Creating A Single Global Electronic Market

# ebXML Test Framework

## Committee Specification Version 1.1 DRAFT

## OASIS ebXML Implementation, Interoperability and Conformance Technical Committee

## 14 April, 2004

**Document identifier:**

ebxml-iic-test-framework-11

**Location:**

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

**Authors/Editors:**

Michael Kass, NIST <michael.kass@nist.gov>

**Contributors:**

Steven Yung, Sun Microsystems <steven.yung@sun.com>
Prakash Sinha, IONA <prakash.sinha@iona.com>
Matthew MacKenzie, Individual <matt@mac-kenzie.net>
Hatem El Sebaaly, IPNet Solutions <hatem@ipnetsolutions.com>
Monica Martin, Sun Microsystems <monica.martin@sun.com>
Jacques Durand, Fujitsu <jdurand@fsw.fujitsu.com>
Christopher Frank < C.Frank@seeburger.de>
Eric VanLydegraf, Kinzan <ericv@kinzan.com>
Jeff Turpin, CycloneCommerce <jturpin@cyclonecommerce.com>
Serm Kulvatunyou, NIST <serm@nist.gov>
Tim Sakach, Drake Certivo, Inc. tsakach@certivo.net
Hyunbo Cho, Postech <hcho@postech.ac.kr>

**Abstract:**

This document specifies ebXML interoperability testing specification for the eBusiness community.

**Status:**

This document has been approved as a committee specification, and is updated periodically on no particular schedule.

Committee members should send comments on this specification to the ebxml-iic@lists.oasis-open.org list. Others should subscribe to and send comments to the ebxml-iic-comment@lists.oasis-open.org list. To subscribe, send an email message to ebxml-iic-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For more information about this work, including any errata and related efforts by this committee, please refer to our home page at http://www.oasis-open.org/committees/ebxml-iic.

**Errata to this version:**

None

# Table of Contents

# 1 Introduction

## 1.1 Summary of Contents of this Document

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

This specification is organized around the following topics:

- Interoperability testing architecture
- Test cases for basic interoperability
- Test data materials

## 1.2 Document Conventions

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

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

- *MUST: This word, or the terms "REQUIRED" or "SHALL", means that the definition is an absolute requirement of the specification.*
- *MUST NOT: This phrase, or the phrase "SHALL NOT", means that the definition is an absolute prohibition of the specification.*
- *SHOULD: This word, or the adjective "RECOMMENDED", means that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications MUST be understood and carefully weighed before choosing a different course.*
- *SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED", means that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.*

- *MAY: This word, or the adjective "OPTIONAL", means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation that does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation that does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides).*

## 1.3 Audience

The target audience for this specification is:

- The community of software developers who implement and/or deploy the ebXML Messaging Service (ebMS) or use other ebXML technologies such a s Registry/Repository (RegRep), Collaboration Profile Protocol/Agreement (CPPA) or Business Process Specification Schema (BPSS)

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

## 1.4 Caveats and Assumptions

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

## 1.5 Related Documents

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

- **ebXML Collaboration Protocol Profile and Agreement Specification [ebCPP]** – CPP defines one business partner's technical capabilities to engage in electronic business collaborations with other partners by exchanging electronic messages. A CPA documents the technical agreement between two (or more) partners to engage in electronic business collaboration. The MS Test Requirements and Test Cases will refer to CPA documents or data as part of their material, or context of verification.

- **ebXML Messaging Service Specification [ebMS]** – defines the messaging protocol and service for ebXML, which provide a secure and reliable method for exchanging electronic business transactions using the Internet.

- **ebXML Test Framework [ebTestFramework]**– describes the test architecture, procedures and material that are used to implement the MS Interoperability *Test Suite*, as well as the test harness for this suite.

- **ebXML MS Conformance Test Suite [ebMSConfTestSuite]**– describes the Conformance test suite and material for Messaging Services.

- **ebXML Registry Specification [ebRS]** – defines how one party can discover and/or agree upon the information the party needs to know about another party prior to sending them a message that complies with this specification. The Test Framework is also designed to support the testing of a registry implementation.

- **ebXML Business Process Specification Schema [BPSS]** – defines how two parties can cooperate through message-based collaborations, which follow particular message choreographies. The Test Framework is also designed to support the testing of a business process implementation.

## 1.6 Minimal Requirements for Conformance

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

- It supports all the mandatory syntax; features and behavior (as identified by the [RFC2119] key words MUST, MUST NOT, REQUIRED, SHALL and SHALL NOT) defined in Part 1.1.1 – Document Conventions.

- It supports all the mandatory syntax, features and behavior defined for each of the components of the Test Framework.

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

# 2 Principles and Methodology of Operations

## 2.1 General Objectives

The ebXML Test Framework is intended to support conformance and interoperability testing for ebXML specifications. It describes a testbed architecture and its software components, how these can be combined to create a test harness for each type of ebXML 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 (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.

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, can be defined in a formal manner (including Test Steps and verification conditions). 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, e.g. in case of interoperability) 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. Besides this kind of interfacing, no application code needs to be written.

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

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.

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

## 2.2 General Methodology

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

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

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

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

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

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

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

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

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

# 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 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 Test Case 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 TestStep levels as well.

**ebXML Message Construction –** Includes any portion of the message

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

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

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

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

**Repeat previously executed Test Steps –** Test Driver MUST be capable of repeating previously executed Test Steps for the current Test Case.

**Send messages** - Either directly at transport layer (e.g. by opening an HTTP connection), or by using Test Service actions.

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

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

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

**Report Conformance Test Results –** Test Driver MUST generate an XML conformance report for all executed tests in the profile. Conformance reports MUST validate to the ebXMLTestReport.xsd schema in Appendix E.

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

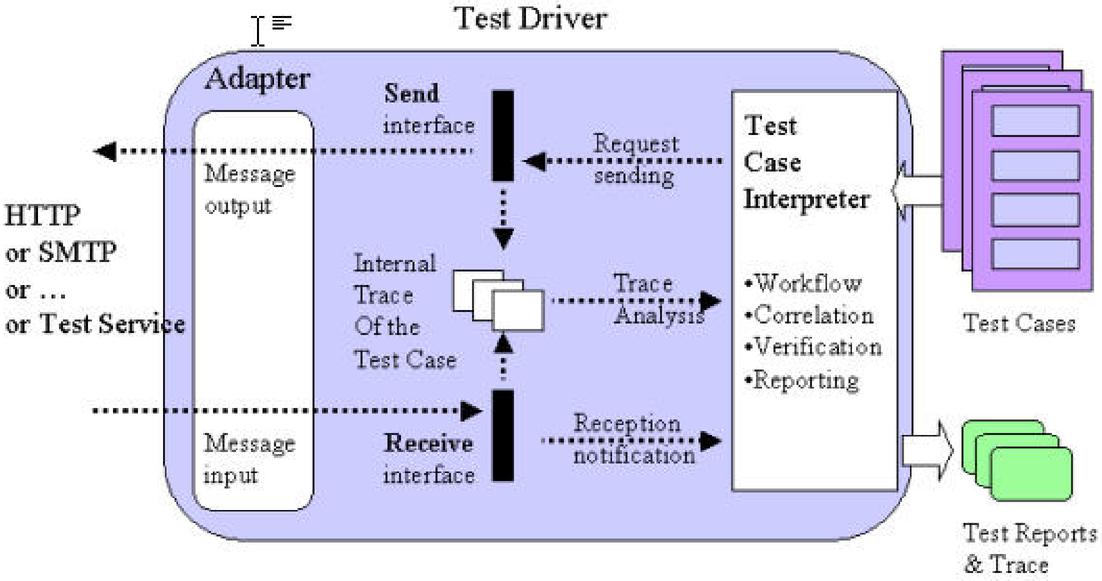

Figure 1- The Test Driver: Functions and Data Flows

### 3.1.2 Using the Test Driver in Connection Mode

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

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

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

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

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

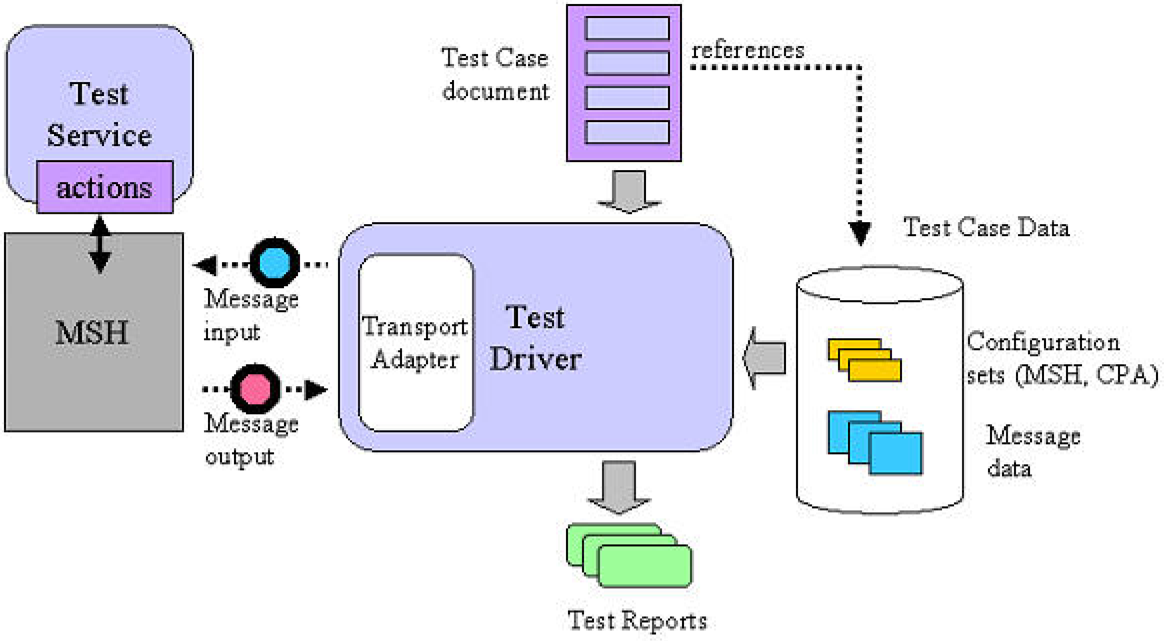

Figure 2- Test Driver:  Connection Mode

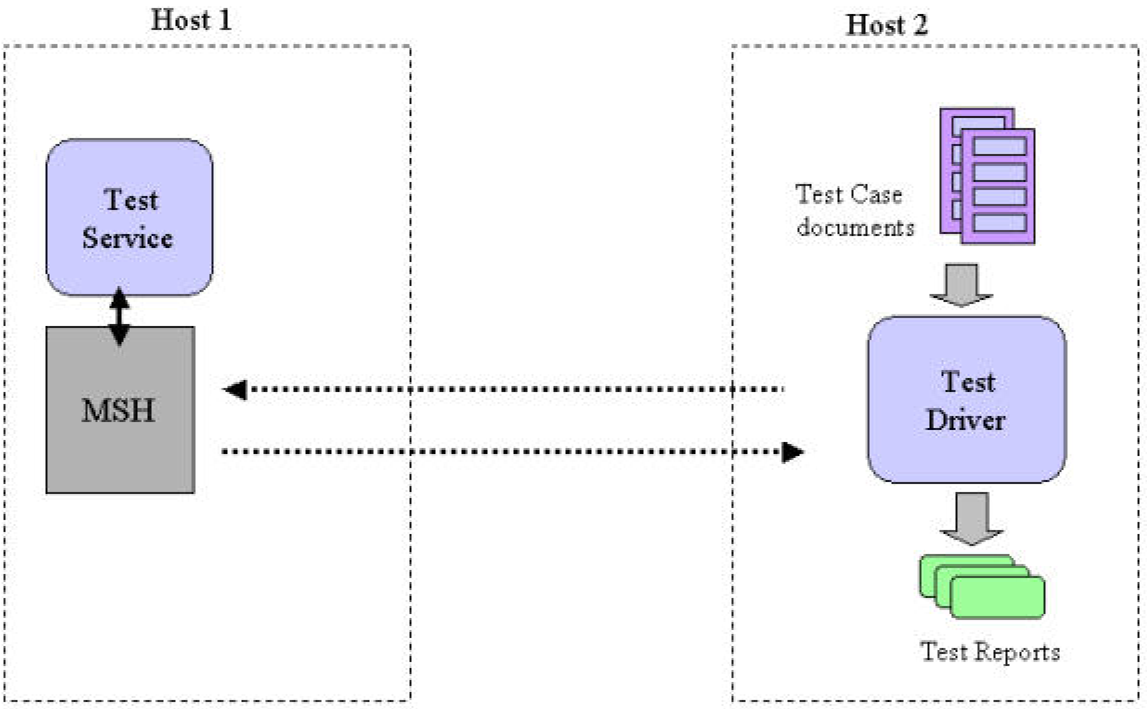

Figure 3 – Test Driver: Remote Connection Mode

### 3.1.3 Using the Test Driver in Service Mode

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

**Sending**: One action of the Test Service, the “Initiator”, serves as a channel to send requests to the MSH it has been interfaced with. This action also MUST provide an interface to the Test Driver at application level. When invoked by a call that contains message data, the action generates a sending request to the MSH API for this message.

