



Creating A Single Global Electronic Market

1  
2  
3



4

---

## **ebXML Test Framework**

5

### **Committee Specification Version 1.1 DRAFT**

6

7

### **OASIS ebXML Implementation, Interoperability and Conformance Technical Committee**

8  
9

**14 July, 2004**

10

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

13 **Location:**  
14 [http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=ebxml-iic](http://www.oasis-open.org/committees/documents.php?wg_abbrev=ebxml-iic)

15 **Authors/Editors:**  
16  
17 Michael Kass, NIST <[michael.kass@nist.gov](mailto:michael.kass@nist.gov)>  
18  
19

20 **Contributors:**  
21 Steven Yung, Sun Microsystems <[steven.yung@sun.com](mailto:steven.yung@sun.com)>  
22 Prakash Sinha, IONA <[prakash.sinha@iona.com](mailto:prakash.sinha@iona.com)>  
23 Matthew MacKenzie, Individual <[matt@mac-kenzie.net](mailto:matt@mac-kenzie.net)>  
24 Hatem El Sebaaly, IPNet Solutions <[hatem@ipnetsolutions.com](mailto:hatem@ipnetsolutions.com)>  
25 Monica Martin, Sun Microsystems <[monica.martin@sun.com](mailto:monica.martin@sun.com)>  
26 Jacques Durand, Fujitsu <[jdurand@fsw.fujitsu.com](mailto:jdurand@fsw.fujitsu.com)>  
27 Christopher Frank <[C.Frank@seeburger.de](mailto:C.Frank@seeburger.de)>  
28 Eric VanLydegraf, Kinzan <[ericv@kinzan.com](mailto:ericv@kinzan.com)>  
29 Jeff Turpin, CycloneCommerce <[jturpin@cyclonecommerce.com](mailto:jturpin@cyclonecommerce.com)>  
30 Serm Kulvatunyou, NIST <[serm@nist.gov](mailto:serm@nist.gov)>  
31 Tim Sakach, Drake Certivo, Inc. [tsakach@certivo.net](mailto:tsakach@certivo.net)  
32 Hyunbo Cho, Postech [hcho@postech.ac.kr](mailto:hcho@postech.ac.kr)  
33 Han Kim Ngo, NIST <[han.ngo@nist.gov](mailto:han.ngo@nist.gov)>

34 **Abstract:**  
35 This document specifies ebXML interoperability testing specification for the eBusiness  
36 community.

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

46  
47 **Errata to this version:**  
48 None  
49

50

---

# 51 Table of Contents

52	1	Introduction.....	5
53	1.1	Summary of Contents of this Document.....	5
54	1.2	Document Conventions .....	5
55	1.3	Audience.....	5
56	1.4	Caveats and Assumptions.....	6
57	1.5	Related Documents .....	6
58	1.6	Minimal Requirements for Conformance.....	6
59	2	Principles and Methodology of Operations .....	8
60	2.1	General Objectives .....	8
61	2.2	General Methodology .....	9
62	3	The Test Framework Components.....	11
63	3.1	The Test Driver .....	11
64	3.1.1	Functions .....	11
65	3.1.2	Using the Test Driver in Connection Mode .....	13
66	3.1.3	Using the Test Driver in Service Mode.....	14
67	3.2	The Test Service.....	16
68	3.2.1	Functions and Interactions .....	16
69	3.2.2	Modes of Operation of the Test Service.....	18
70	3.2.3	Configuration Parameters of the Test Service .....	19
71	3.2.4	The Messaging Actions of the Messaging Services Test Service .....	20
72	3.2.5	Interfaces for Test Driver and Test Service.....	23
73	3.3	Executing Test Cases.....	30
74	3.3.1	A Typical Execution Scenario.....	30
75	3.3.2	Test Case as a Workflow of Threads .....	31
76	3.3.3	Related Message Data and Declarations.....	32
77	3.3.4	Related Testing Configuration Data .....	32
78		Part II: Test Suite Representation.....	34
79	4	Test Suite .....	35
80	4.1	Conformance vs. Interoperability Test Suite.....	35
81	4.2	The Test Suite Document.....	36
82	4.2.1	Test Suite Metadata .....	38
83	4.2.2	The ConfigurationGroup.....	39
84	4.2.3	The TestServiceConfigurator Operation .....	44
85	5	Test Requirements .....	46
86	5.1	Purpose and Structure.....	46
87	5.2	The Test Requirements Document.....	46
88	5.2.1	.....	47
89	5.3	Specification Coverage.....	49
90	5.4	Test Requirements Coverage (or Test Run-Time Coverage) .....	50
91	6	Test Profiles.....	52
92	6.1	The Test Profile Document.....	52
93	6.2	Relationships between Profiles, Requirements and Test Cases.....	53

94	7	Test Cases .....	55
95	7.1	Structure of a Test Case.....	55
96	7.1.1	Test Threads .....	58
97	7.1.2	Thread Operations .....	60
98	7.1.3	Message Store Schema .....	81
99	7.3	Test Service Configurator, Initiator, and Notification Message Formats.....	85
100	7.4	Test Report Schema .....	91
101	8	Test Material.....	93
102	8.1.1	Testing Profile Document.....	93
103	8.1.2	Test Requirements Document.....	93
104	8.1.3	Test Suite Document.....	93
105	8.1.4	Mutator documents.....	94
106	8.1.5	CPAs .....	94
107	9	Test Material Examples.....	95
108	9.1	Example Test Requirements .....	95
109	9.1.1	Conformance Test Requirements .....	95
110	9.1.2	Interoperability Test Requirements .....	97
111	9.2	Example Test Profiles.....	98
112	9.2.1	Conformance Test Profile Example.....	98
113	9.2.3	Interoperability Test Profile.....	99
114	9.3	Example Test Suites.....	99
115	9.3.1	Conformance Test Suite.....	99
116	9.3.2	Interoperability Test Suite.....	101
117	9.3.3	A sample Mutator XSL Document.....	103
118		Appendix A (Normative) The ebXML Test Profile Schema.....	104
119		Appendix B (Normative) The ebXML Test Requirements Schema .....	106
120		Appendix C (Normative) The ebXML Test Suite Message Declaration Schema and Supporting	
121		Subschemas .....	110
122		Appendix D (Normative) The ebXML Message Store Schema (and supporting sub-schemas) .....	137
123		Appendix E (Normative) The Test Report Schema .....	158
124		Appendix F (Normative) ebXML Test Service Message Schema.....	168
125		Appendix G WSDL Definitions for Test Service.....	174
126		Appendix H Terminology.....	178
127		Appendix I References.....	181
128	I.1	Normative References .....	181
129	I.2	Non-Normative References .....	182
130		Appendix J Acknowledgments .....	183
131	J.1	IIC Committee Members.....	183
132		Appendix K Revision History.....	184
133		Appendix L Notices .....	185
134			

---

# 135 1 Introduction

136

## 137 1.1 Summary of Contents of this Document

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

141 This specification is organized around the following topics:

- 142 • Interoperability testing architecture
- 143 • Test cases for basic interoperability
- 144 • Test data materials

145

## 146 1.2 Document Conventions

147 Terms in *Italics* are defined in the Definition of Terms in Appendix H. Terms listed in **Bold Italics**  
148 represent the element and/or attribute content. Terms listed in `Courier` font relate to test data. Notes  
149 are listed in Times New Roman font and are informative (non-normative). Attribute names begin with  
150 lowercase. Element names begin with Uppercase.

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

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

172

## 173 1.3 Audience

174 The target audience for this specification is:

- 175 • The community of software developers who implement and/or deploy the ebXML Messaging  
176 Service (ebMS) or use other ebXML technologies such as a Registry/Repository (RegRep),

177 Collaboration Profile Protocol/Agreement (CPPA) or Business Process Specification Schema  
178 (BPSS)

179

- 180 • The testing or verification authority, which will implement and deploy conformance or  
181 interoperability testing for ebXML implementations.

182

## 183 1.4 Caveats and Assumptions

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

186

## 187 1.5 Related Documents

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

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

212

## 213 1.6 Minimal Requirements for Conformance

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

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

220 It complies with the following interpretation of the keywords OPTIONAL and MAY: When these keywords  
221 apply to the behavior of the implementation, the implementation is free to support these behaviors or not,  
222 as meant in [RFC2119]. When these keywords apply to data and configuration material used by an  
223 implementation of the Test Framework, a conforming implementation of the Test Framework MUST be  
224 capable of processing these optional materials according to the described semantics.

225

226

227

228

229

230

231

232

233

234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276

---

## 2 Principles and Methodology of Operations

### 2.1 General Objectives

The OASIS IIC ebXML Test Framework is intended to support conformance and interoperability testing for ebXML (as well as other eBusiness) specifications. It describes a testbed architecture and its software components, how these can be combined to create a test harness for various types of testing. It also describes the test material to be processed by this architecture, a mark-up language and format for representing test requirements, and test suites (a set of Test Cases).

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

The Test Framework is a foundation for testing all ebXML architectural components such as Messaging, Registry, BPSS, CPA, and Core Components

Because of its generic design, the Test Framework is flexible enough to permit testing beyond ebXML message format, to include XML message envelope and payload testing of any e-Business messaging service

Test Suites and Test Cases that are related to these standards, are defined in a formal manner. They can be automatically processed by the Test Framework, and their execution can easily be reproduced.

The harnessing of an ebXML implementation (or possibly several implementations, in case of interoperability testing) with the Test Framework requires a moderate effort. It generally requires some interfacing work specific to an implementation, in the case no standard interface (API) has been specified. For example, the Test Service (a component of the Test Framework) defines Actions that will need to be called by a particular MSH implementation for ebXML Messaging Services conformance testing. Additionally, MS interoperability testing would require the interfaces defined in section 3.5.5.

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

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

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



277 The verification of each Test Case is done by the Test Driver at run-time, as soon as the Test Case  
278 execution is completed. The outcome of the verification can be obtained as the Test Suite has completed,  
279 and a verification report is generated.

280

## 281 **2.2 General Methodology**

282

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

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

293

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

297

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

302

303 • **Phase 2: Test Requirements Design (MANDATORY)**. A list of Test Requirements is established  
304 for the tested specification, and for the profile/level of conformance/interoperability that is  
305 targeted. These Test Requirements **MUST** refer to the specification document. Jointly to this list,  
306 it is **RECOMMENDED** that Specification Coverage be reported. This document shows, for each  
307 feature in the original specification, the Test Requirements items that address this feature. It also  
308 estimates to which degree the feature is validated by these Test Requirements items.

309

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

315

316 • Phase 4: **Test Suite Design** (MANDATORY). Each Test Requirement from Phase 2 is translated  
317 into one or more Test Cases. A Test Case is defined as a sequence of operations over the Test  
318 Harness. Each Test Case includes: configuration material (CPA data), message material  
319 associated with each operation and test verification conditions that define criteria for passing this  
320 test. All this material, along with any particular operation directives, defines a Test Suite. In order  
321 to be conforming to this specification, a test suite needs to be described as a document (XML)  
322 conforming to part II of this specification.

323

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

328

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

331

332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
  
345  
  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
  
356  
  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371

---

## 3 The Test Framework Components

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

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

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

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

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

### 3.1 The Test Driver

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

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

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

#### 3.1.1 Functions

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

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

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

**Test Service Configuration** – Based upon supplied configuration parameters, Test Service configuration may be done at startup via remote procedure call.

**ebXML (or other type) Message Construction** – Includes any portion of the message, including message envelope and optional message payloads

372 **Persistence of (received) Messages** –received messages MUST persist for the life of a Test Case.  
 373 Persistent messages MUST validate to the ebXMLMessageStore.xsd schema in Appendix D.

374 **Parse and query persistent messages** – Test Driver MUST use XPath query syntax to query persistent  
 375 message content

376 **Parse and query message payloads** – Test Driver MUST support XPath query syntax to query XML  
 377 message payloads of persistent messages.

378 **Control the execution and workflow of the steps of a Test Case.** Some steps may be executed by  
 379 other components, but their initiation is under control of the Test Driver.

380 **Repeat previously executed Test operations**– Test Driver MUST be capable of repeating previously  
 381 executed message requests for the current Test Case.

382 **Send messages through the Test Driver** - Directly at transport layer (e.g. by opening an HTTP  
 383 connection).

384 **Send messages through the Test Service** – Locally (if coupled with the Test Service) or via remote  
 385 procedure call (if not directly interface with the Test Service)

386 **Receive messages** - Either directly at transport layer, or by notification from Test Service actions.

387 **Perform discreet message content validation** – Test Driver MUST be capable of performing discreet  
 388 validation of Time, URI, Signature and the entire XML message

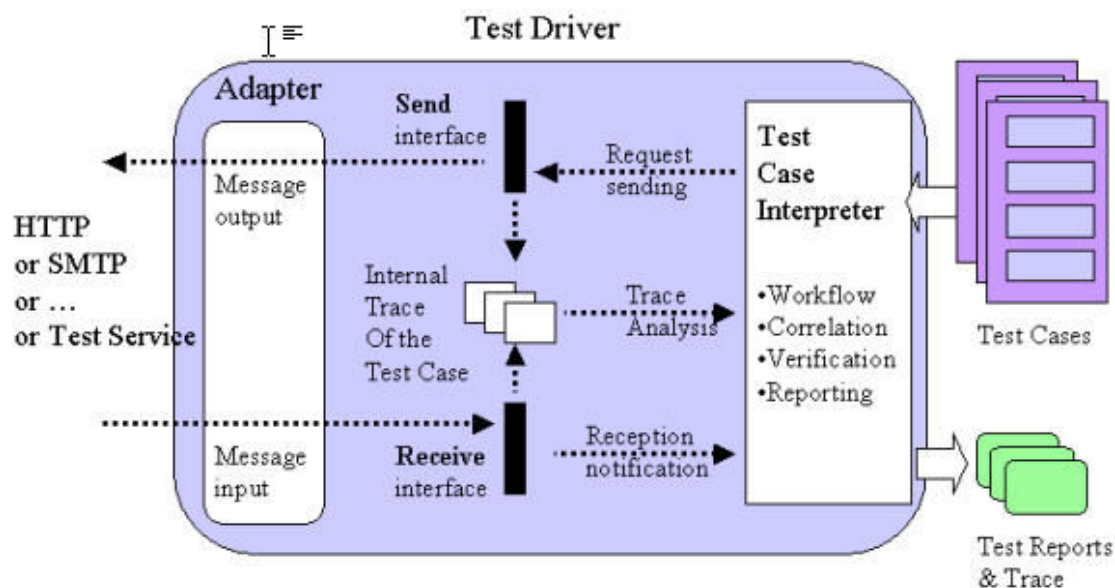
389 **Perform discreet payload content validation** – Test Driver MUST be capable of performing discreet  
 390 validation of Time, URI, Signature and an XML payload

391 **Report Test Results** – Test Driver MUST generate an XML test report for all executed tests in the profile.  
 392 Reports MUST validate to the ebXMLTestReport.xsd schema in Appendix E.

393

394

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



396  
 397 Figure 1- The Test Driver: Functions and Data Flows  
 398  
 399

400  
401

### 402 3.1.2 Using the Test Driver in Connection Mode

403

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

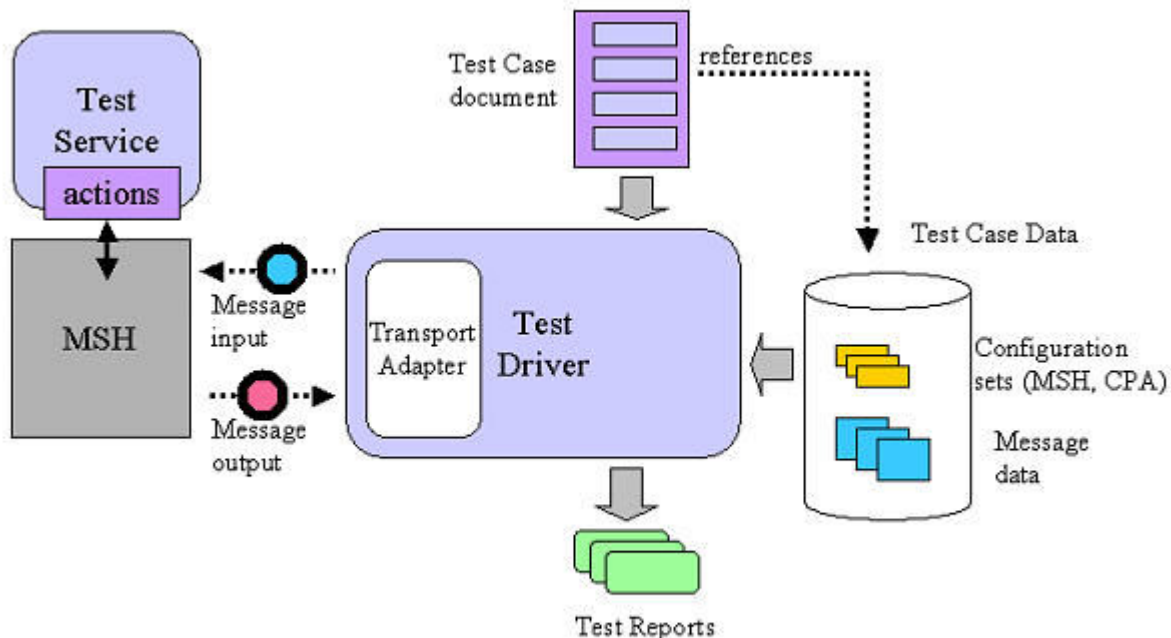
408 Create a message envelope, and generate fully formed messages for this transport.

409 Parse a message envelope and extract header data from a message, as well as from the message  
410 payload in case it is an XML document.

411 Open a message communication channel (connection) with a remote message handler. In that case the  
412 Test Driver is said to operate in connection mode.

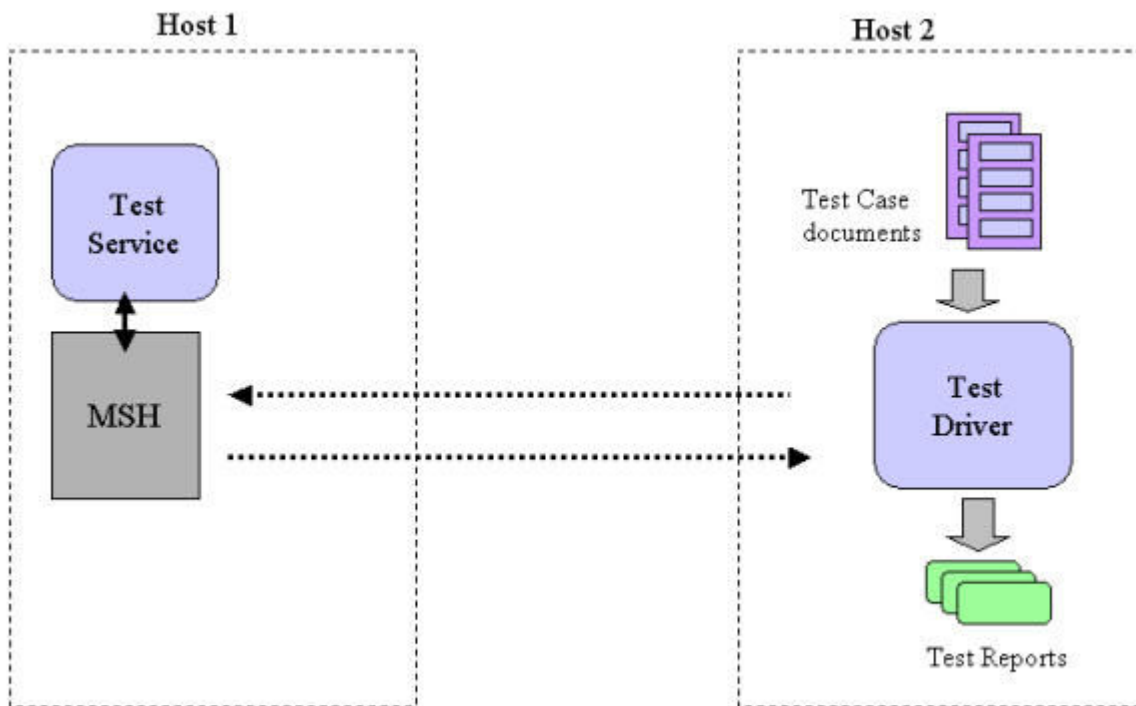
413 When used in connection mode, the Test Driver is acting as a transport end-point that can receive or  
414 send messages with an envelope consistent with the transport protocol (e.g. HTTP, SMTP, or FTP). The  
415 interaction between the MSH and the Test Service is of the same nature as the interaction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

418



419  
420 Figure 2- Test Driver: Connection Mode  
421

422  
423



424  
425 Figure 3 – Test Driver: Remote Connection Mode

### 426 3.1.3 Using the Test Driver in Service Mode

427

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

435

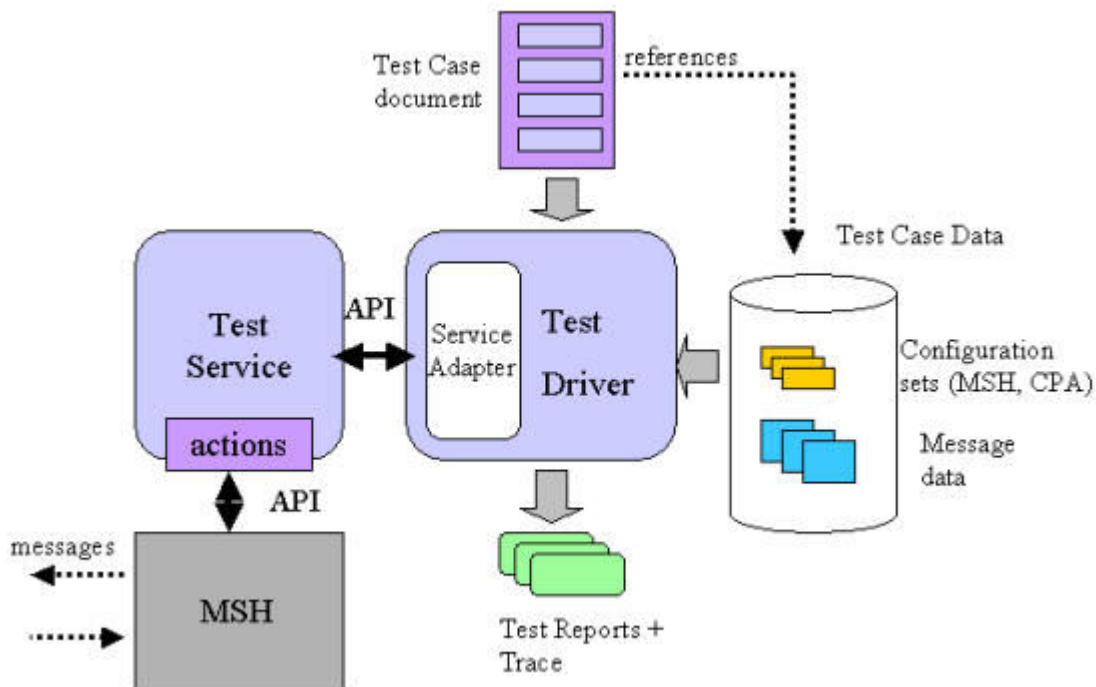
436 **Sending:** One method of the Test Service (represented by the the “Initiator” scripting element) , instructs  
437 the MSH it has been interfaced with to construct and send a message. This method also MUST interface  
438 with the Test Service at application level. When invoked by a call that contains message data, the method  
439 generates a sending request using that MSH’s API.

440

441 **Receiving:** As all actions of the Test Service may participate in the execution of a Test Case (i.e. of its  
442 Threads), the Test Driver needs to be aware of their invocation by incoming notification messages  
443 provided by the Test Service. Each of these actions notify the Test Driver through its “Receive” interface,  
444 passing received message data, as well as response data. In this way, the Test Driver builds an internal  
445 trace (or state) for the Test Case execution, and is able to verify the test based on this data.

446

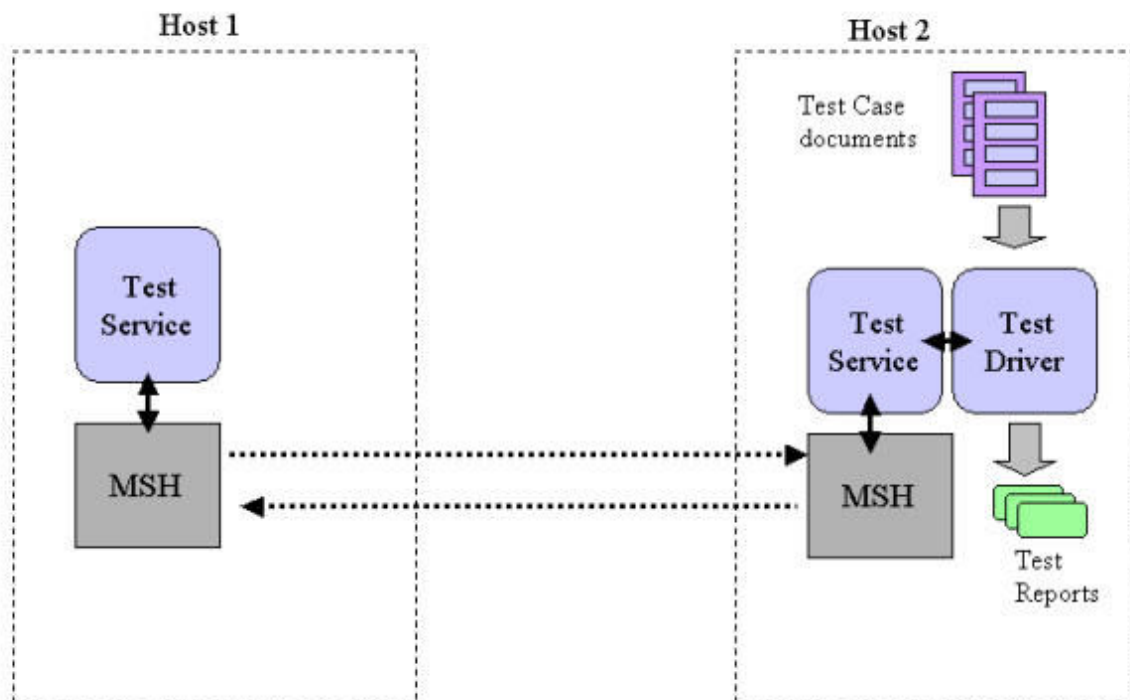
447 The Test Driver MUST support the above communication operations with the Test Service when in  
 448 Service Mode. This may be achieved by using an embedded Service Adapter to bridge the sending and  
 449 receiving functions of the Test Driver, with the Service/Action calls of the Test Service. Figure 4 illustrates  
 450 how the Test Driver operates with a Service Adapter. Alternately, the Test Service MAY notify the Test  
 451 Driver of receive messages and errors via RPC.  
 452



453  
 454 Figure 4 – Test Driver: Service Mode  
 455

456 This design allows for a minimal exposure of the MSH-specific API, to the components of the Test  
 457 Framework. The integration code that needs to be written for connecting the MSH implementation is then  
 458 restricted to an interface with the Service/Actions defined by the framework. Neither the Test Driver, nor  
 459 the Service Adapter, need to be aware of the MSH-specific interface. An example of test harness using  
 460 the Test Driver in Service Mode is shown in Figure 5.  
 461  
 462





463

464 Figure 5 – Test Driver in Service Mode: Point-to-Point Interoperability Testing of Message Handlers

465

## 466 3.2 The Test Service

467

### 468 3.2.1 Functions and Interactions

469

470 The Test Service defines a set of Actions that are useful for executing Test Cases. The Test Service  
 471 represents the application layer for a message handler. It receives message content and error  
 472 notifications from the MSH, and also generates requests to the MSH, which normally are translated into  
 473 messages to be transmitted. The Test Actions are predefined, and are part of the Test Framework). For  
 474 ebXML Messaging Services testing, Service and Actions will map to the Service and Action header  
 475 attributes of ebXML messages generated during the testing.

476

477 For ebXML Messaging Services testing, the Test Service name MUST be: `urn:ebXML:iic:test`.

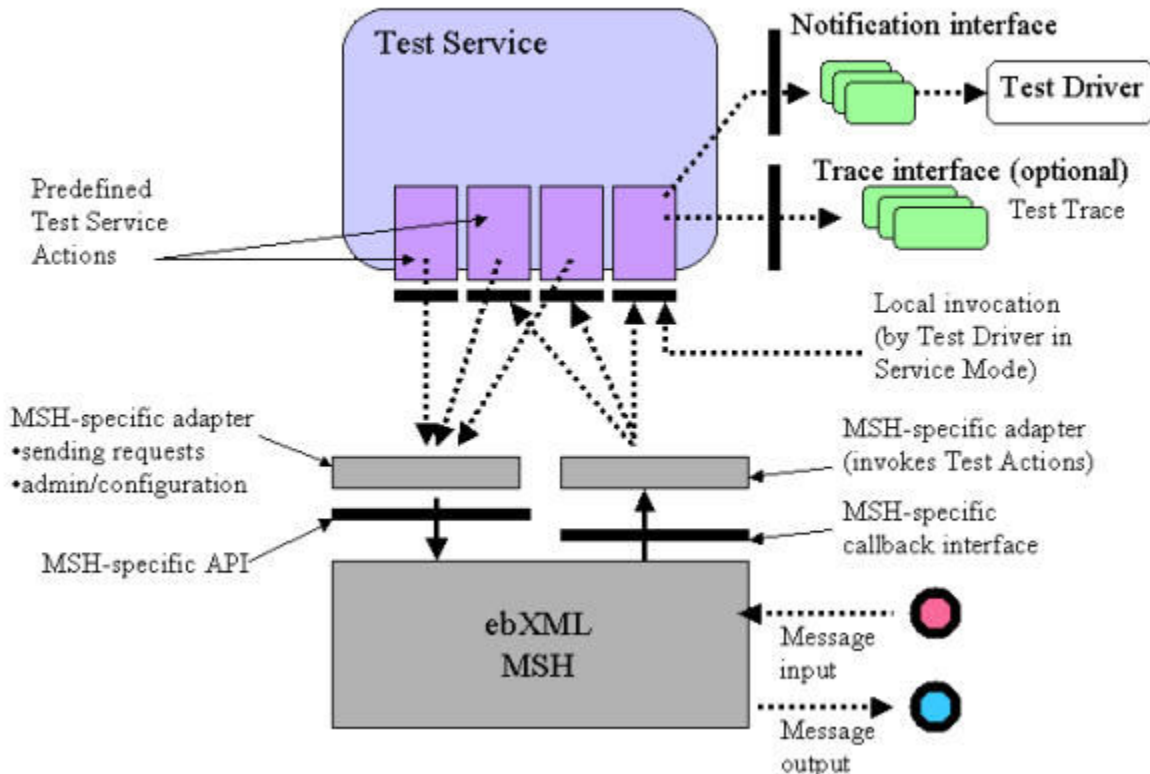
478

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

480

481





482  
483 Figure 6 – The Test Service and its Interfaces

484  
485 The functions of the Test Service are:

- 486
- 487 To implement the actions which map to Service / Action fields in a message header. The set of test
  - 488 actions which are pre-defined in the Test Service will perform diverse functions, which are enumerated
  - 489 below:
  - 490 To notify the Test Driver of incoming messages. This only occurs when the Test Service is deployed in
  - 491 *reporting mode*, which assumes it is coupled with a Test Driver either locally, or via RPC.
  - 492 To perform some message processing, e.g. compare a received message payload with a reference
  - 493 payload (or their digests).
  - 494 To send back a response to the MSH. Depending on the action invoked, the response may range from a
  - 495 pre-defined acknowledgment to a specific message as previously specified.
  - 496 Optionally, to generate a trace of its operations, in order to help trouble-shooting, or for reporting purpose.

497  
498  
499 Although the Test Service simulates an application, it is part of the Test Framework, and does not vary  
500 from one test harness to the other. However, in order to connect to the Test Service, a developer will  
501 have to write wrapper code to the Test Service/Actions that is specific to the MSH implementation that  
502 needs to be integrated. This proprietary code is expected to require a minor effort, but is necessary as the  
503 API and callback interfaces of each MSH are not specified in the [ebMS] standard and is implementation-  
504 dependent.

505

506  
507

### 508 3.2.2 Modes of Operation of the Test Service

509

510 The Test Service operates in two modes: Reporting or Loop mode

511

512 **Reporting mode:** in that mode, the actions of the Test Service instance, when invoked, will send a  
513 notification to the Test Driver. The Test Driver will then be aware of the workflow of the test case. There  
514 are two “sub-modes” of behavior:

515

516 **Local Reporting Mode:** The Test Driver is installed on the same host as the Test Service, and executes  
517 in the same process space. The notification uses the *Receive* interface of the Test Driver, which is  
518 operating in service mode.

519 **Remote Reporting Mode:** The Test Driver is installed on a different host than the Test Service. The  
520 notification is done via messages to the Test Driver, which is operating in connection mode.

521

522 **Loop mode:** in this mode, the actions of the Test Service instance, when invoked, will NOT send a  
523 notification to the Test Driver. The only interaction of the Test Service with external parties, is by sending  
524 back messages via the message handler

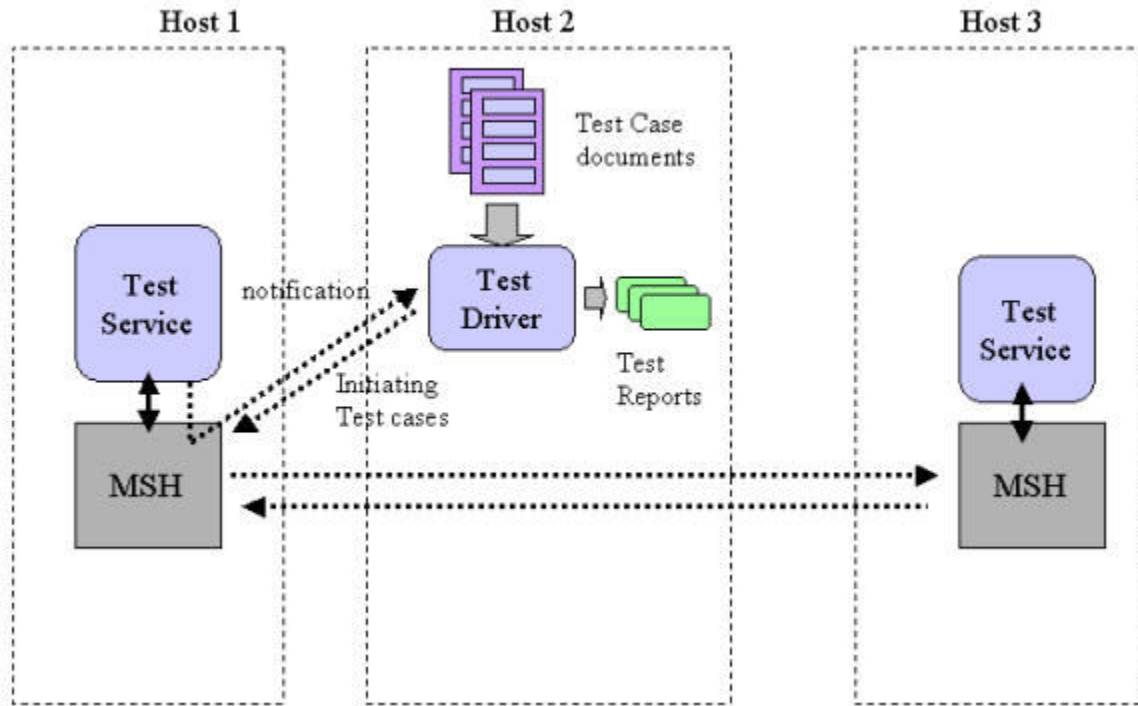
525

526 The Test Service actions operate similarly in both reporting and loop modes. In other words, the mode of  
527 operation does not normally affect the logic of the action. The action may send a response message, to  
528 the requesting party via the “ResponseURL”. In general, the ResponseURL is the same as the requestor  
529 URL.

530

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

536 parties.



537

538

539 Figure 7 – Example of Remote Reporting Mode : The Interoperability Test Center Model

540

541

542

543

544

545

546

547

548

### 549 3.2.3 Configuration Parameters of the Test Service

550

551 Test Service configuration is initially performed when the Test Driver reads the executable Test Suite  
552 XML document, and sends the TestServiceConfigurator element content found at the beginning of the  
553 TestSuite document to the Test Service via its Configuration interface. If the Test Driver is unable to  
554 configure the Test Service, then the Test Driver MUST generate an exception. The Test Driver MAY  
555 handle this exception in a “non-fatal” manner if the Test Service provides an alternate means of initial  
556 configuration.

557

558 Test Service configuration parameters are defined as content within the TestServiceConfiguration  
559 element. There are three parameters that MUST be present to configure the Test Service, and one  
560 “optional” parameter type. The three REQUIRED parameters are:

561

562

563

564

565 OperationMode (either local-reporting, remote-reporting or loop mode)

566 ResponseURL (destination for response messages in any mode)

567 NotificationURL (destination for notification messages, if in local or remote reporting mode)

568

569 Additionally, the content of the PayloadDigests element MAY be passed to the Test Service. These  
570 values are used by the PayloadVerify Test Service action to assert whether a received message payload  
571 is unchanged when received by the MSH.

572

573

574 Outside of these four parameters, the Test Service is considered “stateless”.

575 Test Service configuration MAY be performed locally, if in the same program space as the Test Service.

576 Test Service configuration MUST be performed via RPC to the Test Service Configuration interface’s  
577 “configurator” method if it is in “connection” mode.

578

579

580 In a test harness where an interoperability test suite involves two parties, the test suite (and Test Service  
581 Configuration) will need to be executed twice - alternatively driven from each party. In that case, each  
582 Test Service instance will alternatively be set to a reporting mode (local or remote), while the other will be  
583 set to loop mode. These settings can be set remotely via RPC call to the configurator method of the Test  
584 Service.

585

586

587

588

589

590

591

### 592 3.2.4 The Messaging Actions of the Messaging Services Test Service

593

594 The actions described here are required of the Test Service when performing messaging services testing,  
595 and should suffice in supporting most messaging Test Cases. In the case of ebXML Messaging Services  
596 testing, these actions map to the Service/Action field of a message, and will be triggered on reception of  
597 messages containing these service/action names. However, these actions are generic enough to be  
598 used for any business messaging service.

599

#### 600 3.2.4.1 Common Functions

601

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

604

- 605 • **Generate a response message.** Response messages are destined to the ResponseURL .  
606 They also specify a Service/Action, as they are usually intended for another Test Service  
607 although in case the ResponseURL directly points to the Test Driver in connection mode,  
608 Service/Action will not have the regular MSH semantics.
- 609 • **Notify the Test Driver.** This assumes the Test Service is coupled with a Test Driver. In that  
610 configuration, the Test Service is in reporting mode. The reporting is done by a message  
611 (sent to the Notification URL) when in remote reporting mode, or by a call to the Receive  
612 interface when in local reporting mode.

613

614

### 615 3.2.4.2 Test Service Actions

616

617 The Test Service actions defined below are “generic” types of actions that can be implemented for any  
618 type of messaging service. Specific details regarding Service, Action, MessageID and other elements are  
619 requirements specific to testing ebXML MS. In order to implement these actions for other types of  
620 messaging services the “equivalent” message content would require manipulation. The ebXML test  
621 actions are:

622

#### 623 3.2.4.2.1 Mute action

624

625 Reporting/Loop Mode Action Description: This is an action that does not generate any response  
626 message back. Such an action is used for messages that do not require any effect, except possibly to  
627 cause some side effect in the MSH, for example generating an error.

628 Response Destination: None

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

634

#### 635 3.2.4.2.2 Dummy action

636

637 Reporting/Loop Mode Action Description: This is an action that generates a simple response. On  
638 invocation, this action will generate a canned response message back (no payload, simplest header with  
639 minimally required message content), with no dependency on the received message, except for the  
640 previous MessageID (for correlation) in the RefToMessageID header attribute.

641 Response Destination: A message with a **Mute** action element is sent to the Test Component (Test Driver  
642 or Service) associated with the ResponseURL. This notice serves as proof that the message has been  
643 received, although no assumption can be made on the integrity of its content.

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

647 reporting mode. The notification will report the action name (“Dummy”) and the instance ID of the Test  
648 Service.

649

#### 650 3.2.4.2.3 Reflector

651 3.2.4.2.4 Reporting/Loop Mode Action Description: On invocation, this action generates a response to a  
652 received message, by using the same message material, with minimal changes in the header:

- 653 • Swapping of the to/from parties so that the “to” is now the initial sender.
- 654 • Setting RefToMessageId to the ID of the received message.
- 655 • Removing AckRequested or SyncReply elements if any.
- 656 • All other header elements (except for time stamps) are unchanged. The conversation ID remains  
657 unchanged, as well as the CPAlD. The payload is the same as in the received message, i.e. same  
658 attachment(s).

659 Response Destination: a message with a **Mute** action element is sent to the Test Component (Test Driver  
660 or Service) associated with the ResponseURL. This action acts as a *Reflector* for the initial sending party

661 In Reporting Mode: The action also notifies the associated Test Driver. The notification containing the  
662 received header and payload(s) material, will be done via the Receive interface, if in local reporting mode,  
663 or with a message with Service / Action fields set to “urn:ebXML:iic:test”/ “**Notify**”, if in remote  
664 reporting mode. The notification will report the action name (“Reflector”) and the instance ID of the Test  
665 Service.

666

#### 667 3.2.4.2.5 Initiator action

668

669 Reporting/Loop Mode Action Description: This Test Service action is not invoked through reception of a  
670 request message. Instead, it is invoked via a local method call to the Test Services “Send” interface.  
671 This action may be initiated by a locally interfaced Test Driver, or (via RPC) by a remote Test Driver.

672 On invocation, this action generates a new message. This message may be the first message of a totally  
673 new conversation, or it may be part of an existing conversation (depending upon the message declaration  
674 provided by the Test Driver. The header of the new message can be anything that is specified by the Test  
675 Driver. For example, this action would be used to generate a “first” message of a new conversation,  
676 different from the conversation ID specified in the invoking message.

677 Response Destination: Any party defined by the Test Driver.

678 In Reporting mode: Not Applicable, since this action is invoked directly by the Test Driver only (i.e. no  
679 incoming message is received via MSH).

680

#### 681 3.2.4.2.6 PayloadVerify action

682

683 Reporting/Loop Mode Action Description: On invocation, this action will compare the payload(s) of the  
684 received message, with the expected payload. Instead of using real payloads, to be pre-installed on the  
685 site of the Test Service, it is RECOMMENDED that a digest (or signature) of the reference payloads (files)  
686 be pre-installed on the Test Service host using TestServiceConfiguration parameters supplied by the Test  
687 Driver. The PayloadVerify action will then calculate the digest of each received payload and compare with  
688 the reference digest parameter values. This action will test the service contract between application and  
689 MSH, as errors may originate either on the wire, or at every level of message processing in the MSH until  
690 message data is passed to the application. The action reports to the Test Driver the outcome of the

691 comparison. This is done via an alternate communication channel to ensure that the same system being  
692 tested is not used to report the reliability of its own MSH. A “notification” message is sent via RPC to the  
693 Test Driver. The previous MessageID is reported (for correlation) in the RefToMessageId header attribute  
694 of the response. The previous ConversationId is also reported. The payload message will contain a  
695 verification status notification for each verified payload, as specified in Appendix F.

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

697

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

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

704

705

706

707

### 708 3.2.4.3 Integration of the Test Service with an MSH Implementation

709

710 As previously mentioned, the actions above are predefined and are a required part of the Test Framework  
711 for messaging services testing, and will require some integration code with the MSH implementation, in  
712 form of three adapters, to be provided by the MSH development (or user) team. These adapters are:

713

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

716

717 (2) **MSH control adapter**, which will be invoked by some Test Service actions, and will invoke in turn  
718 the MSH-specific Message Service Interface (or API). Examples of such invocations are for  
719 sending messages (e.g. by actions which send response messages), and MSH configuration  
720 changes (done by the TestServiceConfigurator operation).

721

722 (3) **Error URL adapter**, which is actually independent from the candidate MSH. This adapter will  
723 catch error messages, and invoke the **report** method of the Test Service. The report method  
724 notifies the Test Driver of the error message.

725

### 726 3.2.5 Interfaces for Test Driver and Test Service

727

728 Not all Test Harness communication occurs at the messaging level (i.e. through Test Service actions).  
729 Certain Test Harness functionality can only be safely and reliably guaranteed by decoupling it from the  
730 actual messaging protocol being tested. This is the case for Test Service message initiation,  
731 configuration and error notification. . If the same protocol under test were also used as the infrastructure  
732 for the actions above, then failure of that protocol would result in undetermined/ambiguous Test Case  
733 results.

734 Four interfaces (3 Test Service, 1 Test Driver) are defined to provide a “decoupled” relationship between  
735 the system under test, and the test harness.

736

737 The three interfaces on the Test Service component are:

738

739 **Send** – consists of one method (initiator) that accepts a message declaration, builds the message  
740 envelope, attaches any referenced payloads, and sends the message. The method returns an XML  
741 notification document with a “pass/fail” Result element.

742

743 **Configuration** – Consists of one method, (configurator) which accepts a Configuration Group list of  
744 parameters and their corresponding values. This includes three “required” parameters, and additional  
745 optionalpayload digest name/value parameters. The method returns an XML notification document with a  
746 “pass/fail” Result element.

747

748

749 These two interfaces can be accessed either locally (if the Test Driver and Test Service are running in the  
750 same program space), or remotely (if the Test Driver and Test Service are not local). In the case of  
751 remote communication, these methods MUST be accessible via RPC call.

752

753 The interface on the Test Driver component is:

754

755 **Receive** – Its “notify” method accepts incoming notification messages from the Test Service and passes  
756 them to the Test Service for storage in its Message Store. Notification messages include messages  
757 received by the Test Service (when the Test Service is in “reporting mode”) and application error  
758 messages generated by the Test Service and response messages from the Test Service referencing  
759 success/failure of received message payload verification.

760

### 761 3.2.5.1 Abstract Test Service “Send” Interface

762

763 The abstract interface is defined as:

764

- 765 1. An interface that must be supported by the Test Service
- 766 2. An initiator method that must be supported by that interface
- 767 3. The parameters and responses that must be supported by that method

768

769 This abstract Test Service interface does not specify any particular implementation of a MSH, nor does it  
770 specify a particular language binding.

