebXML Test Framework specification, Version 1.0, Technical Committee
4958 Specification, March 4, 2003,
4959 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic

4960 [ebMS] ebXML Messaging Service Specification, Version 2.0,
4961 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-msg

4962 [ebMSInteropTests] ebXML MS V2.0 Basic Interoperability Profile Test Cases,
4963 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic

4964 [ebMSConfTestSuite] ebXML MS V2.0 Conformance Test Suite,
4965 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic

4966 [ebMSInteropReqs] ebXML MS V2.0 Interoperability Test Requirements,
4967 http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic

4968

4969 [XMLSchema] W3C XML Schema Recommendation,
4970 <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>
4971 <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>
4972 <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

4973 [ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 1.0,
4974 published 10 May, 2001,
4975 <http://www.ebxml.org/specs/ebCPP.doc>

4976 [ebBPSS] ebXML Business Process Specification Schema, version 1.0, published 27 April 2001,
4977 <http://www.ebxml.org/specs/ebBPSS.pdf>.

4978 [ebRS] ebXML Registry Services Specification, version 2.0, published 6 December 2001
4979 <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf>,
4980 published, 5 December 2001.

4981

4982 Appendix I Acknowledgments

4983 The authors wish to acknowledge the support of the members of the OASIS ebXML IIC TC who
4984 contributed ideas, comments and text to this specification by the group's discussion eMail list, on
4985 conference calls and during face-to-face meetings.

4986 I.1 IIC Committee Members

4987 Jacques Durand, Fujitsu <jdurand@fsw.fujitsu.com>
4988 Jeffery Eck, Global Exchange Services <Jeffery.Eck@gxs.ge.com>
4989 Hatem El Sebaaly, IPNet Solutions <hatem@ipnetsolutions.com>
4990 Aaron Gomez, Drummond Group Inc. <aaron@drummondgroup.com>
4991 Michael Kass, NIST <michael.kass@nist.gov>
4992 Matthew MacKenzie, Individual <matt@mac-kenzie.net>
4993 Monica Martin, Sun Microsystems <monica.martin@sun.com>
4994 Tim Sakach, Drake Certivo <tsakach@certivo.net>
4995 Jeff Turpin, Cyclone Commerce <jturpin@cyclonecommerce.com>
4996 Eric van Lydegraf, Kinzan <ericv@kinzan.com>
4997 Pete Wenzel, SeeBeyond <pete@seebeyond.com>
4998 Steven Yung, Sun Microsystems <steven.yung@sun.com>
4999 Boonserm Kulvatunyou, NIST <serm@nist.gov>

5000
5001
5002
5003
5004
5005
5006
5007
5008
5009
5010
5011
5012
5013
5014
5015
5016
5017
5018

5019

Appendix J Revision History

5020

Rev	Date	By Whom	What
cs-10	2003-03-07	Michael Kass	Initial version

5021

Appendix K Notices

5022 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
5023 might be claimed to pertain to the implementation or use of the technology described in this document or
5024 the extent to which any license under such rights might or might not be available; neither does it
5025 represent that it has made any effort to identify any such rights. Information on OASIS's procedures with
5026 respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights
5027 made available for publication and any assurances of licenses to be made available, or the result of an
5028 attempt made to obtain a general license or permission for the use of such proprietary rights by
5029 implementors or users of this specification, can be obtained from the OASIS Executive Director.

5030 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications,
5031 or other proprietary rights which may cover technology that may be required to implement this
5032 specification. Please address the information to the OASIS Executive Director.

5033 **Copyright © OASIS Open 2003. All Rights Reserved.**

5034 This document and translations of it may be copied and furnished to others, and derivative works that
5035 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
5036 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice
5037 and this paragraph are included on all such copies and derivative works. However, this document itself
5038 does not be modified in any way, such as by removing the copyright notice or references to OASIS,
5039 except as needed for the purpose of developing OASIS specifications, in which case the procedures for
5040 copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to
5041 translate it into languages other than English.

5042 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
5043 or assigns.

5044 This document and the information contained herein is provided on an "AS IS" basis and OASIS
5045 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
5046 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
5047 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

5048