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

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

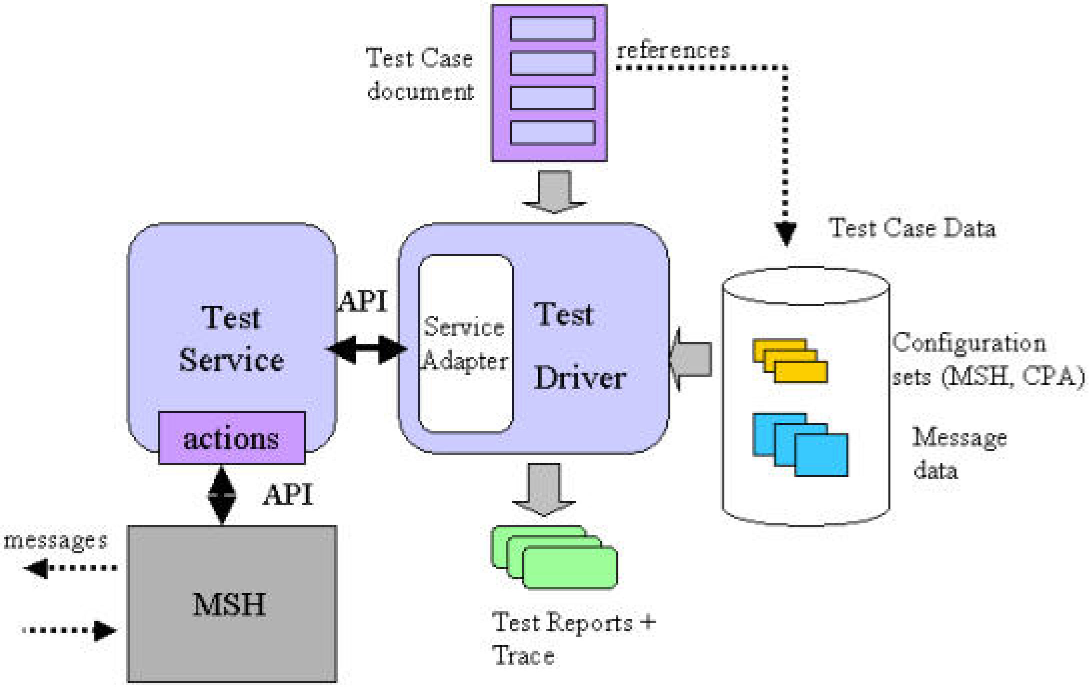

Figure 4 – Test Driver: Service Mode

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

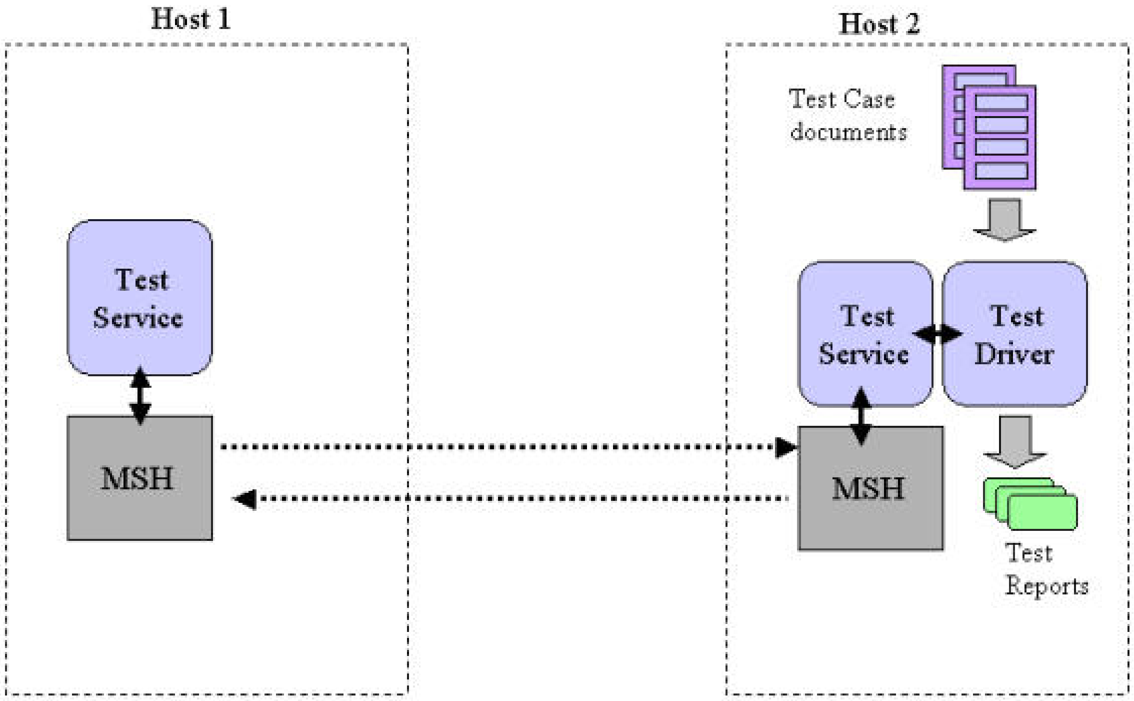

Figure 5 – Test Driver in Service Mode: Point-to-Point Interoperability Testing of Message Handlers

## 3.2 The Test Service

### 3.2.1 Functions and Interactions

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

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

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

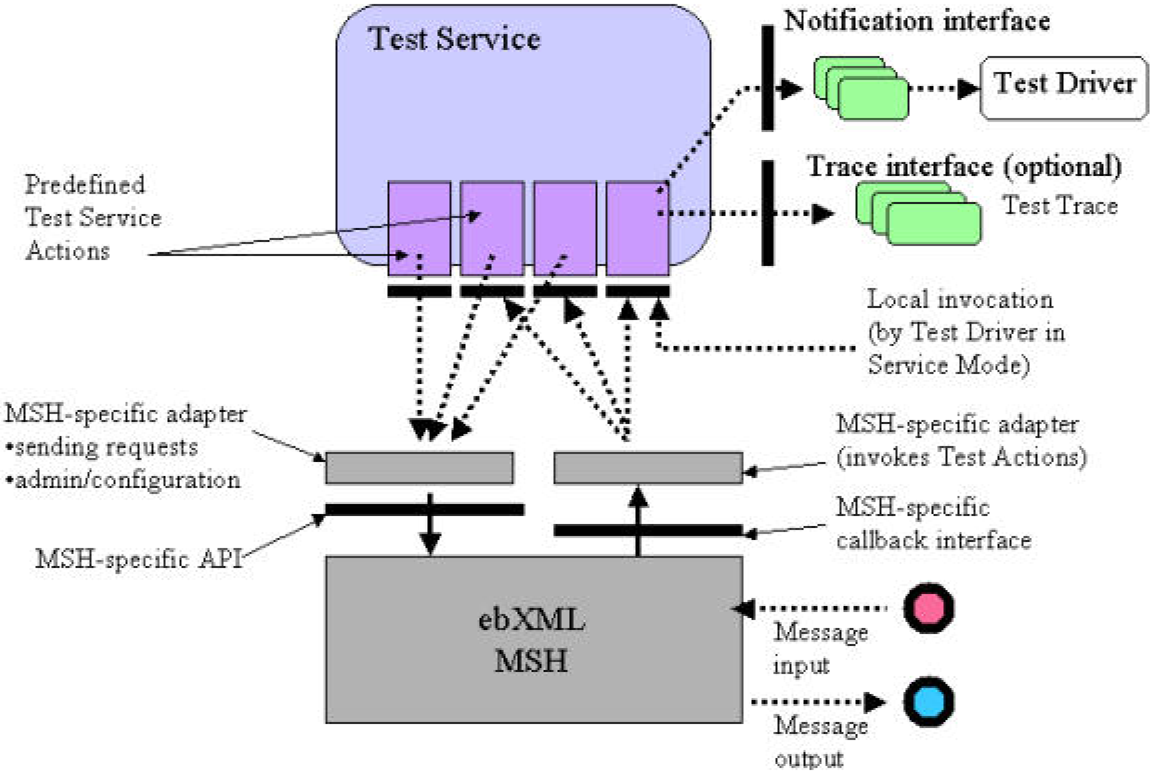
Test Service
Notification interface
Test Driver
Trace interface (optional)
Test Trace
Predefined Test Service Actions
Local invocation (by Test Driver in Service Mode)
MSH-specific adapter
•sending requests
•admin/configuration
MSH-specific adapter (invokes Test Actions)
MSH-specific callback interface
MSH-specific API
ebXML MSH
Message input
Message output

Figure 6 – The Test Service and its Interfaces

The functions of the Test Service are:

To implement the actions which map to Service / Action fields in a message header. The set of test actions which are pre-defined in the Test Service will perform diverse functions, which are enumerated below:

To notify the Test Driver of incoming messages. This only occurs when the Test Service is deployed in *reporting mode*, which assumes it is coupled with a Test Driver.

To perform some message processing, e.g. compare a received message payload with a reference payload (or their digests).

To send back a response to the MSH.  Depending on the action invoked, the response may range from a pre-defined acknowledgment to a specific message as previously specified.

Optionally, to generate a trace of its operations, in order to help trouble-shooting, or for reporting purpose.

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

### 3.2.2 Modes of Operation of the Test Service

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

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

> **Local Reporting Mode:** The Test Driver is installed on the same host as the Test Service, and executes in the same process space. The notification uses the *Receive* interface of the Test Driver, which is operating in service mode.
>
> **Remote Reporting Mode:** The Test Driver is installed on a different host than the Test Service. The notification is done via messages to the Test Driver, which is generally operating in connection mode.