771

Method Return Type	Method Name	Exception
--------------------	-------------	-----------



		Condition
InitiatorResponse (an XML document , returning a synchronous response message containing a boolean Result element)	<p>initiator (MessageDeclaration declarationMessagePayloadList payloads)</p> <p>Passes the constructed message “declaration” to the Test Service initiator action Additionally, any message payloads are passed as an encapsulated list.</p>	Failed to construct or send message

772 Table 1 – Initiator method description

773

774 **Semantic Description:** The Initiator call instructs the Test Service to generate a new message. The  
775 new message content is provided as a argument to the initiator call. Any payload content is provided as  
776 attachments in the SOAP message, and have the same content-Id as defined in the message  
777 Declaration. The envelope header of the new message can be anything that is specified and understood  
778 by the Test Service (e.g. ebXML or RNIF). This action may be used to generate a "first" message of a  
779 new conversation (if no ConversationId is present in the Declaration .

780

781 The method is of return-type InitiatorResponse, meaning the method returns a response XML message  
782 document containing a status message describing the success/failure of the Initiator method call. This is  
783 returned to the Test Driver. A return value of “false” stops execution of the Test Case with a final result of  
784 “undetermined”. A return value of “true” signals the Test Driver to proceed with the testing workflow.

785

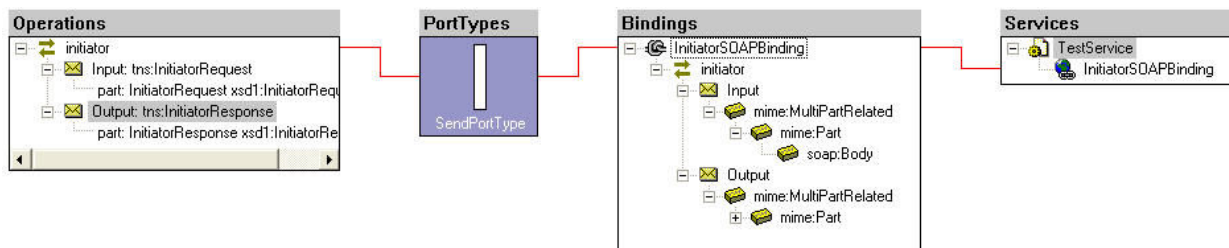
786 3.2.5.2 WSDL representation of the initiator RPC method

787

788 If the Test Driver is “remote” to the Test Service (i.e. resides outside of the program space of the Test  
789 Service), messages may still be initiated by the Test Driver on the remote Test Service via RPC. The  
790 Web Service Description Language (WSDL) document in Appendix H describes the Service, Operation,  
791 Port and (example SOAP) bindings that MUST be implemented in the Test Service in order to perform  
792 remote message initiation via SOAP v1.2 Other RPC bindings may be implemented, as long as the  
793 operations and documents described in this WSDL definition are used, and both the Test Service and  
794 Test Driver are using the same RPC methods and definitions.

795

796



797

798 Figure 8 – WSDL diagram of the Initiator SOAP method

799

800

801

802 3.2.5.3 Abstract Test Service “Configuration” Interface

803

804 The abstract interface is defined as:

805

- 806 1. An interface that must be supported by the Test Service
- 807 2. A configurator method that must be supported by that interface
- 808 3. The parameters and responses that must be supported by that method

809

810 This abstract MSH interface does not specify any particular implementation of a MSH, nor does it specify  
811 a particular language binding.

812

Method Return Type	Method Name	Exception Condition
ConfiguratorResponse (an XML document containing a boolean result element)	configurator (ConfigurationList list) Passes the configuration parameters to the Test Service	Test Service fails to configure properly

813 Table 2 – Configurator method

814

815 **Semantic Description:** The configurator call passes configuration data from the Test Driver to the Test  
816 Service. This includes the three REQUIRED configuration items (ResponseURL, NotificationURL,  
817 ServiceMode), plus additional optional parameters that may be used in payload verification payload  
818 integrity verification.

819 The method is of type ConfiguratorResponse, meaning the method returns a response XML message  
820 document containing a status message describing the success/failure of the configurator method call to  
821 the Test Driver. A return value of “false” stops execution of the Test Case with a final result of  
822 “undetermined”. A return value of “true” signals the Test Driver to proceed with the testing workflow.

823

824

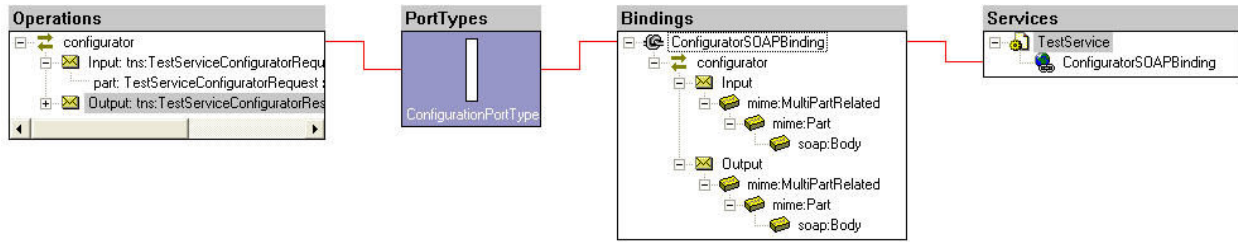
825 3.2.5.3.1 WSDL representation of the configurator SOAP method

826

827 3.2.5.3.2 If the Test Driver is “remote” to the Test Service (i.e. resides outside of the program space of  
828 the Test Service), messages may still be initiated by the Test Driver on the remote Test  
829 Service via RPC. The Web Service Description Language (WSDL) document in Appendix H  
830 describes the Service, Operation, Port and (example) bindings that MUST be implemented in  
831 the Test Service in order to perform remote Test Service configuration via SOAP v1.2 Other  
832 RPC bindings may be implemented, as long as the operations and documents described in  
833 the WSDL definition are used, and the same RPC mechanism is used by both Test Driver and  
834 Test Service implementer.

835 3.2.5.3.3

836



837  
838 Figure 9 – WSDL diagram of the configurator SOAP method

839  
840  
841  
842

843 3.2.5.4 Abstract Test Driver “Receive” Interface

844

845 The Test Driver MUST also have an interface available for communication with the Test Service. The  
846 abstract interface is defined as:

847

- 848 1. An interface that must be supported by the Test Driver
- 849 2. An notify method that must be supported by that interface
- 850 3. The parameters and responses that must be supported by that method

851

852 This abstract MSH interface does not specify any particular implementation of a MSH, nor does it specify  
853 a particular language binding.

854

Method Return Type	Method Name	Exception Condition
NotificationResponse	notify (NotificationMessage message, MessagePayloadList messagePayloads)  Passes the, received notification message envelope and any encapsulated message payloads to the Test Driver	Test Driver fails to accept the notification message

855 Table 4 – WSDL diagram of the notify SOAP method

856

857 **Semantic Description:** The notify method instructs the Test Driver to add the received or generated  
858 message content to the Message Store, along with accompanying service instance id, service action and  
859 other data provided by the Test Service.

860

861 The method is of type NotificationResponse, meaning the method returns a response XML message  
862 document containing a status message describing the success/failure of the notify method call back to the  
863 Test Service.

864

865 The types of notifications that a Test Service may pass to a Test Driver include:

866

867 An **application error notification message** captures specific error notifications from the MSH to its using  
868 application. It is not triggered by reception of an error message, but it is directly triggered by the internal  
869 error module of the MSH local to this Test Service. If the MSH implementation does not support such  
870 direct notification of the application (e.g. instead, it writes such notifications to a log), then an adapter  
871 needs to be written to read this log and invoke this action whenever such an error is notified.

872 Such errors fall into two categories:

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

878

879 Application Error Notification Message Format:

880 Error notification messages have the same characteristics a normal error message (i.e. have a  
881 MessageHeader with refToMessageId, ConversationId, CPAId corresponding to that of the incoming  
882 “offending” message that generated the error). In addition, the message will contain an Error List  
883 conforming to that normally generated by the MSH. This message will be identified as “different” from a  
884 received message by the presence of a “Notification” root element, which contains reporting test service  
885 name, reporting test service instance id, reporting method name (errorAppNotify), synch type  
886 (synchronous or asynchronous), and id.

887

888 An **MSH Error notification message** captures “normal” error notifications from the MSH (i.e. errors  
889 normally returned to the sending MSH). This method is specified to handle cases where the MSH cannot  
890 resolve the error reporting location (not present in CPA) and does not return the error to the sending  
891 MSH. In this case the Test Service Notification interface is utilized to report the error to the Test Driver.

892

893 MSH Error Notification Message Format:

894 Error notification messages will have the same characteristics a normal error message (i.e. have a  
895 MessageHeader with refToMessageId, ConversationId, CPAId corresponding to that of the incoming  
896 “offending” message that generated the error). In addition, the message will contain an Error List  
897 conforming to that normally generated by the MSH. This message will be identified as “different” from a  
898 received message by the presence of a “Notification” root element, which contains reporting test service  
899 name, reporting test service instance id, reporting method name (errorURLNotify), synch type  
900 (synchronous or asynchronous), and id.

901

902 **Received Message notifications** capture messages received by the Test Service. This method is  
903 specified to handle testing scenarios where the Test Service is in “local-reporting” or “remote reporting”  
904 mode. A notification message generated by the notify method is a “copy” of the received message  
905 envelope and an encapsulated list of any attachments provided with the message. The message  
906 contains.

907

908 Received Message Notification Format:

909 All notification messages generated by the report method will have the same characteristics a normal  
910 message (i.e. have a MessageHeader with refToMessageId, ConversationId, CPAId). Additionally, an  
911 encapsulated list of message attachments that were a part of the received message is passed to the Test  
912 Driver. The message will be identified as “different” from a received message by the presence of a  
913 “Notification” root element, which contains reporting test service name, reporting test service instance id,  
914 reporting method name (notify), synch type (synchronous or asynchronous), and id.

915

916 **Payload verification notification messages** inform the Test Driver of the result of the PayloadVerify  
917 action of the Test Service. A notification message consists of a message envelope with the same  
918 characteristics a normal response message (i.e. have a MessageHeader with refToMessageId,  
919 ConversationId, CPAId corresponding to that of the incoming message). This message will be identified  
920 as “different” from a received message by the presence of a “Notification” root element, which contains  
921 reporting test service name, reporting test service instance id, reporting method name (payloadVerify),  
922 synch type (synchronous or asynchronous), and id.

923

924 Received Payload Verification Format:

925 All payload verification messages will have the same characteristics a normal message (i.e. have a  
926 MessageHeader with refToMessageId, ConversationId, CPAId). Additionally, the notify method will pass  
927 to the Test Driver an XML document describing the result of the payload verification. This message will  
928 be identified as “different” from a received message by the presence of a “Notification” root element,  
929 which contains reporting test service name, reporting test service instance id, reporting method name  
930 (messageNotify), synch type (synchronous or asynchronous), and id.

931

932 The XML format used use for all of the above notification messages is described in the section 7.1.12  
933 (“Service Messages”).

934

935 Additional note:

936 Notification messages do not contain any artifacts pertaining to the protocol that carried them. For  
937 example, no HTTP or MIME headers are passed along with the notification message; because the Test  
938 Service does not normally have access to this message content at the application level. Only message  
939 envelopes, and accompanying message payloads are passed on to the Test Driver’s “Receive” interface.

940

941

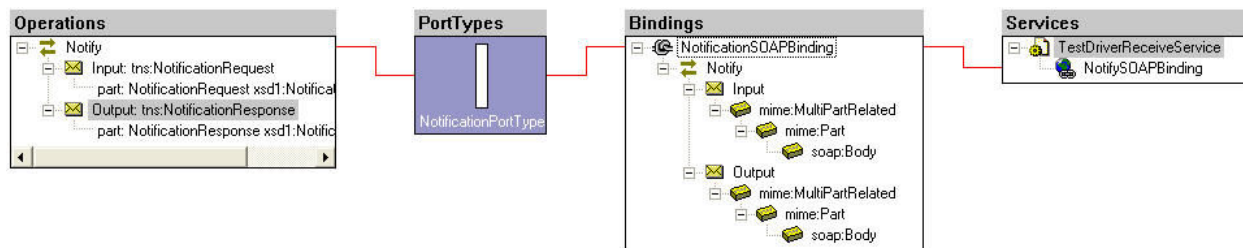
942 3.2.5.4.1 WSDL representation of the Test Driver notify SOAP method

943

944 If the Test Driver is “remote” to the Test Service (i.e. resides outside of the program space of the Test  
945 Service), messages may still be initiated by the Test Service via RPC. The Web Service Description  
946 Language (WSDL) document in Appendix H describes the Service, Operation, Port and (example)  
947 Bindings that MUST be implemented in the Test Service in order to perform remote Test Service  
948 configuration via SOAP v1.2 Other RPC methods may be implemented, as long as the operations and  
949 documents described in the WSDL definition are used, and the same RPC mechanism is used by both  
950 Test Driver and Test Service implementer.

951

952



953  
954 Figure 10 – WSDL diagram of the Test Driver notify SOAP method

955  
956

### 957 3.3 Executing Test Cases

958  
959

960 A Test Suite document contains a collection of Test Cases. Each Test Case is an XML script, intended to  
961 be interpreted by a Test Driver. Using the Test Suite document, the Test Driver MUST be able to:

962  
963  
964  
965

966 **Configure Itself** – Define necessary parameters that permit the Test Driver to send messages and verify  
967 and/or validate received message content

968 **Configure the Test Service** – Define necessary parameters that permit the Test Service to set its mode of  
969 operation, and send notification messages to the Test Driver (if required).

970 **Access all necessary testing material** – Test Requirements documents, message content, payload  
971 content

972 **Execute Test Cases** – Interpret a formalized and valid XML scripting language that permits  
973 unambiguous, repeatable results each time it is interpreted and executed

974 **Generate a Test Report** – After executing the Test Cases, a Test Driver MUST be able to generate a Test  
975 Report using the material provided in the Test Suite, and collateral test material that is part of the Test  
976 Suite.

977  
978  
979

#### 980 3.3.1 A Typical Execution Scenario

981

982 In order to get an idea of how the Test Framework operates, a brief description of how a Test Driver  
983 would typically execute a Test Suite is described below. This is an “overview” description of how the Test  
984 Framework executes. In order to fully understand the details and requirements of implementing this  
985 specification, the remaining portion of this specification must be examined.

986

987 A typical execution model for the Test Harness would be:

988  
989           A Test Driver is installed on a networked computer.  
990 An implementer wishing to test an ebXML (or other) implementation invokes the Test Driver executable.  
991 The Test Driver asks the tester for fundamental information (e.g. mode of testing to be used by the Test  
992 Driver, message and error reporting URL for the candidate implementation)  
993 The Test Driver “self configures” based upon user preferences.  
994 The Test Driver performs any local or remote configuration of the candidate implementation if necessary.  
995 The Test Driver presents the tester with a list of conformance or interoperability testing profiles that  
996 he/she may select from for testing the candidate implementation.  
997 The tester chooses a testing requirements profile.  
998           Execution of Test Cases against the specified testing requirements profile begins.  
999 A standard Test Report Document is generated by the Test Driver, providing a trace of all testing  
1000 operations performed for each Test Case, with accompanying Test Case results, indicating a final result  
1001 of “pass”, “fail” or “undetermined” for each Test Case, based upon detailed results of each operation  
1002 within each Test Case.  
1003 If a candidate implementation passes all Test Cases in the Test Suite, it can be considered conformant or  
1004 interoperable for that particular testing profile.  
1005 If a candidate implementation fails some Test Cases, but the Test Requirement that they tested against  
1006 were “OPTIONAL”, “HIGHLY RECOMMENDED” or “RECOMMENDED”, then that implementation may  
1007 still be conformant for all REQUIRED features tested.  
1008 If the optional features tested were actually implemented on the candidate, and it failed any Test Cases  
1009 that test against those features then the candidate would be considered “non-conformant” for those  
1010 optional features.  
1011 If any Test Case results were “undetermined” (due to network problems, or due to missing prerequisite  
1012 candidate features that are not under the control of the Test Harness) then ultimate  
1013 conformance/interoperability of the candidate implementation is deemed “undetermined” for that testing  
1014 profile. In such cases, resolution of the underlying system issue must be resolved or the Testing Profile  
1015 must be redefined to test only those features that are truly supported by the candidate implementation.  
1016  
1017  
1018  
1019 The above list represents an “overall” view of how a Test Harness operates. Detailed descriptions of the  
1020 testing material that drives the Test Harness, and implementation requirements for the Test Driver and  
1021 Test Service follow.  
1022

### 1023 3.3.2 Test Case as a Workflow of Threads

1024 :  
1025 A Test Case is a workflow of Test Threads. A Thread can be executed either in a synchronous or  
1026 asynchronous manner. If a particular operation consists of a logically grouped sequence of message  
1027 “send” and “receive” operations, then a Thread is a logical container to group those operations. In  
1028 addition, a Thread may test an assertion of expected message content from a received message. A  
1029 Thread may also include conditional actions (testing preconditions) that are a basis for proceeding to the  
1030 execution of the assertion test.  
1031 A Test Case Instance is the execution of a particular sequence of test operations, Two instances of the  
1032 same Test Case will be distinguished by distinct ConversationId and MessageId values in the generated  
1033 messages (referred to as the message “context”). An example of a sequence of Threads associated with  
1034 an MS Conformance Test Case is:

1035

1036 Thread 1: Test driver sends a sample message to the Reflector action of the Test Service. Message  
1037 header data is obtained from message header declaration, and message payload from the received file.

1038 Thread 2: Test driver receives the response message and adds it to the stored messages received for  
1039 this Test Case instance Step 3: Correlation with Step 3 is done based on the ConversationId attribute,  
1040 which should be identical to the MessageId of Step 2. Test driver verifies the test condition on response  
1041 message, for example that the SOAP envelope and extensions are valid.

1042

### 1043 3.3.3 Related Message Data and Declarations

1044

1045

1046 Some Threads will require construction of message data. This message data MUST be specified using a  
1047 Declaration (see Section 7). A Declaration is an XML-based script interpreted by the Test Driver (or Test  
1048 Service if doing interoperability testing) to construct a message envelope and its content. Payload  
1049 material is not included in the messages declaration, but may be referenced by it (for example, in the  
1050 case of ebXML Messaging, via the Manifest element).

1051

1052 The Test Driver MUST be capable of interpreting these scripts in order to:

1053

1054 Assemble a message from script material and referenced payloads.

1055 Analyze and select a received message based on header and envelope content (as well as based on  
1056 payload content if the payload is in XML).

1057

### 1058 3.3.4 Related Testing Configuration Data

1059

1060 Test Cases MAY be executed under a pre-defined collaboration agreement. For example, when testing  
1061 ebXML Messaging Services, this agreement is a CPA [ebCPP]. This agreement will configure the ebXML  
1062 Candidate Implementations involved in the testing, and define the collaborations that execute on these  
1063 implementations.

1064

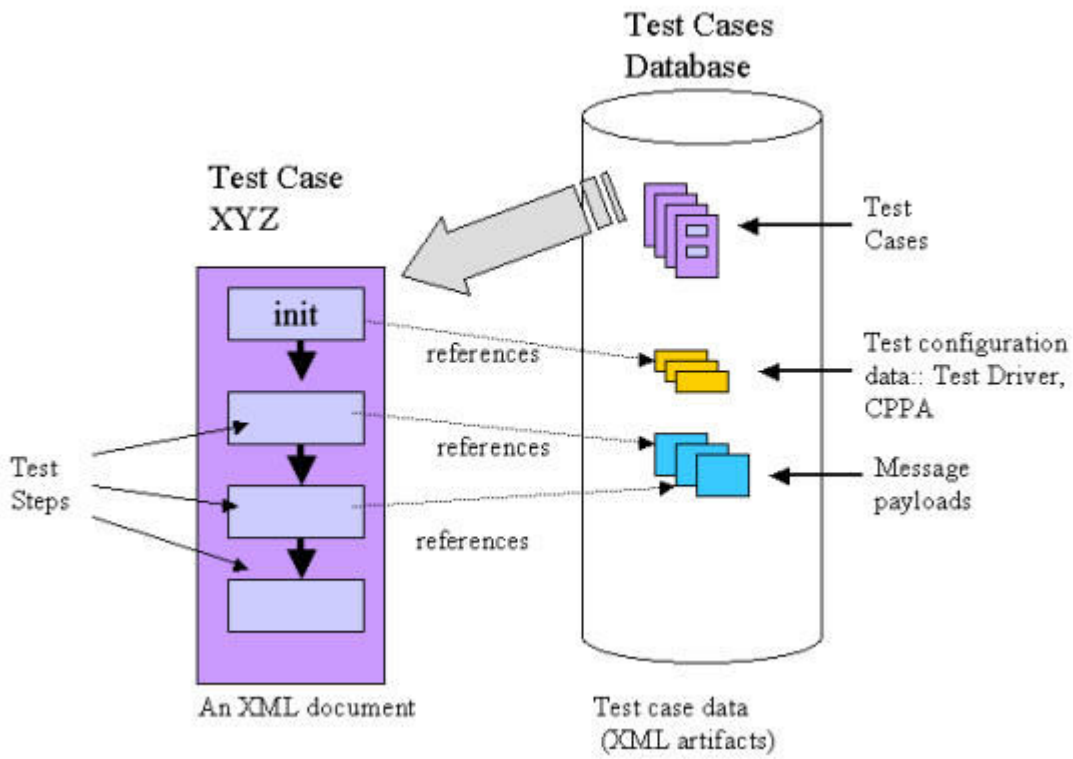
1065 Test Driver Configuration data (found in the Test Suite XML document) parameters define the operational  
1066 mode of the test driver itself. The Test Driver is agnostic to any type of collaboration agreement, but does  
1067 have its own set of configuration parameters and requirements. This information is provided (or  
1068 referenced via URI) in the Test Suite document..

1069

1070

1071





1072  
 1073 Figure 11 – Test Case Document and Database  
 1074  
 1075  
 1076

1077

## Part II: Test Suite Representation

1078

---

## 1079 4 Test Suite

1080

### 1081 4.1 Conformance vs. Interoperability Test Suite

1082

1083 We distinguish two types of test suites, which share similar document schemas and architecture  
1084 components, but serve different purposes:

1085

- 1086 ▪ **Conformance Test Suite.** The objective is to verify the adherence or non-adherence of a Candidate  
1087 Implementation to the target specification. The test harness and Test Cases will be designed around  
1088 a single (candidate) implementation. The suite material emphasizes the target specification, by  
1089 including a comprehensive set of Test Requirements, as well as a clear mapping of these to the  
1090 original specification (e.g. in form of an annotated version of this specification).
- 1091
- 1092 ▪ **Interoperability Test Suite.** The objective is to verify that two implementations (or more) of the same  
1093 specification, or that an implementation and its operational environment, can interoperate according  
1094 to an agreement or contract (which is compliant with the specification, but usually restricts further the  
1095 requirements). These implementations are assumed to be conforming (i.e. have passed conformance  
1096 tests or have achieved the level of function of such tests), so the reference to the specification is not  
1097 as important as in conformance. Such a test suite involves two or more Candidate Implementations of  
1098 the target specification. The test harness and Test Cases will be designed in order to drive and  
1099 monitor these implementations.

1100

1101 A conformance test suite is composed of:

1102

1103 One or more **Test Profile** documents (XML). Such documents represent the level or profile of  
1104 conformance to the specification, as verified by this Test Suite.

1105 Design of a **Test Harness** for the Candidate Implementation that is based on components of the ebXML  
1106 IIC Test Framework.

1107 A **Test Requirements** document. This document contains a list of conformance test assertions that are  
1108 associated with the test profile to be tested.

1109 An **annotation** of the target specification, that indicates the degree of Specification Coverage for each  
1110 specification feature or section, that this set of Test Requirements provides.

1111 A **Test Suite** document. This document implements the Test Requirements, described using the Test  
1112 Framework material (XML mark-up, etc.)

1113

1114 An Interoperability Test Suite is composed of:

1115

1116 One or more **Test Profile** documents (XML). Such documents represent a set of features specific to a  
1117 particular functionality, represented in a Test Suite through Test Cases that only test those particular  
1118 features, and hence, that profile.

1119 Design of a **Test Harness** for two or more interoperating implementations of the specification that is  
1120 based on components of the ebXML Test Framework.

1121 A **Test Requirements** document. This document contains a list of test assertions associated with this  
1122 profile (or level) of interoperability.

1123 A **Test Suite** document. This document implements the Test Requirements, described using the Test  
1124 Framework material (XML mark-up, etc.)

1125

1126

## 1127 4.2 The Test Suite Document

1128

1129 The Test Suite XML document is a collection of Test Driver configuration data, documentation and  
1130 executable Test Cases.

1131 ▪ **Test Suite Metadata** provides documentation used by the Test Driver to generate a Test Report for  
1132 all executed Test Cases.

1133 ▪ **Test Driver Configuration data** provide basic Test Driver parameters used to modify the  
1134 configuration of the Test Driver to accurately perform and evaluate test results. It also contains  
1135 configuration data for the candidate ebXML implementation(s).

1136 ▪ **Message data** is a collection of pre-defined XML payload messages that can be referenced for  
1137 inclusion in an ebXML test message.

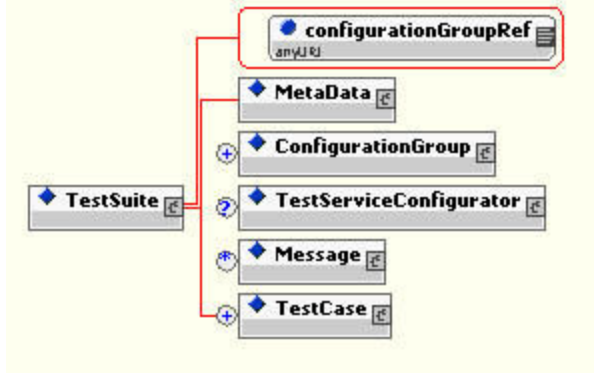
1138 ▪ **Test Cases** are a collection of discrete Threads. Each Thread can execute any number of test  
1139 Operations (including sending, receiving, and examining returned messages). An ebXML Test Suite  
1140 document MUST validate against the ebTest.xsd file in Appendix C.

1141 ▪ **Message Payloads** provide XML and non-XML content for use as material for test messages, as well  
1142 as message data for Test Services linked to the Test Driver.

1143

1144

1145



1146  
1147

1148 Figure 12 – Graphic representation of basic view of TestSuite schema

1149

1150

1151

1152

1153

1154

1155

1156

1157

1158

1159

1160 **Definition of Content**

1161

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
TestSuite	Container for all configuration,		Required	

	documentation and tests			
configurationGroupRef	Reference ID of the ConfigurationGroup data used to configure theTest Driver (in connection mode) or Test Service/MSH ( when in service mode)		Required	ConfigurationGroup not found
Metadata	Container for general documentation of the entire Test Suite		Required	
ConfigurationGroup	Container for Test Driver configuration instance data		Optional	
TestServiceConfigurator	Container for Test Service configuration instance data		Required	Unable to configure Test Service (non-fatal)
Message	Container element for "wildcard" message content (i.e. any well-formed XML content)		Optional	
TestCase	Container for an individual Test Case		Required	

1162 Table 5 provides a list of TestSuite element and attribute content

1163

1164

1165

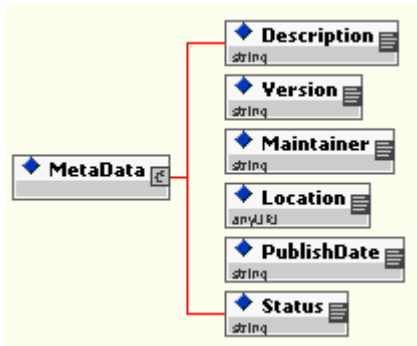
#### 1166 4.2.1 Test Suite Metadata

1167

1168 Documentation for the ebXML MS Test Suite is done through the Metadata element. It is a container  
1169 element for general documentation.

1170

1171



1172

1173 Figure 13 – Graphic representation of expanded view of the Metadata element

1174

1175

1176

1177

1178 **Definition of Content**

Name	Description	Default Value From Test Driver	Required/Optional	Exception Conditions
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	

1179 Table 6 provides a list of Metadata element and attribute content

1180

1181 **4.2.2 The ConfigurationGroup**

1182

1183 The ConfigurationGroup element contains configuration data for both the Test Driver as well as modifying  
1184 the content of test messages constructed by the Test Driver (when in “connection” mode) or message  
1185 declarations passed to the Test Service (when in “service” mode).

1186 ConfigurationGroups may be referenced throughout a Test Suite, in a hierarchical fashion. By default, a  
1187 “global” ConfigurationGroup is required for the entire Test Suite, and MUST be referenced by the  
1188 TestSuite element in the Executable Test Suite document. This established a “base” configuration for the  
1189 Test Driver.

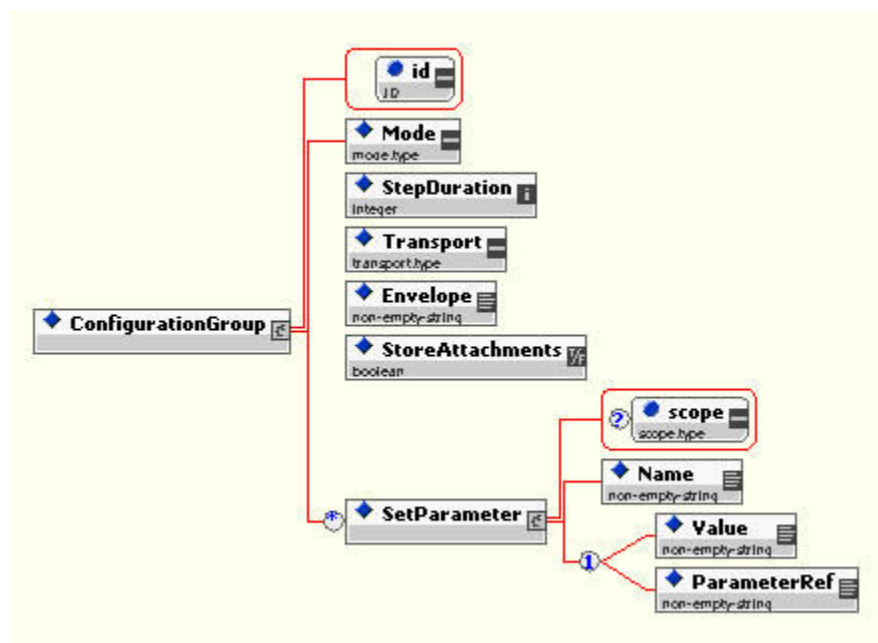
1190 Subsequent re-configurations of the Test Driver may be done at the Test Case and Thread levels of the  
1191 test object hierarchy. At each level, a reference to a ConfigurationGroup via the “configurationGroupRef”  
1192 attribute takes precedence and defines the Test Driver configuration for the current test object and any  
1193 “descendent” test objects (e.g. any Test Cases and sub-Threads will inherit the Test Driver configuration  
1194 defined by their parent Thread). Logically, when workflow control of the Test Case returns to a higher  
1195 level in the object hierarchy, then the ConfigurationGroup defined at that level again takes precedence  
1196 over any defined at a lower level by a descendent test object.

1197

1198

1199

1200



1201

1202

1203



1204 Figure 14 – Graphic representation of expanded view of the ConfigurationGroup element

1205

1206

1207 **Definition of Content**

1208

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
ConfigurationGroup	Container Test Driver/MSH configuration data		Required	
id	Unique URI used to identify this set of configuration data		Required	
Mode	One of two types for the Test Driver, (service   connection)		Required	
StepDuration	Timeout (in seconds) of a message send or receiver operation		Required	
Transport	Directs the Test Driver as to which transport protocol to use to carry messages.		Required	
Envelope	Directs the Test Driver as to which Messaging envelope type it is constructing		Required	
StoreAttachments	Toggle switch directing Test Driver to ignore (false) or store (true) incoming message attachments		Required	
SetParameter	Container for "ad-hoc" 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		Optional	
ParameterRef	Name of previously defined parameter, whose value is substituted for the value of this parameter		Optional	

1209 Table 7 provides a list of ConfigurationGroup element and attribute content

1210

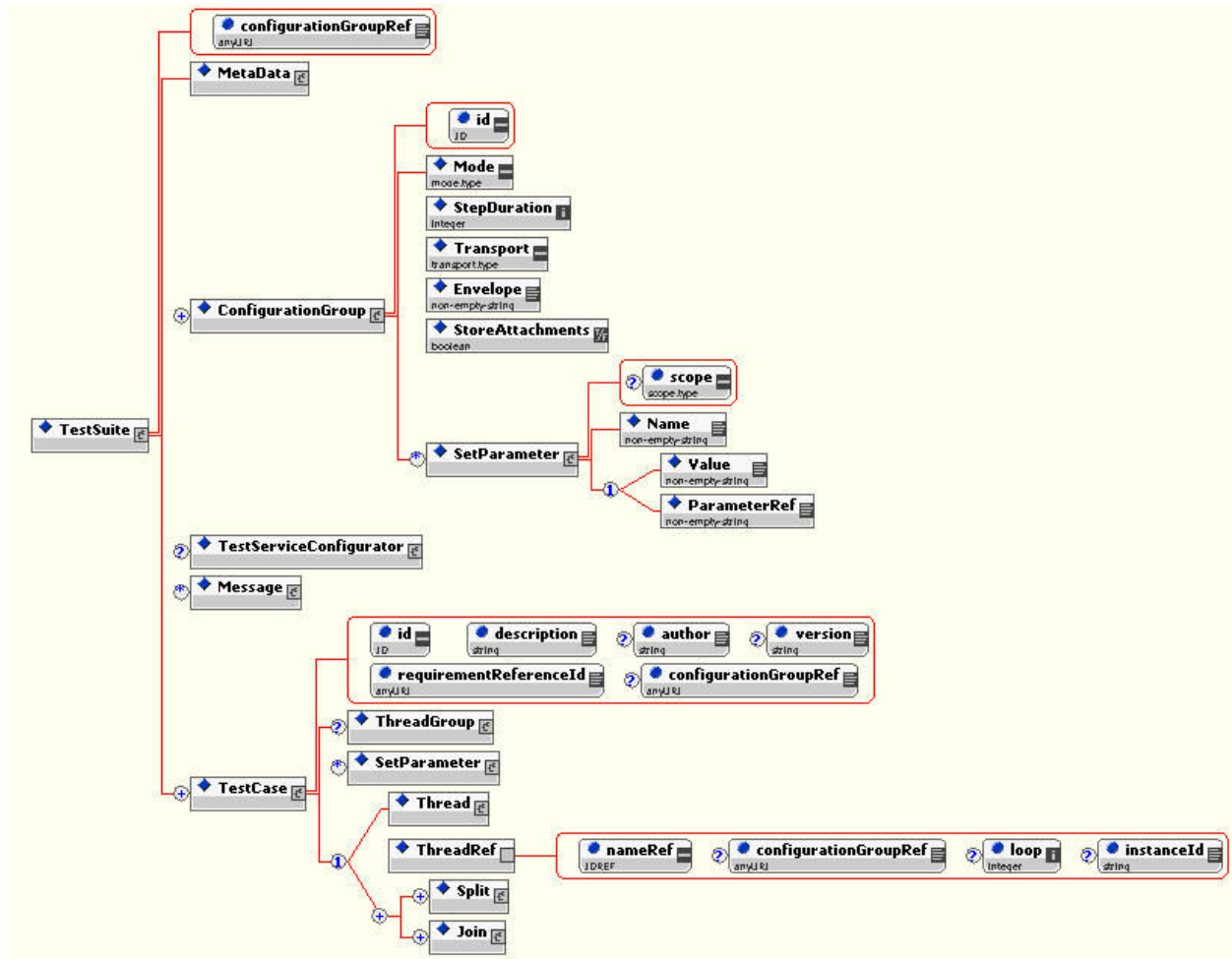
1211

1212

1213

1214

1215



1216

1217

1218 Figure 15 – Graphic representation of hierarchical use of the ConfigurationGroup via reference at  
1219 TestSuite, TestCase and ThreadRef Thread levels in the test object hierarchy

1220

1221

1222 4.2.2.1 Precedence Rules for Test Driver/MSH configuration parameters used in message construction

1223

1224 In order to generate messages correctly, the Test Driver MUST follow the precedence rules for  
1225 interpreting a Configuration Group parameter reference. The precedence rules are:

1226  
1227 Certain portions of a message are auto-generated by the Test Driver (or MSH) at run-time  
1228  
1229 This includes the following run time generated parameters:  
1230  
1231  
1232 ConversationId – Unique to each new Test Case  
1233 MessageId – Unique to each message generated by the PutMessage instruction  
1234 Timestamp - Unique to each message generated by the PutMessage instruction  
1235 These run time parameters MUST have the names specified above (case sensitive).  
1236  
1237  
1238 Additional message content MAY be provided through parameter definitions in the current  
1239 ConfigurationGroup, or through a SetParameter or SetXPathParameter operation within a Thread. This  
1240 includes message content such as:  
1241  
1242 CPA Id  
1243 Service  
1244 Action  
1245 Sender Party Id  
1246 Receiver Party Id  
1247  
1248 The parameters listed above can be given any parameter name the test writer chooses. However, the test  
1249 writer MUST reference the parameter in XSLT mutator stylesheets, or in XPath expressions using the  
1250 identical name (case sensitive) with which it was defined using the SetParameter instruction.  
1251  
1252 The following rule describes how a Test Driver MUST interpret parameter values and their precedence of  
1253 assignment within a Test Suite.  
1254  
1255 The TestSuite element's "configurationGroupRef" attribute points to the global parameter definition for the  
1256 entire Test Suite. This acts as the "basel" parameter definition before Test Suite execution begins.  
1257 Parameters MAY be used by an XSL stylesheet or XUpdate document to "mutate" a Declaration into a  
1258 valid message. They are passed to the XSL or XUpdate processor via name reference.  
1259 Parameters MAY be used by the VerifyContent operation through reference in an XPath expression.  
1260 Parameter names are referenced in XPath expressions with a preceding "\$" character. The Test Driver  
1261 MUST dereference the parameter prior to performing an XPath query on a FilterResult document object.  
1262 If a parameter is defined in a ConfigurationGroup or via a SetParameter operation, the parameter  
1263 definition takes precedence over any "auto-generated" definition of that parameter by the Test Driver.  
1264 Care should be taken to only "override" such values at the TestCase or ) Thread Thread level, so that  
1265 "side effects" are not passed on through the Test Suite object hierarchy (i.e. influencing message  
1266 construction beyond the scope of the Thread that is intended).  
1267 Any descendent Thread T element with a "configurationGroupRef" attribute "redefines" a parameters  
1268 value for itself and any of its descendent Threads (i.e. it limits the scope of the parameter definition to all  
1269 of its descendents).

1270 Any "SetParameter" instruction within a TestCase or Thread element supersedes its current definition  
 1271 within the currently defined ConfigurationGroup. The scope of the parameter definition is limited to the  
 1272 current Thread and any descendent Threads. .  
 1273

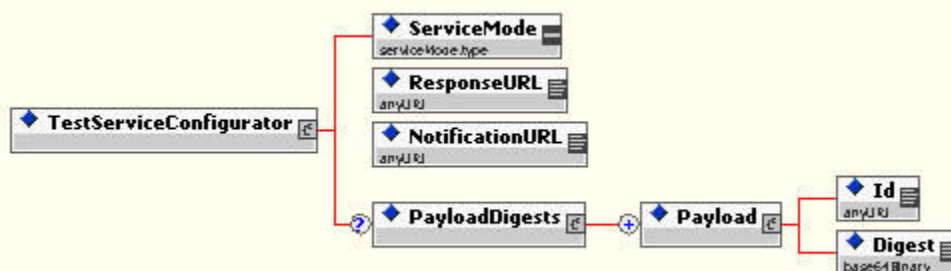
1274 4.2.2.1.1 Exception Conditions

1275  
 1276  
 1277 A Test Driver MUST generate an exception and terminate the Test Case with a result of "undetermined" if  
 1278 it cannot mutate a message due to an undefined parameter.  
 1279

1280 A Test Driver MUST generate an exception and terminate the Test Case with a result of "undetermined" if  
 1281 it cannot verify a message due to an undefined parameter in an XPath query.  
 1282  
 1283

1284 4.2.3 The TestServiceConfigurator Operation

1285  
 1286 The TestServiceConfiguration element instructs the Test Driver to configure the Test Service. A Test  
 1287 Service MUST provide both a Configuration interface to the Test Service, with a "configurator" method,  
 1288 like that specified in section 3.2.5. The Test Driver MAY access the Configuration interface either locally  
 1289 or remotely (via RPC), depending upon the mode of the Test Driver.  
 1290  
 1291



1292  
 1293 Figure 16 – Graphic representation of the TestServiceConfigurator content  
 1294  
 1295

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
TestServiceConfigurator	Container for Test Service configuration data		Required (as a child of a TestSuite element only), optional elsewhere	Unable to configure Test Service

ServiceMode	Switch to set to one of three modes (loop   local-reporting   remote-reporting)		Required	
ResponseURL	Endpoint to send response messages		Required	
NotificationURL	Endpoint to send message and error notifications (typically the Test Driver URL)		Required	
PayloadDigests	Container for one or more payload identifiers and corresponding MD5 digest value		Optional	
Payload	Container for individual payload information		Required	
Id	Id of the message payload		Required	
Digest	MD5 digest value of the payload		Required	

1296

1297 4.2.3.1 TestServiceConfigurator behavior in Connection and Service mode

1298

1299 **In Connection Mode:** The “TestServiceConfigurator” operation instructs the Test Driver to pass  
1300 configuration parameters to a remote Test Service Configuration interface, using its “configurator”  
1301 method. The Test Service MUST respond with a status of “success” or “fail”.

1302

1303

1304 **In Service Mode:** The “TestServiceConfigurator” operation instructs the Test Driver to pass configuration  
1305 parameters to the local Test Service via its Configuration interface, and its “configurator” method. The  
1306 Test Service MUST respond with a status of “success” or “fail”.

---

## 1307 5 Test Requirements

1308

### 1309 5.1 Purpose and Structure

1310

1311 The next step in designing a test suite is to define Test Requirements. This material, when used in a  
1312 conformance-testing context, is also called Test Assertions in NIST and OASIS terminology (see  
1313 definition in glossary in Appendix).

1314 When used for conformance testing, each Test Requirement defines a test item to be performed, that  
1315 covers a particular requirement of the target specification. It rewords the specification element into a  
1316 “testable form”, closer to the final corresponding Test Case, but unlike the latter, independently from the  
1317 test harness specifics. In the ebXML Test Framework, a Test Requirement will be made of three parts:

1318

1319 **Pre-condition** The pre-condition defines the context or situation under which this test item applies. It  
1320 should help a reader understand in which case the corresponding specification requirement applies. In  
1321 order to verify this Test Requirement, the test harness will attempt to create such a situation, or at the  
1322 very least to identify when it occurs. If for some reason the pre-condition is not satisfied when doing  
1323 testing, then it does not mean that the outcome of this test is negative – only that the situation in which it  
1324 applies did not occur. In that case, the corresponding specification requirement could simply not be  
1325 validated, and the subsequent Assertion will not be tested.

1326

1327 **Assertion** The assertion actually defines the specification requirement, as usually qualified by a MUST or  
1328 SHALL keyword. In the test harness, the verification of an assertion will be attempted only if the pre-  
1329 condition is itself satisfied. When doing testing, if the assertion cannot be verified while the pre-condition  
1330 was, then the outcome of this test item is negative.

1331

1332 **Requirement Level** Qualifies the degree of requirement in the specification, as indicated by such  
1333 keywords as RECOMMENDED, SHOULD, MUST, and MAY. Three levels can be distinguished: (1)  
1334 “required” (MUST, SHALL), (2) “recommended” ([HIGHLY] RECOMMENDED, SHOULD), (3) “optional”  
1335 (MAY, OPTIONAL). Any level lower than “required” qualifies a Test Requirement that is not mandatory  
1336 for Conformance testing. Yet, lower requirement degrees may be critical to interoperability tests. The test  
1337 requirement level can be override by explicit declaration in the Test Profile document, in case a lower or  
1338 higher level is required.

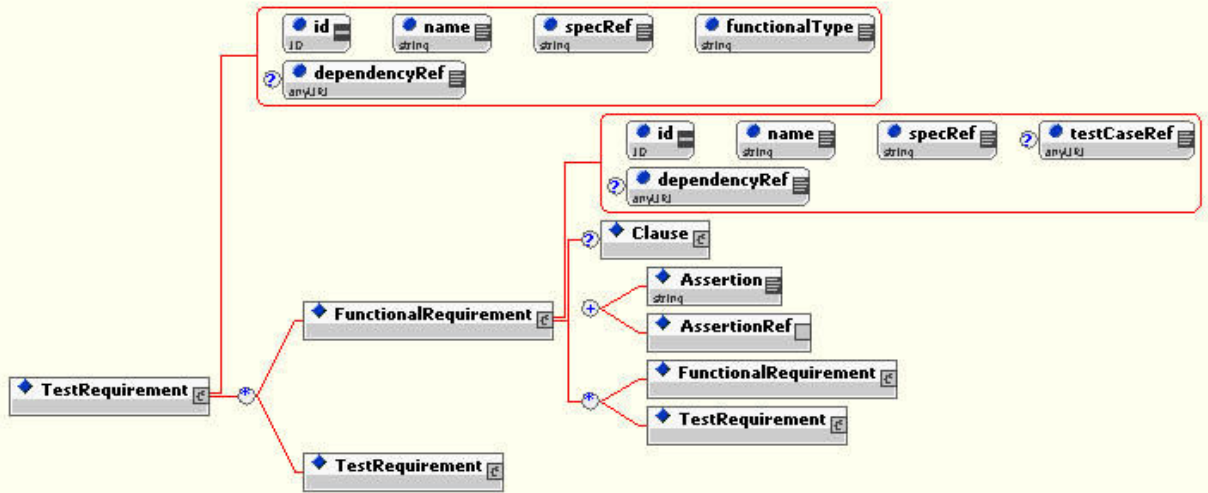
1339

### 1340 5.2 The Test Requirements Document

1341

1342 The Test Requirements XML document provides metadata describing the Testing Requirements, their  
1343 location in the specification, and their requirement type (REQUIRED, HIGHLY RECOMMENDED,  
1344 RECOMMENDED, or OPTIONAL). A Test Requirements XML document MUST validate against the  
1345 ebXMLTestRequirements.xsd file found in Appendix B. The ebXML MS Conformance Test Requirements  
1346 instance file can be found in Appendix E.

1347



1348  
 1349 Figure 17 – Graphic representation of ebXMLTestRequirements.xsd schema

1350  
 1351  
 1352 Definition of Content

1353 5.2.1

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
Requirements	Container for all test requirements		Required	
MetaData	Container for requirements metadata, including Description, Version, Maintainer, Location, Publish Date and Status		Required	
TestRequirement	Container for all testable sub-requirements (FunctionalRequirements) of a single generalized Test Requirement. A TestRequirement may also contain other TestRequirement elements as children		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	
dependencyRef	ID of "prerequisite" TestRequirement or FunctionalRequirement that must be successfully tested prior to testing this requirement		Optional	
FunctionalRequirement	Sub-requirement for the main Test Requirement. This is an actual testable requirement, not a "container" of requirements.		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	
dependencyRef	ID of "prerequisite" TestRequirement or FunctionalRequirement that must be successfully tested first prior to testing this requirement		Optional	
TestCaseId	Identifier of Test Case(s) that test this requirement		Optional	
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		Optional	
And/Or	Union/Intersection operators for Conditions		Optional	
Assertion	Axiom expressing expected behavior of an implementation under conditions specified by any Clause		Required	
AssertionRef	Reference (via id attribute) to existing Assertion element already defined in the Test Requirements document		Optional	
requirementType	Enumerated Assertion descriptor (REQUIRED, OPTIONAL...etc.)		Required	

1354 Table 8 provides a list of the testing Requirements element and attribute content



### 1355 5.3 Specification Coverage

1356

1357 A Test Requirement is a formalized way to express a requirement of the target specification. The  
1358 reference to the specification is included in each Test Requirement, and is made of one or more section  
1359 numbers. There is no one-to-one mapping between sections of a specification document and the Test  
1360 Requirement items listed in the test material for this specification:

1361

1362 A specification section may map to several Test Requirements.

1363 A Test Requirement item may also cover (partially or not) more than one section or sub-section.

1364

1365 A Test Requirement item may then cover a subset of the requirements that are specified in a section.

1366 For these reasons, it is important to determine to which degree the requirements of each section of a  
1367 specification, are fully satisfied by the set of Test Requirements listed in the test suite document.  
1368 Establishing the Specification Coverage by the Test Requirements does this.

1369

1370 The Specification Coverage document is a separate document containing a list of all sections and  
1371 subsections of a specification document, each annotated with:

1372

- 1373 • A coverage qualifier.
- 1374 • A list of Test Requirements that map to this section.

1375

1376 The coverage qualifier may have values:

1377

- 1378 • **Full:** The requirements included in the specification document section are fully covered by  
1379 the associated set of Test Requirements. This means that if each one of these Test  
1380 Requirements is satisfied by an implementation, then the requirements of the corresponding  
1381 document section are fulfilled. When the tests requirements are about conformance: The  
1382 associated set of test requirement(s) are a clear indicator of conformance to the specification  
1383 item, i.e. if a Candidate Implementation passes a Test Case that implements this test  
1384 requirement(s) in a verifiable manner, there is a strong indication that it will behave similarly  
1385 in all situations identified by the spec item.

1386

- 1387 • **None:** This section of the specification is not covered at all. Either there is no associated set  
1388 of Test Requirements, or it is known that the test requirements cannot be tested even  
1389 partially, at least with the Test Framework on which the test suite is to be implemented, and  
1390 under the test conditions that are defined.

1391

- 1392 • **Partial:** The requirements included in this document section are only partially covered by the  
1393 associated (set of) Test Requirement(s). This means that if each one of these Test  
1394 Requirements is satisfied by an implementation, then it cannot be asserted that all the  
1395 requirements of the corresponding document section are fulfilled: only a subset of all  
1396 situations identified by the specification item are addressed. Reasons may be:

1397

- 1398 ○ (1) The pre-condition(s) of the test requirement(s) ignores on purpose a subset of  
1399 situations that cannot be reasonably tested under the Test Framework.
- 1400 ○ (2) The occurrence of situations that match the pre-condition of a Test Requirement  
1401 is known to be under control of the implementation (e.g. implementation-dependent)

1402 or of external factors, and out of the control of the testbed. (See *contingent run-time*  
1403 *coverage* definition, Section 7).

1404 When the tests requirements are about conformance: The associated set of test  
1405 requirement(s) are a weak indicator of conformance to the specification item. A negative test  
1406 result will indicate non-conformance of the implementation.

1407

## 1408 **5.4 Test Requirements Coverage (or Test Run-Time Coverage)**

1409

1410 In a same way as Test Requirements may not be fully equivalent to the specification items they represent  
1411 (see Specification Coverage, Section 5.3), the Test Cases that implement these Test Requirements may  
1412 not fully verify them, for practical reasons.

1413

1414 Some Test Requirements may be difficult or impossible to verify in a satisfactory manner. The reason for  
1415 this generally resides in an inability to satisfy the pre-condition. When processing a Test Case, the Test  
1416 Harness will attempt to generate an operational context or situation that intends to satisfy the pre-  
1417 condition, and that is supposed to be representative enough of real operational situations. The set of such  
1418 real-world situations that is generally covered by the pre-condition of the Test Requirement is called the  
1419 *test requirements (or test run-time) coverage* of this test Requirement. This happens in the following  
1420 cases:

1421

1422 **Partial run-time coverage:** It is in general impossible to generate all the situations that should verify a  
1423 test. It is however expected that the small subset of run-time situations generated by the Test Harness, is  
1424 representative enough of all real-world situations that are relevant to the pre-condition. However, it is in  
1425 some cases obvious that the Test Case definition (and its processing) will not generate a representative-  
1426 enough (set of) situation(s). It could be that a significant subset of situations identified by the pre-condition  
1427 of a Test Requirement cannot be practically set-up and verified. For example, this is the case when some  
1428 combinations of events or of configurations of the implementation will not be tested due to the  
1429 impracticality to address the combinatorial nature of their aggregation. Or, some time-related situations  
1430 cannot be tested under expected time constraints.

1431

1432 **Contingent run-time coverage:** It may happen that the test harness has no complete control in  
1433 producing the situation that satisfies the pre-condition of a Test Requirement. This is the case for Test  
1434 Requirements that only concern optional features that an implementation may or may not decide to  
1435 exhibit, depending on factors under its own control and that are not understood or not easy to control by  
1436 the test developers. An example is: “ IF the implementation chooses to bundle together messages [e.g.  
1437 under some stressed operation conditions left to the appreciation of this implementation] THEN the  
1438 bundling must satisfy condition XYZ”.

1439

1440 When a set of Test Cases is written for a particular set of Test Requirements, the degree of coverage of  
1441 these Test Requirements by these Test Cases SHOULD be assessed. The Test Requirements coverage  
1442 – not to be confused with the Specification Coverage - is represented by a list of the Test Requirements  
1443 Ids, which associates with each Test Requirement:

1444

1445 The Test Case (or set of Test Cases) that cover it,

1446 The coverage qualifier, which indicates the degree to which the Test Requirement is covered.

1447

1448 The coverage qualifier may have values:

1449

- 1450 • **Full:** the Test Requirement item is fully verified by the set of Test Cases.
- 1451 • **Contingent:** The run-time coverage is contingent (see definition).
- 1452 • **Partial:** the Test Requirement item is only partially verified by the associated set of Test
- 1453 Cases. The run-time coverage is partial (see definition).
- 1454 • **None:** the Test Requirement item is not verified at all: there is no relevant Test Case.
- 1455

## 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 is relevant to the conformance or interoperability profile to be tested.

The document drives 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.

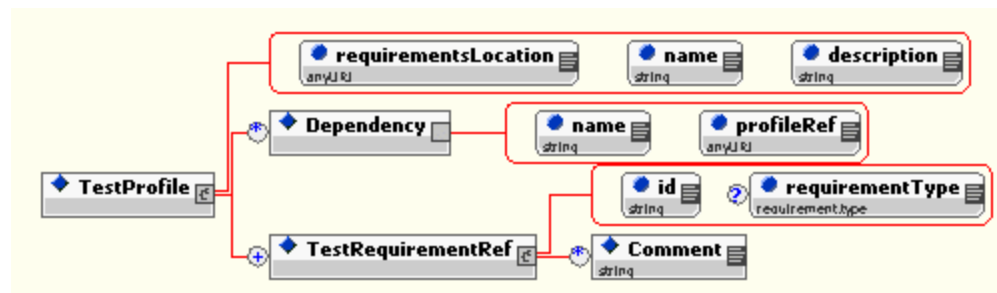


Figure 18 – Graphic representation of ebXMLTestProfile.xsd schema

Definition of Content

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
TestProfile	Container for all references to test requirements		Required	
requirementsLocation	URI of test requirements XML file		Required	Requirements document not found
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 before executing this one		Required	Profile document not found
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	

1474 Table 9 provides a list of TestProfile element and attribute content

1475

## 1476 6.2 Relationships between Profiles, Requirements and Test Cases

1477

1478 Creation of a testing profile requires selection of a group of Test Requirement references that fulfill a  
 1479 particular testing profile. For example, to create a testing profile for a Core Profile would require the  
 1480 creation of an XML document referencing Test Requirements 1,2,3,4,5 and 8.

1481

1482 The Test Driver would read this list, and select (from the Test Requirements Document) the  
 1483 corresponding Test Requirements (and their "sub" Functional Requirements). The Test Driver then  
 1484 searches the Executable Test Suite document to find all Test Cases that "point to" the selected Functional  
 1485 Requirements. If more than one Test Case is necessary to satisfactorily test a single Functional  
 1486 Requirement (as is the case for Functional Requirement #1) there may be more than one Test Case  
 1487 pointing to it. The Test Driver would then execute Test Cases #1, #2 and #3 in order to fully test an  
 1488 ebXML application against Functional Requirement #1.

1489

1490 The only test material outside of the three documents below that MAY require an external file reference  
 1491 from within a Test Case are large, or non-XML message Payloads

1492

1493

1494

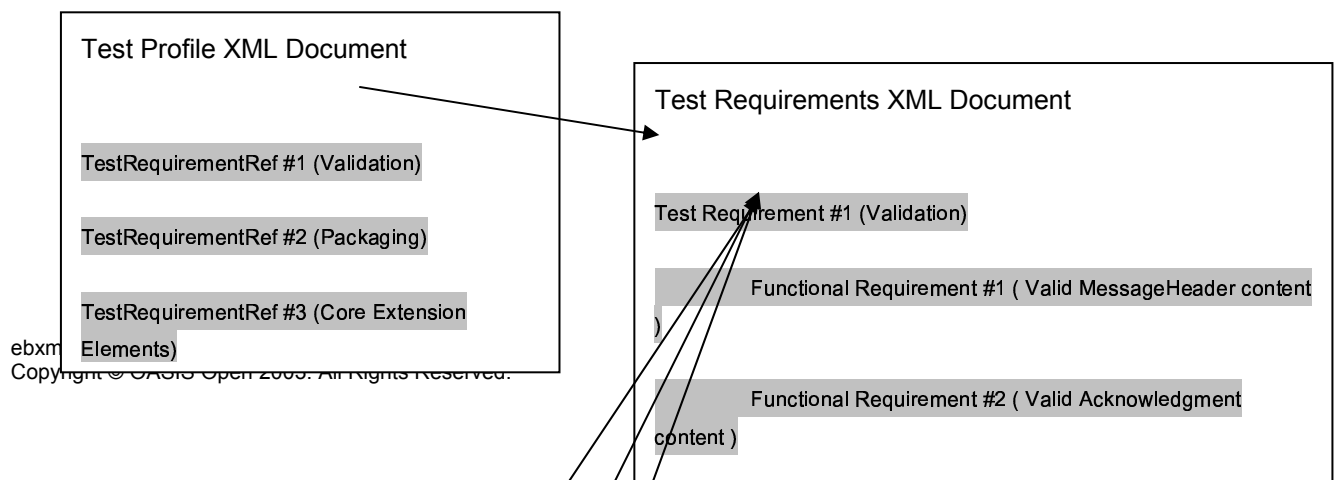
1495

1496

1497

1498

1499



1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528

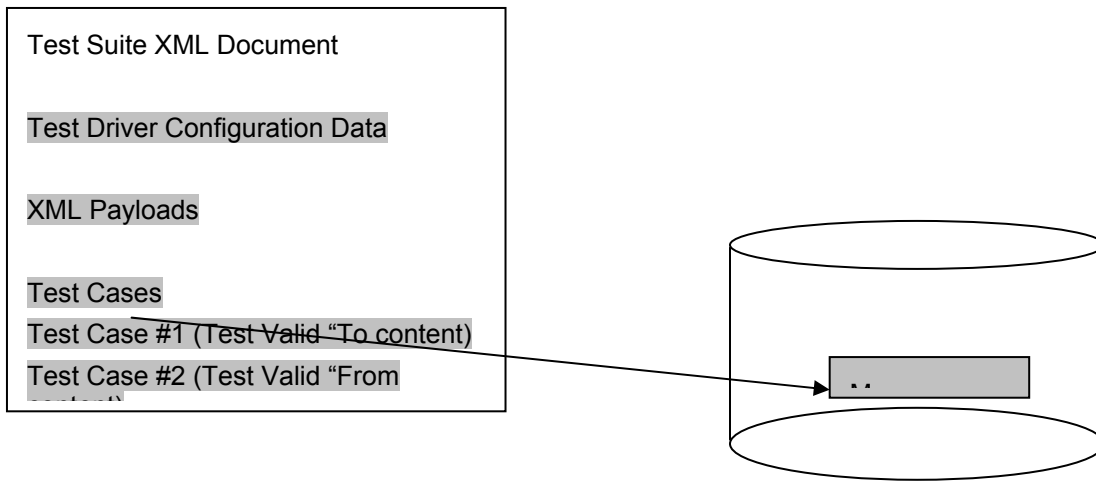


Figure 19 – Test Framework documents and their relationships

---

## 1529 7 Test Cases

1530

### 1531 7.1 Structure of a Test Case

1532

1533 An Executable Test Case is the translation of a Test Requirement (or a part of a Test Requirement), in an  
1534 executable form, for a particular Test Harness. A Test Case includes the following information:

1535

1536 Test Requirement reference.

1537 A workflow of Test Threads

1538 Testable assertion(s) of success or of failure of operations within those Threads.

1539

1540 NOTE: The same Test Case may consolidate several Test Requirement items, i.e. a successful outcome  
1541 of its execution will verify the associated set of Test Requirement items. This is usually the case when  
1542 each of these Test Requirement items can make use of the same sequence of operations, varying only in  
1543 the final test condition. When several Test Requirement items are covered by the same Test Case, the  
1544 processing of the latter SHOULD produce separate verification reports for each Test Requirement.

1545

1546

1547 Test Cases MUST evaluate to a value of “pass, fail, or undetermined”. The Test Case result is based  
1548 upon the final state of the Test Driver as it traverses the logic tree defined by the sequence of Test  
1549 Threads and logical branches. Ultimately, a Test Case result is determined by the state set by the  
1550 TestAssertion operations in the Test Case Workflow.

1551

1552 **A Test Case has a final state of “pass” if:**

1553 The last executed “TestAssertion” operation in the workflow sets the Test Case state to “pass”, and the  
1554 workflow executes to completion without any exception conditions.

1555

1556 **A Test Case has a final state of “fail” if:**

1557

1558 The last executed “TestAssertion” operation in the workflow sets the Test Case state to “fail”, and the  
1559 workflow executes to completion without any exception conditions.

1560

1561

1562

1563

1564 **A Test Case has a final state of “undetermined” if:**

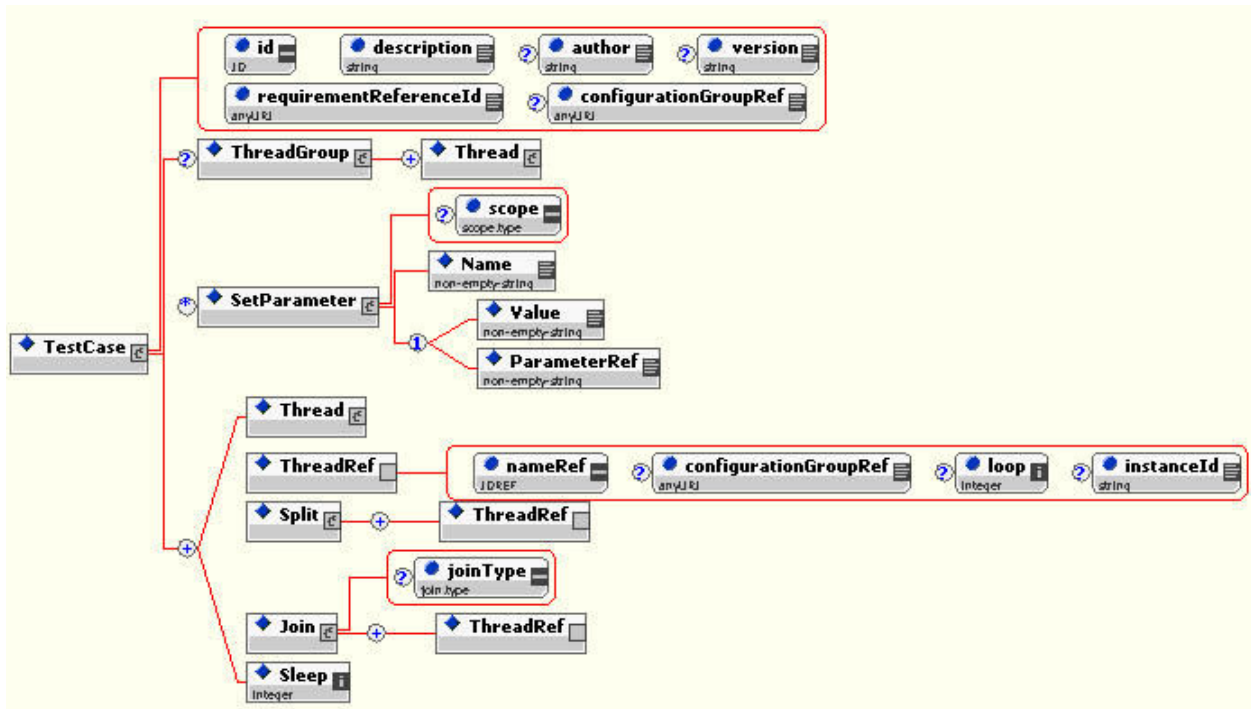
1565

1566 The last executed “TestAssertion” operation in the workflow sets the Test Case state to “undetermined”,  
1567 at which point the Test Driver automatically ceases execution o the Test Case.

1568

1569

1570  
 1571 OR  
 1572  
 1573  
 1574  
 1575 A system exception condition (as defined for each individual operation) occurs in the Workflow. For  
 1576 example, a protocol error occurring in a PutMessage or GetMessage operation will generate such an  
 1577 exception.  
 1578  
 1579  
 1580  
 1581  
 1582  
 1583  
 1584  
 1585  
 1586  
 1587



1588  
 1589 Figure 20 – Graphic representation of expanded view of the TestCase element

1590  
 1591  
 1592  
 1593 Definition of Content  
 1594



Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
TestCase	Container element for all test case content		Optional	
id	Unique identifier for this Test Case		Required	
description	Short description of TestCase		Optional	
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	Test Requirement not found
configurationGroupRef	URI pointing to a ConfigurationGroup instance used to reconfigure Test Driver		Optional	Configuration Group not found
ThreadGroup	Container for all Threads declared for this Test Case		Optional	
Thread	Definition of a subprocess of operations and/or Threads that may be forked synchronously or asynchronously		Required	
SetParameter	Contains name/value pair to be used by subsequent Threads in this Test Case		Optional	
TestServiceConfigurator	Container of configuration content for Test Service when Test Driver is in "service" mode		Optional	Unable to configure Test Service
ThreadRef	Name of the Thread to be executed in this TestCase		Optional	Thread not found
Split	Parallel execution of referenced sub-threads inside of the Split element		Optional	
Join	Evaluation of results of named threads ( as "andjoin" or "orjoin" ) permits execution of operations that follow the Join element		Optional	
Sleep	Instruction to the Test Driver		Optional	

	to “wait” for the specified time interval (in seconds). May be invoked anywhere in the script			
--	---	--	--	--

1595 Table 10 provides a list of TestCase element and attribute content

1596

### 1597 7.1.1 Test Threads

1598

1599 Test Threads are a workflow of operations and/or other sub-threads. One can think of a Thread as a  
 1600 collection of related operations (such as a sequences of operations performing message transmissions  
 1601 and receptions for a common business process). Operations and sub-threads contained in a Test  
 1602 Thread are executed sequentially as they appear in that Thread script.

1603

1604 Sub-threads MAY be executed in parallel if they are the child of a Split element.

1605 The Test Driver interprets a ThreadRef element as an instruction to execute the Thread instance whose  
 1606 name matches that defined in the ThreadRef. A Thread will be executed serially if its ThreadRef is not  
 1607 the child of a Split element.

1608

1609

1610

1611 A Join operation “synchs” the execution of the Test Case, waiting until one (orJoin) or all ( andJoin)  
 1612 Threads defined as children within the Join complete execution Concurrent Threads MUST be “joined”  
 1613 anywhere in the scripting AFTER the Split but within the same Thread in which they were invoked.

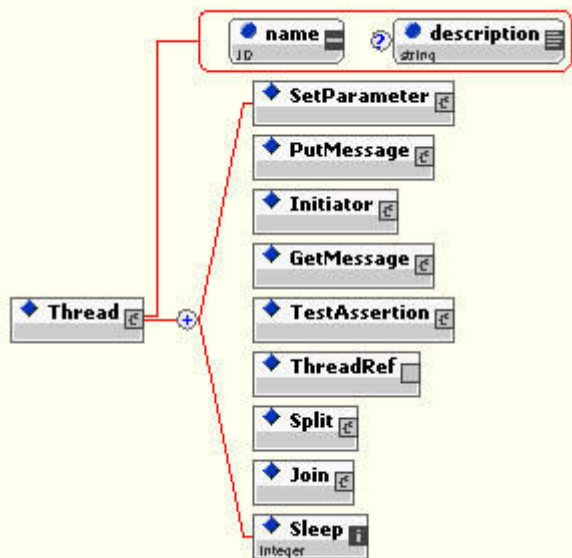
1614 Split Threads that are not Joined MUST generate an error message from the Test Driver and cause  
 1615 execution of the Test Case to cease, with a final Test Case result status of “undetermined”.

1616 A Join operation is by default an “andJoin”, unless specifically set otherwise by the “type” attribute of the  
 1617 Join element.

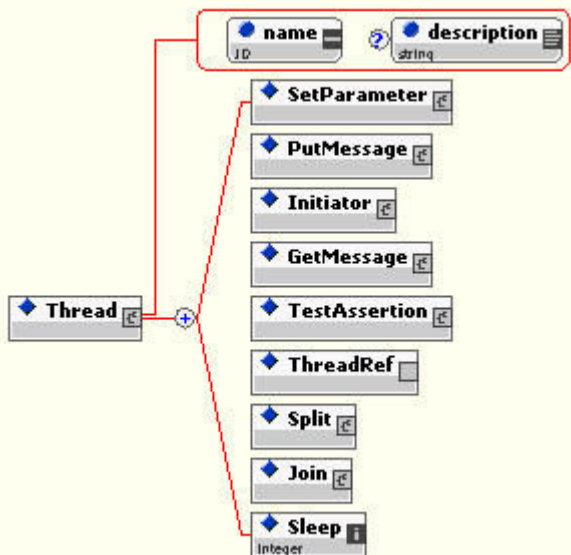
1618

1619

1620



1621



1622

1623

Figure 21 – The Thread content diagram

1624

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
name	Short name for the Thread		Optional	
description	Description of the Thread		Optional	
SetParameter	Set name/value pair to be used by subsequent Thread operations		Optional	
PutMessage	Instruction to Test Driver to send a message		Optional	Message could not be sent

Initiator	Instruction to Test Driver to pass a message declaration to the Test Service for sending		Optional	Message could not be initiated by Test Service
GetMessage	Instruction to Test Driver to retrieve message(s) from the Message Store		Optional	Protocol error occurred
TestAssertion	Instruction to the Test Driver to perform an evaluation		Optional	
ThreadRef	Reference via name to Thread to execute serially		Optional	Thread not found
Split	Directive to run the referenced Thread(s) enclosed in the Split element in parallel		Optional	Thread not found
Join	Directive to evaluate the boolean result of the enclosed referenced Thread(s) in a previous Split		Optional	Thread not found
Sleep	Instruction to "wait" (specified in integer seconds) a period of time before executing the next operation in the script		Optional	

1625 Table 11 – Thread Content Description

1626

1627

1628 **7.1.2 Thread Operations**

1629

1630 These operations may be performed by the Test Driver in one of two modes: connection (Test Driver is  
 1631 remote from Test Service) or service (Test Driver is interfaced with Test Service). The section below  
 1632 describes these operations and their behavior in various modes (if applicable) in more detail.

1633

1634

1635

1636

1637

1638

1639

1640

1641 DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclaration

1642

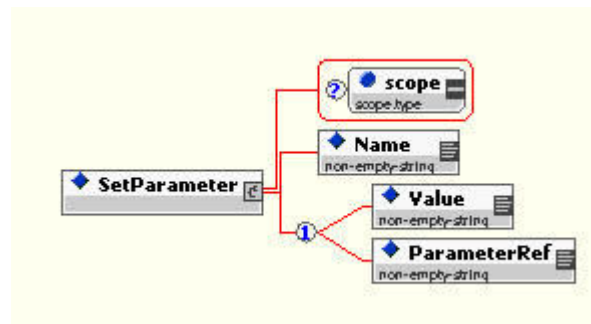
1643 Thread

1644 Table 12 – Expanded list of SetParameter element content

1645  
 1646  
 1647  
 1648  
 1649  
 1650  
 1651  
 1652

7.1.2.1 SetParameter: Setting Parameter values

The “SetParameter” operation instructs the Test Driver to create a name/value pair that can be used by reference by any subsequent operation in the current Thread, as well as any operation in any descendent Threads. Parameter names can be included in XSL stylesheets of message Mutators, or they may be referenced in XPath expressions to verify message content.



1653  
 1654  
 1655  
 1656  
 1657  
 1658  
 1659  
 1660  
 1661

Figure 22 – Graphic representatin of expanded view of SetParameter element Thread

Definition of Content

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
SetParameter	Instruction for Test Driver to store a name/value pair		Optional	
scope	Attribute to control visibility of parameter (selfAndDescendents   self )	selfAndDescendents	Optional	
Name	Parameter Name		Required	Not a valid name
Value	String representation of parameter value		Optional	Not a valid value
ParameterRef	Name of another parameter whose value you wish to store in this parameter		Optional	Parameter not found

1662  
 1663  
 1664  
 1665

Semantics of the SetParameter operation:

1666  
1667 Parameters that could be used to manipulate sent message content, or to evaluate received message  
1668 content, can be assigned for use by Thread operations in three ways:  
1669  
1670 Through assignment as a parameter name/value pair within the current ConfigurationGroup.  
1671 Using SetParameter at the beginning of a Thread  
1672 Using SetXPathParameter operation in a GetMessage operation (to extract a message content value via  
1673 XPath and assign it to a parameter)  
1674

#### 1675 7.1.2.1.1 Scope of a parameter

1676  
1677 These same semantic rules apply to parameters referenced via **ConfigurationGroup**. The  
1678 "configurationGroupRef" attribute is available for use at the TestSuite, TestCase, and Thread levels. A  
1679 hierarchical relationship exists for any parameters defined in the ConfigurationGroup. A  
1680 configurationGroupRef at the TestSuite level is "global", meaning any parameter definitions defined at the  
1681 TestSuite level are exposed to descendent TestCase or Thread. If a parameter is "redefined" at any of  
1682 those "lower levels" in the object hierarchy, then that definition takes precedence for that object and any  
1683 "descendent" objects, until the logical workflow of the TestCase moves back to the current level in the  
1684 object hierarchy. When that occurs, whichever previous definition of a parameter (via a  
1685 configurationGroupRef or SetParameter operation) takes precedence.

1686  
1687 The **SetParameter** operation dynamically creates (or redefines) a single parameter whose value is  
1688 available to the current Test Object (TestCase or Thread) it is defined in. For example, if it is defined  
1689 within a Thread, then it is available to any operation in that Thread, as well as any descendent Threads..  
1690 If it is defined within a Thread, then its definition exists for the lifecycle of that Thread. When the  
1691 workflow execution moves to a "higher" level (i.e. to the parent Thread containing the Thread) then that  
1692 parameter **a**) ceases to exist if it was not already defined at a higher level in the workflow hierarchy or **b**) if  
1693 defined at a higher level, takes the previously value defined at the next highest level in the workflow  
1694 hierarchy.

1695  
1696  
1697  
1698  
1699  
1700

#### 1701 7.1.2.1.2 Referencing/Dereferencing parameters in PutMessage and GetMessage operations

1702  
1703 In the case of a **PutMessage** operation, a parameter defined with the ConfigurationGroup and/or the  
1704 SetParameter operation can be passed to an XSL or XUpdate processor and referenced within an XSL  
1705 stylesheet or XUpdate "mutator" document (via its name) and used to provide/mutate message content of  
1706 the newly constructed message A Test Driver MUST make pass these parameters to the XSL or XUpdate  
1707 processors for use in mutating a Declaration.

1708 In the case of a **GetMessage** operation, a parameter defined with the ConfigurationGroup and/or the  
1709 SetParameter operation can be passed to the XPath processor used for the Filter or VerifyContent  
1710 operations. Within the XPath expression, the parameter MUST be referenced with the same name (case  
1711 sensitive) with which it has been assigned, and MUST be preceded by a '\$' character. The Test Driver

1712 MUST recognize the parameter within the XPath expression, and substitute its value prior to evaluating  
1713 the XPath expression

1714 How parameters are stored and retrieved by the Test Driver is an implementation detail.

1715

### 1716 7.1.2.1 PutMessage: Message Construction and Transmission

1717

1718

1719

1720 The “PutMessage” directive instructs the Test Driver to construct a message and transmit it to the  
1721 designated party. The PutMessage element contains a Declaration (i.e. an XML script) that is used as a  
1722 template to construct the message. The Test Driver must successfully construct and send the message;  
1723 otherwise it must generate an exception.

1724 The PutMessage operation instructs the Test Driver to build and send a messageDeclaration. A minimal  
1725 Declaration (contained within its child Declaration element) is required to construct a message and an  
1726 optional XSL stylesheet or XUpdate document MAY mutate that message declaration. Dynamic message  
1727 content such as timestamps, message ids and conversation ids are passed to the XSLT or XUpdate  
1728 processor through parameters created by the Test Driver. Additional message content may be passed to  
1729 the XSLT or XUpdate processor through parameter definitions defined by the test writer (using the  
1730 configurationGroupRef attribute or the SetParameter directive to define a name/value pair).

1731 DeclarationDeclarationDeclarationDeclaration

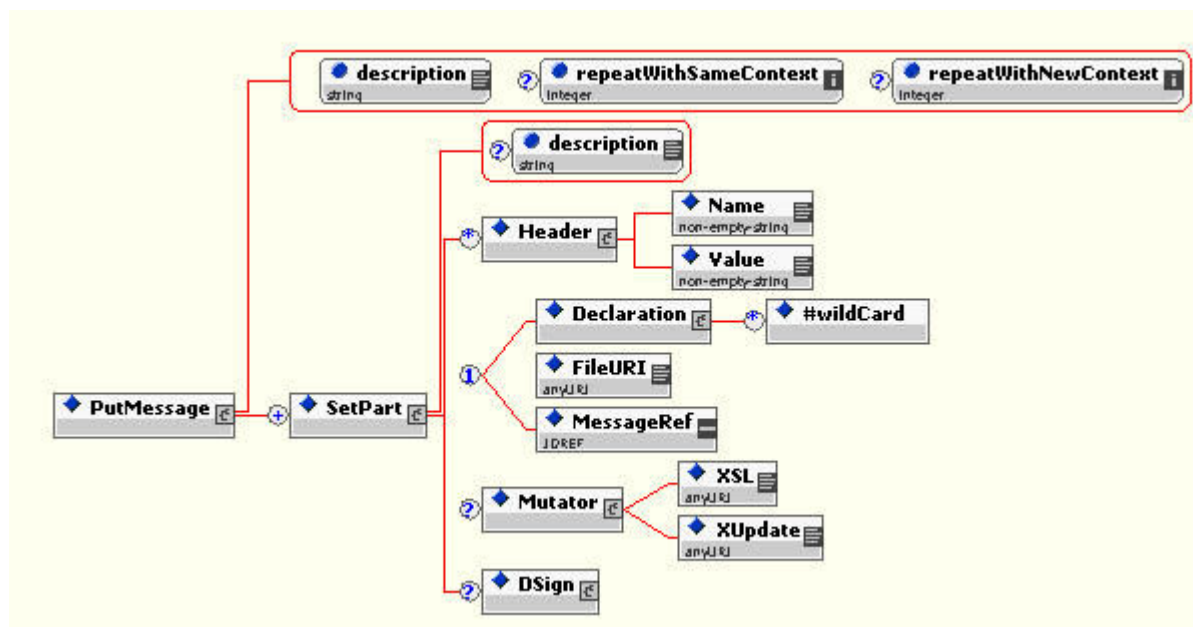
1732

1733

1734

1735

1736



1737

1738 Figure 23 – Graphic representation of expanded view of the PutMessage element

1739

1740

1741

1742 7.1.2.2

1743

1744 Definition of Content

1745

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
PutMessage	Container element for message construction and sending operation		Optional	Protocol error prevented message transmission
description	Metadata describing the nature of the PutMessage operation		Required	
repeatWithSameContext	Integer looping parameter, using same message context ( MessageId and Timestamp )		Optional	
repeatWithNewContext	Integer looping parameter, using new message context ( MessageId and Timestamp )		Optional	
SetPart	Container for construction of a part of the message to be sent		Required	
description	Description of the portion of the message being added		Required	
Header	Instruction to Test Driver to add an attribute name/value pair to this message portion		Optional	
Name	Message part attribute name		Required	
Value	Message part attribute value		Required	
Declaration	XML content defines message envelope to be created (or mutated) by Test Driver		Optional	
FileURI	Reference to message declaration contained in a file		Optional	File not found
MessageRef	Reference to an ID in the Test Suite whose parent is a Messageelement		Optional	Invalid Id
Mutator	Container element for a reference to either an XSL		Optional	



	stylesheet or XUpdate document			
XSL	URI to an XSL stylesheet		Optional	Stylesheet not found
XUpdate	URI to an XUpdate document		Optional	XUpdate script not found
DSign	Container element for XML Digital Signature declaration(s) for this message, used to sign any portion (envelope or payload(s)) of the message		Optional	

1746 Table 13 defines the content of the PutMessage element

1747

1748 Semantics of the PutMessage operation:

1749

#### 1750 7.1.2.2.1 The Declaration

1751

1752 The IIC Test Framework is a generalized testing framework, agnostic to any particular messaging  
 1753 protocol. As a result, it is designed in a very flexibly way. Any type of message can be expressed inside  
 1754 an XML Declaration element. As long as a “mutator” XSL styleesheet or Xupdate document is used to  
 1755 interpret that declaration and generate an actual message, it is up to the discretion of the test writer how  
 1756 they wish to express their message declaration.

1757

1758

1759

1760 DeclarationAs a best practice, the XML content necessary to describe a basic message should be  
 1761 minimal, with default parameter values supplied by the Test Driver for most common and reuseable  
 1762 message content (such as ConversatoinId, CPAId, Sending Party Id..etc) . If the test developer wishes to  
 1763 “override” the default element and attribute values, they may do so by explicitly declaring those values in  
 1764 the XML markup.

1765

1766 Default values for message content are typically set using the Test Suite ConfigurationGroup parameters.  
 1767 Setting parameter values at the Test Suite level makes them “global” for use by any Test Case in the Test  
 1768 Suite. Parameters such as CPAId, ConversationId, Service, Action, ToPartyId and FromPartyId (or their  
 1769 equivalent) would typically be set globally for a messaging Test Suite. They could be optionally  
 1770 “overridden” locally within each Test Case by use of an individual “SetParameter” instruction in the Test  
 1771 Case scripting.

1772

1773 A test writer may additionally override any Test Driver parameter value by explicitly specifying a value in  
 1774 the Declaration itself. For example, explicitly providing a ConversationId in the Declaration can be used  
 1775 as a way to override the Test Driver supplying it in its mutator transformation if the mutator is designed to  
 1776 allow it.

1777

1778 Two parameters (using the exact names specified below) are generated by the Test Driver, and CAN  
 1779 NOT be overridden using parameter definitions. They are MessageId and Timestamp. These two

1780 values can however, be explicitly defined in a Declaration if the test writer wishes to substitute an explicit  
1781 value for that supplied by the Test Driver in their mutator transformation.

1782  
1783 Although it is not required with the IIC Test Framework, it is generally helpful if a community testing a  
1784 particular eBusiness application uses an “agreed upon” schema and semantics to represent the format of  
1785 their Declaration. This simplifies understanding of test scripts among that community.

1786  
1787 As an example, a Declaration schema for ebXML Messaging Services v2.0 is described in Appendix C of  
1788 this document. Both a semantic description, as well as a normative schema is provided for test writers  
1789 wishing to write tests for ebXML MS v2.0.

1790  
1791 If the Declaration is not “inlined” as content in the Test Case script, it MAY be included via the FileURI  
1792 element content, or via the MessageRef (an IDREF pointing to a static Declaration already defined in the  
1793 Test Suite document.

1794  
1795 **7.1.2.2.2 Header: A Name/Value attribute for the message part**

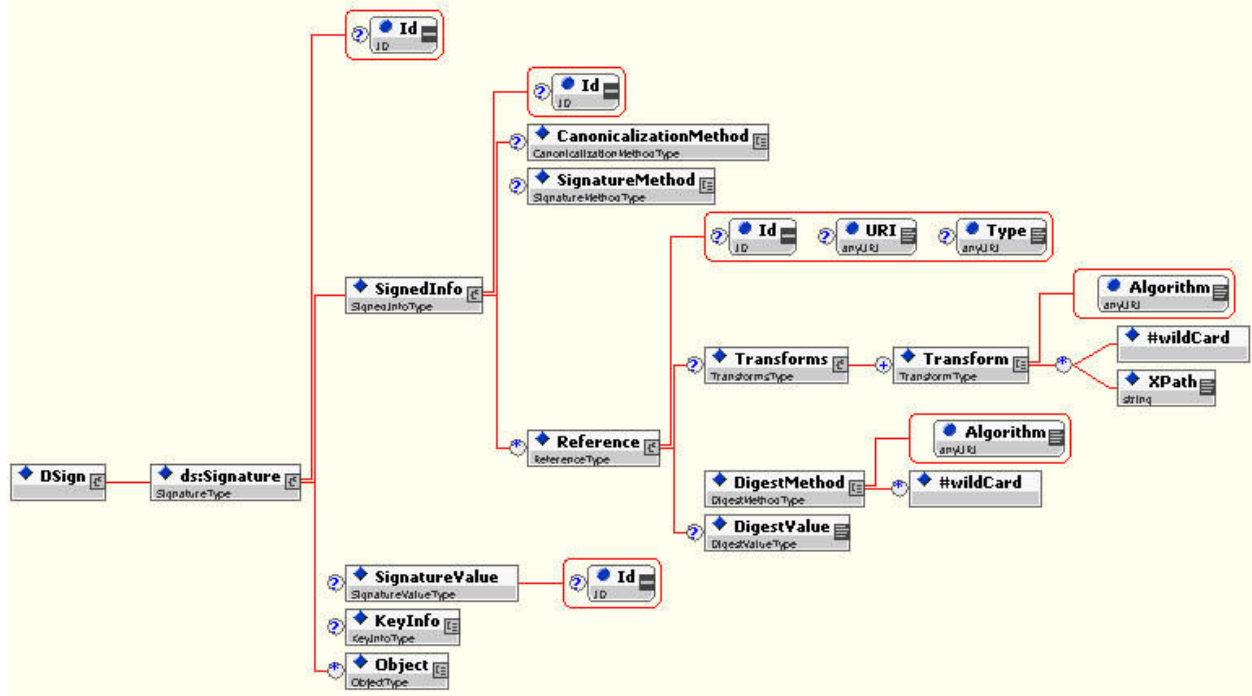
1796  
1797 Because the IIC Test Framework is agnostic to the messaging protocol used, a generic “name/value pair”  
1798 scheme is used to add attributes for the particular the message part.

1799  
1800 **7.1.2.2.3 Mutator: Turning a Declaration into an actual Message**

1801  
1802 A Declaration generally will need to be transformed into a complete and valid message. Additional  
1803 information such as a message timestamp, message identifier and other “run time” information may need  
1804 to be added to complete the message. A Mutator element provides a URI to an XSL or XUpdate  
1805 document that would transform the Declaration into a valid message.

1806  
1807  
1808  
1809 **7.1.2.2.4 DSign: Applying an XML Signature to the message**

1810  
1811 **The DSign instruction tells the Test Driver (or Test Service if doing interoperability testing) to**  
1812 **create and include an XML Signature element with the XML message ( or XML payload) after it is**  
1813 **constructed.**



1814  
 1815 Figure 39 – Graphic representation of expanded view of the DSign element  
 1816

1817  
 1818 **Definition of Content**  
 1819

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
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 Canonicalization and Signature algorithms and References		Required	
CanonicalizationMethod	Modify default container element	Container auto-generated by Test Driver	Optional	Method not supported

				by Test Driver
Algorithm	Modify default attribute and value	<a href="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">http://www.w3.org/TR/2001/REC-xml-c14n-20010315</a>	Required	Algorithm not supported by Test Driver
#wildCard	Generate content "inline"		Optional	
SignatureMethod	Create container element		Required	
Algorithm	Create attribute and value		Required	Algorithm not supported by Test Driver
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 element		Optional	
Transform	Generate element with its value		Optional	
Algorithm	Modify default attribute value	<a href="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">http://www.w3.org/TR/2001/REC-xml-c14n-20010315</a>	Required	Algorithm not supported by Test Driver
#wildCard	Generate content "inline"		Optional	
XPath	Generate element with its value		Optional	Invalid XPath expression

DigestMethod	Generate element with its value		Required	Method not supported by Test Driver
Algorithm	Generate attribute and value		Required	Algorithm not supported by Test Driver
#wildCard	Generate content "inline"		Optional	
DigestValue	Generate element with its value	Set by Test Driver, based upon URI 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	Invalid Key data
Object	Generate container element		Optional	

1820 Table 27 - Content of the Dsign element

1821

1822

1823

1824

1825

1826



1827

1828 Declaration

1829

1830

1831

1832

1833

1834 7.1.2.3 Initiator: Passing message construction directives to the Test Service

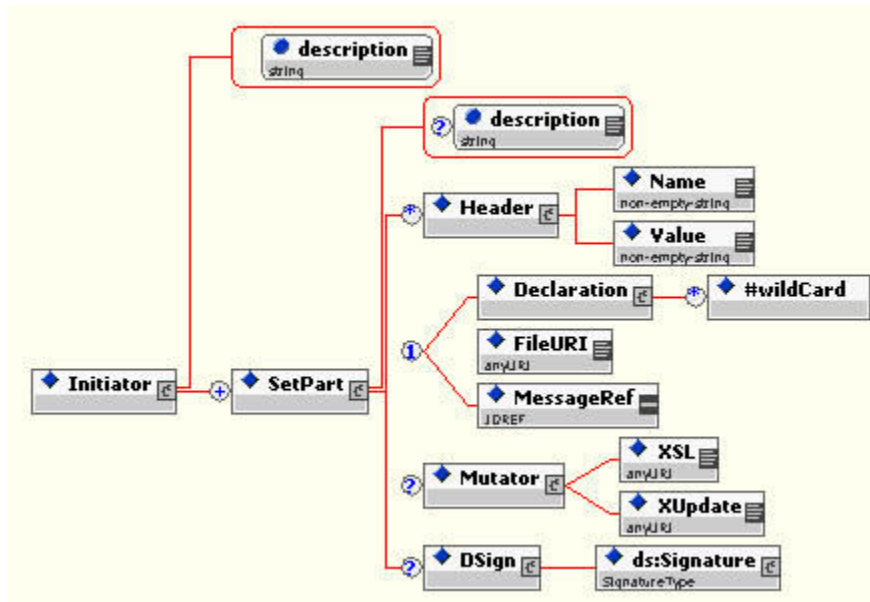
1835

1836 Unlike the “PutMessage” operation, in which the Test Driver constructs and sends a message, the  
1837 “Initiator” operation instructs the Test Driver to instead pass a Declaration (and any associated message  
1838 payloads) to the Test Service Initiation interface, via its “initiator” method. The initiator method of the Test  
1839 Service must successfully interpret the Declaration; construct the message Declaration and send the  
1840 message through its host messaging service. The Test Service initiator method must return a response  
1841 message (defined in Appendix F) to the Test Driver indicating success or failure.

1842 Semantically, all the “sub-operations” of Initiator are identical to that of PutMessage. The only difference  
1843 is that none of the actual message building or sending occurs within the Test Driver, but instead, the  
1844 message is built and sent by the Test Service through its MSH API.

1845

1846



1847

1848

1849 Figure 24 – Graphic representation of expanded view of the Initiator element

1850

1851 Definition of Content

1852

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
Initiator	Container element for message construction and sending operation		Optional	Protocol error prevented message transmission
description	Metadata describing the nature of the PutMessage operation		Required	

SetPart	Container for construction of a part of the message (Declaration or Payload) to be sent		Required	
description	Description of the portion of the message being added		Optional	
Header	Instruction to Test Serviceto add an attribute name/value pair to this message portion		Optional	
Name	Message part attribute name		Required	
Value	Message part attribute value		Required	
Declaration	XML content defines message envelope to be created by Test Service		Optional	
FileURI	Reference to message declaration contained in a file		Optional	File not found
MessageRef	Reference to an ID in the Test Suite whose parent is a Messageelement		Optional	
Mutator	Container element for a reference to either an XSL stylesheet or XUpdate document to mutate the Declaration, prior to passing it to the Test Service		Optional	
XSL	URI to an XSL stylesheet		Optional	Stylesheet not found
XUpdate	URI to an XUpdate document		Optional	XUpdate script not found
DSign	Container element for XML Digital Signature declaration(s) for this message, used to direct the Test Service to sign any portion (envelope or payload(s)) of the message		Optional	

1853 Table 14 defines the content of the Initiator element

1854

1855

1856

1857

1858

1859

1860

1861

1862

1863

1864

1865

1866

1867

DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclaration
DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclaration
DeclarationDeclaration
Declaration

1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877

Declaration

---



---



---



---



---



---



---



---

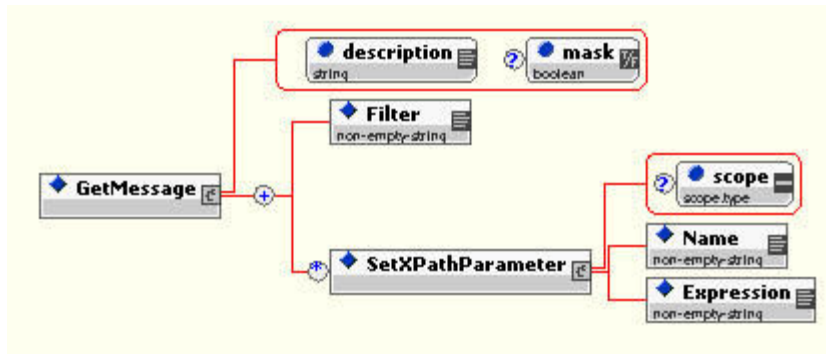
1878  
1879

1880 7.1.2.4 GetMessage: Message Retrieval

1881

1882 The “GetMessage” Thread operation is used by the Test Driver to retrieve incoming messages (when the  
1883 Test Driver is in Connection mode) and message notifications (when the Test Driver is in Service mode).  
1884 Incoming messages for a Test Case are maintained in a persistent Message Store for the life of a Test  
1885 Case.