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

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

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

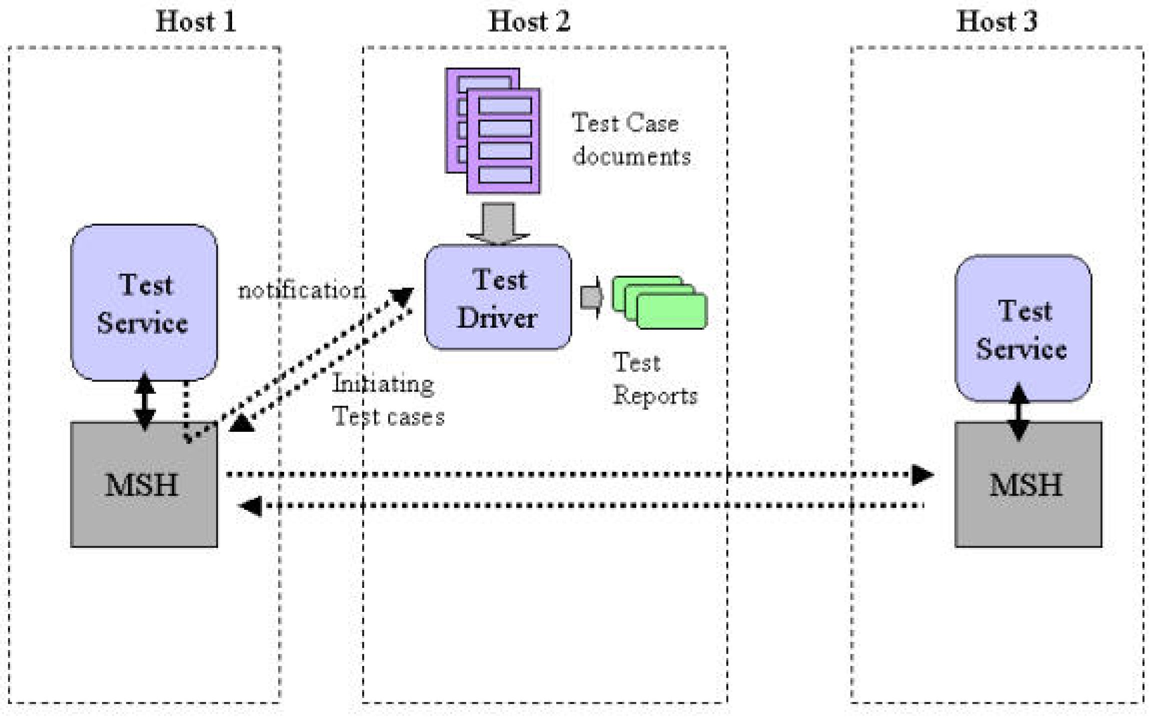

Figure 7 – Example of Remote Reporting Mode : The Interoperability Test Center Model

### 3.2.3 Configuration Parameters of the Test Service

Test Service configuration is initially performed when the Test Driver reads the executable Test Suite XML document, and loads the TestServiceConfigurator content found at the beginning of the TestSuite. The Test Driver attempts to configure the Test Service (through a local or remote Configuration interface). If the Test Driver is unable to configure the Test Service, then the Test Driver MUST generate an exception. The Test Driver MAY handle this exception in a “non-fatal” manner if the Test Service provides an alternate means of initial configuration.

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

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

ResponseURL (destination for response messages)

NotificationURL (destination for notification messages, if applicable)

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

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

Test Service configuration MAY be performed locally, if the Test Driver is in “service” mode (in the same program space as the Test Service). Test Service configuration MUST be performed via RPC to the Test Service Configuration interface’s “configurator” method if it is in “connection” mode.

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

## 3.2.4 The Messaging Actions of the Test Service

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

### 3.2.4.1 Common Functions

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

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

### 3.2.4.2 Test Service Actions

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

#### **3.2.4.2.1 Mute action**

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

Response Destination: None

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

#### 3.2.4.2.2 Dummy action

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

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

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

#### 3.2.4.2.3 Reflector

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

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

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

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

#### 3.2.4.2.5 **Initiator** action

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

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

Response Destination: Any party defined by the Test Driver.

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

#### 3.2.4.2.6 PayloadVerify action

Reporting/Loop Mode Action Description: On invocation, this action will compare the payload(s) of the received message, with the expected payload. Instead of using real payloads, to be pre-installed on the site of the Test Service, it is RECOMMENDED that a digest (or signature) of the reference payloads (files) be pre-installed on the Test Service host using configuration parameters. The PayloadVerify action will then calculate the digest of each received payload and compare with the reference digest parameter values. This action will test the service contract between application and MSH, as errors may originate either on the wire, or at every level of message processing in the MSH until message data is passed to the application. The action reports (via RPC) to the Test Driver the outcome of the comparison. This is done via an alternate communication channel to ensure that the same system being tested is not used to report the reliability of its own MSH. A “notification” message is sent via RPC to the Test Driver. The previous MessageID is reported (for correlation) in the RefToMessageId header attribute of the response. The previous ConversationId is also reported. The payload message will contain a verification status notification for each verified payload, as specified in Appendix F.

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

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

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

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

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

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

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

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

### 3.2.5 Interfaces for Test Driver and Test Service

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

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

The three interfaces on the Test Service component are:

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

**Configuration** – Consists of one method, (configuration) which accepts a Configuration Group list of parameters and their corresponding values. This includes three "required" parameters, and additional optional parameters that may be used in message construction by the "initiator" method. The method returns an XML document with a "pass/fail" Result element.

**Notification** – consists of one method (report), which passes a "notification" message (either an application error, a messaging error, or a received message) to the Test Driver's Receive interface via its "notify" method. The Test Driver MUST return an XML document with a "pass/fail" Result element to the Test Service report method.

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

The interface on the Test Driver component is:

**Receive** – The "notify" method accepts incoming notification messages from the Test Service and returns a required response for each. Notification messages include messages received from the Test Service (when the Test Service is in "reporting mode", error messages from the Test Service and response messages from the Test Service referencing success/failure of Test Service configuration, or message initialization.

#### 3.2.5.1 Abstract Test Service "Send" Interface

The abstract interface is defined as:

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

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

| **Method Return Type** | **Method Name** | **Exception Condition** |
|---|---|---|
| InitiatorResponse | initiator (messageDeclaration, messagePayloadList payloads) | Failed to construct or send |

| | | |
|---|---|---|
| | Passes the constructed message “declaration” and encapsulated message payloads to the Test Service Initiator action, returning a response message (pass/fail) to the Test Driver | message |

Table 1 – Initiator method description

**Semantic Description:** The Initiator call instructs the Test Service to generate a new message. The new message material (message declaration and encapsulated payloads) is provided as two separate arguments to the initiator call. The header of the new message can be anything that is specified and understood by the Test Service (e.g. ebXML or RNIF). This action may be used to generate a "first" message of a new conversation (if no ConversationId is present in the Message Declaration and no “global” ConversationId was provided to the Test Service via a previous call to the “configurator” method). If a global ConversationId was provided to the Test Service through the “configurator” method, then the same ConversationId may be used again by a Test Service to carry on an existing conversation.

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

### 3.2.5.2 WSDL representation of the initiator RPC method

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

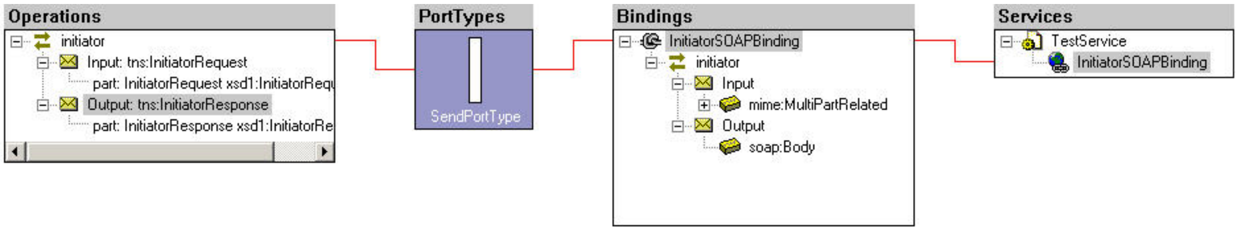

Figure 8 – WSDL diagram of the Initiator SOAP method

#### 3.2.5.3 Abstract Test Service "Configuration" Interface

The abstract interface is defined as:

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

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

| **Method Return Type** | **Method Name** | **Exception Condition** |
|---|---|---|
| ConfiguratorResponse | configurator (ConfigurationList list)<br>Passes the configuration parameters to the Test Service | Test Service fails to configure properly |

Table 2 – Configurator method

**Semantic Description:** The configurator call passes configuration data from the Test Driver to the Test Service. This includes the three REQUIRED configuration items (ResponseURL, NotificationURL, ServiceMode), plus additional optional parameters that may be used in message construction by the "initiator" method; including Service, Action, CPAId, SenderParty, ReceiverParty, Additionally, "ad-hoc" parameters MAY be added for construction of non-ebXML messaging envelopes. Ad-hoc parameter names and values MUST be agreed to by parties whose goal is to develop open test suites for particular messaging applications using the OASIS IIC Test Framework.

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

#### 3.2.5.3.1 WSDL representation of the configurator SOAP method

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

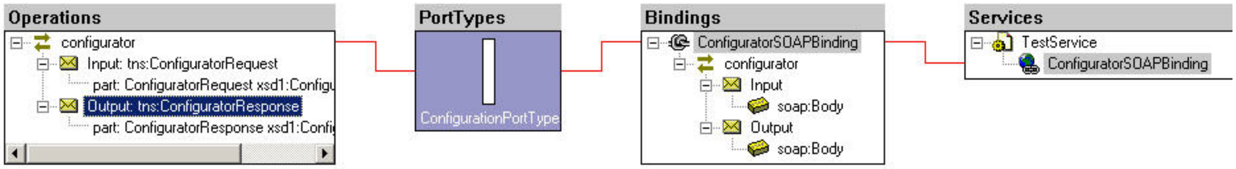

Figure 9 – WSDL diagram of the Configurator SOAP method

### 3.2.5.4 Abstract Test Service "Notification" Interface

The abstract interface is defined as:

4. An interface that must be supported by the Test Service
5. An "errorAppNotify", "errorURLNotify" and "messageNotify" method that must be supported by that interface
6. The parameters and responses that must be supported by that method

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

| | | **Exception** |
|---|---|---|

| Method Return Type | Method Name | Condition |
|---|---|---|
| boolean | errorAppNotify (MessageEnvelope envelope)<br>Passes the application error notification message to the Test Driver<br>errorURLNotify (MessageEnvelope envelope)<br>Passes the application error notification message to the Test Driver<br>messageNotify (MessageEnvelope envelope, MessagePayloadList payloads)<br>Passes received message to the Test Driver | Test Driver fails to respond or responds with a NotificatonResponse Success value of "false" |

Table 3 – report method description

**Semantic Description:** The report method passes a notification message to the Test Driver. The notification message may be a "virtual copy" of a message received by the Test Service, a message containing an Error that is directed to the Test Driver due to an inability of the Test Service to resolve an error reporting location, or it may be an "application error" message, generated by the Test Service, to be forwarded to the Test Driver. The semantics of each case are described below:

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

Such errors fall into two categories:

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

Notification Message Format:

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

The **errorURLNotify** method will capture "normal" error notifications from the MSH (i.e. errors normally returned to the sending MSH). This method is specified to handle cases where the MSH cannot resolve the error reporting location (not present in CPA) and does not return the error to the sending MSH. In this case the Test Service Notification interface is utilized to report the error to the Test Driver.

Notification Message Format:

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

The **messageNotify** method will capture messages received by the Test Service. This method is specified to handle testing scenarios where the Test Service is in "local-reporting" or "remote reporting" mode. A notification message generated by the messageNotify method is a "copy" of the received message envelope and an encapsulated list of any attachments provided with the message. The message contains.

Notification Message Format:

All notification messages generated by the messageNotify method will have the same characteristics a normal message (i.e. have a MessageHeader with refToMessageId, ConversationId, CPAId). Additionally, the messageNotify method will pass to the Test Driver an encapsulated list of message attachments that were a part of the received message. This message will be identified as "different" from a received message by the presence of a "Notification" root element, which contains reporting test service name, reporting test service instance id, reporting method name (messageNotify), synch type (synchronous or asynchronous), and id.

Additional note:

Notfication messages do not contain any artifacts pertaining to the protocol that carried them. For example, no HTTP or MIME headers are passed along with the notification message; becase the Test

Service does not normally have access to this message content. Only message envelopes, and accompanying message payloads are passed on to the Test Driver's "Receive" interface.