1886  
1887  
1888  
1889  
1890  
1891



1892

1893 Figure 38 – Graphic representation of expanded view of the GetMessage element

1894  
1895  
1896  
1897  
1898  
1899  
1900

1901 Definition of Content

1902

Name	Description	Default Value	Required/Optional	Exception Condition
GetMessage	Container for instructions to retrieve a message(s) from the Message Store			



description	Metadata describing the nature of the SetPayload operation		Required	
mask	Instruction to hide any Filtered content from subsequent Filter XPath queries (true   false)	false	Optional	
Filter	Container for XPath query that is used to retrieve message content from Message Store		Required	
SetXPathParameter	Instruction directint the Test Driver to extract message content and store it in a parameter that is available (by default) to the current Thread and its descendent Threads		Optional	
scope	Visibility of parameter to current and descendent Threads (selfAndDescendents   self)	selfAndDescendents	Optional	
Name	Name of new parameter		Required	
Expression	XPath expression that returns message content to be used as the value of the parameter		Required	

1903 Table 26 defines the content of the GetMessage element

1904

1905

1906

1907

1908

1909

1910

1912

#### 1913 7.1.2.5 The Initiator Operation

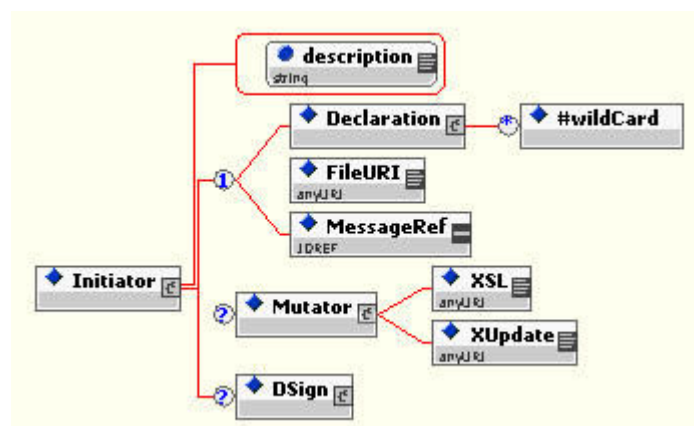
1914

1915 The Initiator Operation provides a means to initiate a conversation from the candidate MSH. The Test  
1916 Driver through the "Send" interface of the Test Service performs the Initiator operation. This is  
1917 accomplished programmatically if the Test Driver is "local" to the Test Service. If this is not the case,  
1918 then this is accomplished through a remote procedure call (RPC), described in section 3.2.4. The Test  
1919 Driver passes on the XML content illustrated and described below to the Test Service "initiator" RPC  
1920 method to construct a message. The type of content in the Declaration element will vary with the  
1921 message envelope type (e.g. ebXML, RNIF..etc.). Also, because it is the Test Service that is actually

1922 constructing the message (not the Test Driver), message declarations MUST only contain directives that  
 1923 the MSH API can execute. For example MIME and SOAP content is generally not available for  
 1924 manipulation by an ebXML MSH API. Therefore, MIME and SOAP message construction directives  
 1925 SHOULD NOT be present as Declaration content, or if present, MUST be ignored by the initiator method  
 1926 of the Send interface.

1927  
 1928 The schema illustrating the Declaration content for ebXML Messaging Services v2.0 testing can be found  
 1929 in Appendix C.

1930



1931  
 1932 Figure 40 – Graphic representation of expanded view of the generic Initiator element

1933

1934

1935 Definition of Content

1936

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
Initiator	Container element for message construction directives and message payloads to be passed to MSH via RPC		Optional	Protocol error prevented message transmission
description	Metadata describing the nature of the Initiator operation		Required	
SetMessageEnvelope	Content defines message envelope to be created (or mutated) by Test Driver		Optional	
Declaration	Message construction directives to be passed to MSH for interpretation and message generation		Optional	
FileURI	Reference to message declaration contained in a		Optional	File not found

	file			
MessageRef	Reference to an ID in the Test Suite whose parent is a Message element		Optional	
DSign	Container element for XML Digital Signature declaration(s) for this message, used to sign any portion (envelope or payload(s)) of the message		Optional	

1937 Table 28 – Content of the Initiator operation

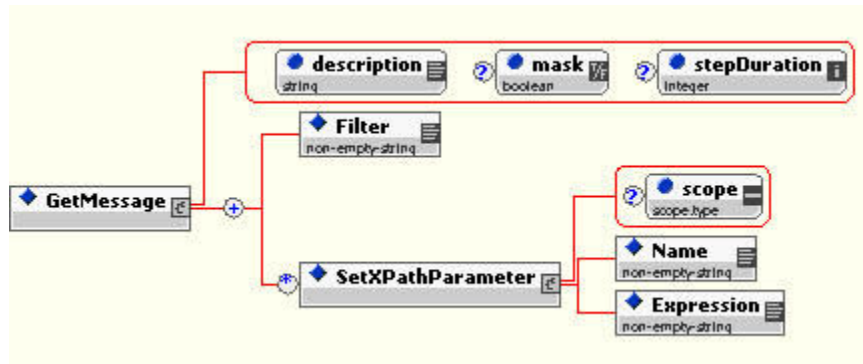
1938

1939

1940 7.1.2.6 The GetMessage Operation

1941

1942 The GetMessage Operation, using its child XPath Filter instruction, retrieves a node-list of Messages from  
 1943 the Message Store of the Test Driver. The content of the node-list is dependent upon the XPath Filter  
 1944 provided. The resulting node-list MAY then be queried for adherence to a particular Test Assertion  
 1945 Additionally, parameter values that may be used later in the Test Case script can be assigned using the  
 1946 SetXPathParameter instruction.



1947

1948 Figure 41 – Graphic representation of expanded view of the GetMessage element

Name	Description	Default Value from Test Driver	Required/Optional	Exception Condition
GetMessage	Container element for filtering, verifying and validating message and payload content		Optional	
description	Description the nature of the GetMessage operation		Required	
mask	Boolean attribute, when set to "true" will "mask" ( hide) the message(s) which satisfy the XPath expression. When "false", the Test Driver will NOT mask any messages	false	Optional	
Filter	Select a node list from Message Store based upon the XPath query supplied as		Required	Not a valid XPath or well formed XPath expression
stepDuration	Maximum time (in seconds) that the Test Driver MUST wait to satisfy the XPath expression before continuing script execution		Optional	
SetXPathParameter	Set the value of a parameter with the value of a node returned by an XPath query against a Filtered message retrieved from the Message Store		Optional	Invalid XPath syntax in Expression element
scope	Constraint on visibility of parameter to other Threads (self   selfAndDescendents)	(selfAndDescendents)	Required	
Name	Parameter name		Required	
Expression	XPath expression used to capture parameter value from FilterResult		Required	Not a valid XPath expression

1949

1950 Table 29 defines the content of the GetMessage element

1951

1952 7.1.2.6.1 Semantics of the GetMessage operation

1953

1954

1955 A fundamental aspect of the GetMessage operation is its behavior and effect over the Message Store.  
1956 The Message Store is an XML document object created by the Test Driver that contains an XML  
1957 representation of all synchronous and asynchronously received ebXML messages for a Test Case. The  
1958 received messages for a particular Test Case MUST persist in the Message Store for the life of the Test  
1959 Case. Messages in the Message Store MAY contain an XML representation of all MIME, SOAP, ebXML  
1960 or other types of message content, represented as an XML document (the schema permits any type of  
1961 XML representation of a messaging envelope, with each representation specified in a "best practice"  
1962 document for a particular testing community). The particular XML representation of a message in the  
1963 Message Store is based upon a "best practice" schema for representing a particular message type. If the  
1964 messages being stored are ebXML messages using HTTP transport and a SOAP envelope, the XML  
1965 format of the Message Store document MUST validate against the ebXMLMessageStore.xsd schema in  
1966 appendix D. The scope of message content stored in the Message Store is "global", meaning its content

1967 is accessible at any time by any Thread (even concurrently executing Threads) during the execution of a  
1968 Test Case. Message Store content changes dynamically with each received message or notification.

1969

1970 The GetMessage "Filter" operation queries the Message Store document object, and retrieves the XML  
1971 content that satisfies the XPath expression specified in its Filter child element. As the MessageStore is  
1972 updated every time a new message comes in, a GetMessage operation will automatically execute as  
1973 often as needed, until either (1) its XPath Filter is satisfied (evaluates to "true"), or (2) the timeout  
1974 (stepDuration) expires.

1975

1976 The XPath query used as content for a Filter operation MUST yield a node-list of 0 or more XML  
1977 elements. Although the content of a message may vary (e.g. ebXML, RNIF, SOAP), all node-list results  
1978 from a Filter operation MUST contain XML elements in order to permit the creation of a FilterResult  
1979 document object, which can then be examined by the TestAssertion operation. The required structure of  
1980 the FilterResult document object is defined in the Filter Result schema in Appendix D.

1981

1982 Message Masking:

1983

1984 All the message items available for querying are children of the MessageStore document object. The  
1985 Xpath expression in the Filter will typically select Message Store content that satisfies the filter. Such  
1986 content MUST be a node list of XML elements. If they are not, the Test Driver MUST generate an  
1987 exception and terminate the Test Case with a final result of "undetermined".

1988 The elements returned by the XPath query are appended as children of a FilterResult element, available  
1989 for further querying, by the TestAssertion operation.

1990 When the mask attribute is set to "true", the messages ( or XML elements) that have been selected  
1991 by a GetMessage operation are "invisible" to future GetMessage operations in the same test case. By  
1992 default, filtering is not performed by the Test Driver.

1993

1994 Setting Parameters using user-defined or received Message Content:

1995

1996 In addition to storing message content, the Message Store MAY also store parameter values to be used  
1997 in the evaluation of subsequent received messages. This is not an implementation requirement however.

1998 As in the case of the SetParameter operation described in 4.2.2.1 , parameters may also be  
1999 defined/redefined through the SetXPathParameter operation. This operation extracts message content  
2000 from the Message Store and stores it as a parameter value. Whether it is a message header, or an XML  
2001 message payload being examined, the test writer may assign a parameter name, and an XPath pointing  
2002 to the content to be stored as a parameter. Each parameter value is a string representation of the  
2003 nodelist content retrieved by the XPath query.

2004

2005

2006

2007

2008

2009

2010

2011

2012

2013

2014  
2015  
2016  
2017

2018 •

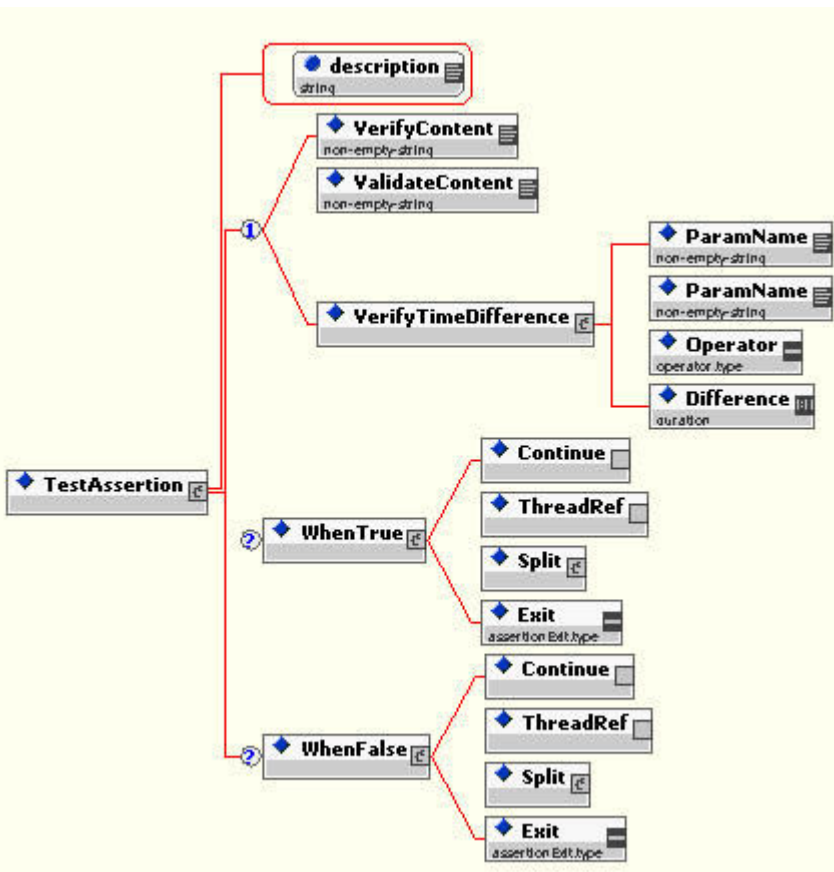
2019  
2020  
2021

2022

2023 7.1.2.7 The TestAssertion Operation

2024

2025 The TestAssertion Operation verifies a Test Requirement through one of three possible sub-operations.  
2026 These sub-operations are: VerifyContent (compare message content to expected values),  
2027 ValidateContent (validate the structure of a document, or a single item in the document) and  
2028 VerifyTimeDifference (compare a computed time difference between two parameters against an expected  
2029 value).



2030  
2031  
2032

Figure 44 – Graphic representation of expanded view of the TestAssertion element

2033  
2034  
2035

Definition of Content

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
description	Metadata describing the nature of the TestPreCondition operation		Required	
VerifyContent	XPath expression to evaluate content of message(s)		Optional	Invalid XPath expression
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	Invalid XPath expression
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] or a URI of a Schematron schema		Optional	Schema not found
VerifyTimeDifference	Instruction to Test Driver to compute the time difference between two parameters and determine if the difference is less than equal or greater to an expected value		Optional	
ParamName	Parameter used in computation of time difference		Required	
Operator	(lessThen lessThanOrEqual equal greaterThan greaterThanOrEqual)		Required	
Difference	Expected value		Required	
WhenTrue	Branching instruction based upon boolean result of the TestAssertion operation		Optional	
WhenFalse	Branching instruction based upon boolean result of the TestAssertion operation		Optional	

2036 Table 32 defines the content of the TestAssertion element

2037

2038 7.1.2.7.1 Semantics of the TestAssertion operation

2039

2040 The TestAssertion operation MUST return either a true or false result (or semantically a pass/fail result) to  
2041 the Test Driver.

2042

2043 If TestAssertion includes a VerifyContent sub-operation, the VerifyContent operation MUST yield a  
2044 boolean value of true/false. If the verification is an XPath operation, the VerifyContent XPath expression  
2045 may yield a node-set, boolean, number or string object. All of these resulting objects MUST be evaluated  
2046 using the “boolean” function described in [XPath]. Those evaluation rules are:

2047

- 2048 • a returned node-set object evaluates to true if and only if it is non-empty
- 2049 • a returned boolean object evaluates to true if it evaluates to “true” and false if it evaluates to  
2050 “false”
- 2051 • a returned number object evaluates to true if and only if it is neither positive or negative zero nor  
2052 NaN
- 2053 • a returned string object evaluates to true if and only if its length is non-zero

2054

2055

2056

2057 **If the TestAssertion sub-operation is ValidateContent, then the content pointed to by the XPath**  
2058 **expression contained in the text content MUST validate according to its contentType attribute .** The  
2059 ValidateContent operation MUST yield a boolean value of true/false. Rules for determining the resulting  
2060 Boolean value are:

2061

- 2062 • if the contentType attribute value is XMLSchema, as defined in [XML] , the operation evaluates to  
2063 true if the content at the specified XPath validates according to the schema defined in the  
2064 “schemaLocation” attribute
- 2065 • if the contentType is URI, as defined in [XMLSCHEMA], the operation evaluates to true if the  
2066 content at the specified XPath is a valid URI
- 2067 • if the contentType is dateTime, as defined in [XMLSCHEMA], the operation evaluates to true if  
2068 the content at the specified XPath is a valid dateTime
- 2069 • if the contentType is signature, as defined in [XMLDSIG], the operation evaluates to true if the  
2070 content at the specified XPath is a valid signature.

2071 If the TestAssertion sub-operation is VerifyTimeDifference, then two dateTime parameter values are  
2072 compared, with an operator of “lessThan, lessThanOrEqual, equal, greaterThan, greaterThanOrEqual”.  
2073 The TestAssertion operation evaluates to “true” if the equation is satisfied, otherwise it returns a value of  
2074 “false” to the Test Driver. The Test Driver MUST generate an exception and exit the Test Case if any of  
2075 the parameters used in VerifyTimeDifference operation are not a dateTime type.

2076

2078

2079

2080

2081

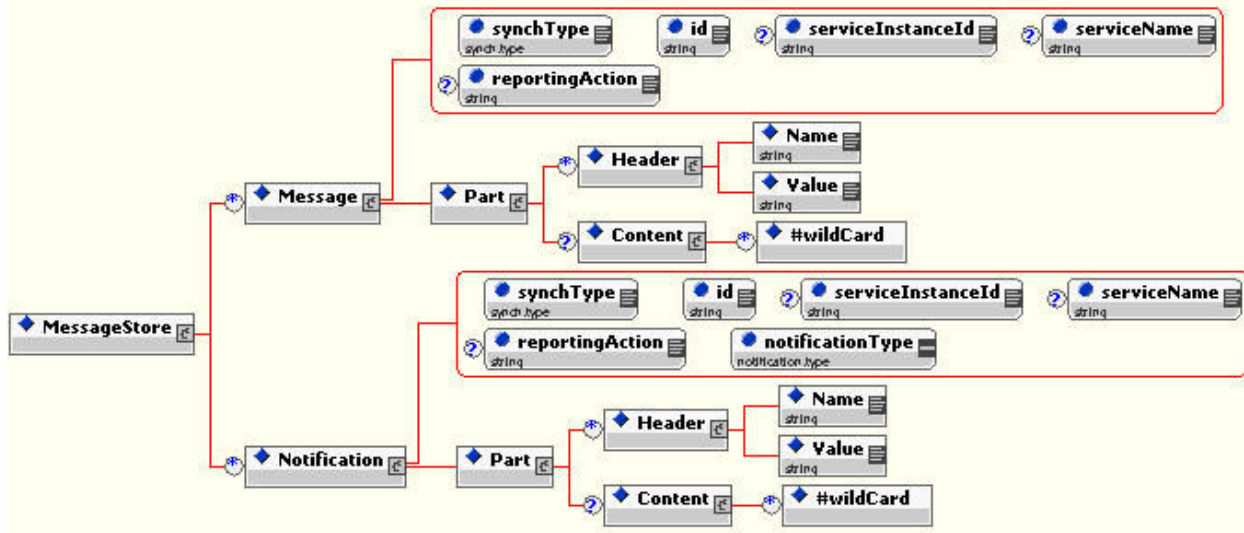


2082  
2083  
2084

2085 **7.1.3 Message Store Schema**

2086

2087 The Generic Message Store schema (Appendix D) describes the XML document format required for a  
2088 Test Driver implementation. The schema facilitates a standard XPath query syntax to be used for retrieval  
2089 and evaluation of received messages, notifications and (optionally) parameter names and values by the  
2090 Test Driver. The “generic” schema design of the Message Store document object permits virtually any  
2091 type of XML format for messages and notifications to be stored and queried via XPath.



2092

2093

2094

2095

2096

2097

2098

2099 Figure 47 – Graphic representation of expanded view of the generic Test Driver MessageStore schema

2100

2101

2102 Description of Content

2103

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
MessageStore	Container for all message, notification and possibly parameter values for a		Required	

	Test Case instance			
Message	Container for a received message, along with some overhead attributes describing the type of message, its origin etc		Optional	
synchType	Descriptor of type of how message was received (synchronous asynchronous)		Required	
id	Test driver provided unique identifier of received message		Required	
serviceInstanceid	Unique identifier of the Test Service that generated the received message		Optional	
serviceName	Name of the Service that generated the received message		Optional	
reportingAction	Name of the action that generated the received message		Optional	
Part	Container for content of a single portion of entire message		Required	
Header	Container of any name/value attribute associated with this particular message part			
Name	Container for actual message part attribute name		Required	
Value	Container for actual message part attribute value		Required	
Content	Container for actual XML message. If message part is not XML, then no Content element is present		Optional	
#wildcard	Any XML representation of message content (typically conforming to specified schemas)		Required	
Notification	Container for any type of message received by a Test Service and reported to the Test Driver		Optional	
notificationType	Type of notification (message, errorURL, errorApp)		Required	

2104  
2105  
2106

### 2107 7.1.3.1 Semantics of the Message Store

2108  
2109

2110 The Message schema permits any type of message representation. Messages are required to have a  
2111 unique ID within the Message Store, and a “synchType” attribute, identifying the message as received  
2112 either synchronously or asynchronously. Messages (unlike Notifications) are received directly by the  
2113 Test Driver (i.e. the Test Driver is in “connection” mode). Hence message content is more complete ,  
2114 since it was received “over the wire”, and all content is accessible to the Test Driver.

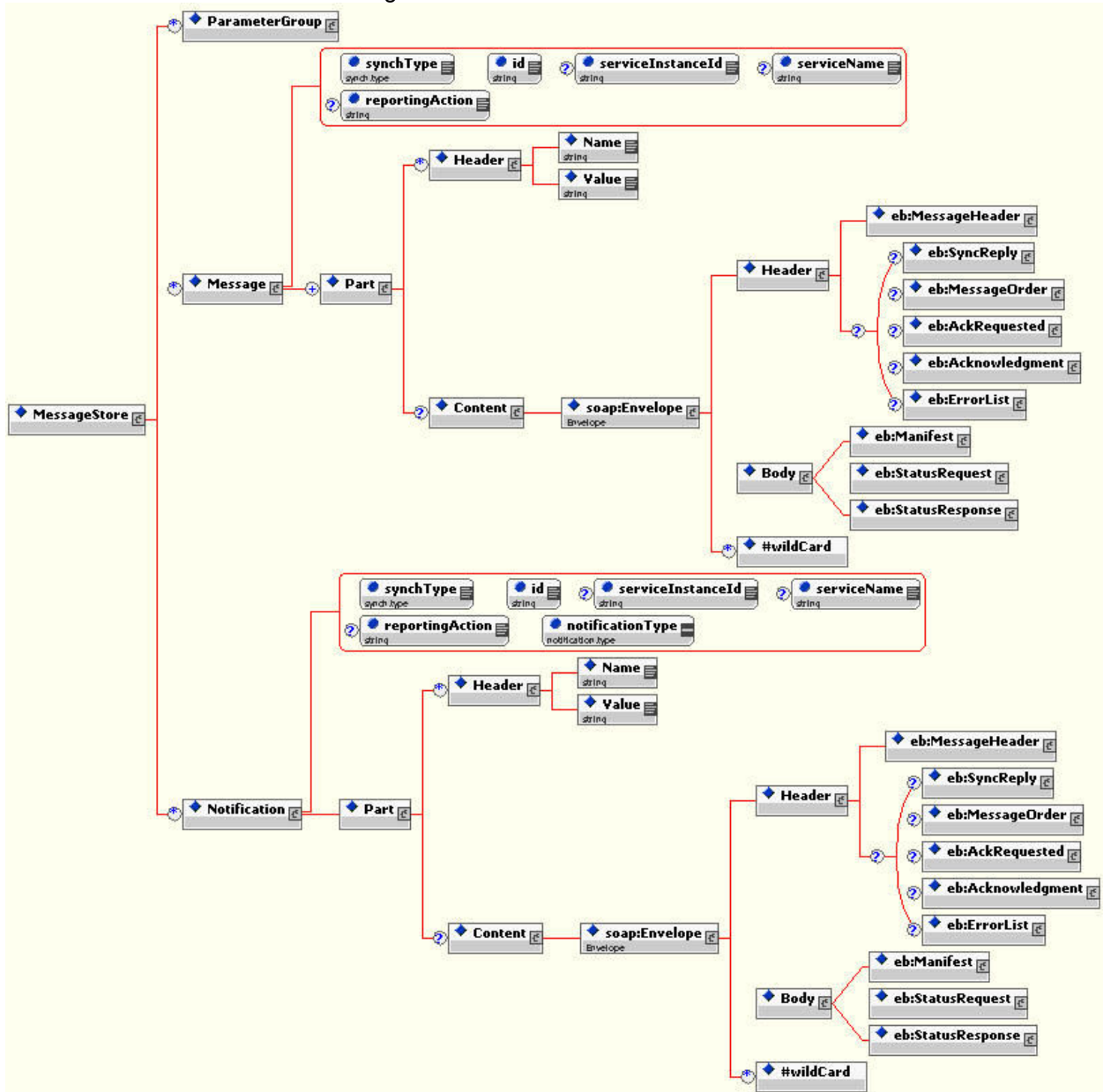
2115

2116 Notification messages are received via an interface from the Test Service. Because the messaging  
2117 system under test cannot be trusted to provide the notifications, they are either passed locally (via the  
2118 Test Service Notification interface) or remotely (via RPC) between Test Service and Test Driver via the

2119 Test Driver "Receive" interface. As a result, message content is restricted to what part of the message  
 2120 was exposed to the Test Service application layer. Therefore the representation or received messages  
 2121 passed via notification is less complete than message content directly received by the Test Driver (for  
 2122 example, MIME content may not be exposed to a Test Service application, therefore MIME headers are  
 2123 not represented in the Notification message). For all other purposes however, the format of the  
 2124 Notification message content is identical to that of a message directly received by the Test Driver.  
 2125

2126 7.1.3.2 ebXML Specific Message Store Schema

2127  
 2128 The ebXML MS v2.0 Message Store Schema (Appendix D) defines the structure of an individual ebXML  
 2129 MS version 2.0 message received over HTTP. This schema MUST be used to define the message  
 2130 structure for ebXML MS V2.0 messages and notifications.



2131

2132 Figure 48 – Graphic representation of expanded view of Message Store content model, specifically for  
 2133 ebXML/SOAP messaging services

2134  
 2135  
 2136  
 2137  
 2138

2139 Definition of Content

2140

2141 The content represented in the figure 48 above is that defined for a Message Store containing ebXML  
 2142 message content received directly by the Test Driver (in connection mode) and message notification  
 2143 content received by the Test Service (with the Test Driver in service mode). All SOAP and ebXML  
 2144 Messaging Services message content validates to the schemas defined in their respective [SOAP] and  
 2145 [ebMS] specifications.

2147 Table 35 defines the content of the MessageStore element

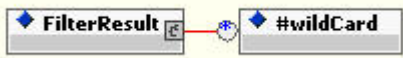
2148  
 2149

2150 7.1.3.3 Filter Result Schema

2151  
 2152  
 2153

2154 Like the Message Store, the Filter Result is a document object that can be queried for content testing and  
 2155 verification. Unlike the MessageStore, the FilterResult document object only needs to exist for the  
 2156 lifecycle of a single Thread. The Filter Result document is identical (in structure) to the MessageStore  
 2157 document, with one exception. The root node of the Filter Result document is a FilterResult element, not  
 2158 a MessageStore element. The content of the Filter Result MUST be a node list object whose node(s) are  
 2159 XML elements. This means that any Filter XPath expression MUST always query for elements within the  
 2160 Message Store. Doing so means that the Test Driver will be able to construct a document object from the  
 2161 Filter node list, and use it for subsequent VerifyContent and ValidateContent operations.

2162



2163

2164 Figure 49 – Generic Filter Result schema, permitting any Message Store element content

2165  
 2166  
 2167  
 2168  
 2169

2170 Definition of Content

2171

Name	Declaration Description	Default Value From Test Driver	Required/Optional
FilterResult	Container for XML representation of all messages received by Test Driver		Required

	for a given Test Case		
#wildcard	Any Message Store element content		Optional

2172 Table 36 defines the content of the FilterResult element

2173

2174

### 2175 7.3 Test Service Configurator, Initiator, and Notification Message Formats

2176

2177 The Test Service Message Schema (Appendix F) describes an XML syntax that MUST be followed for  
 2178 passing Test Service configuration, message construction and message notification data between the  
 2179 Test Driver to the Test Service when the Test Driver is either interfaced with the Test Service, or is  
 2180 remote to the Test Service but is receiving notification messages from the Test Service via RPC.

2181

2182 If the Test Service is in “local reporting mode”, configuration and message initiation information is passed  
 2183 from the Test Driver to the Test Service via the Test Service “Send” and “Configuration” interfaces.

2184 The Send interface provides the “initiator” method to start a new conversation or to construct a message  
 2185 with the conversationId already provided by the Test Driver.

2186 The Configuration interface provides the “configurator” method, which provides the t fundamental  
 2187 parameters for setting the state of the Test Service (ResponseURL, NotificationURL ,ServiceMode and  
 2188 PayloadDigests).

2189

2190

2191

2192 The message initiation and Test Service configuration use the same methods if the Test Service is in  
 2193 “remote reporting mode”. The only difference is that the messages are passed between the two test  
 2194 components via a Remote Procedure Call (RPC) instead of via local calls to respective interfaces.

2195

2196

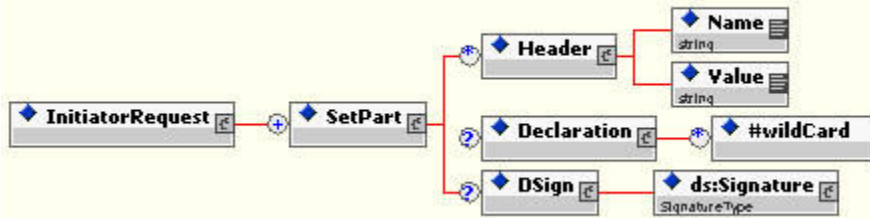
2197

2198 Using an alternate channel for Test Service configuration, message initiation and message reporting  
 2199 separates the implementation under test from the actual testing infrastructure. This helps to isolate  
 2200 failures in conformance and interoperability from failures in the test harness.

2201 The particular alternate communication binding that a test driver and test service implement is not  
 2202 mandated in this specification, however (as an example) an abstract definition and WSDL definition with a  
 2203 SOAP binding is provided in section 3.2.5. The list below describes each of the alternate channel  
 2204 messages defined in Appendix H.

2205

2206 **InitiatorRequest** – XML message content to be interpreted by the Test Service initiator method to  
 2207 construct an ebXML Message (or any other message envelope). This XML request is passed to a  
 2208 candidate MSH Test Service via the Send interface (if the Test Driver is in service mode) or via a remote  
 2209 procedure call to the Test Service (if the Test Driver is in connection mode). The first argument carries  
 2210 the message envelope construction declarations. The second argument is a list of message payloads to  
 2211 be added to the message. If the Test Driver is in “service” mode, the configuration parameters are  
 2212 passed to the Send interface via the initiator method call. If the Test Driver is in “loop” mode, the two  
 2213 parameters are passed to the Test Service via RPC call to the initiator method.



2214  
 2215 Figure 51 – Initiator request content  
 2216  
 2217 Definition of Content  
 2218

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
InitiatorRequest	Container for message declaration		Required	
SetPart	Container for message component declarations		Required	
Header	Generic		Optional	
Name	Name of message part header		Required	
Value	Value assigned to message part header		Required	
DSign	Instruction to Test Service to digitally sign (using [XMLDSIG] the appropriate part of the message		Optional	

2219 Table 37 – Describes the content of the InitiatorRequest element

2220  
 2221  
 2222 **InitiatorResponse** – XML message content to be interpreted by the Test Driver, with a result of “success”  
 2223 or “failure” returned by the Test Service. The response is passed to Test Driver through its Receive  
 2224 interface (if Test Driver is in Service mode) or sent to the getMessage method of the Test Driver Receive  
 2225 RPC Service (if Test Driver is in Loop mode). In both cases, the getMessage method is invoked on the  
 2226 Test Driver. The response message is added to the Message Store by appending its content to a  
 2227 Message Store “Message” element. The Test Driver will automatically evaluate the result of the response  
 2228 message, and exit the Test Case with a final status of “undetermined” if the initiator result is “failure”.  
 2229 Otherwise, the Test Case will proceed to the next operation. Response message content is appended to  
 2230 a Message Store Message element “as is”, with appropriate service instance, reporting action and other  
 2231 information provided as



2232  
 2233

2234 Figure 52 – Graphical representation of the InitiatorResponse schema

2235

2236 Definition of Content

2237

Name	Declaration Description	Default Value From Test Service	Required/Optional	Exception Condition
InitiatorResponse	Container for response from Test Service		Required	
Success	Boolean result (true   false) for conversation initiation from Test Service		Required	

2238

2239 Table 38 – Describes the content of the InitiatorResponse element

2240

2241

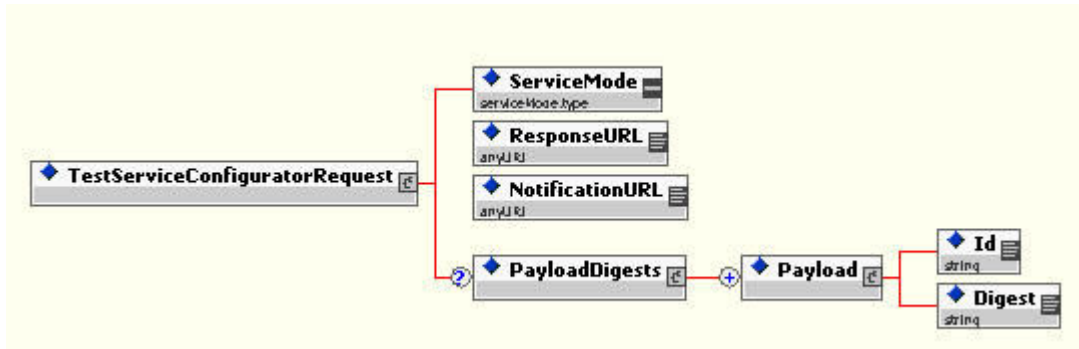
2242

2243

2244

2245 **TesetServiceConfiguratorRequest** – XML message content passed to a candidate MSH Test Service,  
 2246 to be interpreted by the configurator method call. Content consists of three required parameter names  
 2247 and their corresponding values and types. If the Test Driver is in “service” mode, the configuration  
 2248 parameters are passed to the Test Service Configuration interface via the configurator method call. If the  
 2249 Test Driver is in “loop” mode, the parameters are passed to the Test Service via RPC call to the  
 2250 configurator method.

2251



2252

2253 Figure 53 – A Graphical representation of the ConfiguratorRequest content schema

2254

2255

2256 Definition of Content

2257

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition

OperationMode	Toggle mode to ( local-reporting   remote-reporting   loop )		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	
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	

2258 Table 39 – Describes the content of the ConfigurationRequest element

2259

2260 **TestServiceConfiguratorResponse** – XML message content to be interpreted by the getMessage  
2261 method of the Test Driver Receive interface. The response is passed to Test Driver through its Receive  
2262 interface (if Test Driver is in Service mode) or sent to the Test Driver Receive RPC Service (if Test Driver  
2263 is in Loop mode). In both cases, the getMessage method is invoked on the Test Driver. The Test Driver  
2264 will automatically evaluate the result of the response message, and exit the Test Case with a final status  
2265 of “undetermined” if the XML content in the response message indicates “failure” to configure the Test  
2266 Service. Otherwise, the Test Case will proceed to the next operation. Response message content is  
2267 appended to a Message Store Message element “as is”, and providing the required service instance,  
2268 reporting action and other information.

2269



2270

2271 Figure 54 - A graphical representation of the ConfiguratorResponse content schema

2272 Definition of Content

2273

Name	Declaration Description	Default Value From Test Service	Required/Optional	Exception Condition
TestServiceConfiguratorResponse	Container for response from Test Service		Required	
Success	Boolean result (true   false) for Test Service configuration		Required	

2274 Table 40 – Description of content for the ConfiguratorResponse element



2275  
2276  
2277  
2278  
2279  
2280  
2281  
2282  
2283  
2284  
2285  
2286  
2287  
2288  
2289  
2290  
2291  
2292  
2293  
2294  
2295  
2296  
2297  
2298  
2299  
2300  
2301  
2302  
2303  
2304  
2305  
2306  
2307  
2308  
2309  
2310  
2311  
2312

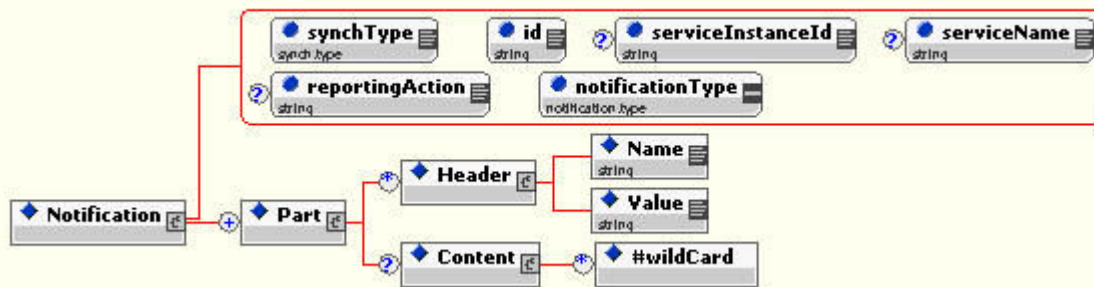
**Notification** – XML message envelope and payloads passed from the Test Service to the Test Driver. This includes errorURL notifications, errorApp notifications and any messages received by the Test Service while operating in “reporting” mode. Notifications are passed to Test Driver through its Receive interface (if Test Driver is in Service mode) or sent to the Test Driver via messaging to the Test Driver “Notify” action. In both cases, the Test Driver will automatically append the received Notification element and content the root element of the Message Store. Additional message payloads associated with the message MUST be stored by the Test Driver for examination by a “GetPayload” operation if necessary. If a particular Test Case must verify that a particular message was received by the candidate implementation, then a GetMessage operation examining the MessageStore for that particular notification message MUST be performed to verify conformance or interoperability.

Although the Notification message format is stored the same way in the MessageStore, there are important differences for each type of notification.

A Notification message with a notificationType attribute of “message”, looks in many ways like a message received directly by a Test Driver, with the exception that some information may not be present (such as MIME header content), since this portion of the message may not be exposed to the methods of the Test Service Notification interface.

A Notification message with a notificationType attribute value of “errorURL” is similar to a generic “message” notification, with the exception that the message was passed to the Test Driver in response to an erroneous message received by the candidate MSH. The content of the notification is the error message that the candidate MSH would normally send to the requesting party or to an identified error reporting URI if one were defined.

A Notification message with a notificationType attribute value of “errorApp” is identical to an “errorURL” notification, with the exception that error list provided in the notification contains “application-level” errors that are not normally returned to the sending party, but are handled internally by the candidate implementation under test.



2313

2314  
 2315  
 2316  
 2317  
 2318

Figure 55 – Graphical representation of the Notification element content schema  
 Definition of Content

Name	Declaration Description	Default Value From Test Driver	Required/Optional	
Notification	Container for reported message content		Optional	
synchType	Descriptor of type of how message was received by Test Service		Required	
id	Test Service provided unique identifier of received message		Required	
serviceInstanceid	Unique identifier of the Test Service that generated the notification		Optional	
serviceName	Name of the Service that generated the notification		Optional	
reportingAction	Name of the action that generated the notification		Optional	
notificationType	Type of notification message. (ErrorURL   ErrorApp   Message)		Required	
Part	Portion of the message received by the Test Service		Required	
Header	Generic container for any attributes and their values associated with this message part		Optional	
Name	Message part attribute name		Required	
Value	Message part attribute value		Required	
Content	Container for any XML representation of message content for this part of the message		Optional	
Content			Optional	

2319 Table 41 – Description of MessageNotification element content

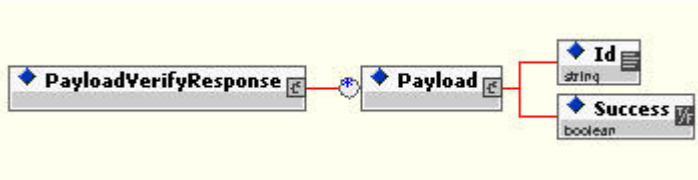
2320  
 2321  
 2322  
 2323  
 2324  
 2325

**NotificationResponse** – XML message content to be interpreted by the Test Service. The response is returned by the notify method of the Test Driver or sent to the Test Service as an RPC response.



2326  
2327 Figure 56 - A graphical representation of the NotificationResponse content schema  
2328 Definition of Content

2329  
2330 **PayloadVerifyResponse** – XML message content to be interpreted by the “notify” method of the Test  
2331 Driver’s “Receive” interface. This message content is an attachment to the notification message.  
2332



2333  
2334 Figure 57 - A graphical representation of the PayloadVerifyResponse content schema  
2335 Definition of Content  
2336  
2337 Definition of Content  
2338

Name	Declaration Description	Default Value From Test Driver	Required/Optional	
PayloadVerifyResponse	Container for results of comparison of message payload received by candidate MSH with their MD5 digest values		Required	
Payload	Container for individual payload verification result		Required	
Id	ID of the payload		Required	
Success	Boolean comparison result for an individual payload		Required	

2339 Table 42 – Description of PayloadVerifyResponse content  
2340

## 2341 7.4 Test Report Schema

2342  
2343 The Test Report schema (Appendix G ) describes the XML report document format required for Test  
2344 Driver implementations. The schema facilitates a standard XML syntax for reporting results of Test Cases  
2345 and their Threads.  
2346 The Test Report is a “full trace” of the Test Case. All XML content in the XML Test Case is available in  
2347 the Test Report. Additionally, a “result” element is appended to certain operation elements in the trace, to  
2348 provide diagnostic information. The “result” attribute MUST have a value of “pass”, “fail” or  
2349 “undetermined”. The Test Report schema is too large to graphically display on this page. Please consult  
2350 Appendix G if you wish to examine the normative schema.

2351  
2352  
2353  
2354  
2355  
2356

---

## 2357 8 Test Material

2358

2359 Test material to support the ebXML Testing Framework includes:

2360

2361 A Testing Profile XML document

2362 A Test Requirements XML document

2363 A Test Suite XML document

2364 Message Declaration Mutator document

2365 Collaboration Agreement document (if needed to configure an MSH)

2366

2367

2368

### 2369 8.1.1 Testing Profile Document

2370

2371 Both conformance and interoperability testing require the creation of a Testing Profile XML document,  
2372 which lists the Test Requirements against which Test Cases will be executed. A Test Profile document  
2373 MUST be included in an interoperability of conformance test suite. The Testing Profile document MUST  
2374 validate against the ebProfile.xsd schema in Appendix A.

2375

### 2376 8.1.2 Test Requirements Document

2377

2378 Both conformance and interoperability testing require the existence of a Test Requirements document.  
2379 While Test Requirements for conformance testing are specific and detailed against an ebXML  
2380 specification, interoperability Test Requirements may be more generic, and less rigorous in their  
2381 description and in their reference to a particular portion of an ebXML specification. However, both types  
2382 of testing MUST provide a Test Requirements XML document that validates against the  
2383 ebXMLTestRequirements.xsd schema in Appendix B.

2384

### 2385 8.1.3 Test Suite Document

2386

2387 Both conformance and interoperability testing require the existence of a Test Suite XML document that  
2388 validates against the ebTest.xsd schema in Appendix C. It is important to note that test case scripting  
2389 inside the Test Suite document MUST take into account the test harness architecture. Although a Test  
2390 Driver in Connection Mode can manipulate low-level message content (such as HTTP or MIME header  
2391 content) such content may not be accessible by a Test Driver in Service Mode, as the MSH does not  
2392 communicate this data to the application layer. Therefore, the following test scripting rules SHOULD be  
2393 followed when designing Test Cases:

2394 Message content described in a Message Declaration MUST be restricted to the business envelope and  
2395 its content, and not include references to the transport protocol content. Transport level content MAY be  
2396 described via the Header (name/value pair) child element of the message Part.

2397

2398  
2399  
2400

#### 2401 8.1.4 Mutator documents

2402 When the Test Driver is in “connection mode”, a message declaration content MAY be “mutated” via an  
2403 XSL or XUpdate processor into a valid message for transmission by the Test Driver. Likewise, when a  
2404 Test Driver is in “service mode”, a message declaration content MAY be “mutated” in to a format suitable  
2405 for interpretation by the Test Service Receive interface, and its message “initiator” method.

2406 Because a message Declaration element content can be any well-formed XML content, message Mutator  
2407 content can also be any valid XSLT or XUpdate format that will mutate its corresponding Declaration  
2408 content. It is HIGHLY RECOMMENDED that a particular testing community agree to a common message  
2409 Declaration and Mutator content schema in order to provide understandability and minimize the  
2410 duplication of effort in constructing conformance and interoperability test suites within that community.

2411 The OASIS IIC has adopted a message declaration schema for ebXML Messaging Services v2.0  
2412 conformance and interoperability testing. It has also defined an XSL stylesheet to mutate that declaration  
2413 into an ebXML message. The schema and stylesheet are available in Appendix G.

2414

2415 Likewise, communities wishing to test other messaging services, or other web applications SHOULD  
2416 devise a schema and stylesheet for their particular testing purpose. These documents SHOULD be  
2417 published as a “recommended practice” for that particular testing community, to minimize the work  
2418 involved in creating test suites that can be used with any IIC Test Framework implementation.

2419

#### 2420 8.1.5 CPAs

2421

2422 For ebXML Messaging Services (MS) testing), both conformance and interoperability testing require the  
2423 existence of a “base” CPA configuration that describes the “bootstrap” configuration of the candidate  
2424 MSH for conformance and interoperability testing. Additional CPAs MAY be needed if testing requires  
2425 different configurations of the candidate MSH. All CPA configurations MUST be uniquely defined (via a  
2426 CPA ID) and documented in the Conformance or Interoperability Test Suite Specification document  
2427 accompanying the Executable Test Suite. How the CPA configuration is presented to the candidate MSH  
2428 implementation is not defined in this specification.

2429

2430

---

## 2431 9 Test Material Examples

2432

2433 This section includes example test material to illustrate

2434

2435 A Test Requirements Document – Listing all Test Requirements for an ebXML implementation

2436 A Test Profile Document – Listing all selected Test Requirements to be exercised

2437 A Test Suite Document – Listing all Executable Test Cases for an ebXML implementation

2438 A Mutator XSL Stylesheet

2439

### 2440 9.1 Example Test Requirements

2441

2442 Below are two XML documents illustrating how Test Requirements are constructed, in this case for an  
2443 ebXML MS 2.0 implementation. In this particular case, the two documents represent Conformance and  
2444 Interoperability Test Requirements for an ebXML Messaging Services V2.0 implementation. The  
2445 example XML documents below include a subset of testing requirements defined for implementations of  
2446 the ebXML Messaging Services v2.0 Specification. Each Test Requirement may have one or more  
2447 Functional Requirements that together must be satisfied in order for an implementation to fully meet that  
2448 Test Requirement.

2449

2450

#### 2451 9.1.1 Conformance Test Requirements

2452

2453 In the example below, a “packaging” TestRequirement element contains two FunctionalRequirement  
2454 elements. The first Functional Requirement states that the primary SOAP message MUST be the first  
2455 MIME part of the message. The second packaging Functional Requirement states that the Content-Type  
2456 MIME header of the Message Package MUST be “text/xml”. If all Test Cases having a requirement  
2457 reference to these two Functional Requirements “pass”, then an ebXML MS v2.0 implementation would  
2458 be deemed “conformant” to the specification for the “Packaging” of ebXML messages. Of course, this is a  
2459 limited set of Test Requirements for illustrative purposes only.

2460

```
2461 <?xml version="1.0" encoding="UTF-8" ?>
2462 <Requirements xmlns="http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs"
2463 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2464 xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs/
2465 ebXMLTestRequirements.xsd">
2466 <MetaData>
2467 <Description>Master Requirements File: ebXML Messaging Services 2.0</Description>
2468 <Version>1.0</Version>
2469 <Maintainer>Michael.Kass<Michael.kass@nist.gov></Maintainer>
2470 <Location>http://www.oasis-open.org/committees/ebxml-
2471 iic/ebmsg/requirements1.0.xml</Location>
2472 <PublishDate>20 Feb 2003</PublishDate>
2473 <Status>DRAFT</Status>
2474 </MetaData>
2475 <!--Main Test Requirement, for message packaging-->
2476 <TestRequirement id="req_id_2" name="PackagingSpecification" specRef="ebMS-2#2.1"
2477 functionalType="packaging">
```

```

2478 <!--Define first sub-requirement to fulfill packaging testing-->
2479 <FunctionalRequirement id="funreq_id_2"
2480 name="GenerateConformantSOAPWithAttachMIMEHeaders" specRef="ebMS-2#2.1.2">
2481 <Clause>
2482 <!--Set first condition of the message is of type "multipart-mime" -->
2483 <Condition id="condition_id_2" requirementType="required">For each generated message,
2484 if it is multipart MIME</Condition>
2485 <Or />
2486 <!--Set alternate condition that the message is not "text/xml" -->
2487 <Condition id="condition_id_305" requirementType="required">if it is not
2488 text/xml</Condition>
2489 </Clause>
2490 <!--Define the Assertion that the first part of message is a SOAP message -->
2491 <Assertion id="assert_id_2" requirementType="required">The primary SOAP message is
2492 carried in the root body part of the message.</Assertion>
2493 </FunctionalRequirement>
2494 <!--Define a second sub-requirement to fulfill packaging testing-->
2495 <FunctionalRequirement id="funreq id 4" name="GenerateCorrectMessagePackageContent-Type"
2496 specRef="ebMS-2#2.1.2">
2497 <Clause>
2498 <!--Define condition that the candidate MSH generates a message -->
2499 <Condition id="condition_id_4" requirementType="required">For each generated
2500 message</Condition>
2501 </Clause>
2502 <!--Define the Assertion that the Content-Type of MIME header of that message is
2503 "text/xml" -->
2504 <Assertion id="assert_id_4" requirementType="required">The Content-Type MIME header in
2505 the Message Package contains a type attribute of "text/xml".</Assertion>
2506 </FunctionalRequirement>
2507 </TestRequirement>
2508 <!--Define a new Test Requirement, for the Core Extension Elements of messaging-->
2509 <TestRequirement id="req_id_3" name="CoreExtensionElements" specRef="ebMS-2#3.1.1"
2510 functionalType="packaging">
2511 <!--Define a sub-requirement to test the CPAId extension element-->
2512 <FunctionalRequirement id="funreq_id_35" name="ReportFailedCPAIDResolution"
2513 specRef="ebMS-2#3.1.2">
2514 <Clause>
2515 <!--First , set condition of a candidate MSH receiving a message with an unresolvable
2516 CPAId-->
2517 <Condition id="condition_id_40" requirementType="required">For each received message,
2518 if value of the CPAId element on an inbound message cannot be resolved</Condition>
2519 </Clause>
2520 <!--Next , define the Assertion that the candidate MSH MUST ( since requirementType is
2521 "required") respond with an Error-->
2522 <Assertion id="assert_id_35" requirementType="required">The MSH responds with an error
2523 (ValueNotRecognized/Error).</Assertion>
2524 </FunctionalRequirement>
2525 <!--Define a sub-requirement to test continuity in message ConversationId-->
2526 <FunctionalRequirement id="funreq id 36" name="ProvideConversationIdIntegrity"
2527 specRef="ebMS-2#3.1.3">
2528 <Clause>
2529 <!--First , set condition of all messages generated by a Candidate Implementation
2530 pertaining to a single CPAId-->
2531 <Condition id="condition_id_41" requirementType="required">For each generated message
2532 within the context of the specified CPAId</Condition>
2533 </Clause>
2534 <!--Next , define the Assertion that a ConversationId element is always present-->
2535 <Assertion id="assert_id_36" requirementType="required">The generated ConversationId
2536 will be present in all messages pertaining to the given conversation.</Assertion>
2537 </FunctionalRequirement>
2538 </TestRequirement>
2539 </Requirements>
2540

```

2541  
2542



## 2543 9.1.2 Interoperability Test Requirements

2544

2545 In the example below, a “basic interoperability profile” TestRequirement element contains two  
2546 FunctionalRequirement elements. The first Functional Requirement states that ebXML MS  
2547 implementation MUST be able to receive and send a basic ebXML message without a payload. The  
2548 second packaging Functional Requirement states that an ebXML MS implementation MUST be able to  
2549 process and return a simple ebXML message with one payload. If all Test Cases having a requirement  
2550 reference to these two Functional Requirements “pass”, then an ebXML MS v2.0 implementation would  
2551 be deemed “interoperable” to the Basic Interoperability Profile Specification for ebXML Messaging. Of  
2552 course, this is a limited set of Test Requirements for illustrative purposes only.

2553

```
2554 <?xml version="1.0" encoding="UTF-8" ?>
2555 <Requirements xmlns="http://www.oasis-open.org/tc/ebxml-iic/interop/reqs"
2556 xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
2557 xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/interop/reqs
2558 ebXMLTestRequirements.xsd">
2559 <MetaData>
2560 <Description>Interoperability Requirements File: ebXML Messaging Services
2561 2.0</Description>
2562 <Version>1.0</Version>
2563 <Maintainer>Michael Kass <michael.kass@nist.gov></Maintainer>
2564 <Location>http://www.oasis-open.org/committees/ebxml-
2565 iic/ebmsg/ms_2.0_interop_requirements1.0.xml</Location>
2566 <PublishDate>11 Feb 2003</PublishDate>
2567 <Status>DRAFT</Status>
2568 </MetaData>
2569 <!--Main Test Requirement, for basic interoperability testing-->
2570 <TestRequirement id="req_id_1" name="Basic Interoperability Profile" specRef="MS 2.0 BIP
2571 0.8" functionalType="basic interoperability">
2572 <!--Define first sub-requirement to fulfill basic testing, sending a "no payload"
2573 message-->
2574 <FunctionalRequirement id="funreq_id_1" name="BasicExchangeNoPayload" specRef="ebMS 2.0
2575 BIP#3.2.1">
2576 <Clause>
2577 <!--First , set condition of a candidate MSH receiving a message with no payload-->
2578 <Condition id="condition_id_1" requirementType="required">For each received ebXML
2579 message with no payload, received by the "Dummy" action</Condition>
2580 </Clause>
2581 <!--Next , define the Assertion of expected behavior for the Dummy Action-->
2582 <Assertion id="assert_id_1" requirementType="required">The message is received and
2583 processed, and a simple response message is returned</Assertion>
2584 </FunctionalRequirement>
2585 <!--Define second sub-requirement to fulfill basic testing, sending a "one payload"
2586 message-->
2587 <FunctionalRequirement id="funreq id 2" name="BasicExchangeOnePayload" specRef="ebMS 2.0
2588 BIP#3.2.2">
2589 <Clause>
2590 <!--Set condition of a candidate MSH receiving a message with one payload-->
2591 <Condition id="condition_id_2" requirementType="required">For each received ebXML
2592 message with one payload, received by the "Reflector" action </Condition>
2593 </Clause>
2594 <!--Define the Assertion of expected behavior for the Reflector Action-->
2595 <Assertion id="assert_id_2" requirementType="required">The message is received and
2596 processed, and a simple response message with the identical payload is
2597 returned</Assertion>
2598 </FunctionalRequirement>
2599 <!--Define third sub-requirement to fulfill basic testing, sending a "three payload"
2600 message-->
2601 <FunctionalRequirement id="funreq id 3" name="BasicExchangeThreePayloads" specRef="ebMS
2602 2.0 BIP#3.2.3">
2603 <Clause>
2604 <!--Set condition of a candidate MSH receiving a message with three payloads-->
```

```

2606 <Condition id="condition_id_3" requirementType="required">For each received ebXML
2607 message with three payloads, received by the "Reflector" action</Condition>
2608 </Clause>
2609 <!--Define the Assertion of expected behavior for the Reflector Action-->
2610 <Assertion id="assert_id_3" requirementType="required">The message is received and
2611 processed, and a simple response message with the identical three payloads are
2612 returned</Assertion>
2613 </FunctionalRequirement>
2614 <!--Define third sub-requirement to fulfill basic testing, generating Error messages-->
2615 <FunctionalRequirement id="funreq_id_4" name="BasicExchangeGenerateError" specRef="ebMS
2616 2.0 BIP#3.2.4">
2617 <Clause>
2618 <!--Set condition of a candidate MSH receiving an erroneous message-->
2619 <Condition id="condition_id_4" requirementType="required">For each received basic
2620 ebXML message that should generate an Error </Condition>
2621 </Clause>
2622 <!--Define the Assertion of expected behavior for the candidate MSH -->
2623 <Assertion id="assert_id_4" requirementType="required">The message is received and,
2624 the MSH returns a message to the originating party with an ErrorList and appropriate
2625 Error message </Assertion>
2626 </FunctionalRequirement>
2627 </TestRequirement>
2628 </Requirements>

```

2629  
2630

## 2631 9.2 Example Test Profiles

2632

2633 Below are two XML documents illustrating how a Test Profile document is constructed, in this case for an  
2634 ebXML MS v2.0 implementation. The example XML documents below represent a subset of test  
2635 requirements to be exercised. The Test Profile document provides a list of ID references (pointers) to  
2636 Test Requirements or Functional Requirements in an external Test Requirements document (see above).  
2637 A Test Harness would read this document, resolve the location of the Test Requirements document, and  
2638 then execute all Test Cases in the Test Suite document that point to (via ID reference) the Test  
2639 Requirements listed below. Note that a Test Driver can execute Test Cases pointing to a Functional  
2640 Requirement (discreet requirement) or a Test Requirement (a container of a group of Functional  
2641 Requirements). If the TestRequirementRef id attribute value points to a Test Requirement, then all Test  
2642 Cases for all child Functional Requirements will be executed by the Test Harness (This is a way to  
2643 conveniently execute a cluster of Test Cases by specifying a single Test Requirement.). This method is  
2644 used for both conformance and interoperability testing.

2645

### 2646 9.2.1 Conformance Test Profile Example

2647

2648 The Test Profile document below would be used to drive a Test Harness, by executing all Test Cases that  
2649 point (via ID) to the listed Test Requirement references (including individual Functional Requirements and  
2650 a single Test Requirement listed in the above example Conformance Test Requirements document.

2651

```

2652 <?xml version="1.0" encoding="UTF-8" ?>
2653 <TestProfile xmlns="http://www.oasis-open.org/tc/ebxml-iic/test-profile"
2654 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
2655 open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-
2656 profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml"
2657 name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing
2658 profile for ebXML MS v2.0 implementations">

```

```
2659 <TestRequirementRef id="funreq_id_2" /> <!--Execute all Test Casses that reference the
2660 Basic SOAP message structure Functional Requirement-->
2661 <TestRequirementRef id="funreq_id_4" /> <!--Execute all Test Cases that reference Message
2662 Packaeg Content Type Functional Requirement-->
2663 <TestRequirementRef id="req_id_2" /> <!--Execut all Test Cases that reference all
2664 Functional Requirements within the Core Extension Elements Test Requirement-->
2665 </TestProfile>
```

2666

## 2667 9.2.3 Interoperability Test Profile

2668

2669 The Test Profile document below would be used to drive a Test Harness, by executing all Test Cases that  
2670 point ( via ID ) to the listed Test Requirement references ( including individual Functional Requirements  
2671 and a single Test Requirement listed in the above example Interoperability Test Requirements document.

2672

```
2673 <?xml version="1.0" encoding="UTF-8" ?>
2674 <TestProfile xmlns="http://www.oasis-open.org/tc/ebxml-iic/test-profile"
2675 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
2676 open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-
2677 profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml"
2678 name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing
2679 profile for ebXML MS v2.0 implementations">
2680 <TestRequirementRef id="funreq_id_1.1" /> <!--Execute all Test Casses that reference the
2681 "Basic Exchange, No Payload" Functional Requirement-->
2682 <TestRequirementRef id="funreq_id_1.2" /> <!--Execute all Test Casses that reference the
2683 "Basic Exchange, One Payload" Functional Requirement-->
2684 </TestProfile>
```

2685

2686

2687

## 2688 9.3 Example Test Suites

2689

2690 Below are two XML documents illustrating how Test Cases are constructed, in this case for testing an  
2691 ebXML MS v2.0 implementation. Each Test Case has a required "requirementReferenceld" attribute,  
2692 pointing to a Functional Requirement in the Test Requirements document. A Test Driver executes all  
2693 Test Cases in this document that have a requirementReferenceld value matching the particular Semantic  
2694 Test Requirement being exercised.

2695

### 2696 9.3.1 Conformance Test Suite

2697

2698

2699 For brevity, only one Test Case is included in the Test Suite below. The complete ebXML MS v2.0  
2700 Conformance Test Suite is available at the OASIS IIC Technical Committee web site.

2701 A Test Driver executing conformance Test Cases operates in "connection" mode, meaning it is not  
2702 interfaced to any MSH, and is acting on its own. The Test Case exercises a Functional Requirement  
2703 listed in section 10.1 The Test Case below verifies that a ConversationId element is present in an  
2704 ebXML response message

2705

2706  
2707  
2708  
2709  
2710  
2711  
2712  
2713  
2714  
2715  
2716  
2717  
2718  
2719  
2720  
2721  
2722  
2723  
2724  
2725  
2726  
2727  
2728  
2729  
2730  
2731  
2732  
2733  
2734  
2735  
2736  
2737  
2738  
2739  
2740  
2741  
2742  
2743  
2744  
2745  
2746  
2747  
2748  
2749  
2750  
2751  
2752  
2753  
2754  
2755  
2756  
2757  
2758  
2759  
2760  
2761  
2762  
2763  
2764  
2765  
2766  
2767  
2768  
2769  
2770

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--
Copyright (C) The Organization for the Advancement of Structured Information Standards [OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and derivative works that
comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
and distributed, in whole or in part, without restriction of any kind, provided that the above copyright
notice and this paragraph are included on all such copies and derivative works. However, this document
itself may not be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the procedures for
copyrights defined in the OASIS Intellectual Property Rights document MUST be followed, or as required
to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
or assigns.
-->

<ebTest:TestSuite xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
ebTest:configurationGroupRef = "base" xmlns:ds = "http://www.oasis-open.org/tc/ebxml-
iic/tests/xmldsig" xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime" xmlns:soap =
"http://www.oasis-open.org/tc/ebxml-iic/tests/soap" xmlns:eb = "http://www.oasis-open.org/tc/ebxml-
iic/tests/eb" xmlns:xlink = "http://www.w3.org/1999/xlink" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation = "http://www.oasis-
open.org/tc/ebxml-iic/tests/schemas/ebTest.xsd">
  <ebTest:MetaData>
    <ebTest:Description> Test for presence of ConversationId in ebXML MessageHeader
element</ebTest:Description>
    <ebTest:Version>0.1</ebTest:Version>
    <ebTest:Maintainer>Michael Kass</ebTest:Maintainer>
    <ebTest:Location>ScriptingTestSuite.xml</ebTest:Location>
    <ebTest:PublishDate>05/20/2004</ebTest:PublishDate>
    <ebTest:Status>DRAFT</ebTest:Status>
  </ebTest:MetaData>
  <ebTest:ConfigurationGroup ebTest:id = "mshc_basic">
    <ebTest:Mode>connection</ebTest:Mode>
    <ebTest:StepDuration>300</ebTest:StepDuration>
    <ebTest:Transport>HTTP</ebTest:Transport>
    <ebTest:Envelope>ebXML</ebTest:Envelope>
    <ebTest:StoreAttachments>>false</ebTest:StoreAttachments>
    <ebTest:SetParameter>
      <ebTest:Name>SenderParty</ebTest:Name>
      <ebTest:Value>TestDriver</ebTest:Value>
    </ebTest:SetParameter>
    <ebTest:SetParameter>
      <ebTest:Name>ReceiverParty</ebTest:Name>
      <ebTest:Value>TestService</ebTest:Value>
    </ebTest:SetParameter>
    <ebTest:SetParameter>
      <ebTest:Name>Service</ebTest:Name>
      <ebTest:Value>urn:ebxml:iic:test</ebTest:Value>
    </ebTest:SetParameter>
    <ebTest:SetParameter>
      <ebTest:Name>Action</ebTest:Name>
      <ebTest:Value>Dummy</ebTest:Value>
    </ebTest:SetParameter>
  </ebTest:ConfigurationGroup>
  <ebTest:TestCase ebTest:id = "testcase_1" ebTest:description = "ConversationId is present in
message" ebTest:requirementReferenceId = "funreq_id_36">
    <ebTest:ThreadGroup>
      <ebTest:Thread ebTest:name = "thread_01">
        <ebTest:PutMessage ebTest:description = "Send a message to the
Dummy action">
```

```

2771         <ebTest:SetPart>
2772             <ebTest:Declaration>
2773                 <soap:Envelope>
2774                     <soap:Header>
2775
2776             <eb:MessageHeader>
2777
2778             <eb:Action>Dummy</eb:Action>
2779
2780             </eb:MessageHeader>
2781
2782                 </soap:Header>
2783                 </soap:Envelope>
2784             </ebTest:Declaration>
2785             <ebTest:Mutator>ebXMLEnvelope.xsl</ebTest:Mutator>
2786         </ebTest:SetPart>
2787     </ebTest:PutMessage>
2788     <ebTest:GetMessage ebTest:description = "Retrieve response message ">
2789
2790     <ebTest:Filter>/TEST:MessageStore/mime:Message[mime:Container[1]/soap:Envelope/soap:Header
2791 /eb:MessageHeader[eb:CPAId='mshc_Basic' and eb:MessageData/eb:RefToMessageId=$MessageId and
2792 eb:Action='Mute']]</ebTest:Filter>
2793         </ebTest:GetMessage>
2794     <ebTest:TestAssertion ebTest:description = "Verify that a ConversationId
2795 element is present in response">
2796
2797     <ebTest:VerifyContent>/FilterResult/Message/soap:Envelope/soap:Header/eb:MessageHeader/eb:Co
2798 nversationId</ebTest:VerifyContent>
2799         </ebTest:TestAssertion>
2800     </ebTest:Thread>
2801 </ebTest:ThreadGroup>
2802     <ebTest:Thread ebTest:name = "main">
2803         <ebTest:ThreadRef ebTest:nameRef = "thread_01"/>
2804     </ebTest:Thread>
2805 </ebTest:TestCase>
</ebTest:TestSuite>

```

### 2806 9.3.2 Interoperability Test Suite

2807

2808 In the example below, a series of four Test Cases make up an Interoperability Test Suite. A Test Driver  
2809 executing conformance Test Cases operates in "service" mode, meaning it is interfaced to a MSH. The  
2810 Test Case exercises a Functional Interoperability Requirement. The Test Case below performs a basic  
2811 message exchange with no message payload. The complete ebXML Basic Interoperability Profile Test  
2812 Suite is available online at the OASIS IIC Technical Commite web site.

```

2813 <?xml version = "1.0" encoding = "UTF-8"?>
2814
2815 <!--
2816 Copyright (C) The Organization for the Advancement of Structured Information Standards [OASIS]
2817 January 2002. All Rights Reserved.
2818 This document and translations of it may be copied and furnished to others, and derivative works that
2819 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
2820 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright
2821 notice and this paragraph are included on all such copies and derivative works. However, this document
2822 itself may not be modified in any way, such as by removing the copyright notice or references to OASIS,
2823 except as needed for the purpose of developing OASIS specifications, in which case the procedures for
2824 copyrights defined in the OASIS Intellectual Property Rights document MUST be followed, or as required
2825 to translate it into languages other than English.
2826 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors
2827 or assigns.
2828 -->
2829
2830

```

```

2831 <ebTest:TestSuite xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
2832 ebTest:configurationGroupRef = "base" xmlns:ds = "http://www.oasis-open.org/tc/ebxml-
2833 iic/tests/xmldsig" xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime" xmlns:soap =
2834 "http://www.oasis-open.org/tc/ebxml-iic/tests/soap" xmlns:eb = "http://www.oasis-open.org/tc/ebxml-
2835 iic/tests/eb" xmlns:xlink = "http://www.w3.org/1999/xlink" xmlns:xsi =
2836 "http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation = "http://www.oasis-
2837 open.org/tc/ebxml-iic/tests schemas\ebTest.xsd">
2838   <ebTest:MetaData>
2839     <ebTest:Description>ebXML MS Interoperability Test Suite </ebTest:Description>
2840     <ebTest:Version>0.1</ebTest:Version>
2841     <ebTest:Maintainer>Michael Kass</ebTest:Maintainer>
2842     <ebTest:Location>ScriptingTestSuite.xml</ebTest:Location>
2843     <ebTest:PublishDate>05/20/2004</ebTest:PublishDate>
2844     <ebTest:Status>DRAFT</ebTest:Status>
2845   </ebTest:MetaData>
2846   <ebTest:ConfigurationGroup ebTest:id = "mshc_basic">
2847     <ebTest:Mode>connection</ebTest:Mode>
2848     <ebTest:StepDuration>300</ebTest:StepDuration>
2849     <ebTest:Transport>HTTP</ebTest:Transport>
2850     <ebTest:Envelope>ebXML</ebTest:Envelope>
2851     <ebTest:StoreAttachments>>false</ebTest:StoreAttachments>
2852     <ebTest:SetParameter>
2853       <ebTest:Name>SenderParty</ebTest:Name>
2854       <ebTest:Value>TestDriver</ebTest:Value>
2855     </ebTest:SetParameter>
2856     <ebTest:SetParameter>
2857       <ebTest:Name>ReceiverParty</ebTest:Name>
2858       <ebTest:Value>TestService</ebTest:Value>
2859     </ebTest:SetParameter>
2860     <ebTest:SetParameter>
2861       <ebTest:Name>Service</ebTest:Name>
2862       <ebTest:Value>urn:ebxml:iic:test</ebTest:Value>
2863     </ebTest:SetParameter>
2864     <ebTest:SetParameter>
2865       <ebTest:Name>Action</ebTest:Name>
2866       <ebTest:Value>Dummy</ebTest:Value>
2867     </ebTest:SetParameter>
2868   </ebTest:ConfigurationGroup>
2869   <ebTest:TestCase ebTest:id = "testcase_1" ebTest:description = "Basic request/response test"
2870 ebTest:requirementReferenceId = " funreq_id_1.1">
2871     <ebTest:ThreadGroup>
2872       <ebTest:Thread ebTest:name = "thread_01">
2873         <ebTest:PutMessage ebTest:description = "Send a message to the
2874 Dummy action">
2875           <ebTest:SetPart>
2876             <ebTest:Declaration>
2877               <soap:Envelope>
2878                 <soap:Header>
2879
2880             <eb:MessageHeader>
2881
2882             <eb:Action>Dummy</eb:Action>
2883
2884             </eb:MessageHeader>
2885
2886               </soap:Header>
2887             </soap:Envelope>
2888           </ebTest:Declaration>
2889           <ebTest:Mutator>ebXMLEnvelope.xsl</ebTest:Mutator>
2890         </ebTest:SetPart>
2891       </ebTest:PutMessage>
2892     <ebTest:GetMessage ebTest:description = "Retrieve response message ">
2893
2894     <ebTest:Filter>/TEST:MessageStore/mime:Message[mime:Container[1]/soap:Envelope/soap:Header
2895 /eb:MessageHeader[eb:CPAId='mshc_Basic' and eb:MessageData/eb:RefToMessageId=$MessageId and

```



```
2896         </ebTest:GetMessage>
2897         <ebTest:TestAssertion ebTest:description = "Verify that an ebXML
2898 Message was received">
2899     <ebTest:VerifyContent>/FilterResult/Message/soap:Envelope/soap:Header/eb:MessageHeader
2900 </ebTest:VerifyContent>
2901         </ebTest:TestAssertion>
2902     </ebTest:Thread>
2903 </ebTest:ThreadGroup>
2904 <ebTest:Thread ebTest:name = "main">
2905     <ebTest:ThreadRef ebTest:nameRef = "thread_01"/>
2906 </ebTest:Thread>
2907 </ebTest:TestCase>
2908 </ebTest:TestSuite>
```

2911  
2912  
2913

### 2914 9.3.3 A sample Mutator XSL Document

2915

2916 The XML document below is an XSLT stylesheet that is used by an XSL processor to interpret anXML  
2917 message Declaration element and its ebXML content, and generate a valid ebXML message envelope.  
2918 This stylesheet can be used in any number of Test Cases, with variations in the resulting message based  
2919 upon variations in the Declaration content in the Test Case.

2920

2921 The stylesheet below was developed by the IIC for creating a valid ebXML Message by transforming a  
2922 message declaration conforming to the IIC ebXML MS v2.0 Message Declaration Schema defined in  
2923 Appendix G.

2924

2925 Testing communities wishing to perform conformance and/or interoperability testing of other messaging  
2926 services, or other XML-based business applications SHOULD define a message declaration schema, and  
2927 any stylesheets they need to construct their messages using the IIC Test Framework.

2928

2929 [To Be added](#)

2930

2931

2932

2933  
2934

## Appendix A (Normative) The ebXML Test Profile Schema

2935 The OASIS ebXML Implementation and Interoperability Committee has provided a version of the ebXML  
2936 Test Profile schema using the schema vocabulary that conforms to the W3C XML Schema  
2937 Recommendation specification [XMLSchema].

2938  
2939  
2940  
2941  
2942  
2943  
2944  
2945  
2946  
2947  
2948  
2949  
2950  
2951  
2952  
2953  
2954  
2955  
2956  
2957  
2958  
2959  
2960  
2961  
2962  
2963  
2964  
2965  
2966  
2967  
2968  
2969  
2970  
2971  
2972  
2973  
2974  
2975  
2976  
2977  
2978  
2979  
2980  
2981  
2982  
2983  
2984  
2985  
2986  
2987  
2988  
2989  
2990  
2991  
2992  
2993  
2994  
2995

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/test-profile"
  xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/test-profile"
  >
  <!--
Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
-->

  <!--
  $Id: TestProfile.xsd,v 1.2 2002/07/02 15:28:27 matt Exp $
  -->

  <element name = "TestProfile">
    <complexType>
      <sequence>
        <element ref = "tns:Dependency" minOccurs = "0" maxOccurs =
"unbounded"/>
        <element ref = "tns:TestRequirementRef" maxOccurs =
"unbounded"/>
      </sequence>
      <attribute name = "requirementsLocation" use = "required" type =
"anyURI"/>
      <attribute name = "name" use = "required" type = "string"/>
      <attribute name = "description" use = "required" type = "string"/>
    </complexType>
  </element>
  <element name = "Dependency">
    <complexType>
      <attribute name = "name" use = "required" type = "string"/>
      <attribute name = "profileRef" use = "required" type = "anyURI"/>
    </complexType>
  </element>
  <element name = "TestRequirementRef">
    <!--
To override the conformance type of the underlying requirement ...
-->
    <complexType>
      <sequence>
        <element name = "Comment" type = "string" minOccurs = "0"
maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "id" use = "required" type = "string"/>
      <attribute name = "requirementType" use = "optional" type =
"tns:requirement.type"/>
    </complexType>
  </element>
  <simpleType name = "requirement.type">
    <restriction base = "string">
      <enumeration value = "required"/>
      <enumeration value = "strongly recommended"/>
      <enumeration value = "recommended"/>
      <enumeration value = "optional"/>
    </restriction>
  </simpleType>
</schema>
```



2996  
2997



---

## Appendix B (Normative) The ebXML Test Requirements Schema

2998  
2999

3000 The OASIS ebXML Implementation and Interoperability Committee has provided a version of the ebXML  
3001 Test Requirements schema using the schema vocabulary that conforms to the W3C XML Schema  
3002 Recommendation specification [XMLSchema].

3003

3004

3005

3006

3007

3008

3009

3010

3011

3012

3013

3014

3015

3016

3017

3018

3019

3020

3021

3022

3023

3024

3025

3026

3027

3028

3029

3030

3031

3032

3033

3034

3035

3036

3037

3038

3039

3040

3041

3042

3043

3044

3045

3046

3047

3048

3049

3050

3051

3052

3053

3054

3055

3056

3057

3058

3059

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs"
  xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs"
  >
  <group name = "FunctionalRequirementGroup">
    <sequence>
      <element ref = "tns:FunctionalRequirement"/>
    </sequence>
  </group>

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or
its successors or assigns.
-->

  <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2000/10/XMLSchema--
  >

  <!-- OASIS/ebXML Test Suite Framework
    Description: Schema used to define ebXML Test Requirements instance document

    Author: Michael Kass
    Organization: NIST

    Author: Matthew MacKenzie
    Organization: XML Global

    Date: 03/31/2002
    Version 1.0
    -->

  <!-- CHANGES:
    Version 1.0 (Matt):
    - added attributes requirementType and name to Level.
    - added other to functional.type enumeration.
  -->

  <element name = "TestRequirement">
```

```

3060     <complexType>
3061         <sequence>
3062             <element ref = "tns:Clause" minOccurs = "0"/>
3063             <choice maxOccurs = "unbounded">
3064                 <element ref = "tns:Assertion"/>
3065                 <element ref = "tns:AssertionRef"/>
3066             </choice>
3067             <choice minOccurs = "0" maxOccurs = "unbounded">
3068                 <element ref = "tns:FunctionalRequirement"/>
3069                 <element ref = "tns:TestRequirement"/>
3070             </choice>
3071         </sequence>
3072         <attribute name = "id" use = "required" type = "ID"/>
3073         <attribute name = "name" use = "required" type = "string"/>
3074         <attribute name = "specRef" use = "required" type = "string"/>
3075         <attribute name = "functionalType" use = "required" type = "string"/>
3076         <attribute name = "dependencyRef" use = "optional" type = "anyURI"/>
3077     </complexType>
3078 </element>
3079 <element name = "FunctionalRequirement">
3080     <complexType>
3081         <sequence>
3082             <element ref = "tns:Clause" minOccurs = "0"/>
3083             <choice maxOccurs = "unbounded">
3084                 <element ref = "tns:Assertion"/>
3085                 <element ref = "tns:AssertionRef"/>
3086             </choice>
3087             <choice minOccurs = "0" maxOccurs = "unbounded">
3088                 <element ref = "tns:FunctionalRequirement"/>
3089                 <element ref = "tns:TestRequirement"/>
3090             </choice>
3091         </sequence>
3092         <attribute name = "id" use = "required" type = "ID"/>
3093         <attribute name = "name" use = "required" type = "string"/>
3094         <attribute name = "specRef" use = "required" type = "string"/>
3095         <attribute name = "testCaseRef" use = "optional" type = "anyURI"/>
3096         <attribute name = "dependencyRef" use = "optional" type = "anyURI"/>
3097     </complexType>
3098 </element>
3099 <element name = "Clause">
3100     <complexType>
3101         <sequence>
3102             <choice>
3103                 <element ref = "tns:Clause"/>
3104                 <choice>
3105                     <element ref = "tns:Condition"/>
3106                     <element ref = "tns:ConditionRef"/>
3107                 </choice>
3108             </choice>
3109             <sequence minOccurs = "0" maxOccurs = "unbounded">
3110                 <choice>
3111                     <element ref = "tns:And"/>
3112                     <element ref = "tns:Or"/>
3113                 </choice>
3114                 <choice>
3115                     <element ref = "tns:Clause"/>
3116                     <choice>
3117                         <element ref = "tns:Condition"/>
3118                         <element ref = "tns:ConditionRef"/>
3119                     </choice>
3120                 </choice>
3121             </sequence>
3122         </sequence>
3123     </complexType>
3124 </element>
3125 <element name = "Condition">
3126     <complexType>
3127         <simpleContent>
3128             <extension base = "string">
3129                 <attribute name = "id" use = "required" type = "ID"/>

```

```

3130         <attribute name = "requirementType" use = "optional"
3131 type = "tns:requirement.type"/>
3132         </extension>
3133     </simpleContent>
3134 </complexType>
3135 </element>
3136 <element name = "ConditionRef">
3137     <complexType>
3138         <attribute name = "id" use = "required" type = "IDREF"/>
3139     </complexType>
3140 </element>
3141 <element name = "And" type = "string"/>
3142 <element name = "Or" type = "string"/>
3143 <element name = "Assertion">
3144     <complexType>
3145         <simpleContent>
3146             <extension base = "string">
3147                 <attribute name = "requirementType" use = "required"
3148 type = "tns:requirement.type"/>
3149                 <attribute name = "id" use = "required" type = "ID"/>
3150             </extension>
3151         </simpleContent>
3152     </complexType>
3153 </element>
3154 <element name = "MetaData">
3155     <complexType>
3156         <sequence>
3157             <element ref = "tns:Description"/>
3158             <element ref = "tns:Version"/>
3159             <element ref = "tns:Maintainer"/>
3160             <element ref = "tns:Location"/>
3161             <element ref = "tns:PublishDate"/>
3162             <element ref = "tns:Status"/>
3163         </sequence>
3164     </complexType>
3165 </element>
3166 <element name = "Description" type = "string"/>
3167 <element name = "Version" type = "string"/>
3168 <element name = "SourceControlInfo" type = "string"/>
3169 <element name = "Maintainer" type = "string"/>
3170 <element name = "Location" type = "anyURI"/>
3171 <element name = "PublishDate" type = "string"/>
3172 <element name = "Status" type = "tns:pubStatus.type"/>
3173 <simpleType name = "pubStatus.type">
3174     <restriction base = "string">
3175         <enumeration value = "DRAFT"/>
3176         <enumeration value = "FINAL"/>
3177         <enumeration value = "RETIRED"/>
3178     </restriction>
3179 </simpleType>
3180 <simpleType name = "requirement.type">
3181     <restriction base = "string">
3182         <enumeration value = "required"/>
3183         <enumeration value = "strongly recommended"/>
3184         <enumeration value = "recommended"/>
3185         <enumeration value = "optional"/>
3186     </restriction>
3187 </simpleType>
3188 <simpleType name = "testLevel.type">
3189     <restriction base = "string">
3190         <enumeration value = "full"/>
3191         <enumeration value = "most"/>
3192         <enumeration value = "partial"/>
3193         <enumeration value = "none"/>
3194     </restriction>
3195 </simpleType>
3196 <simpleType name = "functional.type">
3197     <restriction base = "string">
3198         <enumeration value = "security"/>
3199         <enumeration value = "reliable messaging"/>
3200         <enumeration value = "packaging"/>

```

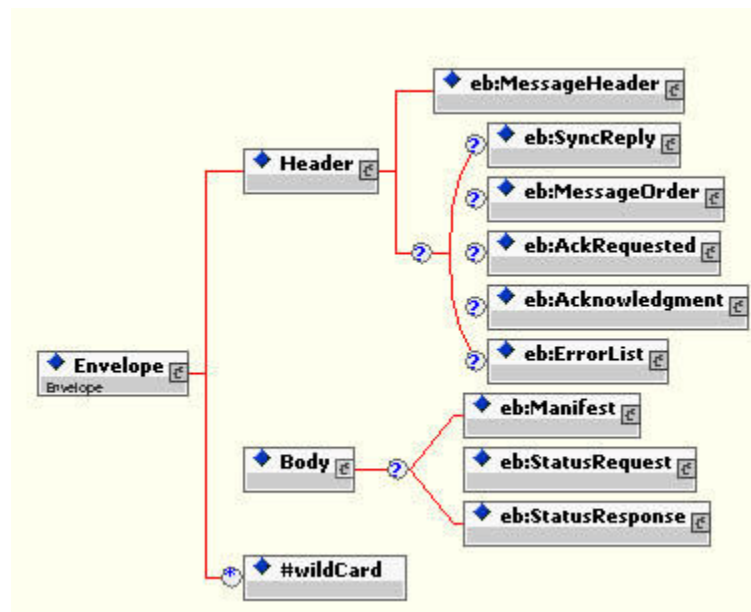
```
3201         <enumeration value = "other"/>
3202     </restriction>
3203 </simpleType>
3204 <simpleType name = "layerList">
3205     <list itemType = "string"/>
3206 </simpleType>
3207 <element name = "Requirements">
3208     <complexType>
3209         <sequence>
3210             <element ref = "tns:MetaData"/>
3211             <element ref = "tns:TestRequirement" maxOccurs = "unbounded"/>
3212         </sequence>
3213     </complexType>
3214 </element>
3215 <element name = "AssertionRef">
3216     <complexType>
3217         <attribute name = "id" use = "required" type = "IDREF"/>
3218     </complexType>
3219 </element>
3220 </schema>
```

3221

3222  
3223  
3224  
  
3225  
3226  
3227  
3228  
3229  
3230  
3231

## Appendix C (Normative) The ebXML Test Suite Message Declaration Schema and Supporting Subschemas

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



3232  
3233  
3234  
  
3235  
3236  
3237  
3238  
3239  
3240  
3241  
3242  
3243  
3244  
3245  
3246  
3247

Figure 24 – Image of Message Envelope Declaration

### 9.3.3.1.1 Schema for ebXML Declaration using SOAP

**MIME header data:** is not expressed in the message Envelope declaration, because it is transport specific. MIME (or other transport) header data MAY be expressed in the Test Case script using the “Header” element defined outside of the message declaration schema.

**SOAP header and body data:** SOAP message content MUST be created or modified using the Declaration content syntax described above and in the soap.xsd schema in Appendix EA Test Driver operating in “service” mode MAY ignore the SOAP portion of a Declaration, since message SOAP manipulation may be unavailable at the application level interface used for an MSH implementation. Test drivers in “connection” mode MUST properly interpret the SOAP portion of a Declaration and generate the appropriate SOAP header/body content.

3248 **ebXML MS 2.0 Message data:** ebXML message content MUST be created or modified using the  
3249 Declaration content syntax illustrated above and described in the eb.xsd schema described in Appendix  
3250 X. . Test drivers operating in both “connection” and “service” modes MUST properly interpret the ebXML  
3251 portion of a Declaration, and generate the appropriate ebXML content or declaration (respectively).

3252  
3253 **Other Types of Message Envelopes and Payloads:** RNIF, BizTalk or other XML Message Envelopes  
3254 and payloads can be constructing using any implementation-specific XML message declaration syntax in  
3255 combination with an XSL stylesheet or XUpdate declaration. It is HIGHLY RECOMMENDED that the  
3256 schemas used to define the Declaration and the Message Store structure be published as a “best  
3257 practice” in order to provide conformity and reusability of conformance and interoperability test suites  
3258 across this Test Framework.

3259  
3260  
3261 Below is a sample ebXML Declaration. The Test Driver mutates the Declaration (using an XSL  
3262 stylesheet), inserting element and attribute content wherever it knows default content should be, and  
3263 declaring, or overriding default values where they are explicitly defined in the Declaration.

3264  
3265

```
3266 <ebTest:Declaration>  
3267   <soap:Envelope>  
3268     <soap:Header>  
3269       <eb:MessageHeader/>  
3270     </soap:Header>  
3271     <soap:Body />  
3272   </soap:Envelope>  
3273 </ebTest:Declaration>
```

3274  
3275

3276 For illustrative purposes, the resulting message can be represented by the example message below. The  
3277 Test Driver, after parsing the simple Declaration above, and mutating it through an XSL stylesheet, would  
3278 generate the following MIME message with enclosed SOAP/ebXML content.

3279  
3280

```
3281 Content-Type: multipart/related; type="text/xml"; boundary="boundaryText";  
3282 start=messagepackage@oasis.org  
3283  
3284 --boundaryText  
3285  
3286 Content-ID: <messagepackage@oasis.org>  
3287 Content-Type: text/xml; charset="UTF-8"  
3288  
3289 <soap:Envelope xmlns:xlink="http://www.w3.org/1999/xlink"  
3290   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
3291   xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"  
3292   xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-  
3293 2_0.xsd" xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/  
3294   http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
```

3296  
3297  
3298  
3299  
3300  
3301  
3302  
3303  
3304  
3305  
3306  
3307  
3308  
3309  
3310  
3311  
3312  
3313  
3314  
3315  
3316  
3317  
3318

```
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd">
<soap:Header>
  <eb:MessageHeader soap:mustUnderstand="1" eb:version="2.0">
    <eb:From>
      <eb:PartyId>urn:oasis:iic:testdriver</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>urn:oasis:iic:testservice</eb:PartyId>
    </eb:To>
    <eb:CPAId> mshc_basic</eb:CPAId>
    <eb:ConversationId> 987654321</eb:ConversationId>
    <eb:Service>urn:ebXML:iic:test</eb:Service>
    <eb:Action>Dummy</eb:Action>
    <eb:MessageData>
      <eb:MessageId>0123456789</eb:MessageId>
      <eb:Timestamp>2000-07-25T12:19:05</eb:Timestamp> MessageData>
    </eb:MessageHeader>
  </soap:Header>
</soap:Envelope>
```

3319

3320 Dynamic ebXML message content values (highlighted above) are supplied by the Test Driver.

3321

3322 The ebXMLMessage.xsd schema in Appendix X defines the format for element and attribute content  
3323 declaration for ebXML MS testing. However, the schema alone DOES NOT define default XML element  
3324 content, since this is beyond the capability of schemas. Therefore, Test Driver implementers MUST  
3325 consult the "Definition of Content" tables for this section of the specification to determine what default  
3326 XML content must be generated by the Test Driver or MSH to create a valid ebXML message.

3327

3328 The following sections describe how a Test Driver or MSH MUST interpret the Declaration content in  
3329 order to be conformant to this specification for ebXML MS testing.

3330

3331

3333

3334 The following XML represents all the information necessary to permit a Test Driver to construct a MIME  
3335 message that may contain a SOAP envelope in its first MIME container. The XML document below  
3336 validates against the mime.xsd schema in Appendix C.

3337

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

3341

3342

### 3343 9.3.3.1.2 Interpreting the SOAP portion of the ebXML Declaration

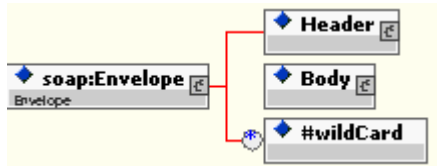
3344

3345 The XML syntax interpreted by the Test Driver to construct the SOAP message content consists of the  
3346 declaration of a SOAP Envelope element, which in turn is a container for the SOAP Header, Body and  
3347 non-SOAP XML content. Construction of the SOAP Header and Body content is simple for the Test  
3348 Driver, requiring only the creation of the two container elements with their namespace properly declared,



3349 and valid according to the [SOAP]. The Test Driver only constructs the SOAP Body element if it is  
 3350 explicitly declared in the content.

3351  
 3352



3353  
 3354 Figure 26 – Graphic representation of expanded view of the soap:Envelope element declaration

3355  
 3356  
 3357  
 3358

3359 Definition of Content

3360

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

3361 Table 15 defines the SOAP message content of the Declaration element in a message declaration

3362  
 3363  
 3364

3365 An Example of Minimal SOAP Declaration Content

3366

3367 The following XML represents all the information necessary to permit a Test Driver to construct a minimal  
 3368 SOAP message. It validates against the soap.xsd schema in appendix X.

3369