Response type:

All methods of the Test Service Notification interface return a result of type boolean , meaning the method returns a true/false result to its calling process. A result of "true" indicates that the notification method received a "success" response from the Test Driver, indicating successful notification. A result of "false" indicates an unsuccessful attempt to notify the Test Driver with this message.

#### 3.2.5.4.1 WSDL representation of the errorAppNotify, errorURLNotify and messageNotify SOAP methods

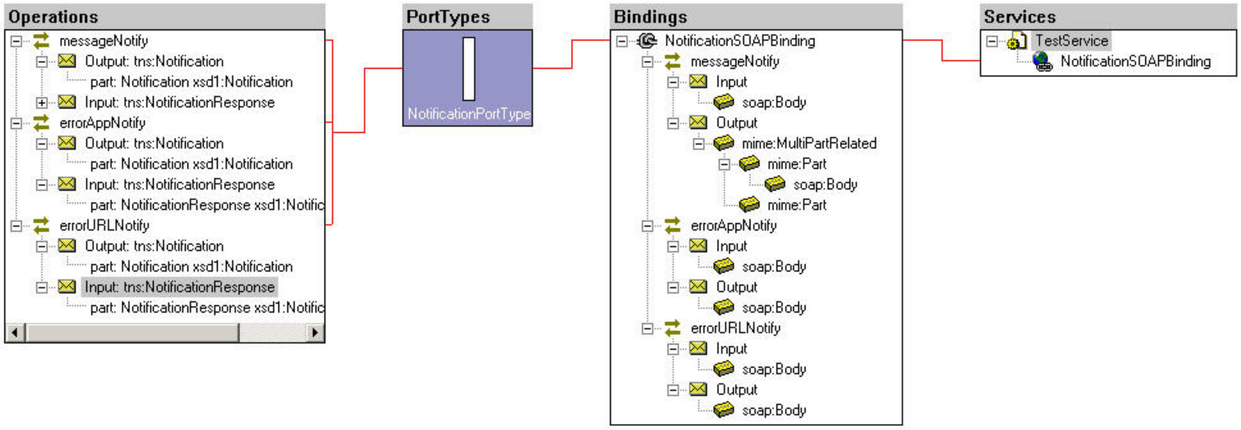

Figure 9 – WSDL diagram of the notification SOAP method

### 3.2.5.5 Abstract Test Driver "Receive" Interface

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

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

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

| Method Return Type | Method Name | Exception Condition |
|---|---|---|
| NotificationResponse | notify (MessageInfo message, messagePayloadList payloads)<br>Passes the, received message envelope and encapsulated message payloads to the Test Driver | Test Driver fails to accept the notification message |

Table 4 – WSDL diagram of the notify SOAP method

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

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

3.2.5.5.1 WSDL representation of the notify SOAP method

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

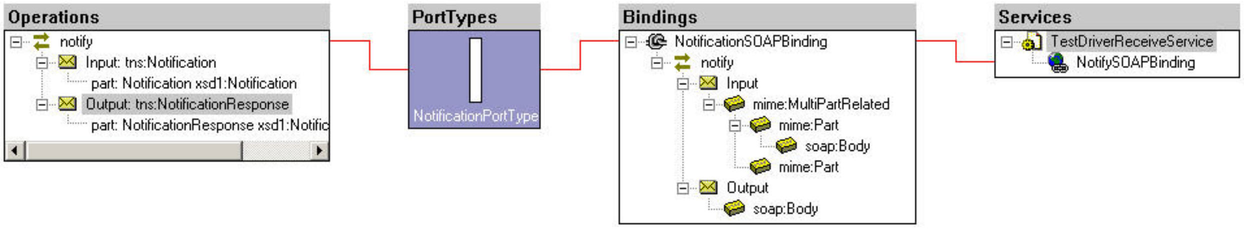

Figure 10 – WSDL diagram of the Notification SOAP method

## 3.3 Executing Test Cases

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

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

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

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

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

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

### 3.3.1 A Typical Execution Scenario

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

A typical execution model for the Test Harness would be:

A Test Driver is installed on a networked computer.

An implementer wishing to test an exam (or other) implementation invokes the Test Driver executable.

The Test Driver asks the tester for fundamental information (e.g. mode of testing to be used by the Test Driver, message and error reporting URL for the candidate implementation)

The Test Driver “self configures” based upon user preferences.

The Test Driver performs any local or remote configuration of the candidate implementation.

The Test Driver presents the tester with a list of conformance or interoperability testing profiles that he/she may select from for testing the candidate implementation.

The tester chooses a profile.

Execution of Test Cases against the specified profile begins.

A standard Test Report Document is generated by the Test Driver, providing a trace of all testing operations performed for each Test Case, with accompanying Test Case results, indicating a final result of “pass”, “fail” or “undetermined” for each Test Case, based upon detailed results of each operation within each Test Case.

If a candidate implementation passes all Test Cases in the Test Suite, it can be considered conformant or interoperable for that particular testing profile.

If a candidate implementation fails some Test Cases, but the Test Requirement that they tested against were “OPTIONAL”, “HIGHLY RECOMMENDED” or “RECOMMENDED”, then that implementation may still be conformant for all REQUIRED features tested.

If the optional features tested were actually implemented on the candidate, and it failed any Test Cases that test against those features then the candidate would be considered “non-conformant” for those optional features.

If any Test Case results were “undetermined” (due to network problems, or due to missing prerequisite candidate features that are not under the control of the Test Harness) then ultimate conformance/interoperability of the candidate implementation is deemed “undetermined” for that testing profile. In such cases, resolution of the underlying system issue must be resolved or the Testing Profile must be redefined to test only those features that are truly supported by the candidate implementation.

The above list represents an “overall” view of how a Test Harness operates. Detailed descriptions of the testing material that drives the Test Harness, and implementation requirements for the Test Driver and Test Service follow.

### 3.3.2 Test Case as a Workflow of Threads and Test Steps

:

A Test Case is a workflow of Test Threads and/or Test Steps. A Thread can be executed either in a synchronous or asynchronous manner. If a particular operation consists of a logically grouped sequence of message “send” and “receive” operations, then a Thread is a logical container to group those operations. . A Test Step can be thought of as test operation (either a single message sending or message receiving operation). In addition, a Test Step may test an assertion of expected message content from a received message. A Test Step may also include conditional actions (testing preconditions) that are a basis for proceeding to the execution of the assertion test.

A Test Case Instance is the execution of a particular Test Case, identified by specific message attribute values. For example, two instances of the same Test Case will be distinguished by distinct ConversationId and MessageId values in the generated messages. An example of a sequence of Test Steps associated with an MS Conformance Test Case is:

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

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

### 3.3.3 Related Message Data and Message Declarations

Some Test Steps will require construction of message data. This message data MUST be specified using a Message Declaration (see Section 7). A Message Declaration is an XML-based script interpreted by the Test Driver to construct a message envelope and its content. Payload material is not included in the messages declaration, but is referenced by it (for example, in the case of ebXML Messaging, via the Manifest element).

A test step may also include operations that allow for extraction of a payload from or for adding a payload to a message.

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

Assemble a message from script material and referenced payloads.

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

### 3.3.4 Related Configuration Data

Test Cases MAY be executed under a pre-defined collaboration agreement.  For example, when testing ebXML Messaging Services, this agreement is a CPA [ebCPP]. This agreement will configure the ebXML Candidate Implementations involved in the testing, or the collaborations that execute on these implementations. A Test Driver Configuration Document:  contains XML content for Test Driver configuration.  Included in this document will be parameters defining the operational mode of the test driver and (if applicable) a reference to configuration data for the candidate implementation(s) to be tested.

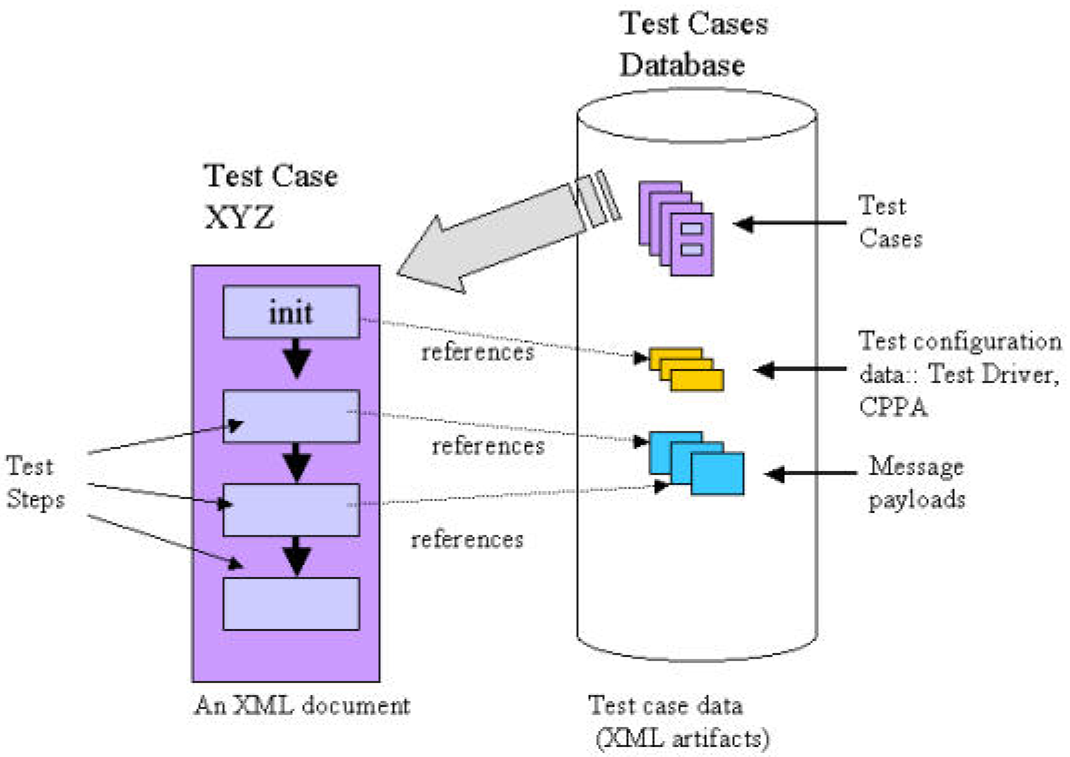

Figure 11 – Test Case Document and Database

# Part II: Test Suite Representation

# 4 Test Suite

## 4.1 Conformance vs. Interoperability Test Suite

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

§ **Conformance Test Suite**. The objective is to verify the adherence or non-adherence of a Candidate Implementation to the target specification. The test harness and Test Cases will be designed around a single (candidate) implementation. The suite material emphasizes the target specification, by including a comprehensive set of Test Requirements, as well as a clear mapping of these to the original specification (e.g. in form of an annotated version of this specification).

§ **Interoperability Test Suite**. The objective is to verify that two implementations (or more) of the same specification, or that an implementation and its operational environment, can interoperate according to an agreement or contract (which is compliant with the specification, but usually restricts further the requirements). These implementations are assumed to be conforming (i.e. have passed conformance tests or have achieved the level of function of such tests), so the reference to the specification is not as important as in conformance. Such a test suite involves two or more Candidate Implementations of the target specification. The test harness and Test Cases will be designed in order to drive and monitor these implementations.