```

3370 <soap:Envelope>
3371   <soap:Header/>
3372 </soap:Envelope>
  
```

3373

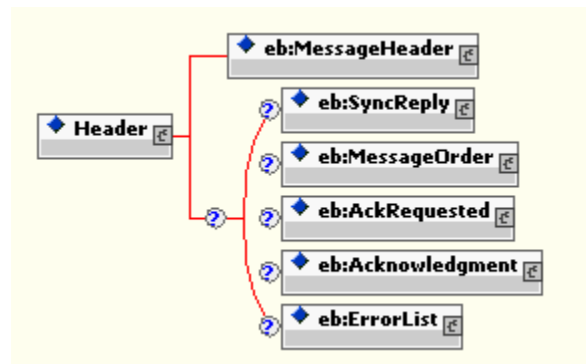
3374 9.3.3.1.3 Interpreting the SOAP Header Extension Element Declaration

3375

3376 The declarative syntax interpreted by the Test Driver to construct the ebXML Header extension message  
 3377 content consists of the declaration of a SOAP Header element, which in turn is a container for the ebXML  
 3378 Header extension elements and their content. The only extension element that is required in the container  
 3379 is the eb:MessageHeader element, which directs the Test Driver to construct an ebXML MessageHeader  
 3380 element, along with its proper namespace declaration, as defined in [EBMS]. The Test Driver does not  
 3381 construct any other Header extension elements unless they are explicitly declared as content in the  
 3382 SOAP Header Declaration.

3383

3384



3385

3386 Figure 27 – Graphic representation of expanded view of the soap:Header element declaration

3387

3388

3389

3390

3391 Definition of Content

3392

3393

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

3394 Table 16 defines the MIME message content of the SOAP Header element in a message declaration

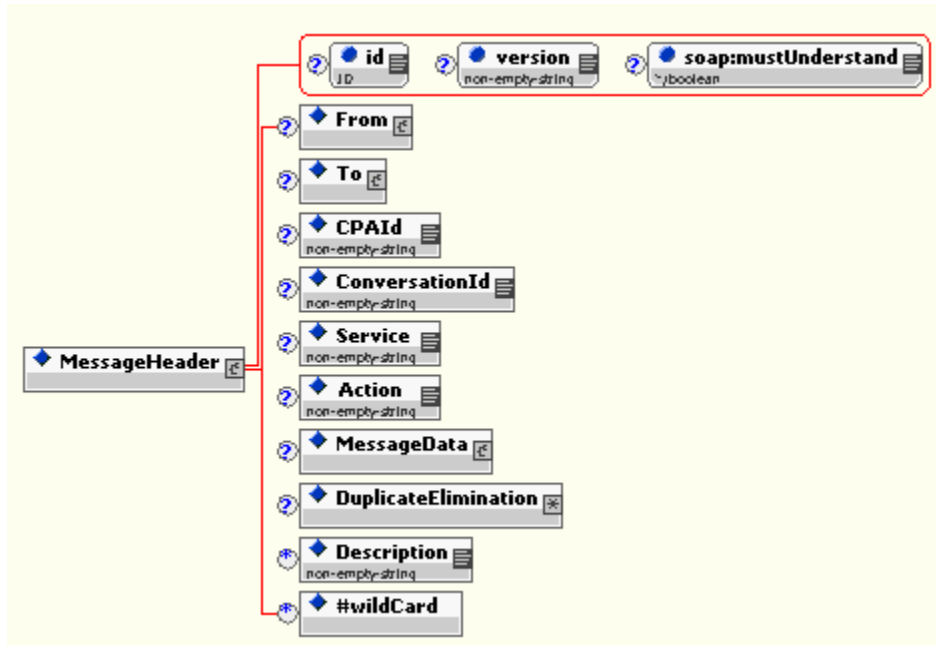
3395  
3396

3397 9.3.3.1.4 Interpreting the ebXML MessageHeader Element Declaration

3398

3399 The XML syntax interpreted by the Test Driver to construct the ebXML MessageHeader extension content  
 3400 consists of the declaration of a MessageHeader element, and a required declaration of CPAId and Action  
 3401 elements within it. This is the "minimum" declaration a Test Driver needs to generate an ebXML Message  
 3402 Header. All other required content, as defined in the schema in the ebXML MS v2.0 Specification, is  
 3403 provided by the Test Driver through either default parameters defined in the ebTest.xsd schema in  
 3404 Appendix C, or directly generated by the Test Driver (e.g. to generate necessary message container  
 3405 elements) or by explicit declaration of content in the Declaration. The figure below illustrates the schema  
 3406 for an ebXML Message Header declaration to be interpreted by the Test Driver.

3407  
3408



3409

3410 Figure 28 – Graphic representation of expanded view of the ebXML MessageHeader element declaration

3411

3412 Definition of Content

3413

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
eb:MessageHeader	Generate MessageHeader element and all of its default element/attribute content		Required	
id	Generate attribute with		Optional	

	declared value			
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 specified Action name	Generated by Test 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	

3414 Table 17 defines the content of the ebXML MessageHeader element in a message declaration

3415

3416

3417 An Example of a Minimal ebXML MessageHeader Content Declaration

3418

3419 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML  
3420 MessageHeader element with all necessary content to validate against the ebXML MS V2.0 schema. All  
3421 declared content must validate the ebTest.xsd schema in Appendix C.

3422

3423 `<eb:MessageHeader/>`

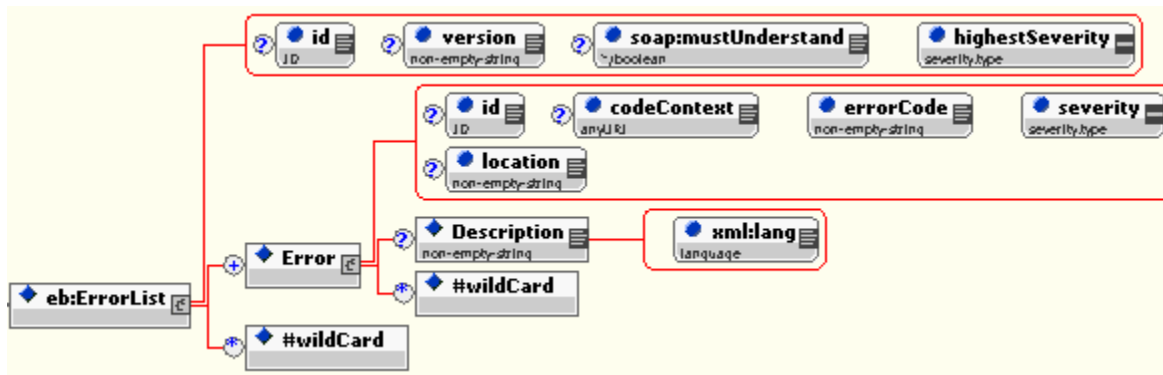
3424 9.3.3.1.5 Interpreting the ebXML ErrorList Element Declaration

3425

3426 The XML syntax interpreted by the Test Driver to construct the ebXML ErrorList extension content  
3427 consists of the declaration of an ErrorList element, and a required declaration of one or more Error  
3428 elements within it. All required content, as defined in the schema in the ebXML MS V2.0 Specification, is  
3429 provided through either default parameters defined in the ebTest.xsd schema and included by the Test  
3430 Driver, or by explicit declaration.

3431

3432



3433  
 3434 Figure 29 - Graphic representation of expanded view of the ebXML ErrorList element declaration

3435  
 3436  
 3437 Definition of Content

3438

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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	

3439 Table 18 defines the content of the ErrorList element in a message declaration

3440  
 3441  
 3442  
 3443  
 3444  
 3445  
 3446  
 3447  
 3448  
 3449  
 3450  
 3451  
 3452  
 3453  
 3454  
 3455  
 3456  
 3457  
 3458  
 3459  
 3460  
 3461  
 3462  
 3463  
 3464  
 3465  
 3466  
 3467  
 3468  
 3469  
 3470

An Example of a Minimal ebXML ErrorList Content Declaration

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

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

9.3.3.1.6 Interpreting the ebXML SyncReply Element Declaration

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

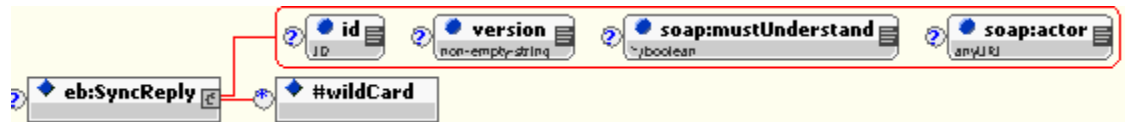


Figure 30 – Graphic representation of expanded view of the ebXML SyncReply element declaration

Definition of Content

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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	true	Optional	

	value			
soap:actor	Modify default attribute value	http://schemas.xmlsoap.org/soap/actor/next	Optional	
#wildCard	Generate content "inline"		Optional	

3471 Table 19 defines the content of the SyncReply element in a message declaration

3472

3473

3474

3475

3476 An Example of a Minimal ebXML SyncReply Content Declaration

3477

3478 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML  
3479 AckRequested element with all necessary content to validate against the [EBMS] schema schema.

3480

3481 `<eb:SyncReply/>`

### 3482 9.3.3.1.7 Interpreting the ebXML AckRequested Element Declaration

3483

3484 The XML syntax interpreted by the Test Driver to construct the ebXML AckRequested extension content  
3485 consists of the declaration of an AckRequested element. All required content as defined in the [EBMS]  
3486 schema, is provided by the Test Driver or by explicit declaration.

3487

3488



3489

3490 Figure 31 – Graphic representation of expanded view of the ebXML AckRequested element declaration

3491

3492

3493 Definition of Content

3494

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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	

3495 Table 20 defines the content of the AckRequested element in a message declaration

3496

3497

3498 An Example of a Minimal ebXML AckRequested Content Declaration

3499

3500 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML  
 3501 AckRequested element with all necessary content to validate against the [EBMS] schema.

3502

3503

```
<eb:AckRequested/>
```

3504

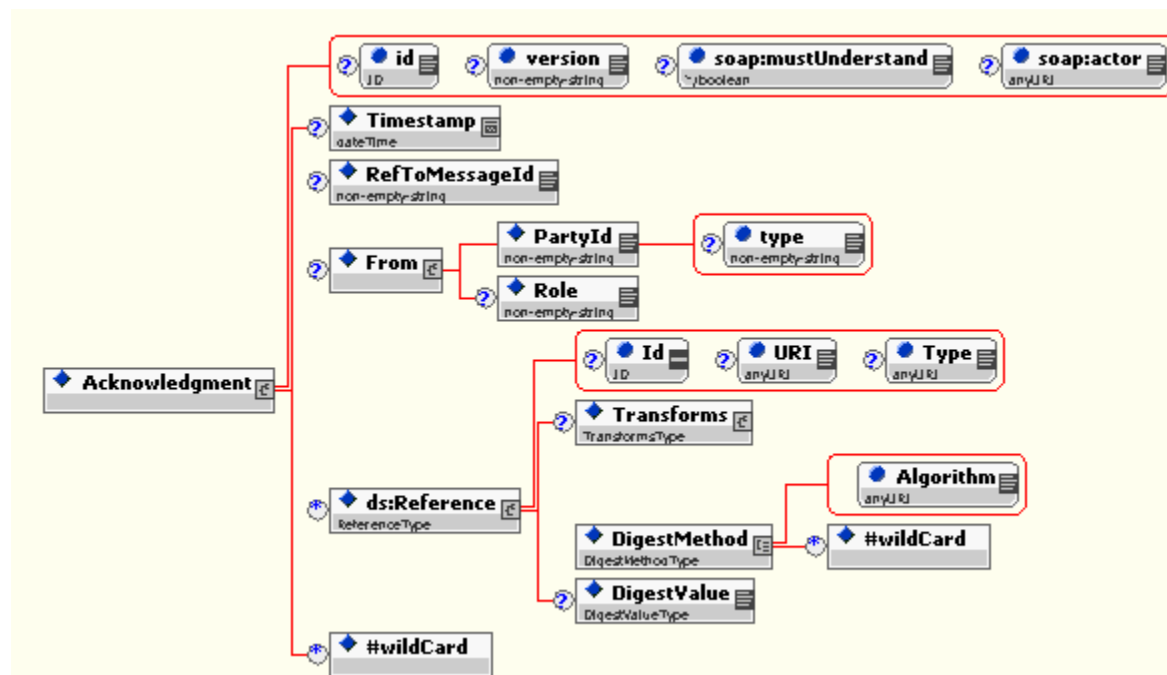
### 3505 9.3.3.1.8 Interpreting the ebXML Acknowledgment Element Declaration

3506

3507 The XML syntax interpreted by the Test Driver to construct the ebXML Acknowledgment extension  
 3508 content consists of the declaration of an Acknowledgment element. All required content, as defined in the  
 3509 [EBMS] schema, is provided by the Test Driver or through explicit declaration.

3510

3511



3512

3513 Figure 32 – Graphic representation of expanded view of the ebXML Acknowledgment element declaration  
 3514  
 3515  
 3516 Definition of Content  
 3517

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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 runtime	Optional	
RefToMessageId	Modify default element value	Generated by Test Driver at runtime	Optional	
From	Modify default container	Generated by Test Driver at runtime	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	

3518 Table 21 defines the content of the Acknowledgment element in a message declaration

3519

3520

3521 An Example of a Minimal "unsigned" ebXML Acknowledgment Content Declaration

3522

3523 The following XML represents the minimum information necessary to permit a Test Driver to construct an  
3524 ebXML Acknowledgment element.

3525

3526 `<eb:Acknowledgment/>`

3527

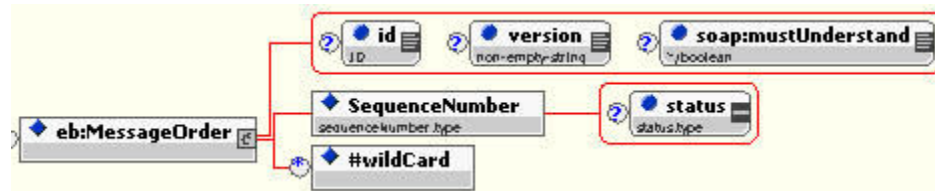
3528

3529 9.3.3.1.9 Interpreting the ebXML MessageOrder Element Declaration

3530

3531 The XML syntax interpreted by the Test Driver to construct the ebXML MessageOrder extension content  
3532 consists of the declaration of a MessageOrder element. All required content, as defined in the [EBMS]  
3533 schema, is provided by the Test Driver or through explicit declaration.

3534



3535  
 3536 Figure 33 – Graphic representation of expanded view of the ebXML MessageOrder element declaration  
 3537  
 3538  
 3539 Definition of Content  
 3540

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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	

3541 Table 22 defines the content of the MessageOrder element in a message declaration

3542  
 3543 An Example of a Minimal ebXML MessageOrder Content Declaration  
 3544

3545 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML  
 3546 MessageOrder element.

3547

3548  
 3549  
 3550

```
<eb:MessageOrder>
  <eb:SequenceNumber>1</eb:SequenceNumber>
</eb:MessageOrder>
```

3551

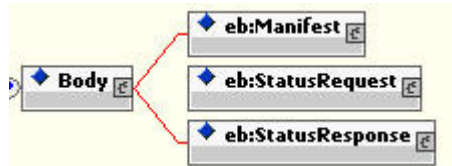
3552 9.3.3.1.10 Interpreting the SOAP Body Extension Element Declaration

3553

3554 The XML syntax used by the Test Driver to construct the ebXML Body extension message content  
 3555 consists of the declaration of a SOAP Body element, which in turn is a container for the ebXML Manifest,  
 3556 StatusRequest or StatusResponse elements.

3557 The Test Driver does not construct any of these SOAP Body extension elements unless they are explicitly  
 3558 declared as content in the SOAP Body Declaration.

3559  
 3560

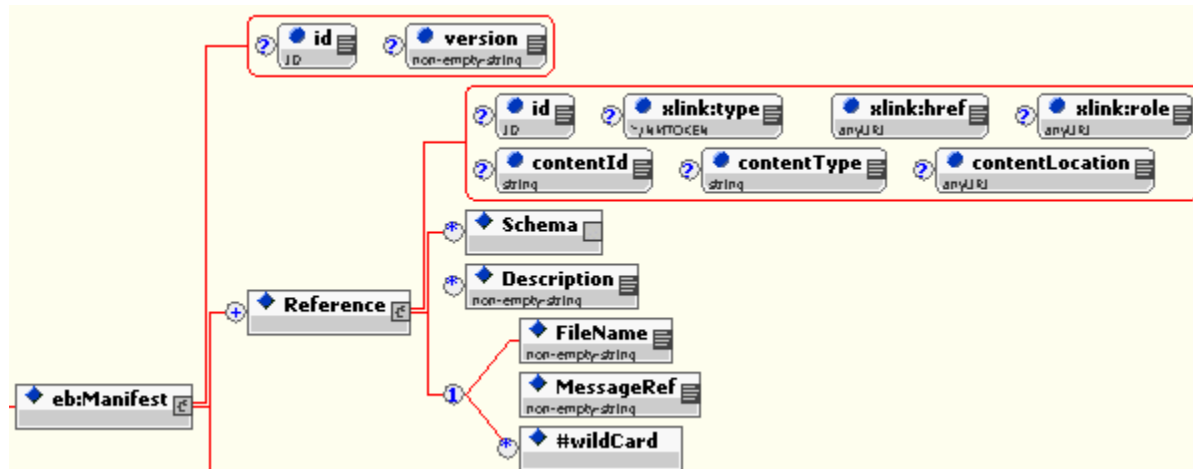


3561  
 3562 Figure 34 – Graphic representation of expanded view of the soap:Body element declaration  
 3563

3564 9.3.3.1.11 Interpreting the ebXML Manifest Element Declaration

3565  
 3566 The XML syntax interpreted by the Test Driver to construct the ebXML Manifest extension content  
 3567 consists of the declaration of a Manifest element. All required content, as defined in the [EBMS] schema,  
 3568 is provided by the Test Driver or through explicit declaration

3569  
 3570



3571  
 3572 Figure 35 – Graphic representation of expanded view of the ebXML Manifest element declaration  
 3573

3574  
 3575 Definition of Content  
 3576

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

	value			
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 the Content-Type MIME header of the payload		Optional	
contentLocation	Set the 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	File not found
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	

3577 Table 23 defines the content of the Manifest element in a message declaration

3578

3579 An Example of a Minimal ebXML Manifest Content Declaration

3580

3581 The following XML represents the minimum information necessary to permit a Test Driver to construct an  
3582 ebXML Manifest element with all necessary content to validate against the ebXML MS v2.0 schema.

3583

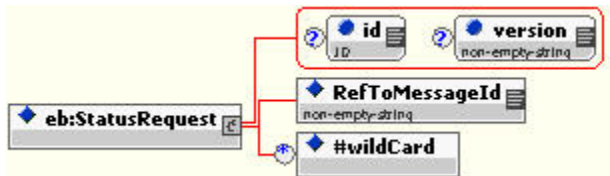
3584  
3585  
3586

```
<eb:Manifest>
<eb:Reference xlink:href="cid:payload 1"/>
</eb:Manifest>
```

3587 9.3.3.1.12 Interpreting the ebXML StatusRequest Element Declaration

3588

3589 The XML syntax interpreted by the Test Driver to construct the ebXML StatusRequest extension content  
3590 consists of the declaration of a StatusRequest element. All required content, as defined in the [EBMX]  
3591 schema. All required content, as defined in the [EBMS] schema, is provided by the Test Driver or through  
3592 explicit declaration  
3593



3594

3595 Figure 36 – Graphic representation of expanded view of the ebXML StatusRequest element declaration

3596

3597

3598 Definition of Content

3599

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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	

3600 Table 24 defines the content of the StatusRequest element in a message declaration

3601

3602 An Example of a Minimal ebXML StatusRequest Content Declaration

3603

3604 The following XML represents all the minimum information necessary to permit a Test Driver to construct  
3605 an ebXML StatusRequest element with all necessary content to validate against the [EBMS] schema.

3606

3607

3608

3609

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

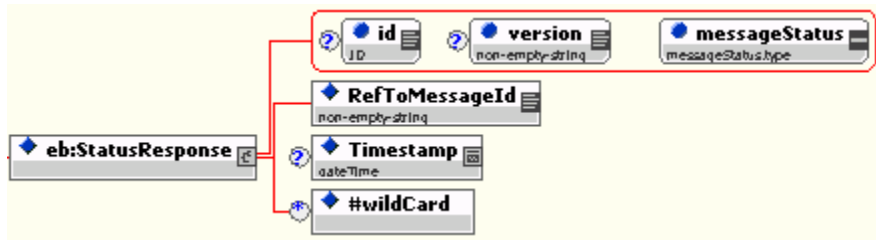
3610

3611

3612 9.3.3.1.13 Interpreting the ebXML StatusResponse Element Declaration

3613

3614 The XML syntax used by the Test Driver to construct the ebXML StatusResponse extension content  
 3615 consists of the declaration of a StatusResponse element with required and optional element/attribute  
 3616 content.  
 3617



3618

3619 Figure 37 – Graphic representation of expanded view of the ebXML StatusResponse element declaration

3620

3621

3622 Definition of Content

3623

Name	Declaration Description	Default Value From Test Driver	Required/Optional	Exception Condition
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 value		Optional	
RefToMessageId	Generate element and its value		Required	
Timestamp	Modify default value	Generated by Test Driver at run time	Optional	
#wildCard	Generate content "inline"		Optional	

3624 Table 25 defines the content of the StatusResponse element in a message declaration

3625

3626 An Example of a Minimal ebXML StatusResponse Content Declaration

3627

3628 The following XML represents all the information necessary to permit a Test Driver to construct an ebXML  
 3629 StatusResponse element with all necessary content to validate against the [EBMX] schema.

3630

3631 <eb:StatusResponse messageStatus="Processed"/>



3632  
3633  
3634  
3635  
3636  
3637  
3638

## SOAP Portion of the ebXML Declaration Schema

3639  
3640  
3641  
3642  
3643  
3644  
3645  
3646  
3647  
3648  
3649  
3650  
3651  
3652  
3653  
3654  
3655  
3656  
3657  
3658  
3659  
3660  
3661  
3662  
3663  
3664  
3665  
3666  
3667  
3668  
3669  
3670  
3671  
3672  
3673  
3674  
3675  
3676  
3677  
3678  
3679  
3680  
3681  
3682  
3683  
3684  
3685  
3686  
3687  
3688  
3689  
3690  
3691  
3692  
3693  
3694  
3695  
3696

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
  xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
  xmlns:xs = "http://www.w3.org/2001/XMLSchema"
  xmlns:eb = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
  xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb" schemaLocation
= "eb.xsd"/>
  <import namespace = "http://www.w3.org/2000/09/xmldsig#" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xmldsig-core-schema.xsd"/>
  <group name = "optionElements">
    <all minOccurs = "0">
      <element ref = "eb:SyncReply" minOccurs = "0"/>
      <element ref = "eb:MessageOrder" minOccurs = "0"/>
      <element ref = "eb:AckRequested" minOccurs = "0"/>
      <element ref = "eb:Acknowledgment" minOccurs = "0"/>
      <element ref = "eb:ErrorList" minOccurs = "0"/>
      <element ref = "ds:Signature" minOccurs = "0"/>
    </all>
  </group>
  <attributeGroup name = "encodingStyle">
    <attribute name = "encodingStyle" type = "tns:encodingStyle"/>
  </attributeGroup>

  <!-- Schema for the SOAP/1.1 envelope

  This schema has been produced using W3C's SOAP Version 1.2 schema
  found at:

  http://www.w3.org/2001/06/soap-envelope

  Copyright 2001 Martin Gudgin, Developmentor.

  Changes made are the following:
  - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
  - reverted mustUnderstand to only allow 0 and 1 as lexical values

  Copyright 2003 OASIS

  Changes made are the following:
  - SOAP Header and Body element content models constrained to include ebXML content

  Original copyright:

  Copyright 2001 W3C (Massachusetts Institute of Technology,
  Institut National de Recherche en Informatique et en Automatique,
  Keio University). All Rights Reserved.
  http://www.w3.org/Consortium/Legal/

  This document is governed by the W3C Software License [1] as
  described in the FAQ [2].

  [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
  [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
```

```

3697 -->
3698
3699
3700 <!-- Envelope, header and body -->
3701
3702 <element name = "Envelope" type = "tns:Envelope"/>
3703 <complexType name = "Envelope">
3704 <sequence>
3705 <element ref = "tns:Header"/>
3706 <element ref = "tns:Body"/>
3707 <any namespace = "##other" processContents = "lax" minOccurs = "0"
3708 maxOccurs = "unbounded"/>
3709 </sequence>
3710 <anyAttribute namespace = "##other" processContents = "lax"/>
3711 </complexType>
3712 <element name = "Header">
3713 <complexType>
3714 <sequence>
3715 <element ref = "eb:MessageHeader"/>
3716 <group ref = "tns:optionElements"/>
3717 </sequence>
3718 </complexType>
3719 </element>
3720 <complexType name = "Header">
3721 <sequence>
3722 <any namespace = "##other" processContents = "lax" minOccurs = "0"
3723 maxOccurs = "unbounded"/>
3724 </sequence>
3725 <anyAttribute namespace = "##other" processContents = "lax"/>
3726 </complexType>
3727 <element name = "Body">
3728 <complexType>
3729 <choice minOccurs = "0">
3730 <element ref = "eb:Manifest"/>
3731 <element ref = "eb:StatusRequest"/>
3732 <element ref = "eb:StatusResponse"/>
3733 </choice>
3734 </complexType>
3735 </element>
3736 <complexType name = "Body">
3737 <annotation>
3738 <documentation>
3739 Prose in the spec does not specify that attributes are allowed on the Body
3740 element
3741 </documentation>
3742 </annotation>
3743 <sequence>
3744 <any namespace = "##any" processContents = "lax" minOccurs = "0"
3745 maxOccurs = "unbounded"/>
3746 </sequence>
3747 <anyAttribute namespace = "##any" processContents = "lax"/>
3748 </complexType>
3749
3750 <!-- Global Attributes. The following attributes are intended to be usable via
3751 qualified attribute names on any complex type referencing them. -->
3752
3753 <attribute name = "mustUnderstand" default = "0">
3754 <simpleType>
3755 <restriction base = "boolean">
3756 <pattern value = "0|1"/>
3757 </restriction>
3758 </simpleType>
3759 </attribute>
3760 <attribute name = "actor" type = "anyURI"/>
3761 <simpleType name = "encodingStyle">
3762 <annotation>
3763 <documentation>
3764 'encodingStyle' indicates any canonicalization conventions followed in the
3765 contents of the containing element. For example, the value
3766 'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern described in SOAP
3767 specification

```

```

3768     </documentation>
3769         </annotation>
3770         <list itemType = "anyURI"/>
3771     </simpleType>
3772     <complexType name = "Fault"
3773         final = "extension">
3774         <annotation>
3775             <documentation>
3776                 Fault reporting structure
3777             </documentation>
3778         </annotation>
3779         <sequence>
3780             <element name = "faultcode" type = "QName"/>
3781             <element name = "faultstring" type = "string"/>
3782             <element name = "faultactor" type = "anyURI" minOccurs = "0"/>
3783             <element name = "detail" type = "tns:detail" minOccurs = "0"/>
3784         </sequence>
3785     </complexType>
3786     <complexType name = "detail">
3787         <sequence>
3788             <any namespace = "##any" processContents = "lax" minOccurs = "0"
3789             maxOccurs = "unbounded"/>
3790         </sequence>
3791         <anyAttribute namespace = "##any" processContents = "lax"/>
3792     </complexType>
3793 </schema>

```

3794

3795 ebMS portion of the ebXML Declaration Schema

3796

```

3797 <?xml version = "1.0" encoding = "UTF-8"?>
3798 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
3799 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
3800     targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
3801     xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
3802     xmlns:xlink = "http://www.w3.org/1999/xlink"
3803     xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3804     xmlns:soap = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3805
3806     version = "1.0"
3807     elementFormDefault = "qualified"
3808     attributeFormDefault = "qualified">
3809     <import namespace = "http://www.w3.org/1999/xlink" schemaLocation =
3810     "http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd"/>
3811     <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
3812     schemaLocation = "xmldsig.xsd"/>
3813     <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
3814     schemaLocation = "soap.xsd"/>
3815     <import namespace = "http://www.w3.org/XML/1998/namespace" schemaLocation =
3816     "http://www.oasis-open.org/committees/ebxml-msg/schema/xml_lang.xsd"/>
3817     <attributeGroup name = "headerExtension.grp">
3818         <attribute ref = "tns:id"/>
3819         <attribute ref = "tns:version" use = "optional"/>
3820         <attribute ref = "soap:mustUnderstand" use = "optional"/>
3821     </attributeGroup>
3822     <attributeGroup name = "bodyExtension.grp">
3823         <attribute ref = "tns:id"/>
3824         <attribute ref = "tns:version" use = "optional"/>
3825     </attributeGroup>
3826
3827     <!--
3828     Copyright (C) The Organization for the Advancement of Structured Information Standards
3829     [OASIS]
3830     January 2002. All Rights Reserved.

```

```

3831 This document and translations of it may be copied and furnished to others, and
3832 derivative works that comment on or otherwise explain it or assist in its implementation
3833 may be prepared, copied, published and distributed, in whole or in part, without
3834 restriction of any kind, provided that the above copyright notice and this paragraph are
3835 included on all such copies and derivative works. However, this document itself may not
3836 be modified in any way, such as by removing the copyright notice or references to OASIS,
3837 except as needed for the purpose of developing OASIS specifications, in which case the
3838 procedures for copyrights defined in the OASIS Intellectual Property Rights document
3839 MUST be followed, or as required to translate it into languages other than English.
3840 The limited permissions granted above are perpetual and will not be revoked by OASIS or
3841 its successors or assigns.
3842 -->
3843
3844
3845 <!-- MANIFEST, for use in soap:Body element -->
3846
3847 <element name = "Manifest">
3848   <complexType>
3849     <sequence>
3850       <element ref = "tns:Reference" maxOccurs = "unbounded"/>
3851       <any namespace = "##other" processContents = "lax" minOccurs =
3852 "0" maxOccurs = "unbounded"/>
3853     </sequence>
3854     <attributeGroup ref = "tns:bodyExtension.grp"/>
3855   </complexType>
3856 </element>
3857 <element name = "Reference">
3858   <complexType>
3859     <sequence>
3860       <element ref = "tns:Schema" minOccurs = "0" maxOccurs =
3861 "unbounded"/>
3862       <element ref = "tns:Description" minOccurs = "0" maxOccurs =
3863 "unbounded"/>
3864       <choice>
3865         <element ref = "tns:FileName"/>
3866         <element ref = "tns:MessageRef"/>
3867         <any namespace = "##other" processContents = "lax"
3868 minOccurs = "0" maxOccurs = "unbounded"/>
3869       </choice>
3870     </sequence>
3871     <attribute ref = "tns:id"/>
3872     <attribute ref = "xlink:type" fixed = "simple"/>
3873     <attribute ref = "xlink:href" use = "required"/>
3874     <attribute ref = "xlink:role"/>
3875     <attribute name = "contentId" use = "optional" type = "string"/>
3876     <attribute name = "contentType" use = "optional" type = "string"/>
3877     <attribute name = "contentLocation" use = "optional" type = "anyURI"/>
3878   </complexType>
3879 </element>
3880 <element name = "Schema">
3881   <complexType>
3882     <attribute name = "location" use = "required" type = "anyURI"/>
3883     <attribute name = "version" type = "tns:non-empty-string"/>
3884   </complexType>
3885 </element>
3886
3887 <!-- MESSAGEHEADER, for use in soap:Header element -->
3888
3889 <element name = "MessageHeader">
3890   <complexType>
3891     <sequence>
3892       <element ref = "tns:From" minOccurs = "0"/>
3893       <element ref = "tns:To" minOccurs = "0"/>
3894       <element ref = "tns:CPAId" minOccurs = "0"/>
3895       <element ref = "tns:ConversationId" minOccurs = "0"/>
3896       <element ref = "tns:Service" minOccurs = "0"/>
3897       <element ref = "tns:Action" minOccurs = "0"/>
3898       <element ref = "tns:MessageData" minOccurs = "0"/>
3899       <element ref = "tns:DuplicateElimination" minOccurs = "0"/>
3900       <element ref = "tns:Description" minOccurs = "0" maxOccurs =
3901 "unbounded"/>

```

```

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

```

```

3973         <attribute name = "signed" use = "optional" type = "boolean"/>
3974     </complexType>
3975 </element>
3976
3977 <!-- ACKNOWLEDGMENT, for use in soap:Header element -->
3978
3979 <element name = "Acknowledgment">
3980     <complexType>
3981         <sequence>
3982             <element ref = "tns:Timestamp" minOccurs = "0"/>
3983             <element ref = "tns:RefToMessageId" minOccurs = "0"/>
3984             <element ref = "tns:From" minOccurs = "0"/>
3985             <element name = "Reference" minOccurs = "0" maxOccurs =
3986 "unbounded"/>
3987             <any namespace = "##other" processContents = "lax" minOccurs =
3988 "0"/>
3989             <element ref = "ds:Reference" minOccurs = "0" maxOccurs =
3990 "unbounded"/>
3991         </sequence>
3992         <attributeGroup ref = "tns:headerExtension.grp"/>
3993         <attribute ref = "soap:actor" default = "urn:oasis:names:tc:ebxml-
3994 msg:actor:toPartyMSH"/>
3995     </complexType>
3996 </element>
3997
3998 <!-- ERROR LIST, for use in soap:Header element -->
3999
4000 <element name = "ErrorList">
4001     <complexType>
4002         <sequence>
4003             <element ref = "tns:Error" maxOccurs = "unbounded"/>
4004             <any namespace = "##other" processContents = "lax" minOccurs =
4005 "0" maxOccurs = "unbounded"/>
4006         </sequence>
4007         <attributeGroup ref = "tns:headerExtension.grp"/>
4008         <attribute name = "highestSeverity" use = "required" type =
4009 "tns:severity.type"/>
4010     </complexType>
4011 </element>
4012 <element name = "Error">
4013     <complexType>
4014         <sequence>
4015             <element ref = "tns:Description" minOccurs = "0"/>
4016             <any namespace = "##other" processContents = "lax" minOccurs =
4017 "0" maxOccurs = "unbounded"/>
4018         </sequence>
4019         <attribute ref = "tns:id"/>
4020         <attribute name = "codeContext" default = "urn:oasis:names:tc:ebxml-
4021 msg:service:errors" type = "anyURI"/>
4022         <attribute name = "errorCode" use = "required" type = "tns:non-empty-
4023 string"/>
4024         <attribute name = "severity" use = "required" type =
4025 "tns:severity.type"/>
4026         <attribute name = "location" type = "tns:non-empty-string"/>
4027     </complexType>
4028 </element>
4029
4030 <!-- STATUS RESPONSE, for use in soap:Body element -->
4031
4032 <element name = "StatusResponse">
4033     <complexType>
4034         <sequence>
4035             <element ref = "tns:RefToMessageId"/>
4036             <element ref = "tns:Timestamp" minOccurs = "0"/>
4037             <any namespace = "##other" processContents = "lax" minOccurs =
4038 "0" maxOccurs = "unbounded"/>
4039         </sequence>
4040         <attributeGroup ref = "tns:bodyExtension.grp"/>
4041         <attribute name = "messageStatus" use = "required" type =
4042 "tns:messageStatus.type"/>
4043     </complexType>

```

```

4044 </element>
4045
4046 <!-- STATUS REQUEST, for use in soap:Body element -->
4047
4048 <element name = "StatusRequest">
4049     <complexType>
4050         <sequence>
4051             <element ref = "tns:RefToMessageId"/>
4052             <any namespace = "##other" processContents = "lax" minOccurs =
4053 "0" maxOccurs = "unbounded"/>
4054         </sequence>
4055         <attributeGroup ref = "tns:bodyExtension.grp"/>
4056     </complexType>
4057 </element>
4058
4059 <!-- COMMON TYPES -->
4060
4061 <complexType name = "sequenceNumber.type">
4062     <simpleContent>
4063         <extension base = "positiveInteger">
4064             <attribute name = "status" default = "Continue" type =
4065 "tns:status.type"/>
4066         </extension>
4067     </simpleContent>
4068 </complexType>
4069 <simpleType name = "status.type">
4070     <restriction base = "NMTOKEN">
4071         <enumeration value = "Reset"/>
4072         <enumeration value = "Continue"/>
4073     </restriction>
4074 </simpleType>
4075 <simpleType name = "messageStatus.type">
4076     <restriction base = "NMTOKEN">
4077         <enumeration value = "Unauthorized"/>
4078         <enumeration value = "NotRecognized"/>
4079         <enumeration value = "Received"/>
4080         <enumeration value = "Processed"/>
4081         <enumeration value = "Forwarded"/>
4082     </restriction>
4083 </simpleType>
4084 <simpleType name = "non-empty-string">
4085     <restriction base = "string">
4086         <minLength value = "1"/>
4087     </restriction>
4088 </simpleType>
4089 <simpleType name = "severity.type">
4090     <restriction base = "NMTOKEN">
4091         <enumeration value = "Warning"/>
4092         <enumeration value = "Error"/>
4093     </restriction>
4094 </simpleType>
4095
4096 <!-- COMMON ATTRIBUTES and ATTRIBUTE GROUPS -->
4097
4098 <attribute name = "id" type = "ID"/>
4099 <attribute name = "version" type = "tns:non-empty-string"/>
4100
4101 <!-- COMMON ELEMENTS -->
4102
4103 <element name = "PartyId">
4104     <complexType>
4105         <simpleContent>
4106             <extension base = "tns:non-empty-string">
4107                 <attribute name = "type" type = "tns:non-empty-
4108 string"/>
4109             </extension>
4110         </simpleContent>
4111     </complexType>
4112 </element>
4113 <element name = "To">
4114     <complexType>

```

```

4115         <sequence>
4116             <element ref = "tns:PartyId"/>
4117             <element name = "Role" type = "tns:non-empty-string" minOccurs
4118 = "0"/>
4119         </sequence>
4120     </complexType>
4121 </element>
4122 <element name = "From">
4123     <complexType>
4124         <sequence>
4125             <element ref = "tns:PartyId"/>
4126             <element name = "Role" type = "tns:non-empty-string" minOccurs
4127 = "0"/>
4128         </sequence>
4129     </complexType>
4130 </element>
4131 <element name = "Description">
4132     <complexType>
4133         <simpleContent>
4134             <extension base = "tns:non-empty-string">
4135                 <attribute ref = "xml:lang" use = "required"/>
4136             </extension>
4137         </simpleContent>
4138     </complexType>
4139 </element>
4140 <element name = "RefToMessageId" type = "tns:non-empty-string"/>
4141 <element name = "Timestamp" type = "dateTime"/>
4142 <element name = "FileName" type = "tns:non-empty-string"/>
4143 <element name = "MessageRef" type = "tns:non-empty-string"/>
4144 </schema>

```



---

## Appendix D (Normative) The ebXML Message Store Schema (and supporting sub-schemas)

The Message Store content schema below is a representation of the ebXML message envelope and any XML payload content that accompanies the message. Although the schema for the Message Store is generic in design, permitting any XML envelope structure, and any XML payload structure, in order to use a particular executable test suite, the structure of the messages within the Message Store MUST have an "agreed upon" schema. By defining a specific Message Store schema structure, Executable Test Cases can be written by any party wishing to contribute to a common Test Case library.

Below is the IIC schema for representing ebXML MS v2.0 message and payload content in the Message Store. Content stored in an IIC Test Driver Message Store MUST conform to this schema in order to execute the IIC MS 2.0 Conformance Test Suite using the IIC Test Framework V2.0.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests"
  xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
  xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmlsig"
  xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
  version = "1.0"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmlsig"
schemaLocation = "xmlsig.xsd"/>
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting poc 07 13 04/schemas/mime.xsd"/>
  <!-- <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmlsig"
schemaLocation = "xmlsig.xsd"/> -->

  <!-- <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting poc 06 27 04/schemas/mime.xsd"/> -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
```

```

4204 The limited permissions granted above are perpetual and will not be revoked by OASIS or
4205 its successors or assigns.
4206 -->
4207
4208     <element name = "TestSuite">
4209         <complexType>
4210             <sequence>
4211                 <element ref = "ebTest:MetaData"/>
4212                 <element ref = "ebTest:ConfigurationGroup" maxOccurs =
4213 "unbounded"/>
4214                 <element ref = "ebTest:TestServiceConfigurator" minOccurs =
4215 "0"/>
4216                 <element ref = "ebTest:Message" minOccurs = "0" maxOccurs =
4217 "unbounded"/>
4218                 <element ref = "ebTest:TestCase" maxOccurs = "unbounded"/>
4219             </sequence>
4220             <attribute name = "configurationGroupRef" use = "required" type =
4221 "anyURI"/>
4222         </complexType>
4223     </element>
4224     <element name = "MetaData">
4225         <complexType>
4226             <sequence>
4227                 <element ref = "ebTest:Description"/>
4228                 <element ref = "ebTest:Version"/>
4229                 <element ref = "ebTest:Maintainer"/>
4230                 <element ref = "ebTest:Location"/>
4231                 <element ref = "ebTest:PublishDate"/>
4232                 <element ref = "ebTest:Status"/>
4233             </sequence>
4234         </complexType>
4235     </element>
4236     <element name = "Description" type = "ebTest:non-empty-string"/>
4237     <element name = "Version" type = "ebTest:non-empty-string"/>
4238     <element name = "Maintainer" type = "ebTest:non-empty-string"/>
4239     <element name = "Location" type = "anyURI"/>
4240     <element name = "PublishDate" type = "ebTest:non-empty-string"/>
4241     <element name = "Status" type = "ebTest:non-empty-string"/>
4242     <element name = "TestCase">
4243         <complexType>
4244             <sequence>
4245                 <element ref = "ebTest:ThreadGroup" minOccurs = "0"/>
4246                 <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
4247 = "unbounded"/>
4248                 <choice maxOccurs = "unbounded">
4249                     <element ref = "ebTest:Thread"/>
4250                     <element ref = "ebTest:ThreadRef"/>
4251                     <element ref = "ebTest:Split"/>
4252                     <element ref = "ebTest:Join"/>
4253                     <element ref = "ebTest:Sleep"/>
4254                 </choice>
4255             </sequence>
4256             <attribute name = "id" use = "required" type = "ID"/>
4257             <attribute name = "description" use = "required" type = "string"/>
4258             <attribute name = "author" use = "optional" type = "string"/>
4259             <attribute name = "version" use = "optional" type = "string"/>
4260             <attribute name = "requirementReferenceId" use = "required" type =
4261 "anyURI"/>
4262             <attribute name = "configurationGroupRef" use = "optional" type =
4263 "anyURI"/>
4264         </complexType>
4265     </element>
4266     <element name = "ConfigurationGroup">
4267         <complexType>
4268             <sequence>
4269                 <element ref = "ebTest:Mode"/>
4270                 <element ref = "ebTest:StepDuration"/>
4271                 <element ref = "ebTest:Transport"/>
4272                 <element ref = "ebTest:Envelope"/>
4273                 <element ref = "ebTest:StoreAttachments"/>

```

```

4274         <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
4275 = "unbounded"/>
4276         </sequence>
4277         <attribute name = "id" use = "required" type = "ID"/>
4278     </complexType>
4279 </element>
4280 <element name = "CPAId" type = "ebTest:non-empty-string"/>
4281 <element name = "Mode" type = "ebTest:mode.type"/>
4282 <element name = "SenderParty" type = "anyURI"/>
4283 <element name = "ReceiverParty" type = "anyURI"/>
4284 <element name = "Service" type = "anyURI"/>
4285 <element name = "Action" type = "ebTest:non-empty-string"/>
4286 <element name = "StepDuration" type = "integer"/>
4287 <element name = "Transport" type = "ebTest:transport.type"/>
4288 <element name = "Envelope" type = "ebTest:non-empty-string"/>
4289 <simpleType name = "mode.type">
4290     <restriction base = "NMTOKEN">
4291         <enumeration value = "local-service"/>
4292         <enumeration value = "remote-service"/>
4293         <enumeration value = "connection"/>
4294     </restriction>
4295 </simpleType>
4296 <simpleType name = "mimeHeader.type">
4297     <restriction base = "NMTOKEN">
4298         <enumeration value = "MIMEMessageContent-Type"/>
4299         <enumeration value = "MIMEMessageStart"/>
4300         <enumeration value = "Content-Type"/>
4301         <enumeration value = "start"/>
4302         <enumeration value = "charset"/>
4303         <enumeration value = "type"/>
4304         <enumeration value = "wildcard"/>
4305     </restriction>
4306 </simpleType>
4307 <simpleType name = "content.type">
4308     <restriction base = "NMTOKEN">
4309         <enumeration value = "XML"/>
4310         <enumeration value = "date"/>
4311         <enumeration value = "URI"/>
4312         <enumeration value = "signature"/>
4313         <enumeration value = "XPointer"/>
4314     </restriction>
4315 </simpleType>
4316 <simpleType name = "method.type">
4317     <restriction base = "NMTOKEN">
4318         <enumeration value = "xpath"/>
4319         <enumeration value = "md5"/>
4320     </restriction>
4321 </simpleType>
4322 <simpleType name = "messageContext.type">
4323     <restriction base = "NMTOKEN">
4324         <enumeration value = "true"/>
4325         <enumeration value = "false"/>
4326     </restriction>
4327 </simpleType>
4328 <simpleType name = "requirement.type">
4329     <restriction base = "NMTOKEN">
4330         <enumeration value = "required"/>
4331         <enumeration value = "stronglyrecommended"/>
4332         <enumeration value = "recommended"/>
4333         <enumeration value = "optional"/>
4334     </restriction>
4335 </simpleType>
4336 <simpleType name = "non-empty-string">
4337     <restriction base = "string">
4338         <minLength value = "1"/>
4339     </restriction>
4340 </simpleType>
4341 <simpleType name = "configAction.type">
4342     <restriction base = "NMTOKEN">
4343         <enumeration value = "query"/>
4344         <enumeration value = "replace"/>

```

```

4345         </restriction>
4346     </simpleType>
4347     <simpleType name = "action.type">
4348         <restriction base = "NMTOKEN">
4349             <enumeration value = "reset"/>
4350             <enumeration value = "modify"/>
4351         </restriction>
4352     </simpleType>
4353     <simpleType name = "configItem.type">
4354         <restriction base = "NMTOKEN"/>
4355     </simpleType>
4356     <simpleType name = "parameter.type">
4357         <restriction base = "NMTOKEN">
4358             <enumeration value = "string"/>
4359             <enumeration value = "parameter"/>
4360         </restriction>
4361     </simpleType>
4362     <simpleType name = "connectivePredicate.type">
4363         <restriction base = "NMTOKEN">
4364             <enumeration value = "and"/>
4365             <enumeration value = "or"/>
4366         </restriction>
4367     </simpleType>
4368     <simpleType name = "thread.type">
4369         <restriction base = "NMTOKEN">
4370             <enumeration value = "synchronous"/>
4371             <enumeration value = "asynchronous"/>
4372         </restriction>
4373     </simpleType>
4374     <simpleType name = "matchResult.type">
4375         <restriction base = "NMTOKEN">
4376             <enumeration value = "pass"/>
4377             <enumeration value = "fail"/>
4378         </restriction>
4379     </simpleType>
4380     <simpleType name = "if.type">
4381         <restriction base = "NMTOKEN">
4382             <enumeration value = "andif"/>
4383             <enumeration value = "orif"/>
4384         </restriction>
4385     </simpleType>
4386     <simpleType name = "split.type">
4387         <restriction base = "NMTOKEN">
4388             <enumeration value = "andsplit"/>
4389             <enumeration value = "orsplit"/>
4390         </restriction>
4391     </simpleType>
4392     <simpleType name = "join.type">
4393         <restriction base = "NMTOKEN">
4394             <enumeration value = "andjoin"/>
4395             <enumeration value = "orjoin"/>
4396         </restriction>
4397     </simpleType>
4398     <simpleType name = "serviceMode.type">
4399         <restriction base = "NMTOKEN">
4400             <enumeration value = "loop"/>
4401             <enumeration value = "local-reporting"/>
4402             <enumeration value = "remote-reporting"/>
4403         </restriction>
4404     </simpleType>
4405     <simpleType name = "time.type">
4406         <restriction base = "NMTOKEN">
4407             <enumeration value = "timeToAcknowledgeReceipt"/>
4408             <enumeration value = "timeToAcknowledgeAcceptance"/>
4409             <enumeration value = "timeToPerform"/>
4410             <enumeration value = "other"/>
4411         </restriction>
4412     </simpleType>
4413     <simpleType name = "operator.type">
4414         <restriction base = "NMTOKEN">
4415             <enumeration value = "equal"/>

```

```

4416         <enumeration value = "lessThan1"/>
4417         <enumeration value = "lessThanOrEqual"/>
4418         <enumeration value = "greaterThan"/>
4419         <enumeration value = "greaterThanOrEqual"/>
4420     </restriction>
4421 </simpleType>
4422 <simpleType name = "assertionExit.type">
4423     <restriction base = "NMTOKEN">
4424         <enumeration value = "pass"/>
4425         <enumeration value = "fail"/>
4426         <enumeration value = "undetermined"/>
4427     </restriction>
4428 </simpleType>
4429 <simpleType name = "preconditionExit.type">
4430     <restriction base = "NMTOKEN">
4431         <enumeration value = "undetermined"/>
4432     </restriction>
4433 </simpleType>
4434 <simpleType name = "scope.type">
4435     <restriction base = "NMTOKEN">
4436         <enumeration value = "self"/>
4437         <enumeration value = "selfAndDescendents"/>
4438     </restriction>
4439 </simpleType>
4440 <simpleType name = "transport.type">
4441     <restriction base = "NMTOKEN">
4442         <enumeration value = "FTP"/>
4443         <enumeration value = "SMTP"/>
4444         <enumeration value = "HTTP"/>
4445     </restriction>
4446 </simpleType>
4447 <element name = "MessageExpression">
4448     <complexType>
4449         <sequence>
4450             <element ref = "ebTest:ErrorMessage"/>
4451         </sequence>
4452     </complexType>
4453 </element>
4454 <element name = "ErrorMessage" type = "ebTest:non-empty-string"/>
4455 <element name = "PutMessage">
4456     <complexType>
4457         <sequence>
4458             <element ref = "ebTest:SetPart" maxOccurs = "unbounded"/>
4459         </sequence>
4460         <attribute name = "description" use = "required" type = "string"/>
4461         <attribute name = "repeatWithSameContext" use = "optional" type =
4462 "integer"/>
4463         <attribute name = "repeatWithNewContext" use = "optional" type =
4464 "integer"/>
4465     </complexType>
4466 </element>
4467 <element name = "GetPayload">
4468     <complexType>
4469         <sequence>
4470             <choice>
4471                 <element ref = "ebTest:Content-ID"/>
4472                 <element ref = "ebTest:Content-Location"/>
4473                 <element ref = "ebTest:Index"/>
4474             </choice>
4475             <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
4476 maxOccurs = "unbounded"/>
4477         </sequence>
4478         <attribute name = "description" use = "required" type = "string"/>
4479     </complexType>
4480 </element>
4481 <element name = "GetMessage">
4482     <complexType>
4483         <sequence maxOccurs = "unbounded">
4484             <element ref = "ebTest:Filter"/>
4485             <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
4486 maxOccurs = "unbounded"/>

```

```

4487         </sequence>
4488         <attribute name = "description" use = "required" type = "string"/>
4489         <attribute name = "mask" use = "optional" type = "boolean"/>
4490     </complexType>
4491 </element>
4492 <element name = "Filter">
4493     <complexType>
4494         <simpleContent>
4495             <extension base = "ebTest:non-empty-string">
4496                 <attribute name = "stepDuration" use = "optional" type
4497 = "integer"/>
4498             </extension>
4499         </simpleContent>
4500     </complexType>
4501 </element>
4502 <element name = "SetPart">
4503     <complexType>
4504         <sequence>
4505             <element ref = "ebTest:Header" minOccurs = "0" maxOccurs =
4506 "unbounded"/>
4507             <choice>
4508                 <element ref = "ebTest:Declaration"/>
4509                 <element ref = "ebTest:FileURI"/>
4510                 <element ref = "ebTest:MessageRef"/>
4511             </choice>
4512             <element ref = "ebTest:Mutator" minOccurs = "0"/>
4513             <element ref = "ebTest:DSign" minOccurs = "0"/>
4514         </sequence>
4515         <attribute name = "description" use = "optional" type = "string"/>
4516     </complexType>
4517 </element>
4518 <element name = "TestAssertion">
4519     <complexType>
4520         <sequence>
4521             <choice>
4522                 <element ref = "ebTest:VerifyContent"/>
4523                 <element ref = "ebTest:ValidateContent"/>
4524                 <element ref = "ebTest:VerifyTimeDifference"/>
4525             </choice>
4526             <element name = "WhenTrue" minOccurs = "0">
4527                 <complexType>
4528                     <choice>
4529                         <element ref = "ebTest:Continue"/>
4530                         <element ref = "ebTest:ThreadRef"/>
4531                         <element ref = "ebTest:Split"/>
4532                         <element name = "Exit" type =
4533 "ebTest:assertionExit.type"/>
4534                     </choice>
4535                 </complexType>
4536             </element>
4537             <element name = "WhenFalse" minOccurs = "0">
4538                 <complexType>
4539                     <choice>
4540                         <element ref = "ebTest:Continue"/>
4541                         <element ref = "ebTest:ThreadRef"/>
4542                         <element ref = "ebTest:Split"/>
4543                         <element name = "Exit" type =
4544 "ebTest:assertionExit.type"/>
4545                     </choice>
4546                 </complexType>
4547             </element>
4548         </sequence>
4549         <attribute name = "description" use = "required" type = "string"/>
4550     </complexType>
4551 </element>
4552 <element name = "MimeHeader" type = "ebTest:mimeHeader.type"/>
4553 <element name = "MimeHeaderValue" type = "ebTest:non-empty-string"/>
4554 <element name = "Content-Location" type = "ebTest:non-empty-string"/>
4555 <element name = "Index" type = "integer"/>
4556 <element name = "FileURI" type = "anyURI"/>
4557 <element name = "PayloadRef" type = "ebTest:non-empty-string"/>

```

```

4558     <element name = "Signature" type = "base64Binary"/>
4559     <element name = "Content-ID" type = "ebTest:non-empty-string"/>
4560     <element name = "MessageDeclaration">
4561         <complexType>
4562             <sequence>
4563                 <any namespace = "##other" processContents = "lax" minOccurs =
4564 "0" maxOccurs = "unbounded"/>
4565             </sequence>
4566         </complexType>
4567     </element>
4568     <element name = "ValidateContent">
4569         <complexType>
4570             <simpleContent>
4571                 <extension base = "ebTest:non-empty-string">
4572                     <attribute name = "contentType" use = "required" type =
4573 "ebTest:content.type"/>
4574                     <attribute name = "schemaLocation" use = "optional"
4575 type = "anyURI"/>
4576                 </extension>
4577             </simpleContent>
4578         </complexType>
4579     </element>
4580     <element name = "VerifyContent" type = "ebTest:non-empty-string"/>
4581     <element name = "Message">
4582         <complexType>
4583             <sequence>
4584                 <any namespace = "##other" processContents = "lax" minOccurs =
4585 "0" maxOccurs = "unbounded"/>
4586             </sequence>
4587             <attribute name = "id" use = "required" type = "ID"/>
4588         </complexType>
4589     </element>
4590     <element name = "SetParameter">
4591         <complexType>
4592             <sequence>
4593                 <element name = "Name" type = "ebTest:non-empty-string"/>
4594                 <choice>
4595                     <element name = "Value" type = "ebTest:non-empty-
4596 string"/>
4597                     <element name = "ParameterRef" type = "ebTest:non-
4598 empty-string"/>
4599                 </choice>
4600             </sequence>
4601             <attribute name = "scope" use = "optional" type =
4602 "ebTest:scope.type"/>
4603         </complexType>
4604     </element>
4605     <element name = "Mutator">
4606         <complexType>
4607             <choice>
4608                 <element ref = "ebTest:XSL"/>
4609                 <element ref = "ebTest:XUpdate"/>
4610             </choice>
4611         </complexType>
4612     </element>
4613     <element name = "XSL" type = "anyURI"/>
4614     <element name = "XUpdate" type = "anyURI"/>
4615     <element name = "BooleanClause">
4616         <complexType>
4617             <attribute name = "booleanPredicate" use = "required" type =
4618 "boolean"/>
4619         </complexType>
4620     </element>
4621     <element name = "DSign">
4622         <complexType>
4623             <sequence>
4624                 <element ref = "ds:Signature"/>
4625             </sequence>
4626         </complexType>
4627     </element>
4628     <element name = "Declaration">

```



```

4629         <complexType>
4630             <sequence>
4631                 <any namespace = "##other" processContents = "lax" minOccurs =
4632 "0" maxOccurs = "unbounded"/>
4633             </sequence>
4634         </complexType>
4635     </element>
4636     <element name = "Thread">
4637         <complexType>
4638             <choice maxOccurs = "unbounded">
4639                 <element ref = "ebTest:SetParameter"/>
4640                 <element ref = "ebTest:PutMessage"/>
4641                 <element ref = "ebTest:Initiator"/>
4642                 <element ref = "ebTest:GetMessage"/>
4643                 <element ref = "ebTest:TestAssertion"/>
4644                 <element ref = "ebTest:ThreadRef"/>
4645                 <element ref = "ebTest:Split"/>
4646                 <element ref = "ebTest:Join"/>
4647                 <element ref = "ebTest:Sleep"/>
4648             </choice>
4649             <attribute name = "name" use = "required" type = "ID"/>
4650             <attribute name = "description" use = "optional" type = "string"/>
4651         </complexType>
4652     </element>
4653     <element name = "ThreadRef">
4654         <complexType>
4655             <attribute name = "nameRef" use = "required" type = "IDREF"/>
4656             <attribute name = "configurationGroupRef" use = "optional" type =
4657 "anyURI"/>
4658             <attribute name = "loop" use = "optional" type = "integer"/>
4659             <attribute name = "instanceId" use = "optional" type = "string"/>
4660         </complexType>
4661     </element>
4662     <element name = "Pass">
4663         <complexType/>
4664     </element>
4665     <element name = "Fail">
4666         <complexType/>
4667     </element>
4668     <element name = "ThreadGroup">
4669         <complexType>
4670             <sequence>
4671                 <element ref = "ebTest:Thread" maxOccurs = "unbounded"/>
4672             </sequence>
4673         </complexType>
4674     </element>
4675     <element name = "Sleep" type = "integer"/>
4676     <element name = "Split">
4677         <complexType>
4678             <sequence maxOccurs = "unbounded">
4679                 <element ref = "ebTest:ThreadRef"/>
4680             </sequence>
4681         </complexType>
4682     </element>
4683     <element name = "Join">
4684         <complexType>
4685             <sequence maxOccurs = "unbounded">
4686                 <element ref = "ebTest:ThreadRef"/>
4687             </sequence>
4688             <attribute name = "joinType" use = "optional" type =
4689 "ebTest:join.type"/>
4690         </complexType>
4691     </element>
4692     <element name = "Initiator">
4693         <complexType>
4694             <sequence>
4695                 <choice>
4696                     <element ref = "ebTest:Declaration"/>
4697                     <element ref = "ebTest:FileURI"/>
4698                     <element ref = "ebTest:MessageRef"/>
4699                 </choice>

```



```

4700         <element ref = "ebTest:Mutator" minOccurs = "0"/>
4701         <element ref = "ebTest:DSign" minOccurs = "0"/>
4702     </sequence>
4703     <attribute name = "description" use = "required" type = "string"/>
4704 </complexType>
4705 </element>
4706 <element name = "TestServiceConfigurator">
4707     <complexType>
4708         <sequence>
4709             <element ref = "ebTest:ServiceMode"/>
4710             <element ref = "ebTest:ResponseURL"/>
4711             <element ref = "ebTest:NotificationURL"/>
4712             <element ref = "ebTest:PayloadDigests" minOccurs = "0"/>
4713         </sequence>
4714     </complexType>
4715 </element>
4716 <element name = "MessageRef" type = "IDREF"/>
4717 <element name = "ConfigurationItem">
4718     <complexType>
4719         <sequence>
4720             <element name = "Name" type = "ebTest:non-empty-string"/>
4721             <element name = "Value" type = "ebTest:non-empty-string"/>
4722         </sequence>
4723     </complexType>
4724 </element>
4725 <element name = "ErrorURL" type = "anyURI"/>
4726 <element name = "NotificationURL" type = "anyURI"/>
4727 <element name = "SetXPathParameter">
4728     <complexType>
4729         <sequence>
4730             <element name = "Name" type = "ebTest:non-empty-string"/>
4731             <element name = "Expression" type = "ebTest:non-empty-
4732 string"/>
4733         </sequence>
4734         <attribute name = "scope" use = "optional" type =
4735 "ebTest:scope.type"/>
4736     </complexType>
4737 </element>
4738 <element name = "ResponseURL" type = "anyURI"/>
4739 <element name = "StoreAttachments" type = "boolean"/>
4740 <element name = "OperationMode" type = "string"/>
4741 <element name = "PayloadDigests">
4742     <complexType>
4743         <sequence>
4744             <element name = "Payload" maxOccurs = "unbounded">
4745                 <complexType>
4746                     <sequence>
4747                         <element name = "Id" type = "anyURI"/>
4748                         <element name = "Digest" type =
4749 "base64Binary"/>
4750                     </sequence>
4751                 </complexType>
4752             </element>
4753         </sequence>
4754     </complexType>
4755 </element>
4756 <element name = "ServiceMode" type = "ebTest:serviceMode.type"/>
4757 <element name = "Transaction">
4758     <complexType>
4759         <sequence maxOccurs = "unbounded">
4760             <choice maxOccurs = "unbounded">
4761                 <element ref = "ebTest:PutMessage"/>
4762                 <element ref = "ebTest:Initiator"/>
4763             </choice>
4764             <element ref = "ebTest:GetMessage" minOccurs = "0" maxOccurs =
4765 "unbounded"/>
4766         </sequence>
4767         <attribute name = "timeToPerform" use = "optional" type = "duration"/>
4768     </complexType>
4769 </element>
4770 <element name = "VerifyTimeDifference">

```

```

4771     <complexType>
4772         <sequence>
4773             <element ref = "ebTest:ParamName"/>
4774             <element ref = "ebTest:ParamName"/>
4775             <element ref = "ebTest:Operator"/>
4776             <element ref = "ebTest:Difference"/>
4777         </sequence>
4778     </complexType>
4779 </element>
4780 <element name = "TimeToAcknowledgeReceipt">
4781     <complexType>
4782         <sequence>
4783             <element ref = "ebTest:XPathExpression"/>
4784         </sequence>
4785     </complexType>
4786 </element>
4787 <element name = "TimeToAcknowledgeAcceptance">
4788     <complexType>
4789         <sequence>
4790             <element ref = "ebTest:XPathExpression"/>
4791         </sequence>
4792     </complexType>
4793 </element>
4794 <element name = "Difference" type = "duration"/>
4795 <element name = "Operator" type = "ebTest:operator.type"/>
4796 <element name = "XPathExpression" type = "ebTest:non-empty-string"/>
4797 <element name = "Continue">
4798     <complexType/>
4799 </element>
4800 <element name = "ParamName" type = "ebTest:non-empty-string"/>
4801 <element name = "VerifyTimeToPerform">
4802     <complexType>
4803         <sequence>
4804             <element ref = "ebTest:ThreadName" maxOccurs = "unbounded"/>
4805         </sequence>
4806         <attribute name = "maxTime" use = "required" type = "duration"/>
4807     </complexType>
4808 </element>
4809 <element name = "ThreadName" type = "IDREF"/>
4810 <element name = "Header">
4811     <complexType>
4812         <sequence>
4813             <element ref = "ebTest:Name"/>
4814             <element ref = "ebTest:Value"/>
4815         </sequence>
4816     </complexType>
4817 </element>
4818 <element name = "Name" type = "ebTest:non-empty-string"/>
4819 <element name = "Value" type = "ebTest:non-empty-string"/>

```

4820 </schema>

4821

4822

4823

4824

4825

4826

4827

4828

4829

4830

4831

4832  
4833  
4834  
4835  
4836  
4837  
4838  
4839

## SOAP Portion of ebXML-Specific Message Store Schema

```
4840 <?xml version = "1.0" encoding = "UTF-8"?>
4841 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
4842 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
4843   targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
4844   xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
4845   xmlns:xs = "http://www.w3.org/2001/XMLSchema"
4846   xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
4847 2 0.xsd">
4848   <import namespace = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
4849 header-2 0.xsd" schemaLocation = "http://www.oasis-open.org/committees/ebxml-
4850 msg/schema/msg-header-2 0.xsd"/>
4851   <group name = "optionElements">
4852     <all minOccurs = "0">
4853       <element ref = "eb:SyncReply" minOccurs = "0"/>
4854       <element ref = "eb:MessageOrder" minOccurs = "0"/>
4855       <element ref = "eb:AckRequested" minOccurs = "0"/>
4856       <element ref = "eb:Acknowledgment" minOccurs = "0"/>
4857       <element ref = "eb:ErrorList" minOccurs = "0"/>
4858     </all>
4859   </group>
4860   <attributeGroup name = "encodingStyle">
4861     <attribute name = "encodingStyle" type = "tns:encodingStyle"/>
4862   </attributeGroup>
4863
4864   <!-- Schema for the SOAP/1.1 envelope
4865
4866   This schema has been produced using W3C's SOAP Version 1.2 schema
4867   found at:
4868
4869   http://www.w3.org/2001/06/soap-envelope
4870
4871   Copyright 2001 Martin Gudgin, Developmentor.
4872
4873   Changes made are the following:
4874   - reverted namespace to http://schemas.xmlsoap.org/soap/envelope/
4875   - reverted mustUnderstand to only allow 0 and 1 as lexical values
4876
4877   Original copyright:
4878
4879   Copyright 2001 W3C (Massachusetts Institute of Technology,
4880   Institut National de Recherche en Informatique et en Automatique,
4881   Keio University). All Rights Reserved.
4882   http://www.w3.org/Consortium/Legal/
4883
4884   This document is governed by the W3C Software License [1] as
4885   described in the FAQ [2].
4886
4887   [1] http://www.w3.org/Consortium/Legal/copyright-software-19980720
4888   [2] http://www.w3.org/Consortium/Legal/IPR-FAQ-20000620.html#DTD
4889 -->
4890
4891
4892 <!-- Envelope, header and body -->
4893
4894 <element name = "Envelope" type = "tns:Envelope"/>
4895 <complexType name = "Envelope">
4896   <sequence>
```

```

4897         <element ref = "tns:Header"/>
4898         <element ref = "tns:Body"/>
4899         <any namespace = "##other" processContents = "lax" minOccurs = "0"
4900 maxOccurs = "unbounded"/>
4901     </sequence>
4902     <anyAttribute namespace = "##other" processContents = "lax"/>
4903 </complexType>
4904 <element name = "Header">
4905     <complexType>
4906         <sequence>
4907             <element ref = "eb:MessageHeader"/>
4908             <group ref = "tns:optionElements"/>
4909         </sequence>
4910     </complexType>
4911 </element>
4912 <complexType name = "Header">
4913     <sequence>
4914         <any namespace = "##other" processContents = "lax" minOccurs = "0"
4915 maxOccurs = "unbounded"/>
4916     </sequence>
4917     <anyAttribute namespace = "##other" processContents = "lax"/>
4918 </complexType>
4919 <element name = "Body">
4920     <complexType>
4921         <choice>
4922             <element ref = "eb:Manifest"/>
4923             <element ref = "eb:StatusRequest"/>
4924             <element ref = "eb:StatusResponse"/>
4925         </choice>
4926     </complexType>
4927 </element>
4928 <complexType name = "Body">
4929     <annotation>
4930         <documentation>
4931             Prose in the spec does not specify that attributes are allowed on the Body
4932 element
4933         </documentation>
4934     </annotation>
4935     <sequence>
4936         <any namespace = "##any" processContents = "lax" minOccurs = "0"
4937 maxOccurs = "unbounded"/>
4938     </sequence>
4939     <anyAttribute namespace = "##any" processContents = "lax"/>
4940 </complexType>
4941
4942 <!-- Global Attributes. The following attributes are intended to be usable via
4943 qualified attribute names on any complex type referencing them. -->
4944
4945 <attribute name = "mustUnderstand" default = "0">
4946     <simpleType>
4947         <restriction base = "boolean">
4948             <pattern value = "0|1"/>
4949         </restriction>
4950     </simpleType>
4951 </attribute>
4952 <attribute name = "actor" type = "anyURI"/>
4953 <simpleType name = "encodingStyle">
4954     <annotation>
4955         <documentation>
4956             'encodingStyle' indicates any canonicalization conventions followed in the
4957 contents of the containing element. For example, the value
4958 'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern described in SOAP
4959 specification
4960         </documentation>
4961     </annotation>
4962     <list itemType = "anyURI"/>
4963 </simpleType>
4964 <complexType name = "Fault">
4965     final = "extension">
4966     <annotation>
4967         <documentation>

```

4968  
4969  
4970  
4971  
4972  
4973  
4974  
4975  
4976  
4977  
4978  
4979  
4980  
4981  
4982  
4983  
4984  
4985

```
    Fault reporting structure
  </documentation>
    </annotation>
    <sequence>
      <element name = "faultcode" type = "QName"/>
      <element name = "faultstring" type = "string"/>
      <element name = "faultactor" type = "anyURI" minOccurs = "0"/>
      <element name = "detail" type = "tns:detail" minOccurs = "0"/>
    </sequence>
  </complexType>
  <complexType name = "detail">
    <sequence>
      <any namespace = "##any" processContents = "lax" minOccurs = "0"
maxOccurs = "unbounded"/>
    </sequence>
    <anyAttribute namespace = "##any" processContents = "lax"/>
  </complexType>
</schema>
```

4986  
4987  
4988  
4989

## ebMS Portion of ebXML-Specific Message Store Schema

4990  
4991  
4992  
4993  
4994  
4995  
4996  
4997  
4998  
4999  
5000  
5001  
5002  
5003  
5004  
5005  
5006  
5007  
5008  
5009  
5010  
5011  
5012  
5013  
5014  
5015  
5016  
5017  
5018  
5019  
5020  
5021  
5022  
5023  
5024  
5025  
5026  
5027  
5028  
5029  
5030  
5031  
5032  
5033  
5034

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
  xmlns:tns = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
  xmlns:xlink = "http://www.w3.org/1999/xlink"
  xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
  xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/"

  version = "1.0"
  elementFormDefault = "qualified"
  attributeFormDefault = "qualified">
  <import namespace = "http://www.w3.org/1999/xlink" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd"/>
  <import namespace = "http://www.w3.org/2000/09/xmldsig#" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xmldsig-core-schema.xsd"/>
  <import namespace = "http://schemas.xmlsoap.org/soap/envelope/" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd"/>
  <import namespace = "http://www.w3.org/XML/1998/namespace" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xml lang.xsd"/>
  <attributeGroup name = "bodyExtension.grp">
    <attribute ref = "tns:id"/>
    <attribute ref = "tns:version" use = "required"/>
  </attributeGroup>
  <attributeGroup name = "headerExtension.grp">
    <attribute ref = "tns:id"/>
    <attribute ref = "tns:version" use = "required"/>
    <attribute ref = "soap:mustUnderstand" use = "required"/>
  </attributeGroup>

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
```

```

5035 The limited permissions granted above are perpetual and will not be revoked by OASIS or
5036 its successors or assigns.
5037 -->
5038
5039
5040 <!-- MANIFEST, for use in soap:Body element -->
5041
5042 <element name = "Manifest">
5043   <complexType>
5044     <sequence>
5045       <element ref = "tns:Reference" maxOccurs = "unbounded"/>
5046       <any namespace = "##other" processContents = "lax" minOccurs =
5047 "0" maxOccurs = "unbounded"/>
5048     </sequence>
5049     <attributeGroup ref = "tns:bodyExtension.grp"/>
5050   </complexType>
5051 </element>
5052 <element name = "Reference">
5053   <complexType>
5054     <sequence>
5055       <element ref = "tns:Schema" minOccurs = "0" maxOccurs =
5056 "unbounded"/>
5057       <element ref = "tns:Description" minOccurs = "0" maxOccurs =
5058 "unbounded"/>
5059       <any namespace = "##other" processContents = "lax" minOccurs =
5060 "0" maxOccurs = "unbounded"/>
5061     </sequence>
5062     <attribute ref = "tns:id"/>
5063     <attribute ref = "xlink:type" fixed = "simple"/>
5064     <attribute ref = "xlink:href" use = "required"/>
5065     <attribute ref = "xlink:role"/>
5066   </complexType>
5067 </element>
5068 <element name = "Schema">
5069   <complexType>
5070     <attribute name = "location" use = "required" type = "anyURI"/>
5071     <attribute name = "version" type = "tns:non-empty-string"/>
5072   </complexType>
5073 </element>
5074
5075 <!-- MESSAGEHEADER, for use in soap:Header element -->
5076
5077 <element name = "MessageHeader">
5078   <complexType>
5079     <sequence>
5080       <element ref = "tns:From"/>
5081       <element ref = "tns:To"/>
5082       <element ref = "tns:CPAId"/>
5083       <element ref = "tns:ConversationId"/>
5084       <element ref = "tns:Service"/>
5085       <element ref = "tns:Action"/>
5086       <element ref = "tns:MessageData"/>
5087       <element ref = "tns:DuplicateElimination" minOccurs = "0"/>
5088       <element ref = "tns:Description" minOccurs = "0" maxOccurs =
5089 "unbounded"/>
5090       <any namespace = "##other" processContents = "lax" minOccurs =
5091 "0" maxOccurs = "unbounded"/>
5092     </sequence>
5093     <attributeGroup ref = "tns:headerExtension.grp"/>
5094   </complexType>
5095 </element>
5096 <element name = "CPAId" type = "tns:non-empty-string"/>
5097 <element name = "ConversationId" type = "tns:non-empty-string"/>
5098 <element name = "Service">
5099   <complexType>
5100     <simpleContent>
5101       <extension base = "tns:non-empty-string">
5102         <attribute name = "type" type = "tns:non-empty-
5103 string"/>
5104       </extension>
5105     </simpleContent>

```

```

5106         </complexType>
5107     </element>
5108     <element name = "Action" type = "tns:non-empty-string"/>
5109     <element name = "MessageData">
5110         <complexType>
5111             <sequence>
5112                 <element ref = "tns:MessageId"/>
5113                 <element ref = "tns:Timestamp"/>
5114                 <element ref = "tns:RefToMessageId" minOccurs = "0"/>
5115                 <element ref = "tns:TimeToLive" minOccurs = "0"/>
5116             </sequence>
5117         </complexType>
5118     </element>
5119     <element name = "MessageId" type = "tns:non-empty-string"/>
5120     <element name = "TimeToLive" type = "dateTime"/>
5121     <element name = "DuplicateElimination"/>
5122
5123     <!-- SYNC REPLY, for use in soap:Header element -->
5124
5125     <element name = "SyncReply">
5126         <complexType>
5127             <sequence>
5128                 <any namespace = "##other" processContents = "lax" minOccurs =
5129 "0" maxOccurs = "unbounded"/>
5130             </sequence>
5131             <attributeGroup ref = "tns:headerExtension.grp"/>
5132             <attribute ref = "soap:actor" use = "required"/>
5133         </complexType>
5134     </element>
5135
5136     <!-- MESSAGE ORDER, for use in soap:Header element -->
5137
5138     <element name = "MessageOrder">
5139         <complexType>
5140             <sequence>
5141                 <element ref = "tns:SequenceNumber"/>
5142                 <any namespace = "##other" processContents = "lax" minOccurs =
5143 "0" maxOccurs = "unbounded"/>
5144             </sequence>
5145             <attributeGroup ref = "tns:headerExtension.grp"/>
5146         </complexType>
5147     </element>
5148     <element name = "SequenceNumber" type = "tns:sequenceNumber.type"/>
5149
5150     <!-- ACK REQUESTED, for use in soap:Header element -->
5151
5152     <element name = "AckRequested">
5153         <complexType>
5154             <sequence>
5155                 <any namespace = "##other" processContents = "lax" minOccurs =
5156 "0" maxOccurs = "unbounded"/>
5157             </sequence>
5158             <attributeGroup ref = "tns:headerExtension.grp"/>
5159             <attribute ref = "soap:actor"/>
5160             <attribute name = "signed" use = "required" type = "boolean"/>
5161         </complexType>
5162     </element>
5163
5164     <!-- ACKNOWLEDGMENT, for use in soap:Header element -->
5165
5166     <element name = "Acknowledgment">
5167         <complexType>
5168             <sequence>
5169                 <element ref = "tns:Timestamp"/>
5170                 <element ref = "tns:RefToMessageId"/>
5171                 <element ref = "tns:From" minOccurs = "0"/>
5172                 <element ref = "ds:Reference" minOccurs = "0" maxOccurs =
5173 "unbounded"/>
5174                 <any namespace = "##other" processContents = "lax" minOccurs =
5175 "0" maxOccurs = "unbounded"/>
5176             </sequence>

```

```

5177         <attributeGroup ref = "tns:headerExtension.grp"/>
5178         <attribute ref = "soap:actor"/>
5179     </complexType>
5180 </element>
5181
5182 <!-- ERROR LIST, for use in soap:Header element -->
5183
5184 <element name = "ErrorList">
5185     <complexType>
5186         <sequence>
5187             <element ref = "tns:Error" maxOccurs = "unbounded"/>
5188             <any namespace = "##other" processContents = "lax" minOccurs =
5189 "0" maxOccurs = "unbounded"/>
5190         </sequence>
5191         <attributeGroup ref = "tns:headerExtension.grp"/>
5192         <attribute name = "highestSeverity" use = "required" type =
5193 "tns:severity.type"/>
5194     </complexType>
5195 </element>
5196 <element name = "Error">
5197     <complexType>
5198         <sequence>
5199             <element ref = "tns:Description" minOccurs = "0"/>
5200             <any namespace = "##other" processContents = "lax" minOccurs =
5201 "0" maxOccurs = "unbounded"/>
5202         </sequence>
5203         <attribute ref = "tns:id"/>
5204         <attribute name = "codeContext" default = "urn:oasis:names:tc:ebxml-
5205 msg:service:errors" type = "anyURI"/>
5206         <attribute name = "errorCode" use = "required" type = "tns:non-empty-
5207 string"/>
5208         <attribute name = "severity" use = "required" type =
5209 "tns:severity.type"/>
5210         <attribute name = "location" type = "tns:non-empty-string"/>
5211     </complexType>
5212 </element>
5213
5214 <!-- STATUS RESPONSE, for use in soap:Body element -->
5215
5216 <element name = "StatusResponse">
5217     <complexType>
5218         <sequence>
5219             <element ref = "tns:RefToMessageId"/>
5220             <element ref = "tns:Timestamp" minOccurs = "0"/>
5221             <any namespace = "##other" processContents = "lax" minOccurs =
5222 "0" maxOccurs = "unbounded"/>
5223         </sequence>
5224         <attributeGroup ref = "tns:bodyExtension.grp"/>
5225         <attribute name = "messageStatus" use = "required" type =
5226 "tns:messageStatus.type"/>
5227     </complexType>
5228 </element>
5229
5230 <!-- STATUS REQUEST, for use in soap:Body element -->
5231
5232 <element name = "StatusRequest">
5233     <complexType>
5234         <sequence>
5235             <element ref = "tns:RefToMessageId"/>
5236             <any namespace = "##other" processContents = "lax" minOccurs =
5237 "0" maxOccurs = "unbounded"/>
5238         </sequence>
5239         <attributeGroup ref = "tns:bodyExtension.grp"/>
5240     </complexType>
5241 </element>
5242
5243 <!-- COMMON TYPES -->
5244
5245 <complexType name = "sequenceNumber.type">
5246     <simpleContent>
5247         <extension base = "positiveInteger">

```



```

5248         <attribute name = "status" default = "Continue" type =
5249 "tns:status.type"/>
5250     </extension>
5251 </simpleContent>
5252 </complexType>
5253 <simpleType name = "status.type">
5254     <restriction base = "NMTOKEN">
5255         <enumeration value = "Reset"/>
5256         <enumeration value = "Continue"/>
5257     </restriction>
5258 </simpleType>
5259 <simpleType name = "messageStatus.type">
5260     <restriction base = "NMTOKEN">
5261         <enumeration value = "Unauthorized"/>
5262         <enumeration value = "NotRecognized"/>
5263         <enumeration value = "Received"/>
5264         <enumeration value = "Processed"/>
5265         <enumeration value = "Forwarded"/>
5266     </restriction>
5267 </simpleType>
5268 <simpleType name = "non-empty-string">
5269     <restriction base = "string">
5270         <minLength value = "1"/>
5271     </restriction>
5272 </simpleType>
5273 <simpleType name = "severity.type">
5274     <restriction base = "NMTOKEN">
5275         <enumeration value = "Warning"/>
5276         <enumeration value = "Error"/>
5277     </restriction>
5278 </simpleType>
5279
5280 <!-- COMMON ATTRIBUTES and ATTRIBUTE GROUPS -->
5281
5282 <attribute name = "id" type = "ID"/>
5283 <attribute name = "version" type = "tns:non-empty-string"/>
5284
5285 <!-- COMMON ELEMENTS -->
5286
5287 <element name = "PartyId">
5288     <complexType>
5289         <simpleContent>
5290             <extension base = "tns:non-empty-string">
5291                 <attribute name = "type" type = "tns:non-empty-
5292 string"/>
5293             </extension>
5294         </simpleContent>
5295     </complexType>
5296 </element>
5297 <element name = "To">
5298     <complexType>
5299         <sequence>
5300             <element ref = "tns:PartyId" maxOccurs = "unbounded"/>
5301             <element name = "Role" type = "tns:non-empty-string" minOccurs
5302 = "0"/>
5303         </sequence>
5304     </complexType>
5305 </element>
5306 <element name = "From">
5307     <complexType>
5308         <sequence>
5309             <element ref = "tns:PartyId" maxOccurs = "unbounded"/>
5310             <element name = "Role" type = "tns:non-empty-string" minOccurs
5311 = "0"/>
5312         </sequence>
5313     </complexType>
5314 </element>
5315 <element name = "Description">
5316     <complexType>
5317         <simpleContent>
5318             <extension base = "tns:non-empty-string">

```

5319  
5320  
5321  
5322  
5323  
5324  
5325  
5326

```
                <attribute ref = "xml:lang" use = "required"/>
            </extension>
        </simpleContent>
    </complexType>
</element>
<element name = "RefToMessageId" type = "tns:non-empty-string"/>
<element name = "Timestamp" type = "dateTime"/>
</schema>
```

5327

5328

5329

5330 Generic FilterResult Schema

5331

5332

5333

5334

5335

5336

5337

5338

5339

5340

5341

5342

5343

5344

5345

5346

5347

5348

5349

5350

5351

5352

5353

5354

5355

5356

5357

5358

5359

5360

5361

5362

5363

5364

5365

5366

5367

5368

5369

5370

5371

5372

5373

5374

5375

5376

5377

5378

5379

5380

5381

5382

5383

5384

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<xsd:schema xmlns = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!--
  Copyright (C) The Organization for the Advancement of Structured Information Standards
  [OASIS]
  January 2002. All Rights Reserved.
  This document and translations of it may be copied and furnished to others, and
  derivative works that comment on or otherwise explain it or assist in its implementation
  may be prepared, copied, published and distributed, in whole or in part, without
  restriction of any kind, provided that the above copyright notice and this paragraph are
  included on all such copies and derivative works. However, this document itself may not
  be modified in any way, such as by removing the copyright notice or references to OASIS,
  except as needed for the purpose of developing OASIS specifications, in which case the
  procedures for copyrights defined in the OASIS Intellectual Property Rights document
  MUST be followed, or as required to translate it into languages other than English.
  The limited permissions granted above are perpetual and will not be revoked by OASIS or
  its successors or assigns.
  -->

  <xsd:element name = "FilterResult">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element ref = "Message" minOccurs = "0" maxOccurs =
"unbounded"/>
        <xsd:element ref = "Notification" minOccurs = "0" maxOccurs =
"unbounded"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name = "GenericMessage" type = "GenericMessageType"/>
  <xsd:simpleType name = "synch.type">
    <xsd:restriction base = "xsd:string">
      <xsd:enumeration value = "synchronous"/>
      <xsd:enumeration value = "asynchronous"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name = "notification.type">
    <xsd:restriction base = "xsd:NMTOKEN">
      <xsd:enumeration value = "message"/>
      <xsd:enumeration value = "errorURL"/>
      <xsd:enumeration value = "errorApp"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:element name = "Message">
```

```

5385         <xsd:complexType>
5386             <xsd:complexContent>
5387                 <xsd:extension base = "GenericMessageType">
5388                     <xsd:attribute name = "type" use = "optional" type =
5389 "xsd:string"/>
5390                     <xsd:attribute name = "contentType" use = "optional"
5391 type = "xsd:string"/>
5392                 </xsd:extension>
5393             </xsd:complexContent>
5394         </xsd:complexType>
5395     </xsd:element>
5396     <xsd:complexType name = "GenericMessageType">
5397         <xsd:sequence>
5398             <xsd:any namespace = "##other" processContents = "lax" minOccurs = "0"
5399 maxOccurs = "unbounded"/>
5400         </xsd:sequence>
5401         <xsd:attribute name = "synchType" use = "required" type = "synch.type"/>
5402         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
5403         <xsd:attribute name = "serviceInstanceId" use = "optional" type =
5404 "xsd:string"/>
5405         <xsd:attribute name = "serviceName" use = "optional" type = "xsd:string"/>
5406         <xsd:attribute name = "reportingAction" use = "optional" type =
5407 "xsd:string"/>
5408         <xsd:anyAttribute namespace = "##any" processContents = "strict"/>
5409     </xsd:complexType>
5410     <xsd:element name = "Notification">
5411         <xsd:complexType>
5412             <xsd:complexContent>
5413                 <xsd:extension base = "GenericMessageType">
5414                     <xsd:attribute name = "notificationType" use =
5415 "required" type = "notification.type"/>
5416                 </xsd:extension>
5417             </xsd:complexContent>
5418         </xsd:complexType>
5419     </xsd:element>
5420 </xsd:schema>

```

5421

5422

## 5423 ebXML Specific Filter Result Schema

```

5424 <?xml version = "1.0" encoding = "UTF-8"?>
5425 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
5426 <xsd:schema xmlns = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
5427     targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
5428     xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
5429     xmlns:soap = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
5430     xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
5431     <xsd:import namespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
5432     schemaLocation =
5433 "file:///E:/ebXML_MS_20_Conformance_Testing_1.0/schemas/messagestore_mime.xsd"/>
5434     <xsd:import namespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
5435     schemaLocation =
5436 "file:///E:/ebXML MS 20 Conformance Testing 1.0/schemas/messagestore soap.xsd"/>
5437     <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->
5438
5439
5440     <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->
5441
5442
5443     <!--
5444     Copyright (C) The Organization for the Advancement of Structured Information Standards
5445     [OASIS]
5446     January 2002. All Rights Reserved.

```

5447 This document and translations of it may be copied and furnished to others, and  
5448 derivative works that comment on or otherwise explain it or assist in its implementation  
5449 may be prepared, copied, published and distributed, in whole or in part, without  
5450 restriction of any kind, provided that the above copyright notice and this paragraph are  
5451 included on all such copies and derivative works. However, this document itself may not  
5452 be modified in any way, such as by removing the copyright notice or references to OASIS,  
5453 except as needed for the purpose of developing OASIS specifications, in which case the  
5454 procedures for copyrights defined in the OASIS Intellectual Property Rights document  
5455 MUST be followed, or as required to translate it into languages other than English.  
5456 The limited permissions granted above are perpetual and will not be revoked by OASIS or  
5457 its successors or assigns.  
5458 -->

```

5459
5460     <xsd:element name = "FilterResult">
5461         <xsd:complexType>
5462             <xsd:choice>
5463                 <xsd:element ref = "Message" minOccurs = "0" maxOccurs =
5464 "unbounded"/>
5465                 <xsd:element ref = "Notification" minOccurs = "0" maxOccurs =
5466 "unbounded"/>
5467             </xsd:choice>
5468         </xsd:complexType>
5469     </xsd:element>
5470     <xsd:element name = "GenericMessage" type = "GenericMessageType"/>
5471     <xsd:simpleType name = "synch.type">
5472         <xsd:restriction base = "xsd:string">
5473             <xsd:enumeration value = "synchronous"/>
5474             <xsd:enumeration value = "asynchronous"/>
5475         </xsd:restriction>
5476     </xsd:simpleType>
5477     <xsd:simpleType name = "notification.type">
5478         <xsd:restriction base = "xsd:NMTOKEN">
5479             <xsd:enumeration value = "errURL"/>
5480             <xsd:enumeration value = "errorApp"/>
5481             <xsd:enumeration value = "message"/>
5482         </xsd:restriction>
5483     </xsd:simpleType>
5484     <xsd:element name = "Message">
5485         <xsd:complexType>
5486             <xsd:complexContent>
5487                 <xsd:extension base = "GenericMessageType">
5488                     <xsd:sequence>
5489                         <xsd:element ref = "mime:MessageContainer"/>
5490                     </xsd:sequence>
5491                     <xsd:attribute name = "type" use = "optional" type =
5492 "xsd:string"/>
5493                     <xsd:attribute name = "contentType" use = "optional"
5494 type = "xsd:string"/>
5495                 </xsd:extension>
5496             </xsd:complexContent>
5497         </xsd:complexType>
5498     </xsd:element>
5499     <xsd:complexType name = "GenericMessageType">
5500         <xsd:attribute name = "synchType" use = "required" type = "synch.type"/>
5501         <xsd:attribute name = "id" use = "required" type = "xsd:string"/>
5502         <xsd:attribute name = "serviceInstanceId" use = "optional" type =
5503 "xsd:string"/>
5504         <xsd:attribute name = "serviceName" use = "optional" type = "xsd:string"/>
5505         <xsd:attribute name = "reportingAction" use = "optional" type =
5506 "xsd:string"/>
5507         <xsd:anyAttribute namespace = "##any" processContents = "strict"/>
5508     </xsd:complexType>
5509     <xsd:element name = "Notification">
5510         <xsd:complexType>
5511             <xsd:complexContent>
5512                 <xsd:extension base = "GenericMessageType">
5513                     <xsd:sequence>
5514                         <xsd:element ref = "soap:Envelope"/>
5515                     </xsd:sequence>
5516                     <xsd:attribute name = "notificationType" use =
5517 "required" type = "notification.type"/>

```

5518  
5519  
5520  
5521  
5522

```
                </xsd:extension>  
            </xsd:complexContent>  
        </xsd:complexType>  
    </xsd:element>  
</xsd:schema>
```

5523

## Appendix E (Normative) The Test Report Schema

5524

5525

5526

5527

5528

5529

5530

5531

5532

5533

5534

5535

5536

5537

5538

5539

5540

5541

5542

5543

5544

5545

5546

5547

5548

5549

5550

5551

5552

5553

5554

5555

5556

5557

5558

5559

5560

5561

5562

5563

5564

5565

5566

5567

5568

5569

5570

5571

5572

5573

5574

5575

5576

5577

5578

5579

5580

5581

5582

5583

5584

5585

5586

5587

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests"
  xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
  xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
  xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
  version = "1.0"
  elementFormDefault = "unqualified"
  attributeFormDefault = "unqualified">
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting/poc/07_13_04/schemas/mime.xsd"/>
  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or
its successors or assigns.
-->

  <element name = "TestSuite">
    <complexType>
      <sequence>
        <element ref = "ebTest:MetaData"/>
        <element ref = "ebTest:ConfigurationGroup" maxOccurs =
"unbounded"/>
        <element ref = "ebTest:TestServiceConfigurator" minOccurs =
"0"/>
        <element ref = "ebTest:Message" minOccurs = "0" maxOccurs =
"unbounded"/>
        <element ref = "ebTest:TestCase" maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "configurationGroupRef" use = "required" type =
"anyURI"/>
    </complexType>
  </element>
  <element name = "MetaData">
    <complexType>
      <sequence>
        <element ref = "ebTest:Description"/>
        <element ref = "ebTest:Version"/>
      </sequence>
    </complexType>
  </element>
</schema>
```

```

5588         <element ref = "ebTest:Maintainer"/>
5589         <element ref = "ebTest:Location"/>
5590         <element ref = "ebTest:PublishDate"/>
5591         <element ref = "ebTest:Status"/>
5592     </sequence>
5593 </complexType>
5594 </element>
5595 <element name = "Description" type = "ebTest:non-empty-string"/>
5596 <element name = "Version" type = "ebTest:non-empty-string"/>
5597 <element name = "Maintainer" type = "ebTest:non-empty-string"/>
5598 <element name = "Location" type = "anyURI"/>
5599 <element name = "PublishDate" type = "ebTest:non-empty-string"/>
5600 <element name = "Status" type = "ebTest:non-empty-string"/>
5601 <element name = "TestCase">
5602     <complexType>
5603         <sequence>
5604             <element ref = "ebTest:ThreadGroup" minOccurs = "0"/>
5605             <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
5606 = "unbounded"/>
5607             <choice>
5608                 <element ref = "ebTest:Thread"/>
5609                 <element ref = "ebTest:ThreadRef"/>
5610                 <sequence maxOccurs = "unbounded">
5611                     <element ref = "ebTest:Split" maxOccurs =
5612 "unbounded"/>
5613                     <element ref = "ebTest:Join" maxOccurs =
5614 "unbounded"/>
5615                 </sequence>
5616             </choice>
5617         </sequence>
5618         <attribute name = "id" use = "required" type = "ID"/>
5619         <attribute name = "description" use = "required" type = "string"/>
5620         <attribute name = "author" use = "optional" type = "string"/>
5621         <attribute name = "version" use = "optional" type = "string"/>
5622         <attribute name = "requirementReferenceId" use = "required" type =
5623 "anyURI"/>
5624         <attribute name = "configurationGroupRef" use = "optional" type =
5625 "anyURI"/>
5626         <attribute name = "result" use = "required" type =
5627 "ebTest:result.type"/>
5628     </complexType>
5629 </element>
5630 <element name = "ConfigurationGroup">
5631     <complexType>
5632         <sequence>
5633             <element ref = "ebTest:Mode"/>
5634             <element ref = "ebTest:StepDuration"/>
5635             <element ref = "ebTest:Transport"/>
5636             <element ref = "ebTest:Envelope"/>
5637             <element ref = "ebTest:StoreAttachments"/>
5638             <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
5639 = "unbounded"/>
5640         </sequence>
5641         <attribute name = "id" use = "required" type = "ID"/>
5642     </complexType>
5643 </element>
5644 <element name = "CPAId" type = "ebTest:non-empty-string"/>
5645 <element name = "Mode" type = "ebTest:mode.type"/>
5646 <element name = "SenderParty" type = "anyURI"/>
5647 <element name = "ReceiverParty" type = "anyURI"/>
5648 <element name = "Service" type = "anyURI"/>
5649 <element name = "Action" type = "ebTest:non-empty-string"/>
5650 <element name = "StepDuration" type = "integer"/>
5651 <element name = "Transport" type = "ebTest:transport.type"/>
5652 <element name = "Envelope" type = "ebTest:non-empty-string"/>
5653 <simpleType name = "mode.type">
5654     <restriction base = "NMTOKEN">
5655         <enumeration value = "local-service"/>
5656         <enumeration value = "remote-service"/>
5657         <enumeration value = "connection"/>
5658     </restriction>