A conformance test suite is composed of:

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

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

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

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

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

An Interoperability Test Suite is composed of:

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

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

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

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

## 4.2 The Test Suite Document

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

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

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

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

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

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

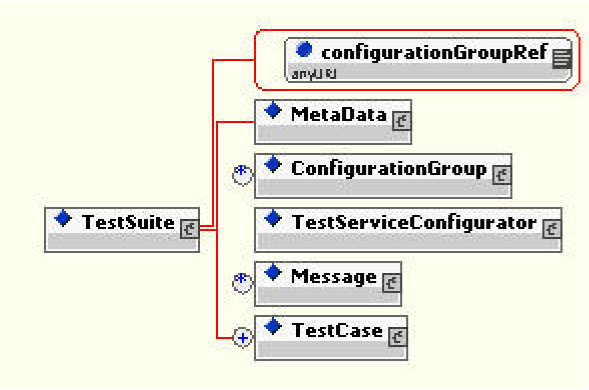

Figure 12 – Graphic representation of basic view of TestSuite schema

**Definition of Content**

| Name | Description | Default Value From Test Driver | Required/Optional | Exception Condition |
|---|---|---|---|---|
| TestSuite | Container for all configuration, documentation and tests | | Required | |
| configurationGroupRef | Reference ID of the ConfigurationGroup data used to configure | | Required | ConfigurationGroup not found |

|  |  |  |  |  |
|---|---|---|---|---|
|  | theTest Driver (in connection mode) or Test Service/MSH ( when in service mode) |  |  |  |
| Metadata | Container for general documentation of the entire Test Suite |  | Required |  |
| ConfigurationGroup | Container for Test Driver configuration instance data |  | Optional |  |
| TestServiceConfigurator | Containter for Test Service configuration instance data |  | Required | Unable to configure Test Service |
| Message | Container element for “wildcard” message content (i.e. any well-formed XML content) |  | Optional |  |
| TestCase | Container for an individual Test Case |  | Required |  |

Table 5 provides a list of TestSuite element and attribute content

### 4.2.1 Test Suite Metadata

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

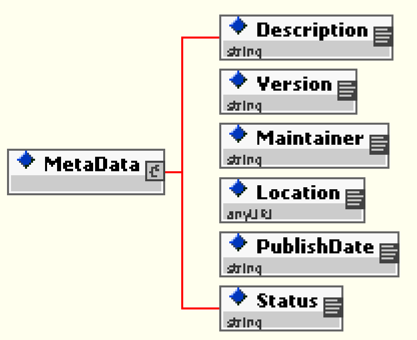

Figure 13 – Graphic representation of expanded view of the Metadata element

**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 | |

Table 6 provides a list of Metadata element and attribute content

### 4.2.2 The ConfigurationGroup

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

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

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

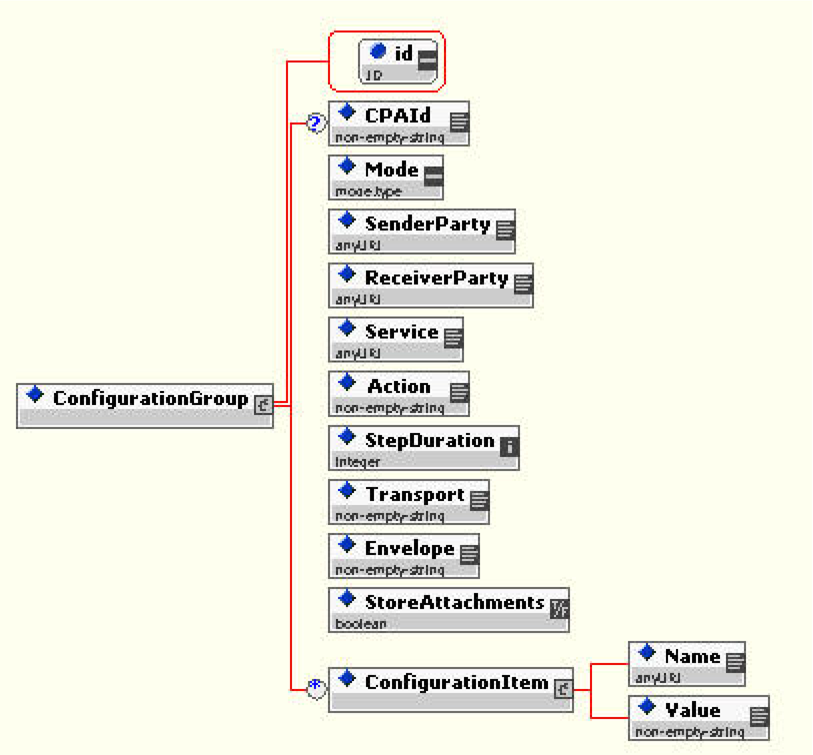

Figure 14 – Graphic representation of expanded view of the BaseConfigurationGroup element

### Definition of Content

| 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 | |
| CPAId | Unique identifier matching one of the testing Collaboration Agreement documents in the Conformance or Interoperability Test Suite. Inserted inside outgoing messages, as content for the CPAId element in an ebXML MessageHeader. Value is also used to configure MSH when in “service” mode. If a CPA is not used in testing (e.g. A2A testing), then this optional parameter may be omitted from the ConfigurationGroup content. | | Optional | |
| Mode | One of two types for the Test Driver, (service \| connection) | | Required | |
| SenderParty | Default identifier used in message header From/PartyId | | Required | |
| ReceiverParty | Default identifier used in message header To/PartyId | | Required | |
| Service | Default Service to be inserted into outgoing message Service element content | | Required | |
| Action | Default Service Action to be inserted into outgoing messages Action element content | | Required | |
| 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 | |
| ConfigurationItem | 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 |  | Required |  |
| Type | Type of ConfigurationItem (namespace or parameter) |  | Required |  |

Table 7 provides a list of ConfigurationGroup element and attribute content

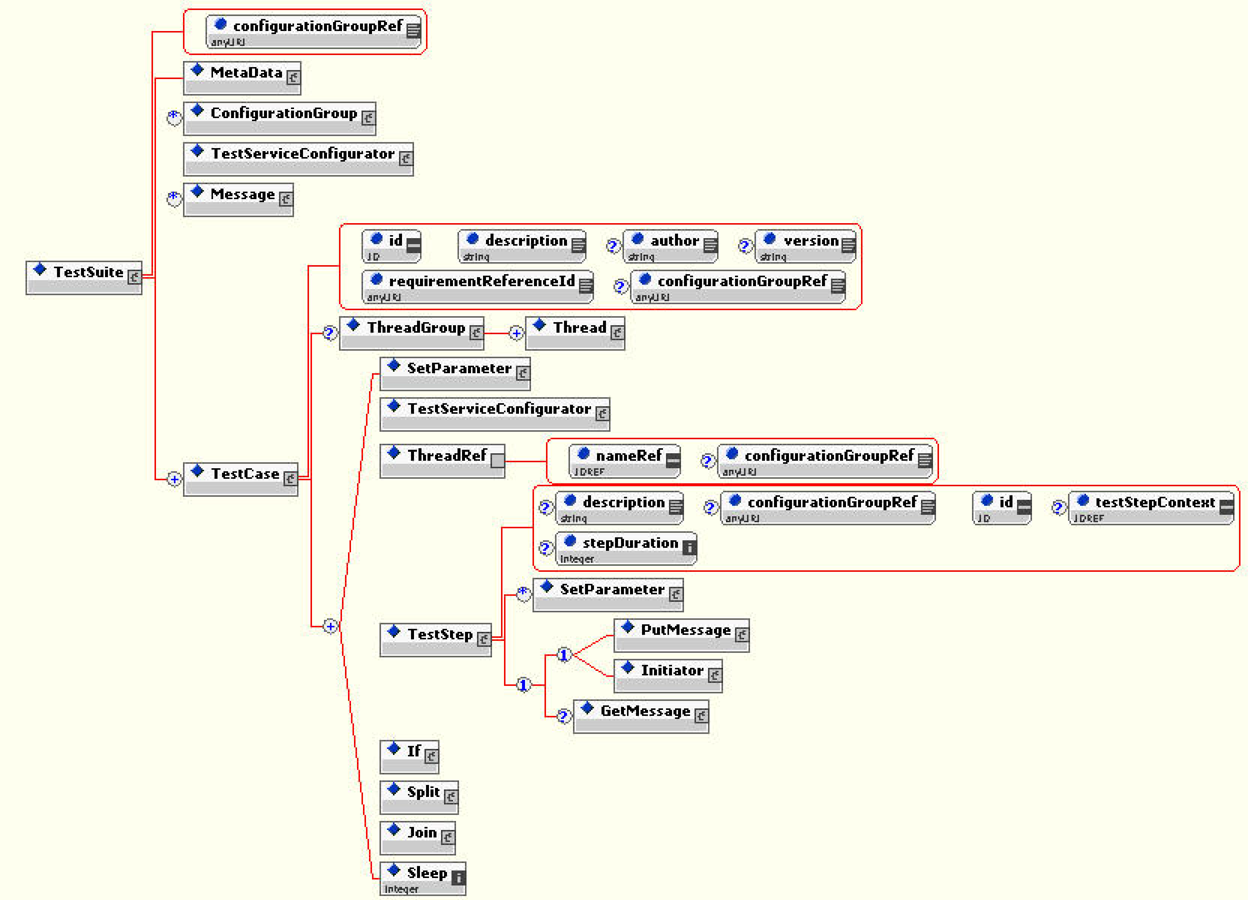

Figure 15 – Graphic representation of hierarchical use of the ConfigurationGroup via reference at TestSuite, TestCase, Thread and TestStep levels in the test object hierarchy

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

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

Certain portions of a message are auto-generated by the Test Driver (or MSH) at run-time

This includes the following parameters:

ConversationId

MessageId

Timestamp

Other message content may be provided through parameter definitions in the current ConfigurationGroup, or through a SetParameter or SetXPathParameter operation. This includes message content such as:

CPAId

Service

Action

SenderParty

ReceiverParty

The following rule describes how a Test Driver MUST interpret parameter values and their precedence of assignment within a Test Suite.

a) The TestSuite element's “configurationGroupRef” attribute points to the “global” parameter definition for the entire Test Suite. This acts as the “baseline” parameter definition before Test Suite execution begins.

b) Parameters MAY be used by an XSL stylesheet or Xupdate document that “mutates” a Message Declaration. They are passed to the XSL or Xupdate processor via name reference.

c) Parameters MAY be used by the VerifyContent operation through reference in an XPath expression. Parameter names are referenced in XPath expressions with a preceding “$” character. The Test Driver

MUST dereference the parameter prior to performing an XPath query on a FilterResult document object.

d) If a parameter is defined in a ConfigurationGroup or via a SetParameter operation, the parameter definition takes precedence over any “auto-generated” definition of that parameter by the Test Driver. Care should be taken to only “override” such values at the (low) TestStep level, so that “side effects” are not passed on through the Test Suite object hierarchy (i.e. influencing message construction beyond the scope of the Test Step or Thread that is intended).

e) Any descendent TestCase, Thread or TestStep element with a “configurationGroupRef” attribute “redefines” a parameters value for itself and any descendent Threads or Test Steps (i.e. it limits the scope of the parameter definition to all of its descendents).

f) Any “SetParameter” elements definition within a TestCase, Thread or Test Step element supersedes its current definition within the current ConfigurationGroup. The scope of the parameter definition is limited to any descendent Threads and/or Test Steps of the current test object.

A Test Driver MUST generate an exception and terminate the Test Case with a result of “undetermined” if it cannot construct a message due to an undefined parameter.

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

### 4.2.3 The TestServiceConfigurator Operation

The TestServiceConfiguration element instructs the Test Driver to configure the Test Service. A Test Service MUST provide both a Configuration interface to the Test Service, with a “configurator” method, like that specified in section 3.2.5. The Test Driver MAY access the Configuration interface either locally or remotely (via RPC), depending upon the current mode of the Test Driver.

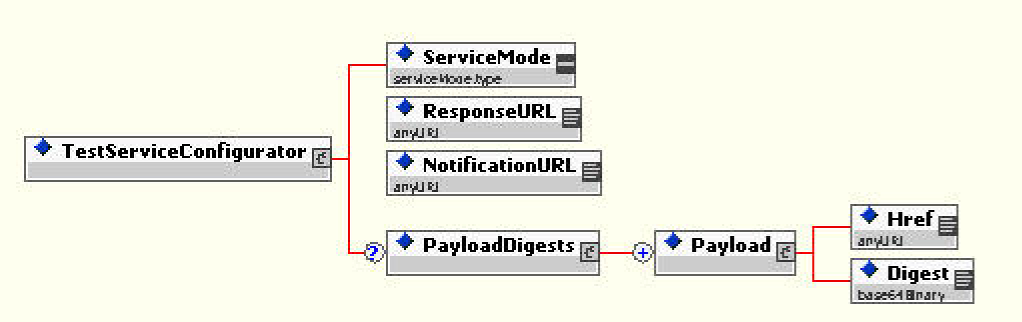

Figure 16 – Graphic representation of the TestServiceConfigurator content

| **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 | |
| Href | URI of the message payload | | Required | |
| Digest | MD5 digest value of the payload | | Required | |

#### 4.2.3.1 TestServiceConfigurator behavior in Connection and Service mode

**In Connection Mode:** The "TestServiceConfigurator" operation instructs the Test Driver to pass configuration parameters (via RPC) to a remote Test Service Configuration interface, using its "configurator" method. The Test Service MUST respond with a status of "success" or "fail".

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

# 5 Test Requirements

## 5.1 Purpose and Structure

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

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

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

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

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

## 5.2 The Test Requirements Document

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

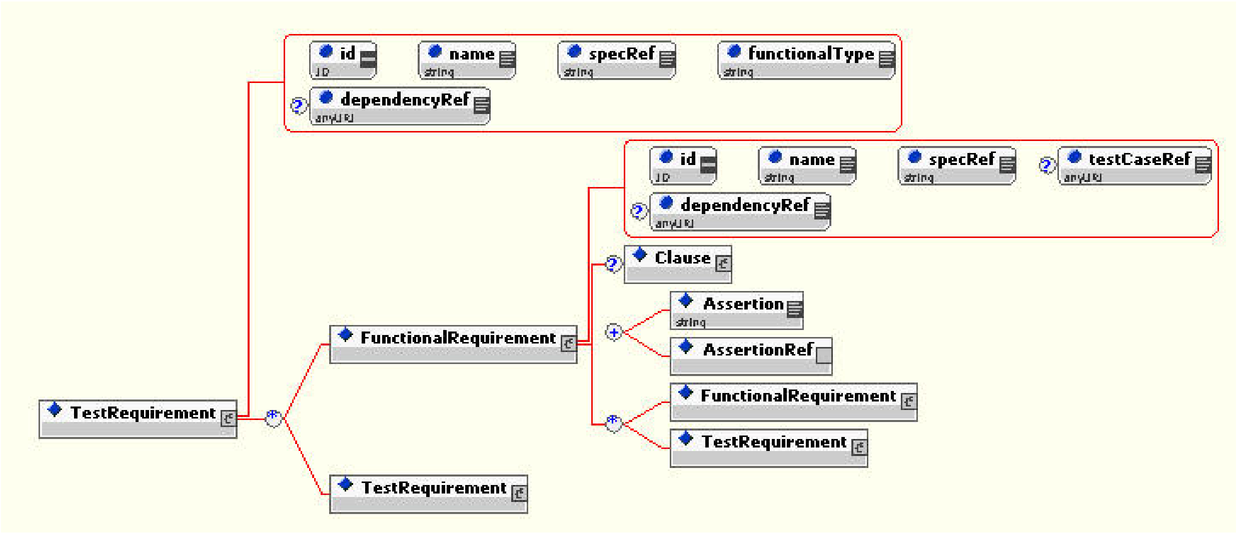

Figure 17 – Graphic representation of ebXMLTestRequirements.xsd schema

Definition of Content

## 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 first 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 | |

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

## 5.3 Specification Coverage

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

A specification section may map to several Test Requirements.

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

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

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

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

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

The coverage qualifier may have values:

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

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

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

    - (1) The pre-condition(s) of the test requirement(s) ignores on purpose a subset of situations that cannot be reasonably tested under the Test Framework.
    - (2) The occurrence of situations that match the pre-condition of a Test Requirement is known to be under control of the implementation  (e.g. implementation-dependent) or of external factors, and out of the control of the testbed. (See *contingent run-time coverage* definition, Section 7).

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

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

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

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

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

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

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

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

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

The coverage qualifier may have values:

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

# 6 Test Profiles

## 6.1 The Test Profile Document

The Test Profile document points to a subset of Test Requirements (in the Test Requirements document), that 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.

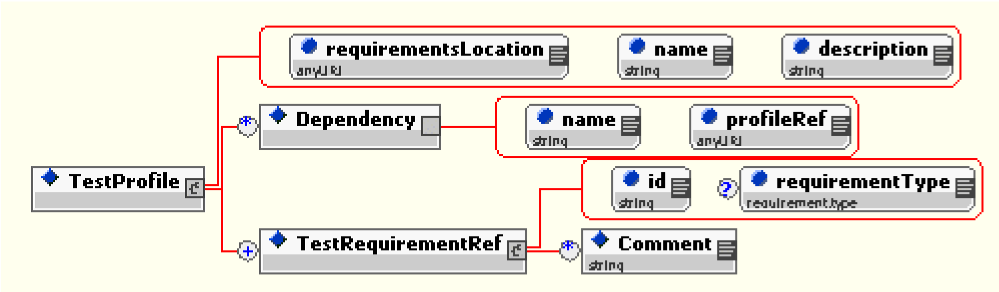

Figure 18 – Graphic representation of ebXMLTestProfile.xsd schema

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 |  |

Table 9 provides a list of TestProfile element and attribute content

## 6.2 Relationships between Profiles, Requirements and Test Cases

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

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

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

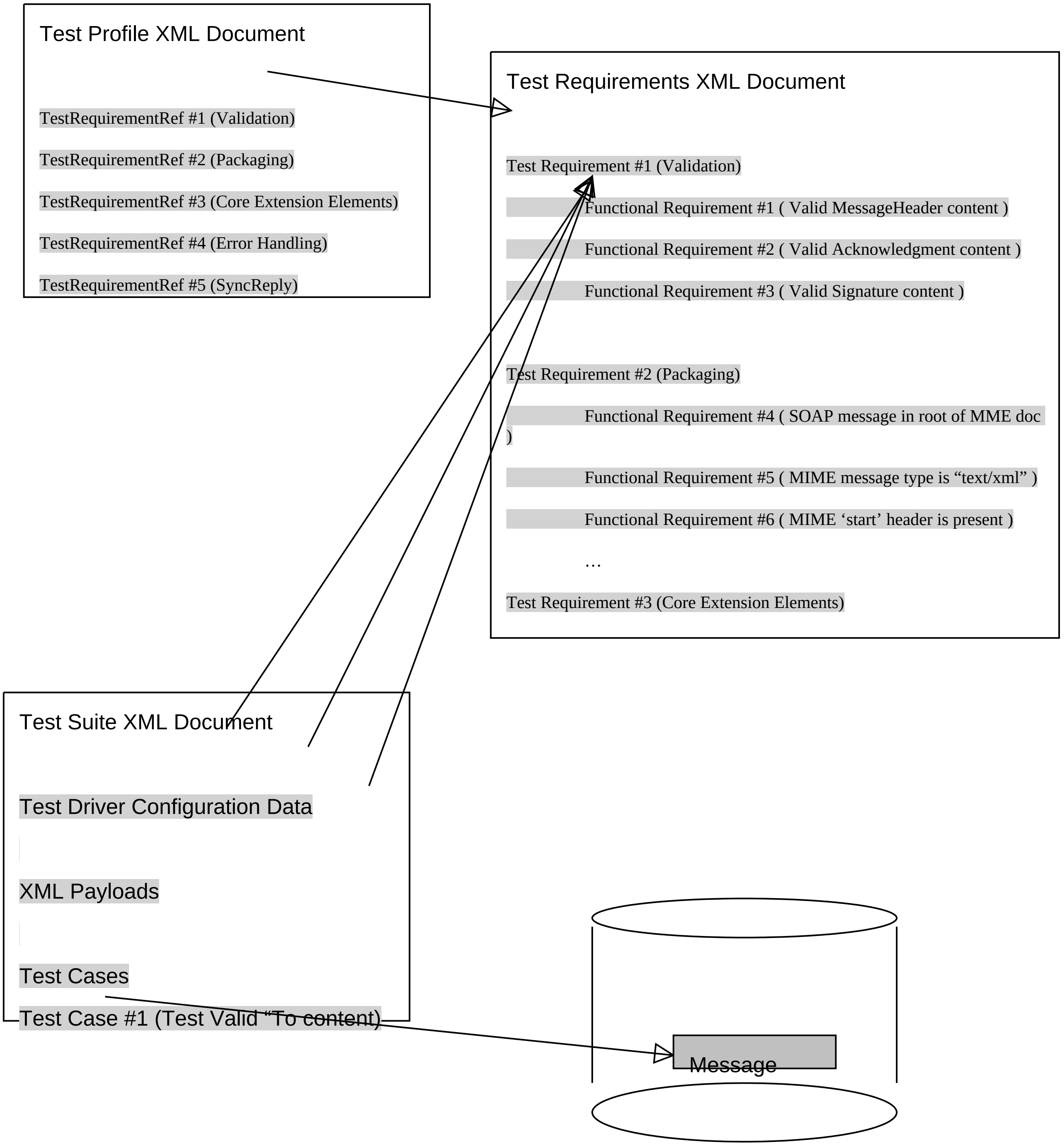
Test Profile XML Document
TestRequirementRef #1 (Validation)
TestRequirementRef #2 (Packaging)
TestRequirementRef #3 (Core Extension Elements)
TestRequirementRef #4 (Error Handling)
TestRequirementRef #5 (SyncReply)
Test Requirements XML Document
Test Requirement #1 (Validation)
Functional Requirement #1 ( Valid MessageHeader content )
Functional Requirement #2 ( Valid Acknowledgment content )
Functional Requirement #3 ( Valid Signature content )
Test Requirement #2 (Packaging)
Functional Requirement #4 ( SOAP message in root of MME doc )
Functional Requirement #5 ( MIME message type is “text/xml” )
Functional Requirement #6 ( MIME ‘start’ header is present )
…
Test Requirement #3 (Core Extension Elements)
Test Suite XML Document
Test Driver Configuration Data
XML Payloads
Test Cases
Test Case #1 (Test Valid “To content)
Message

Figure 19 – Test Framework documents and their relationships

# 7 Test Cases

## 7.1 Structure of a Test Case

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

Test Requirement reference.

A workflow of Test Steps and/or Test Threads

Testable precondition(s) and assertion(s) of success or of failure within those Test Steps.

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

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

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

The last executed “TestAssertion” operation in the workflow evaluates to “true” (or pass).

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

Any TestAssertion that evaluates to a result of "false", **and from which no further workflow execution can occur** (i.e. no branching is possible) based upon its boolean result causes the Test Driver to cease execution of the Test Case, and report a final result of the Test Case as “fail”. In the case of a “false” result in an asynchronous Thread that contains that TestPreCondition, the execution of all other concurrent Threads belonging to that Split MUST also complete so that it can be determined if workflow execution may continue based upon the associated Join operation. . If the Thread in question is not subsequently Joined in the workflow, then the Test Case execution ceases with a final result of “undetermined”.

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

Any TestPreCondition operation that evaluates to a result of "false", **and from which no further workflow execution can occur** (i.e. no branching is possible) based upon its boolean result causes the Test Driver to cease execution of the Test Case, and report a final result of the Test Case as “undetermined”. In the case of a “false” TestPreCondition result within an asynchronous Thread, the execution of all other concurrent Threads belonging to that Split MUST also complete so that it can be determined if workflow execution may continue based upon the associated Join operation. If the Thread in question is not subsequently Joined in the workflow, then the Test Case execution ceases with a final result of “undetermined”.