```

```

5659 </simpleType>
5660 <simpleType name = "mimeHeader.type">
5661   <restriction base = "NMTOKEN">
5662     <enumeration value = "MIMEMessageContent-Type"/>
5663     <enumeration value = "MIMEMessageStart"/>
5664     <enumeration value = "Content-Type"/>
5665     <enumeration value = "start"/>
5666     <enumeration value = "charset"/>
5667     <enumeration value = "type"/>
5668     <enumeration value = "wildcard"/>
5669   </restriction>
5670 </simpleType>
5671 <simpleType name = "content.type">
5672   <restriction base = "NMTOKEN">
5673     <enumeration value = "XML"/>
5674     <enumeration value = "date"/>
5675     <enumeration value = "URI"/>
5676     <enumeration value = "signature"/>
5677     <enumeration value = "XPointer"/>
5678   </restriction>
5679 </simpleType>
5680 <simpleType name = "method.type">
5681   <restriction base = "NMTOKEN">
5682     <enumeration value = "xpath"/>
5683     <enumeration value = "md5"/>
5684   </restriction>
5685 </simpleType>
5686 <simpleType name = "messageContext.type">
5687   <restriction base = "NMTOKEN">
5688     <enumeration value = "true"/>
5689     <enumeration value = "false"/>
5690   </restriction>
5691 </simpleType>
5692 <simpleType name = "requirement.type">
5693   <restriction base = "NMTOKEN">
5694     <enumeration value = "required"/>
5695     <enumeration value = "stronglyrecommended"/>
5696     <enumeration value = "recommended"/>
5697     <enumeration value = "optional"/>
5698   </restriction>
5699 </simpleType>
5700 <simpleType name = "non-empty-string">
5701   <restriction base = "string">
5702     <minLength value = "1"/>
5703   </restriction>
5704 </simpleType>
5705 <simpleType name = "configAction.type">
5706   <restriction base = "NMTOKEN">
5707     <enumeration value = "query"/>
5708     <enumeration value = "replace"/>
5709   </restriction>
5710 </simpleType>
5711 <simpleType name = "action.type">
5712   <restriction base = "NMTOKEN">
5713     <enumeration value = "reset"/>
5714     <enumeration value = "modify"/>
5715   </restriction>
5716 </simpleType>
5717 <simpleType name = "configItem.type">
5718   <restriction base = "NMTOKEN"/>
5719 </simpleType>
5720 <simpleType name = "parameter.type">
5721   <restriction base = "NMTOKEN">
5722     <enumeration value = "string"/>
5723     <enumeration value = "parameter"/>
5724   </restriction>
5725 </simpleType>
5726 <simpleType name = "connectivePredicate.type">
5727   <restriction base = "NMTOKEN">
5728     <enumeration value = "and"/>
5729     <enumeration value = "or"/>

```



```

5730         </restriction>
5731     </simpleType>
5732     <simpleType name = "thread.type">
5733         <restriction base = "NMTOKEN">
5734             <enumeration value = "synchronous"/>
5735             <enumeration value = "asynchronous"/>
5736         </restriction>
5737     </simpleType>
5738     <simpleType name = "matchResult.type">
5739         <restriction base = "NMTOKEN">
5740             <enumeration value = "pass"/>
5741             <enumeration value = "fail"/>
5742         </restriction>
5743     </simpleType>
5744     <simpleType name = "if.type">
5745         <restriction base = "NMTOKEN">
5746             <enumeration value = "andif"/>
5747             <enumeration value = "orif"/>
5748         </restriction>
5749     </simpleType>
5750     <simpleType name = "split.type">
5751         <restriction base = "NMTOKEN">
5752             <enumeration value = "andsplit"/>
5753             <enumeration value = "orsplit"/>
5754         </restriction>
5755     </simpleType>
5756     <simpleType name = "join.type">
5757         <restriction base = "NMTOKEN">
5758             <enumeration value = "andjoin"/>
5759             <enumeration value = "orjoin"/>
5760         </restriction>
5761     </simpleType>
5762     <simpleType name = "serviceMode.type">
5763         <restriction base = "NMTOKEN">
5764             <enumeration value = "loop"/>
5765             <enumeration value = "local-reporting"/>
5766             <enumeration value = "remote-reporting"/>
5767         </restriction>
5768     </simpleType>
5769     <simpleType name = "time.type">
5770         <restriction base = "NMTOKEN">
5771             <enumeration value = "timeToAcknowledgeReceipt"/>
5772             <enumeration value = "timeToAcknowledgeAcceptance"/>
5773             <enumeration value = "timeToPerform"/>
5774             <enumeration value = "other"/>
5775         </restriction>
5776     </simpleType>
5777     <simpleType name = "operator.type">
5778         <restriction base = "NMTOKEN">
5779             <enumeration value = "equal"/>
5780             <enumeration value = "lessThan1"/>
5781             <enumeration value = "lessThanOrEqual"/>
5782             <enumeration value = "greaterThan"/>
5783             <enumeration value = "greaterThanOrEqual"/>
5784         </restriction>
5785     </simpleType>
5786     <simpleType name = "assertionExit.type">
5787         <restriction base = "NMTOKEN">
5788             <enumeration value = "pass"/>
5789             <enumeration value = "fail"/>
5790             <enumeration value = "undetermined"/>
5791         </restriction>
5792     </simpleType>
5793     <simpleType name = "preconditionExit.type">
5794         <restriction base = "NMTOKEN">
5795             <enumeration value = "undetermined"/>
5796         </restriction>
5797     </simpleType>
5798     <simpleType name = "scope.type">
5799         <restriction base = "NMTOKEN">
5800             <enumeration value = "self"/>

```

```

5801         <enumeration value = "selfAndDescendents"/>
5802     </restriction>
5803 </simpleType>
5804 <simpleType name = "transport.type">
5805     <restriction base = "NMTOKEN">
5806         <enumeration value = "FTP"/>
5807         <enumeration value = "SMTP"/>
5808         <enumeration value = "HTTP"/>
5809     </restriction>
5810 </simpleType>
5811 <simpleType name = "result.type">
5812     <restriction base = "NMTOKEN">
5813         <enumeration value = "pass"/>
5814         <enumeration value = "fail"/>
5815         <enumeration value = "undetermined"/>
5816     </restriction>
5817 </simpleType>
5818 <simpleType name = "exception.type">
5819     <restriction base = "NMTOKEN">
5820         <enumeration value = "undetermined"/>
5821     </restriction>
5822 </simpleType>
5823 <element name = "MessageExpression">
5824     <complexType>
5825         <sequence>
5826             <element ref = "ebTest:ErrorMessage"/>
5827         </sequence>
5828     </complexType>
5829 </element>
5830 <element name = "ErrorMessage" type = "ebTest:non-empty-string"/>
5831 <element name = "PutMessage">
5832     <complexType>
5833         <sequence>
5834             <element ref = "ebTest:SetPart" maxOccurs = "unbounded"/>
5835         </sequence>
5836         <attribute name = "description" use = "required" type = "string"/>
5837         <attribute name = "repeatWithSameContext" use = "optional" type =
5838 "integer"/>
5839         <attribute name = "repeatWithNewContext" use = "optional" type =
5840 "integer"/>
5841     </complexType>
5842 </element>
5843 <element name = "GetMessage">
5844     <complexType>
5845         <sequence maxOccurs = "unbounded">
5846             <element ref = "ebTest:Filter"/>
5847             <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
5848 maxOccurs = "unbounded"/>
5849         </sequence>
5850         <attribute name = "description" use = "required" type = "string"/>
5851         <attribute name = "mask" use = "optional" type = "boolean"/>
5852     </complexType>
5853 </element>
5854 <element name = "Filter">
5855     <complexType>
5856         <simpleContent>
5857             <extension base = "ebTest:non-empty-string">
5858                 <attribute name = "stepDuration" use = "optional" type
5859 = "integer"/>
5860                 <attribute name = "result" use = "required" type =
5861 "ebTest:result.type"/>
5862             </extension>
5863         </simpleContent>
5864     </complexType>
5865 </element>
5866 <element name = "SetPart">
5867     <complexType>
5868         <sequence>
5869             <element ref = "ebTest:Header" minOccurs = "0" maxOccurs =
5870 "unbounded"/>
5871         </sequence>

```

```

5872         <element ref = "ebTest:Declaration"/>
5873         <element ref = "ebTest:FileURI"/>
5874         <element ref = "ebTest:MessageRef"/>
5875     </choice>
5876     <element ref = "ebTest:Mutator" minOccurs = "0"/>
5877     <element ref = "ebTest:DSign" minOccurs = "0"/>
5878 </sequence>
5879 <attribute name = "description" use = "optional" type = "string"/>
5880 <attribute name = "result" use = "required" type =
5881 "ebTest:result.type"/>
5882 </complexType>
5883 </element>
5884 <element name = "TestAssertion">
5885     <complexType>
5886         <sequence>
5887             <choice>
5888                 <element ref = "ebTest:VerifyContent"/>
5889                 <element ref = "ebTest:ValidateContent"/>
5890                 <element ref = "ebTest:VerifyTimeDifference"/>
5891             </choice>
5892             <element name = "WhenTrue" minOccurs = "0">
5893                 <complexType>
5894                     <choice>
5895                         <element ref = "ebTest:Continue"/>
5896                         <element ref = "ebTest:ThreadRef"/>
5897                         <element ref = "ebTest:Split"/>
5898                         <element name = "Exit" type =
5899 "ebTest:assertionExit.type"/>
5900                     </choice>
5901                 </complexType>
5902             </element>
5903             <element name = "WhenFalse" minOccurs = "0">
5904                 <complexType>
5905                     <choice>
5906                         <element ref = "ebTest:Continue"/>
5907                         <element ref = "ebTest:ThreadRef"/>
5908                         <element ref = "ebTest:Split"/>
5909                         <element name = "Exit" type =
5910 "ebTest:assertionExit.type"/>
5911                     </choice>
5912                 </complexType>
5913             </element>
5914         </sequence>
5915         <attribute name = "description" use = "required" type = "string"/>
5916         <attribute name = "result" use = "required" type =
5917 "ebTest:result.type"/>
5918     </complexType>
5919 </element>
5920 <element name = "MimeHeader" type = "ebTest:mimeHeader.type"/>
5921 <element name = "MimeHeaderValue" type = "ebTest:non-empty-string"/>
5922 <element name = "Content-Location" type = "ebTest:non-empty-string"/>
5923 <element name = "Index" type = "integer"/>
5924 <element name = "FileURI" type = "anyURI"/>
5925 <element name = "PayloadRef" type = "ebTest:non-empty-string"/>
5926 <element name = "Signature" type = "base64Binary"/>
5927 <element name = "Content-ID" type = "ebTest:non-empty-string"/>
5928 <element name = "MessageDeclaration">
5929     <complexType>
5930         <sequence>
5931             <any namespace = "##other" processContents = "lax" minOccurs =
5932 "0" maxOccurs = "unbounded"/>
5933         </sequence>
5934     </complexType>
5935 </element>
5936 <element name = "ValidateContent">
5937     <complexType>
5938         <simpleContent>
5939             <extension base = "ebTest:non-empty-string">
5940                 <attribute name = "contentType" use = "required" type =
5941 "ebTest:content.type"/>

```

```

5942                                     <attribute name = "schemaLocation" use = "optional"
5943 type = "anyURI"/>
5944                                     <attribute name = "result" use = "required" type =
5945 "ebTest:result.type"/>
5946                                     </extension>
5947                                     </simpleContent>
5948                                     </complexType>
5949 </element>
5950 <element name = "VerifyContent">
5951   <complexType>
5952     <simpleContent>
5953       <extension base = "ebTest:non-empty-string">
5954         <attribute name = "result" use = "required" type =
5955 "ebTest:result.type"/>
5956       </extension>
5957     </simpleContent>
5958   </complexType>
5959 </element>
5960 <element name = "Message">
5961   <complexType>
5962     <sequence>
5963       <any namespace = "##other" processContents = "lax" minOccurs =
5964 "0" maxOccurs = "unbounded"/>
5965     </sequence>
5966     <attribute name = "id" use = "required" type = "ID"/>
5967   </complexType>
5968 </element>
5969 <element name = "SetParameter">
5970   <complexType>
5971     <sequence>
5972       <element name = "Name" type = "ebTest:non-empty-string"/>
5973       <choice>
5974         <element name = "Value" type = "ebTest:non-empty-
5975 string"/>
5976         <element name = "ParameterRef" type = "ebTest:non-
5977 empty-string"/>
5978       </choice>
5979     </sequence>
5980     <attribute name = "scope" use = "optional" type =
5981 "ebTest:scope.type"/>
5982     <attribute name = "result" use = "optional" type =
5983 "ebTest:exception.type"/>
5984   </complexType>
5985 </element>
5986 <element name = "Mutator">
5987   <complexType>
5988     <choice>
5989       <element ref = "ebTest:XSL"/>
5990       <element ref = "ebTest:XUpdate"/>
5991     </choice>
5992     <attribute name = "result" use = "required" type =
5993 "ebTest:result.type"/>
5994   </complexType>
5995 </element>
5996 <element name = "XSL" type = "anyURI"/>
5997 <element name = "XUpdate" type = "anyURI"/>
5998 <element name = "BooleanClause">
5999   <complexType>
6000     <attribute name = "booleanPredicate" use = "required" type =
6001 "boolean"/>
6002   </complexType>
6003 </element>
6004 <element name = "DSign">
6005   <complexType>
6006     <sequence>
6007       <element ref = "ds:Signature"/>
6008     </sequence>
6009     <attribute name = "result" use = "required" type =
6010 "ebTest:result.type"/>
6011   </complexType>
6012 </element>

```

```

6013     <element name = "Declaration">
6014         <complexType>
6015             <sequence>
6016                 <any namespace = "##other" processContents = "lax" minOccurs =
6017 "0" maxOccurs = "unbounded"/>
6018             </sequence>
6019         </complexType>
6020     </element>
6021     <element name = "Thread">
6022         <complexType>
6023             <choice maxOccurs = "unbounded">
6024                 <element ref = "ebTest:SetParameter"/>
6025                 <element ref = "ebTest:PutMessage"/>
6026                 <element ref = "ebTest:Initiator"/>
6027                 <element ref = "ebTest:GetMessage"/>
6028                 <element ref = "ebTest:TestAssertion"/>
6029                 <element ref = "ebTest:ThreadRef"/>
6030                 <element ref = "ebTest:Split"/>
6031                 <element ref = "ebTest:Join"/>
6032                 <element ref = "ebTest:Sleep"/>
6033             </choice>
6034             <attribute name = "name" use = "required" type = "ID"/>
6035             <attribute name = "description" use = "optional" type = "string"/>
6036         </complexType>
6037     </element>
6038     <element name = "ThreadRef">
6039         <complexType>
6040             <attribute name = "nameRef" use = "required" type = "IDREF"/>
6041             <attribute name = "configurationGroupRef" use = "optional" type =
6042 "anyURI"/>
6043             <attribute name = "loop" use = "optional" type = "integer"/>
6044             <attribute name = "instanceId" use = "optional" type = "string"/>
6045             <attribute name = "result" use = "required" type =
6046 "ebTest:result.type"/>
6047         </complexType>
6048     </element>
6049     <element name = "Pass">
6050         <complexType/>
6051     </element>
6052     <element name = "Fail">
6053         <complexType/>
6054     </element>
6055     <element name = "ThreadGroup">
6056         <complexType>
6057             <sequence>
6058                 <element ref = "ebTest:Thread" maxOccurs = "unbounded"/>
6059             </sequence>
6060         </complexType>
6061     </element>
6062     <element name = "Sleep" type = "integer"/>
6063     <element name = "Split">
6064         <complexType>
6065             <sequence maxOccurs = "unbounded">
6066                 <element ref = "ebTest:ThreadRef"/>
6067             </sequence>
6068         </complexType>
6069     </element>
6070     <element name = "Join">
6071         <complexType>
6072             <sequence maxOccurs = "unbounded">
6073                 <element ref = "ebTest:ThreadRef"/>
6074             </sequence>
6075             <attribute name = "joinType" use = "optional" type =
6076 "ebTest:join.type"/>
6077         </complexType>
6078     </element>
6079     <element name = "Initiator">
6080         <complexType>
6081             <sequence>
6082                 <choice>
6083                     <element ref = "ebTest:Declaration"/>

```

```

6084         <element ref = "ebTest:FileURI"/>
6085         <element ref = "ebTest:MessageRef"/>
6086     </choice>
6087     <element ref = "ebTest:Mutator" minOccurs = "0"/>
6088     <element ref = "ebTest:DSign" minOccurs = "0"/>
6089 </sequence>
6090 <attribute name = "description" use = "required" type = "string"/>
6091 <attribute name = "result" use = "required" type =
6092 "ebTest:result.type"/>
6093 </complexType>
6094 </element>
6095 <element name = "TestServiceConfigurator">
6096     <complexType>
6097         <sequence>
6098             <element ref = "ebTest:ServiceMode"/>
6099             <element ref = "ebTest:ResponseURL"/>
6100             <element ref = "ebTest:NotificationURL"/>
6101             <element ref = "ebTest:PayloadDigests" minOccurs = "0"/>
6102         </sequence>
6103     </complexType>
6104 </element>
6105 <element name = "MessageRef" type = "IDREF"/>
6106 <element name = "ConfigurationItem">
6107     <complexType>
6108         <sequence>
6109             <element name = "Name" type = "ebTest:non-empty-string"/>
6110             <element name = "Value" type = "ebTest:non-empty-string"/>
6111         </sequence>
6112     </complexType>
6113 </element>
6114 <element name = "ErrorURL" type = "anyURI"/>
6115 <element name = "NotificationURL" type = "anyURI"/>
6116 <element name = "SetXPathParameter">
6117     <complexType>
6118         <sequence>
6119             <element name = "Name" type = "ebTest:non-empty-string"/>
6120             <element name = "Expression" type = "ebTest:non-empty-
6121 string"/>
6122         </sequence>
6123         <attribute name = "scope" use = "optional" type =
6124 "ebTest:scope.type"/>
6125         <attribute name = "result" use = "required" type =
6126 "ebTest:result.type"/>
6127     </complexType>
6128 </element>
6129 <element name = "ResponseURL" type = "anyURI"/>
6130 <element name = "StoreAttachments" type = "boolean"/>
6131 <element name = "OperationMode" type = "string"/>
6132 <element name = "PayloadDigests">
6133     <complexType>
6134         <sequence>
6135             <element name = "Payload" maxOccurs = "unbounded">
6136                 <complexType>
6137                     <sequence>
6138                         <element name = "Id" type = "anyURI"/>
6139                         <element name = "Digest" type =
6140 "base64Binary"/>
6141                     </sequence>
6142                 </complexType>
6143             </element>
6144         </sequence>
6145     </complexType>
6146 </element>
6147 <element name = "ServiceMode" type = "ebTest:serviceMode.type"/>
6148 <element name = "Transaction">
6149     <complexType>
6150         <sequence maxOccurs = "unbounded">
6151             <choice maxOccurs = "unbounded">
6152                 <element ref = "ebTest:PutMessage"/>
6153                 <element ref = "ebTest:Initiator"/>
6154             </choice>

```

```

6155         <element ref = "ebTest:GetMessage" minOccurs = "0" maxOccurs =
6156 "unbounded"/>
6157     </sequence>
6158     <attribute name = "timeToPerform" use = "optional" type = "duration"/>
6159 </complexType>
6160 </element>
6161 <element name = "VerifyTimeDifference">
6162     <complexType>
6163         <sequence>
6164             <element ref = "ebTest:ParamName"/>
6165             <element ref = "ebTest:ParamName"/>
6166             <element ref = "ebTest:Operator"/>
6167             <element ref = "ebTest:Difference"/>
6168         </sequence>
6169         <attribute name = "result" use = "required" type =
6170 "ebTest:result.type"/>
6171     </complexType>
6172 </element>
6173 <element name = "Difference" type = "duration"/>
6174 <element name = "Operator" type = "ebTest:operator.type"/>
6175 <element name = "XPathExpression" type = "ebTest:non-empty-string"/>
6176 <element name = "Continue">
6177     <complexType/>
6178 </element>
6179 <element name = "ParamName" type = "ebTest:non-empty-string"/>
6180 <element name = "VerifyTimeToPerform">
6181     <complexType>
6182         <sequence>
6183             <element ref = "ebTest:ThreadName" maxOccurs = "unbounded"/>
6184         </sequence>
6185         <attribute name = "maxTime" use = "required" type = "duration"/>
6186     </complexType>
6187 </element>
6188 <element name = "ThreadName" type = "IDREF"/>
6189 <element name = "Header">
6190     <complexType>
6191         <sequence>
6192             <element ref = "ebTest:Name"/>
6193             <element ref = "ebTest:Value"/>
6194         </sequence>
6195     </complexType>
6196 </element>
6197 <element name = "Name" type = "ebTest:non-empty-string"/>
6198 <element name = "Value" type = "ebTest:non-empty-string"/>
6199 </schema>

```

---

## Appendix F (Normative) ebXML Test Service Message Schema

6200

6201

6202

6203

6204

6205

6206

6207

6208

6209

6210

6211

6212

6213

6214

6215

6216

6217

6218

6219

6220

6221

6222

6223

6224

6225

6226

6227

6228

6229

6230

6231

6232

6233

6234

6235

6236

6237

6238

6239

6240

6241

6242

6243

6244

6245

6246

6247

6248

6249

6250

6251

6252

6253

6254

6255

6256

6257

6258

6259

6260

6261

6262

6263

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
  targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
  xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
  xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
  xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <import namespace = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
header-2_0.xsd" schemaLocation = "http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-2_0.xsd"/>
  <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or
its successors or assigns.
-->

  <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2002. All Rights Reserved.
This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its implementation
may be prepared, copied, published and distributed, in whole or in part, without
restriction of any kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this document itself may not
be modified in any way, such as by removing the copyright notice or references to OASIS,
except as needed for the purpose of developing OASIS specifications, in which case the
procedures for copyrights defined in the OASIS Intellectual Property Rights document
MUST be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by OASIS or
its successors or assigns.
-->

  <element name = "InitiatorRequest">
    <complexType>
      <sequence>
        <element ref = "tns:SetPart" maxOccurs = "unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name = "InitiatorResponse">
    <complexType>
      <sequence>
        <element ref = "tns:Success"/>
      </sequence>
    </complexType>
  </element>
</schema>
```



```

6264         </sequence>
6265     </complexType>
6266 </element>
6267 <element name = "NotificationResponse">
6268     <complexType>
6269         <sequence>
6270             <element ref = "tns:Success"/>
6271         </sequence>
6272     </complexType>
6273 </element>
6274 <complexType name = "GenericMessageType">
6275     <attribute name = "synchType" use = "required" type = "tns:synch.type"/>
6276     <attribute name = "id" use = "required" type = "string"/>
6277     <attribute name = "serviceInstanceId" use = "optional" type = "string"/>
6278     <attribute name = "serviceName" use = "optional" type = "string"/>
6279     <attribute name = "reportingAction" use = "optional" type = "string"/>
6280     <anyAttribute namespace = "##any" processContents = "strict"/>
6281 </complexType>
6282 <element name = "NotificationRequest">
6283     <complexType>
6284         <complexContent>
6285             <extension base = "tns:GenericMessageType">
6286                 <sequence>
6287                     <element ref = "tns:Part" maxOccurs =
6288 "unbounded"/>
6289                 </sequence>
6290                 <attribute name = "notificationType" use = "required"
6291 type = "tns:notification.type"/>
6292             </extension>
6293         </complexContent>
6294     </complexType>
6295 </element>
6296 <element name = "TestServiceConfiguratorRequest">
6297     <complexType>
6298         <sequence>
6299             <element name = "ServiceMode" type = "tns:serviceMode.type"/>
6300             <element ref = "tns:ResponseURL"/>
6301             <element ref = "tns:NotificationURL"/>
6302             <element ref = "tns:PayloadDigests" minOccurs = "0"/>
6303         </sequence>
6304     </complexType>
6305 </element>
6306 <element name = "ResponseURL" type = "anyURI"/>
6307 <element name = "TestServiceConfiguratorResponse">
6308     <complexType>
6309         <sequence>
6310             <element ref = "tns:Success"/>
6311         </sequence>
6312     </complexType>
6313 </element>
6314 <element name = "Status" type = "boolean"/>
6315 <element name = "Mode" type = "tns:non-empty-string"/>
6316 <element name = "MessageId" type = "tns:non-empty-string"/>
6317 <simpleType name = "non-empty-string">
6318     <restriction base = "string">
6319         <minLength value = "1"/>
6320     </restriction>
6321 </simpleType>
6322 <simpleType name = "configAction.type">
6323     <restriction base = "NMTOKEN">
6324         <enumeration value = "query"/>
6325         <enumeration value = "replace"/>
6326     </restriction>
6327 </simpleType>
6328 <simpleType name = "result.type">
6329     <restriction base = "NMTOKEN">
6330         <enumeration value = "pass"/>
6331         <enumeration value = "fail"/>
6332     </restriction>
6333 </simpleType>
6334 <simpleType name = "operationMode.type">

```

```

6335         <restriction base = "NMTOKEN">
6336             <enumeration value = "reporting"/>
6337             <enumeration value = "loop"/>
6338         </restriction>
6339     </simpleType>
6340     <simpleType name = "parameter.type">
6341         <restriction base = "NMTOKEN">
6342             <enumeration value = "parameter"/>
6343             <enumeration value = "string"/>
6344         </restriction>
6345     </simpleType>
6346     <simpleType name = "serviceMode.type">
6347         <restriction base = "NMTOKEN">
6348             <enumeration value = "remote-reporting"/>
6349             <enumeration value = "local-reporting"/>
6350             <enumeration value = "loop"/>
6351         </restriction>
6352     </simpleType>
6353     <simpleType name = "parameter.type">
6354         <restriction base = "NMTOKEN">
6355             <enumeration value = "string"/>
6356             <enumeration value = "namespace"/>
6357         </restriction>
6358     </simpleType>
6359     <simpleType name = "notification.type">
6360         <restriction base = "NMTOKEN">
6361             <enumeration value = "errURL"/>
6362             <enumeration value = "errorApp"/>
6363             <enumeration value = "message"/>
6364         </restriction>
6365     </simpleType>
6366     <simpleType name = "synch.type">
6367         <restriction base = "string">
6368             <enumeration value = "synchronous"/>
6369             <enumeration value = "asynchronous"/>
6370         </restriction>
6371     </simpleType>
6372     <element name = "OperationMode" type = "tns:operationMode.type"/>
6373     <element name = "NotificationURL" type = "anyURI"/>
6374     <element name = "DSign">
6375         <complexType>
6376             <sequence>
6377                 <element ref = "ds:Signature"/>
6378             </sequence>
6379         </complexType>
6380     </element>
6381     <element name = "PayloadDigests">
6382         <complexType>
6383             <sequence>
6384                 <element ref = "tns:Payload" maxOccurs = "unbounded"/>
6385             </sequence>
6386         </complexType>
6387     </element>
6388     <element name = "Id" type = "string"/>
6389     <element name = "Digest" type = "string"/>
6390     <element name = "SetPart">
6391         <complexType>
6392             <sequence>
6393                 <element ref = "tns:Header" minOccurs = "0" maxOccurs =
6394 "unbounded"/>
6395                 <element ref = "tns:Declaration" minOccurs = "0"/>
6396                 <element ref = "tns:DSign" minOccurs = "0"/>
6397             </sequence>
6398         </complexType>
6399     </element>
6400     <element name = "Header">
6401         <complexType>
6402             <sequence>
6403                 <element ref = "tns:Name"/>
6404                 <element ref = "tns:Value"/>
6405             </sequence>

```

```
6406         </complexType>
6407     </element>
6408     <element name = "Declaration">
6409         <complexType>
6410             <sequence>
6411                 <any namespace = "##other" processContents = "lax" minOccurs =
6412                 "0" maxOccurs = "unbounded"/>
6413             </sequence>
6414         </complexType>
6415     </element>
6416     <element name = "Payload">
6417         <complexType>
6418             <sequence>
6419                 <element ref = "tns:Id"/>
6420                 <element ref = "tns:Success"/>
6421             </sequence>
6422         </complexType>
6423     </element>
6424     <element name = "Name" type = "string"/>
6425     <element name = "Value" type = "string"/>
6426     <element name = "Part">
6427         <complexType>
6428             <sequence>
6429                 <element ref = "tns:Header" minOccurs = "0" maxOccurs =
6430                 "unbounded"/>
6431                 <element ref = "tns:Content" minOccurs = "0"/>
6432             </sequence>
6433         </complexType>
6434     </element>
6435     <element name = "Content">
6436         <complexType>
6437             <sequence>
6438                 <any namespace = "##other" processContents = "lax" minOccurs =
6439                 "0" maxOccurs = "unbounded"/>
6440             </sequence>
6441         </complexType>
6442     </element>
6443     <element name = "PayloadVerifyResponse">
6444         <complexType>
6445             <sequence>
6446                 <element ref = "tns:Payload" minOccurs = "0" maxOccurs =
6447                 "unbounded"/>
6448             </sequence>
6449         </complexType>
6450     </element>
6451     <element name = "Success" type = "boolean"/>
6452 </schema>
```

6453  
6454  
6455  
6456  
6457  
6458  
6459  
6460  
6461  
6462  
6463  
6464  
6465

6466  
6467  
6468  
6469  
6470  
6471  
6472  
6473  
6474  
6475  
6476  
6477  
6478  
6479



6481

# Appendix G WSDL Definitions for Test Service

6482

6483  
6484

6485

6486

6487 WSDL Definition of the Test Service initiator SOAP method

6488

```

6489 <?xml version="1.0" encoding="UTF-8"?>
6490 <!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mike Kass (Personal) -
6491 ->
6492 <wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
6493 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
6494 iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6495 xmlns:xsd="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6496 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
6497 targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
6498 name="RegistryService">
6499   <wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6500 location="schemas/TestServiceMessages.xsd"/>
6501   <wsdl:message name="InitiatorRequest">
6502     <wsdl:part name="InitiatorRequest" element="xsd:InitiatorRequest"/>
6503   </wsdl:message>
6504   <wsdl:message name="InitiatorResponse">
6505     <wsdl:part name="InitiatorResponse" element="xsd:InitiatorResponse"/>
6506   </wsdl:message>
6507   <wsdl:portType name="SendPortType">
6508     <documentation>Maps to the Initiator interface of Test Framework
6509 spec.</documentation>
6510     <wsdl:operation name="initiator">
6511       <wsdl:input message="tns:InitiatorRequest"/>
6512       <wsdl:output message="tns:InitiatorResponse"/>
6513     </wsdl:operation>
6514   </wsdl:portType>
6515   <wsdl:binding name="InitiatorSOAPBinding" type="tns:SendPortType">
6516     <soap:binding style="document"
6517 transport="http://schemas.xmlsoap.org/soap/http"/>
6518     <wsdl:operation name="initiator">
6519       <soap:operation
6520 soapAction="uri:oasis:ebxml:iic:testservice:Send:initiator"/>
6521       <wsdl:input>
6522         <mime:multipartRelated>
6523           <mime:part>
6524             <soap:body use="literal"/>
6525           </mime:part>
6526         </mime:multipartRelated>
6527       </wsdl:input>
6528       <wsdl:output>
6529         <mime:multipartRelated>
6530           <mime:part>
6531             <soap:body use="literal"/>
6532           </mime:part>
6533         </mime:multipartRelated>
6534       </wsdl:output>
6535     </wsdl:operation>
6536   </wsdl:binding>
6537   <wsdl:service name="TestService">
6538     <documentation>The QueryManager service of OASIS ebXML Test Framework version
6539 1.1</documentation>
6540     <wsdl:port name="InitiatorSOAPBinding" binding="tns:InitiatorSOAPBinding">
6541       <soap:address
6542 location="http://your_URL_to_your_ConfigurationService"/>

```

```
6543         </wsdl:port>
6544     </wsdl:service>
6545     <documentation>This is the the normative abstract WSDL service definition for the
6546     OASIS ebXML Test Service</documentation>
6547 </wsdl:definitions>
```

6548  
6549  
6550  
6551  
6552  
6553  
6554  
6555

## WSDL Definition of the Test Service configure method

```
6556 <?xml version="1.0" encoding="UTF-8"?>
6557 <!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mike Kass (Personal) -
6558 ->
6559 <wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
6560 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
6561 iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6562 xmlns:xsd="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6563 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
6564 targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
6565 name="RegistryService">
6566     <wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6567     location="schemas/TestServiceMessages.xsd"/>
6568     <wsdl:message name="TestServiceConfiguratorRequest">
6569         <wsdl:part name="TestServiceConfiguratorRequest"
6570         element="xsd:TestServiceConfiguratorRequest"/>
6571     </wsdl:message>
6572     <wsdl:message name="TestServiceConfiguratorResponse">
6573         <wsdl:part name="TestServiceConfiguratorResponse"
6574         element="xsd:TestServiceConfiguratorResponse"/>
6575     </wsdl:message>
6576     <wsdl:portType name="ConfigurationPortType">
6577         <documentation>Maps to the Configurator interface of Test Framework
6578         spec.</documentation>
6579         <wsdl:operation name="configurator">
6580             <wsdl:input message="tns:TestServiceConfiguratorRequest"/>
6581             <wsdl:output message="tns:TestServiceConfiguratorResponse"/>
6582         </wsdl:operation>
6583     </wsdl:portType>
6584     <wsdl:binding name="ConfiguratorSOAPBinding" type="tns:ConfigurationPortType">
6585         <soap:binding style="document"
6586         transport="http://schemas.xmlsoap.org/soap/http"/>
6587         <wsdl:operation name="configurator">
6588             <soap:operation
6589             soapAction="uri:oasis:ebxml:iic:testservice:Configuration:configurator"/>
6590             <wsdl:input>
6591                 <mime:multipartRelated>
6592                     <mime:part>
6593                         <soap:body/>
6594                     </mime:part>
6595                 </mime:multipartRelated>
6596             </wsdl:input>
6597             <wsdl:output>
6598                 <mime:multipartRelated>
6599                     <mime:part>
6600                         <soap:body/>
6601                     </mime:part>
6602                 </mime:multipartRelated>
6603             </wsdl:output>
6604         </wsdl:operation>
6605     </wsdl:binding>
6606     <wsdl:service name="TestService">
```

```
6607         <documentation>The QueryManager service of OASIS ebXML Test Framework version
6608 1.1</documentation>
6609         <wsdl:port name="ConfiguratorSOAPBinding"
6610 binding="tns:ConfiguratorSOAPBinding">
6611             <soap:address
6612 location="http://your URL to your ConfigurationService"/>
6613         </wsdl:port>
6614     </wsdl:service>
6615     <documentation>This is the the normative abstract WSDL service definition for the
6616 OASIS ebXML Test Service</documentation>
6617 </wsdl:definitions>
```

6618

6619

6620

6621

6622

### 6623 WSDL Definition of the Test Driver notify method

6624

```
6625 <?xml version="1.0" encoding="UTF-8"?>
6626 <!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Mike Kass (Personal) -
6627 -->
6628 <wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
6629 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
6630 iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6631 xmlns:xsd="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6632 targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
6633 name="RegistryService">
6634     <wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
6635 location="schemas/TestServiceMessages.xsd"/>
6636     <wsdl:message name="NotificationRequest">
6637         <wsdl:part name="NotificationRequest" element="xsd:NotificationRequest"/>
6638     </wsdl:message>
6639     <wsdl:message name="NotificationResponse">
6640         <wsdl:part name="NotificationResponse" element="xsd:NotificationResponse"/>
6641     </wsdl:message>
6642     <wsdl:portType name="NotificationPortType">
6643         <documentation>Maps to the Notification interface of Test Framework
6644 spec.</documentation>
6645         <wsdl:operation name="Notify">
6646             <wsdl:input message="tns:NotificationRequest"/>
6647             <wsdl:output message="tns:NotificationResponse"/>
6648         </wsdl:operation>
6649     </wsdl:portType>
6650     <wsdl:binding name="NotificationSOAPBinding" type="tns:NotificationPortType">
6651         <soap:binding style="document"
6652 transport="http://schemas.xmlsoap.org/soap/http"/>
6653         <wsdl:operation name="Notify">
6654             <soap:operation
6655 soapAction="uri:oasis:ebxml:iic:testservice:Receive:Notification"/>
6656             <wsdl:input>
6657                 <mime:multipartRelated>
6658                     <mime:part>
6659                         <soap:body use="literal"/>
6660                     </mime:part>
6661                 </mime:multipartRelated>
6662             </wsdl:input>
6663             <wsdl:output>
6664                 <mime:multipartRelated>
6665                     <mime:part>
6666                         <soap:body use="literal"/>
6667                     </mime:part>
6668                 </mime:multipartRelated>
6669             </wsdl:output>
6670         </wsdl:operation>
```



```
6671     </wsdl:binding>
6672     <wsdl:service name="TestDriverReceiveService">
6673         <documentation>The Receive service of OASIS ebXML Test Framework version
6674 1.1</documentation>
6675         <wsdl:port name="NotifySOAPBinding" binding="tns:NotificationSOAPBinding">
6676             <soap:address location="http://your URL to your ReceiveService"/>
6677         </wsdl:port>
6678     </wsdl:service>
6679     <documentation>This is the the normative abstract WSDL service definition for the
6680 OASIS ebXML Test Service</documentation>
6681 </wsdl:definitions>
```

6682

## Appendix H Terminology

6683 Several terms used in this specification are borrowed from the Conformance Glossary (OASIS,  
 6684 [ConfGlossary]) and also from the Standards and Conformance Testing Group at NIST.  
 6685 [ConfCertModelNIST]. They are not reported in this glossary, which only reflects (1) terms that are  
 6686 believed to be specific to – and introduced by - the ebXML Test Framework, or (2) terms that have a well  
 6687 understood meaning in testing literature (see above references) and may have additional properties in the  
 6688 context of the Test Framework that is worth mentioning.  
 6689

Term	Definition
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.
<b>Candidate Implementation</b>	(or Implementation Under test): The implementation (realization of a specification) used as a target of the testing (e.g. <u>conformance testing</u> ).
<b>Conformance</b>	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).
<b>Interoperability profile</b>	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.
<b>Interoperability Testing</b>	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.

<b>Loop mode (Test Service in)</b>	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.
<b>MSH</b>	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.
<b>Profile</b>	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 Threads, 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.

6690  
6691  
6692  
6693  
6694  
6695  
6696  
6697  
6698  
6699  
6700  
6701  
6702



6704

## Appendix I References

6705

### I.1 Normative References

6707

6708 [ConfCertModelNIST] Conformance Testing and Certification Model for Software Specifications. L.  
6709 Carnahan, L. Rosenthal, M. Skall. ISACC '98 Conference. March 1998

6710 [ConfCertTestFrmk] Conformance Testing and Certification Framework. L. Rosenthal, M. Skall, L. Carnahan.  
6711 April 2001

6712 [ConfReqOASIS] Conformance Requirements for Specifications. OASIS Conformance Technical  
6713 Committee. March 2002.

6714 [ConfGlossary] Conformance Glossary. OASIS Conformance TC, L. Rosenthal. September 2000.

6715 [RFC2119] Key Words for use in RFCs to Indicate Requirement Levels, Internet Engineering Task  
6716 Force, March 1997

6717 [RFC2045] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message  
6718 Bodies, N Freed & N Borenstein, Published November 1996

6719 [RFC2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. N. Freed, N.  
6720 Borenstein. November 1996.

6721 [RFC2387] The MIME Multipart/Related Content-type. E. Levinson. August 1998.

6722 [RFC2392] Content-ID and Message-ID Uniform Resource Locators. E. Levinson, August 1998

6723 [RFC2396] Uniform Resource Identifiers (URI): Generic Syntax. T Berners-Lee, August 1998

6724 [RFC2821] Simple Mail Transfer Protocol, J. Klensin, Editor, April 2001 Obsoletes RFC 821

6725 [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee,  
6726 "Hypertext Transfer Protocol, HTTP/1.1", June 1999.

6727 [SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor; David  
6728 Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish  
6729 Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand  
6730 Software, Inc.; W3C Note 08 May 2000,  
6731 <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

6732 [SOAPAttach] SOAP Messages with Attachments, John J. Barton, Hewlett Packard Labs; Satish Thatte  
6733 and Henrik Frystyk Nielsen, Microsoft, Published Oct 09 2000  
6734 <http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211>

6735 [XLINK] W3C XML Linking Recommendation, <http://www.w3.org/TR/2001/REC-xlink-20010627/>

6736 [XML] W3C Recommendation: Extensible Markup Language (XML) 1.0 (Second Edition),  
6737 October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>

6738 [XMLC14N] W3C Recommendation Canonical XML 1.0,  
6739 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

6740 [XMLNS] W3C Recommendation for Namespaces in XML, World Wide Web Consortium, 14  
6741 January 1999, <http://www.w3.org/TR/1999/REC-xml-names-19990114/>

6742 [XMLDSIG] Joint W3C/IETF XML-Signature Syntax and Processing specification,  
6743 <http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>.

6744 [XPointer] XML Pointer Language (XPointer) Version 1.0, W3C Candidate Recommendation 11  
6745 September 2001, <http://www.w3.org/TR/2001/CR-xptr-20010911/>  
6746

## 6747 I.2 Non-Normative References

6748 [ebTestFramework] ebXML Test Framework specification, Version 1.0, Technical Committee  
6749 Specification, March 4, 2003,  
6750 [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ebxml-iic](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic)

6751 [ebMS] ebXML Messaging Service Specification, Version 2.0,  
6752 [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ebxml-msg](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-msg)

6753 [ebMSInteropTests] ebXML MS V2.0 Basic Interoperability Profile Test Cases,  
6754 [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ebxml-iic](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic)

6755 [ebMSConfTestSuite] ebXML MS V2.0 Conformance Test Suite,  
6756 [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ebxml-iic](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic)

6757 [ebMSInteropReqs] ebXML MS V2.0 Interoperability Test Requirements,  
6758 [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ebxml-iic](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ebxml-iic)

6759

6760 [XMLSchema] W3C XML Schema Recommendation,  
6761 <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>  
6762 <http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/>  
6763 <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

6764 [ebCPP] ebXML Collaboration Protocol Profile and Agreement specification, Version 1.0,  
6765 published 10 May, 2001,  
6766 <http://www.ebxml.org/specs/ebCCP.doc>

6767 [ebBPSS] ebXML Business Process Specification Schema, version 1.0, published 27 April 2001,  
6768 <http://www.ebxml.org/specs/ebBPSS.pdf>.

6769 [ebRS] ebXML Registry Services Specification, version 2.0, published 6 December 2001  
6770 <http://www.oasis-open.org/committees/repreg/documents/2.0/specs/ebRS.pdf>,  
6771 published, 5 December 2001.

6772

---

6773

## Appendix J Acknowledgments

6774 The authors wish to acknowledge the support of the members of the OASIS ebXML IIC TC who  
6775 contributed ideas, comments and text to this specification by the group's discussion eMail list, on  
6776 conference calls and during face-to-face meetings.

### 6777 J.1 IIC Committee Members

6778 Jacques Durand, Fujitsu <[jdurand@fsw.fujitsu.com](mailto:jdurand@fsw.fujitsu.com)>  
6779 Jeffery Eck, Global Exchange Services <[Jeffery.Eck@gxs.ge.com](mailto:Jeffery.Eck@gxs.ge.com)>  
6780 Hatem El Sebaaly, IPNet Solutions <[hatem@ipnetsolutions.com](mailto:hatem@ipnetsolutions.com)>  
6781 Aaron Gomez, Drummond Group Inc. <[aaron@drummondgroup.com](mailto:aaron@drummondgroup.com)>  
6782 Michael Kass, NIST <[michael.kass@nist.gov](mailto:michael.kass@nist.gov)>  
6783 Matthew MacKenzie, Individual <[matt@mac-kenzie.net](mailto:matt@mac-kenzie.net)>  
6784 Monica Martin, Sun Microsystems <[monica.martin@sun.com](mailto:monica.martin@sun.com)>  
6785 Tim Sakach, Drake Certoivo <[tsakach@certoivo.net](mailto:tsakach@certoivo.net)>  
6786 Jeff Turpin, Cyclone Commerce <[jturpin@cyclonecommerce.com](mailto:jturpin@cyclonecommerce.com)>  
6787 Eric van Lydegraf, Kinzan <[ericv@kinzan.com](mailto:ericv@kinzan.com)>  
6788 Pete Wenzel, SeeBeyond <[pete@seebeyond.com](mailto:pete@seebeyond.com)>  
6789 Steven Yung, Sun Microsystems <[steven.yung@sun.com](mailto:steven.yung@sun.com)>  
6790 Boonserm Kulvatunyou, NIST <[serm@nist.gov](mailto:serm@nist.gov)>

6791

6792

6793

6794

6795

6796

6797

6798

6799

6800

6801

6802

6803

6804

6805

6806

6807

6808

6809

6810

---

## Appendix K Revision History

6811

Rev	Date	By Whom	What
cs-10	2003-03-07	Michael Kass	Initial version
cs-11	2004-03-30	Michael Kass	First revision (DRAFT)
cs-12	2004-04-12	Michael Kass	Second revision (DRAFT)



6812

---

## Appendix L Notices

6813 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that  
6814 might be claimed to pertain to the implementation or use of the technology described in this document or  
6815 the extent to which any license under such rights might or might not be available; neither does it  
6816 represent that it has made any effort to identify any such rights. Information on OASIS's procedures with  
6817 respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights  
6818 made available for publication and any assurances of licenses to be made available, or the result of an  
6819 attempt made to obtain a general license or permission for the use of such proprietary rights by  
6820 implementors or users of this specification, can be obtained from the OASIS Executive Director.

6821 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications,  
6822 or other proprietary rights which may cover technology that may be required to implement this  
6823 specification. Please address the information to the OASIS Executive Director.

6824 Copyright © OASIS Open 2003. *All Rights Reserved.*

6825 This document and translations of it may be copied and furnished to others, and derivative works that  
6826 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published  
6827 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice  
6828 and this paragraph are included on all such copies and derivative works. However, this document itself  
6829 does not be modified in any way, such as by removing the copyright notice or references to OASIS,  
6830 except as needed for the purpose of developing OASIS specifications, in which case the procedures for  
6831 copyrights defined in the OASIS Intellectual Property Rights document must be followed, or as required to  
6832 translate it into languages other than English.

6833 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors  
6834 or assigns.

6835 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
6836 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY  
6837 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
6838 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

6839