OR

A TestPreCondition was the final test operation in the workflow ( not a TestAssertion )

OR

Asystem exception condition (as defined for each individual operation) occurs in the Workflow. For example, a protocol error occurring in a PutMessage or GetMessage operation will generate such an exception.

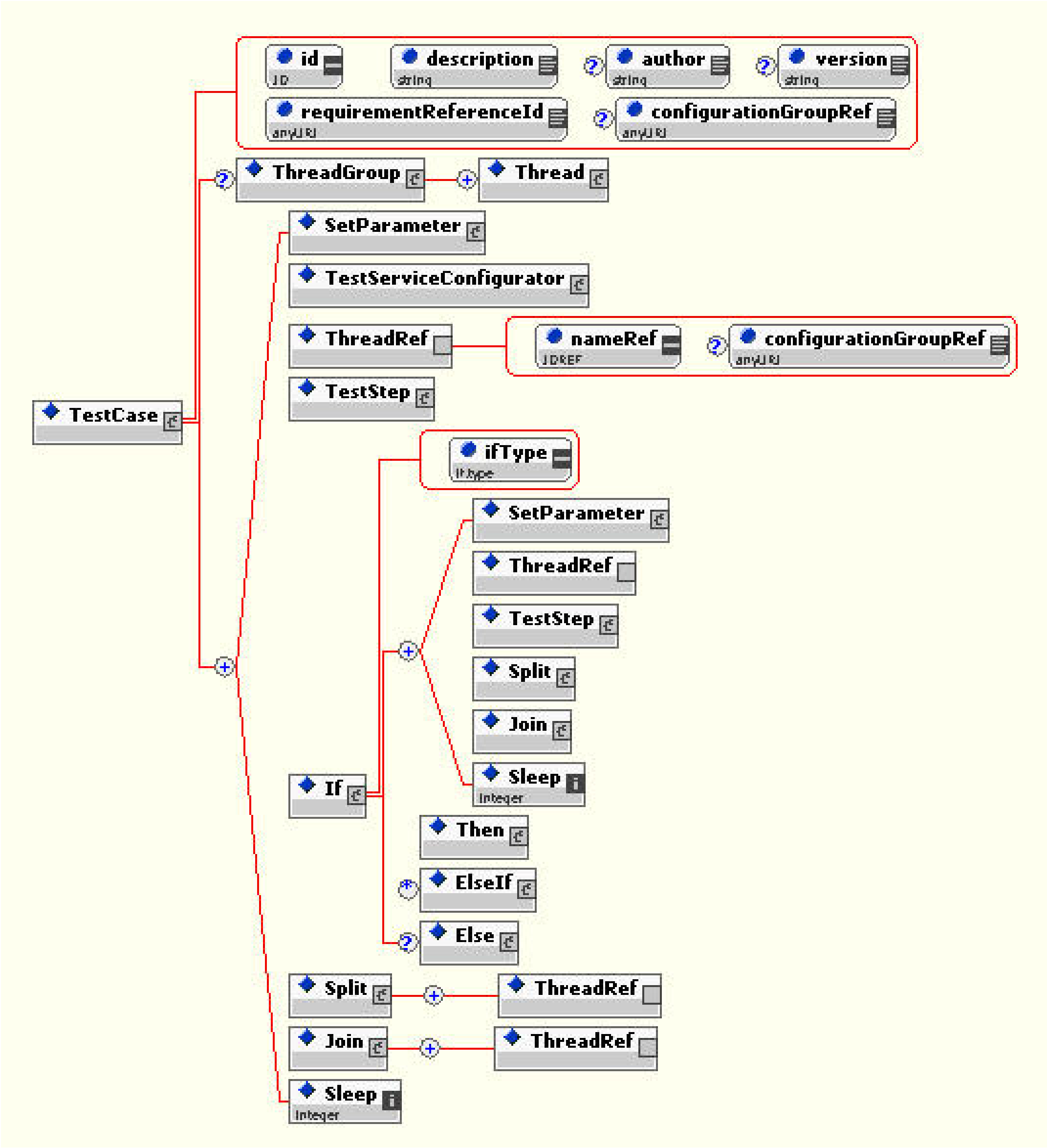

Figure 20 – Graphic representation of expanded view of the TestCase element

Definition of Content

| Name | Description | Default Value From Test Driver | Required/Optional | Exception Condition |
|---|---|---|---|---|
| TestCase | Container element for | | Optional | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | all test case content |  |  |  |
| 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 |  |
| requirementReferenceId | Pointer to the unique ID of the FunctionalRequiremt |  | 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 Test Steps and/or Threads that may be forked synchronously or asynchronously |  | Required |  |
| SetParameter | Contains  name/value pair to be used by subsequent Threads or Test Steps 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 |
| TestStep | Container for send, receive and message verification operations, executed in the same program space as the current thread ( not “forked” ) |  | Required |  |

|  |  |  |  |  |
|---|---|---|---|---|
| If | Branching mechanism for test script, with “andif” or “orif”  option. Boolean result value of Threads and Test Steps determine logical flow |  | Optional |  |
| ifType | Attribute that determins the predicate of the If statement ( either “andif” or “orif” | IfType=”andif” | Required |  |
| Then | Branching mechanisms for test script if the result of If is “true” |  | Required |  |
| Elseif | Additional branch if result of If clause is “false” |  | Optional |  |
| Else | Additional branch if result of If and all ElseIfs is “false” |  |  |  |
| 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 to “wait” for the specified time interval (in seconds). May be invoked anywhere in the script |  | Optional |  |

Table 10 provides a list of TestCase element and attribute content

### 7.1.1 Test Threads

Test Threads are a workflow of Test Steps and/or other sub-threads.   One can think of a Thread as a collection of “related” operations (such as a sequences of Test Steps performing message transmissions and receptions for a common business process).  Test Steps (single send/receive operations) and sub-threads contained in a Test Thread are executed sequentially as they appear in that Thread script.

Thread MAY be executed either serially or in parallel.

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

Threads MUST execute concurrently if its ThreadRef is the child of a Split element.

Threads MAY be “joined” anywhere following the Split in which they were executed.

A Join operation “synchs” the execution of the Test Case, waiting until all Threads defined as children within the Join complete execution. When that occurs, the Join operation will evaluate the boolean result of the Thread(s) to determine if they meet the condition for an “andJoin” (all Threads evaluate to “true”).

An “orJoin” operation is not defined for the Join operator, however this operation can be achieved through the use of the If/Then/Else scripting construct.

Threads defined in a Split MAY NOT necessarily be “joined” at all in a Test Case. This is permissible. However, if a Test Case is scripted in this way, the “lone” Thread(s) MUST evaluate to a boolean result of “true” (pass) in order for the Test Case result to be “pass”.

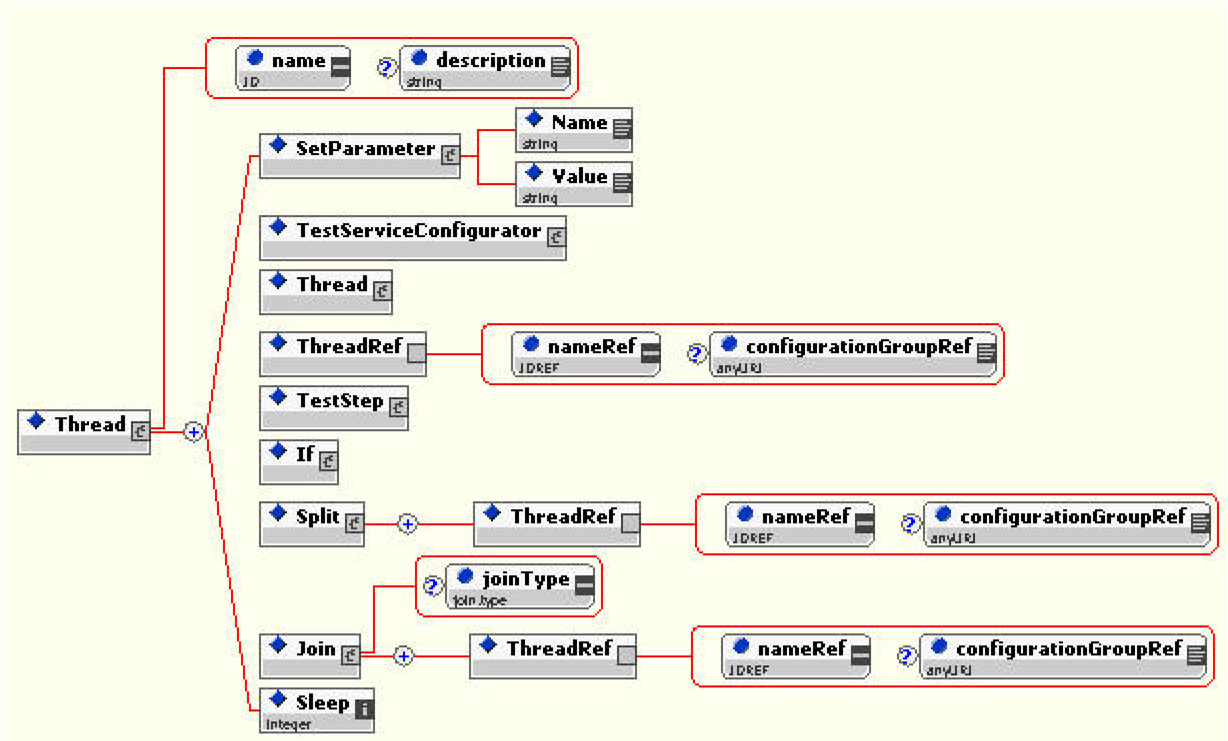

Figure 21 – The Thread content diagram

| 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 Test Step operations | | Optional | |
| ThreadRef | Reference via name to Thread to execute serially | | Optional | Thread not found |
| Thread | Instance of a Thread to be executed synchronously | | Optional | |
| TestStep | A message send/receive operation that is executed in the same program space as the current Thread | | Optional | |
| Split | Directive to run the referenced Thread(s) enclosed in the Split element in parallel | | Optional | Thread not found |

|  |  |  |  |  |
|---|---|---|---|---|
| ThreadRef | Reference via name to Thread to execute concurrently |  |  | Thread not found |
| Join | Directive to evaluate the boolean result of the enclosed referenced Thread(s) in a previous Split |  | Optional | Thread not found |
| joinType | Type of Thread join.. (andjoin \| orjoin ) | andjoin | Optional |  |
| Sleep | Instrution to “wait” (specified in integer seconds) a period of time before executing the next operation in the script |  | Optional |  |

Table 11 – Thread Content Description

## 7.1.2 Test Steps

Test Steps perform a single operation. These operations include message construction and transmission or message verification/validation. Each of these operations may be performed by the Test Driver in one of two modes: connection (Test Driver is remote from Test Service) or service (Test Driver is interfaced with Test Service). The section below describes these operations and their modes in more detail.

### 7.1.2.1 Test Step Operations

#### 7.1.2.1.1 Message Construction and Transmission

**In Connection Mode:**

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

The “Initiator” Test Step operation instructs the Test Driver to pass a Message Declaration (and any required message payloads) to the Test Service via RPC call to the Test Service Initiation interface, via its “initiator” method.  The initiator method must successfully interpret the Message Declaration; construct the message with a new ConversationId (if none is present in the Message Declaration). The Test Service initiator method must return a response message (defined in Appendix F) to the Test Driver indicating success or failure.

**In Service Mode:**

The “PutMessage” Test Step operation instructs the Test Driver to pass on a Message Declaration and accompanying message payload data to the Test Service Send interface and its “initiator” method. Because the Test Service is essentially “stateless” except for its 3 configuration parameters, any Message Declaration sent by the Test Driver MUST contain the required information (i.e. FromPartyId, ToPartyId, CPAId ..etc.)    One exception is the ConversationId. The Test Driver MAY provide its own ConversationId in the Message Declaration.  Otherwise, the initiator method will provide one in the constructed message (therefore starting a new conversation). ). The Test Service initiator method must return a response message to the Test Driver indicating success or failure.

The “Initiator” operation is not used by the Test Driver in “service” mode, since the PutMessage operation invokes the same method (initiator) of the Test Service Send interface, regardless of whether it is controlling the ConversationId or letting the Test Service initiate the conversation.

#### 7.1.2.2 Message Verification/Validation

**In Connection Mode:** The “GetMessage” Test Step operation is used by the Test Driver to verify and/or validate incoming messages to the Test Driver. Incoming messages for a Test Case are maintained in a persistent Message Store for the life of a Test Case. Message content is filtered and verified via [XPath] and validated via [XMLSchema]. Success or failure of the GetMessage operation is based upon the success/failure of TestAssertion operations that verify or validate message content based upon XPath query results, or XML schema validation respectively.

**In Service Mode:** The “GetMessage” Test Step operation is used by the Test Driver to verify and/or validate incoming messages to the Test Service that are passed to the Test Driver via its Receive interface, and its “notify” method. Incoming messages for a Test Case are maintained in a persistent Message Store for the life of a Test Case. Message content is filtered and verified via [XPath] and validated via [XMLSchema]. Success or failure of the GetMessage operation is based upon the success/failure of TestAssertion operations that verify or validate message content based upon XPath query results, or XML schema validation respectively.

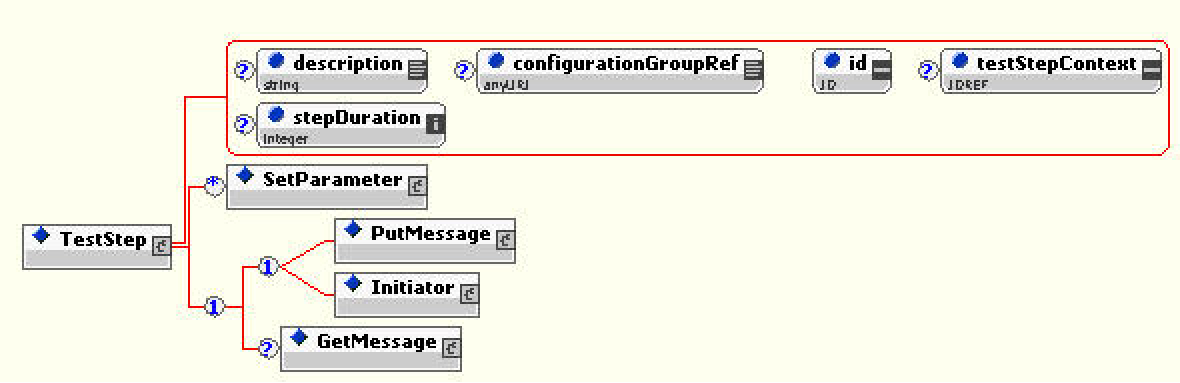

Figure 22 – Graphic representation of expanded view of the TestStep element

Definition of Content

| **Name** | **Description** | **Default Value From Test Driver** | **Required/Optional** | **Exception Condition** |
|---|---|---|---|---|
| TestStep | Container for one of four possible operations | | Optional | |
| description | Short description of Test Step | | Optional | |
| configurationGroupRef | Reference to existing ConfigurationGroup to change current Test Driver configuration for this Test Step | | Optional | Configuration Group not found |
| id | ID of the test step | | Required | |
| testStepContext | Use CPAId, ConversationId, MessageId and RefToMessageId from previous integer step number indicated | | Optional | Test Step ID not found |
| stepDuration | Override the default maximum execution durationof this Test Step | Taken from current ConfigurationGroup value | Optional | |

| | | | | |
|---|---|---|---|---|
| SetParameter | Optionally define a parameter name and value that will be used by the PutMessage or GetMessage operation | | Optional | |
| PutMessage | Directive to construct and send a message | | Optional | |
| GetMessage | Directive to retrieve messages from Message Store in their entirety | | Optional | |
| Initiator | Initiate a message on the local or remote Test Service | | Optional | Test Service is unable to construct or transmit message |

Table 12 provides a list of TestStep element and attribute content

7.1.2.3 Semantics of the SetParameter Operation in a Test Step

Parameters can be assigned for use by Test Step operations in three ways:

2) Through assignment as a parameter name/value pair within the current ConfigurationGroup.

3) Using SetParameter at the beginning of a Test Step

4) Using SetXPathParameter operation in a GetMessage operation (to extract a message content value via XPath and assign it to a parameter)

#### 7.1.2.3.1 Scope of a parameter

These same semantic rules apply to parameters referenced via **ConfigurationGroup.** The "configurationGroupRef" attribute is available for use at the TestSuite, TestCase, Thread and TestStep levels. A hierarchical relationship exists for any parameters defined in the ConfigurationGroup. A configurationGroupRef at the TestSuite level is "global", meaning any parameter definitions defined at the

TestSuite level are available to descendent TestCase, Thread, or TestStep. If a parameter is “redefined” through a reference to another ConfigurationGroup at any of those “lower levels” in the object hierarchy, then that definition takes precedence for that object and any “descendent” objects, until the logical workflow of the TestCase moves to a “higher” level in the object hierarchy. When that occurs, whichever previous definition of a parameter (via a configurationGroupRef or SetParameter operation) takes precedence.

The **SetParameter** operation dynamically creates (or redefines) a single parameter whose value is available to the current Test Object (TestCase, Thread or TestStep) it is defined in. For example, if it is defined within a Thread, then it is available to any operation in that Thread, as well as any descendent Test Steps or Threads of that current Thread. If it is defined within a Test Step, then its definition exists for that Test Step. . When the workflow execution moves to a “higher” level (i.e. to the Thread containing the TestStep) then that parameter **a)** ceases to exist if it was not already defined at a higher level in the workflow hierarchy or **b)** takes the previously value defined at the next highest level in the workflow hierarchy.

The **SetXPathParameter** operation (available within the GetMessage operation of a TestStep) dynamically creates (or redefines) a single parameter whose value is available to the current Test Object it is defined in (and optionally) to its parent Thread object IF the scope attribute is set to “parent”. By default, if the scope attribute is not present the parameter’s scope is restricted to the current TestStep. The value of a parameter defined using this operation is retrieved from the message content defined in the XPath expression used in the operation.

#### 7.1.2.3.2 Referencing/Dereferencing parameters in PutMessage and GetMessage operations

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

In the case of a **GetMessage** operation, a parameter defined with the ConfigurationGroup and/or the SetParameter operation can be passed to the XPath processor used for the Filter or VerifyContent operations. Within the XPath expression, the parameter MUST be referenced with the same name (case sensitive) with which it has been assigned, and MUST be preceeded by a ‘$’ character. The Test Driver MUST recognize the parameter within the XPath expression, and substitute its value prior to evaluating the XPath expression

How parameters are stored and retrieved by the Test Driver is an implementation detail, however to accommodate implementations that may wish to store parameters within the Message Store, the Message Store schema in Appendix D provides a structure that permits DOM access to parameters through a simple name/value pair schema. Parameter values are stored under the MessageStore root element as children of a ParameterGroup element whose ID is a unique value defined by the Test Driver at run time, defining the context in which the parameter was defined (for example, a unique ID assigned to each test object instance (e.g. a Test Step) could be used to bind the current parameter definition within that test object, and disassociate it from other definitions in concurrently running Threads). How unique ParameterGroup ID's are defined is an implementation detail.

.

#### 7.1.2.4 The PutMessage Operation

The PutMessage operation instructs the Test Driver to build and send a message (if the Test Driver is in "connection" mode) or to pass a Message Declaration to the Test Service (if the Test Driver is in "service" mode) . A minimal Message Declaration (contained within its child MessageDeclaration element) is required to construct a message and an optional XSL stylesheet or Xupdate document MAY mutate that message declaration. Dynamic message content such as timestamps, message ids and conversation ids are passed to the XSLT or Xupdate processor through parameters created by the Test Driver. Additional message content may be passed to the XSLT or Xupdate processor through parameter definitions defined by the test writer (using the configurationGroupRef attribute or the SetParameter directive to define a name/value pair).

An important difference between the functionality of the PutMessage operation when in "connection" vs. "service" mode is the role of the Mutator element. If the Test Service is in "connection" mode, the Mutator (and its XSL stylesheet or Xupdate document) "mutates" the Message Declaration into a message envelope suitable for transmission to an MSH. If the Test Service is in "service" mode, the Mutator mutates the Message Declaration, but does not build an actual message. Instead, it simply modifies the Message Declaration, supplying message content via parameter definitions. It DOES NOT construct a message. That is the responsibility of the "initiator" method of the Test Service. The initiator method reads the Message Declaration provided by the Test Driver and actually constructs a message based upon that template.

What this means is that a Test Case designed for use by a Test Driver in "connection" mode will not function properly if the Test Driver is switched to "service mode" because the functionality of the Test Driver is now turned over to the Test Service, using the API of the MSH under test. A new Mutator stylesheet or Xupdate document MUST be used if one wishes to switch Test Driver modes.

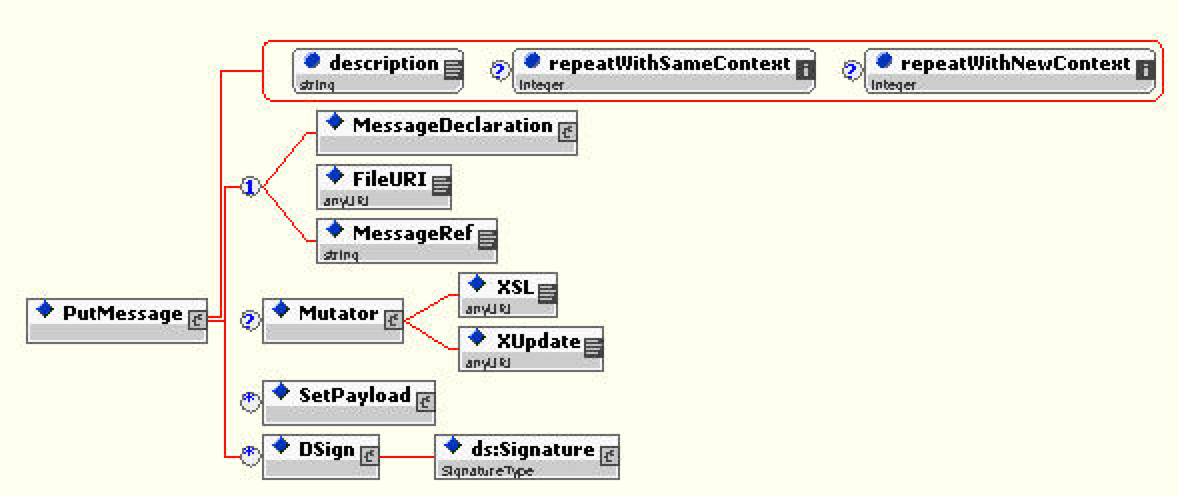

Figure 23 – Graphic representation of expanded view of the PutMessage element

7.1.2.5

**Definition of Content**

| **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 ( i.e. ConversationId, MessageID, Timestamp ) | | Optional | |
| repeatWithNewContext | Integer looping parameter, using new message context ( i.e. | | Optional | |

| | | | | |
|---|---|---|---|---|
| | ConversationId, MessageID, Timestamp ) | | | |
| MessageDeclaration | 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 MessagePayload element | | Optional | |
| Mutator | Container element for a reference to either an XSL stylesheet or Xupdate document | | Optional | |
| XSL | URI to an XSL stylesheet | | Optional | Stylesheet not found |
| XUpdate | URI to an Xupdate document | | Oprional | Xupdate script not found |
| XUpdate | URI reference to Xupdate document | | Optional | |
| SetPayload | Container element for Test Driver directives to create MIME attachments (or Payloads) to message | | Optional | |
| DSign | Container element for XML Digital Signature declaration(s) for this message, used to sign any portion (envelope or payload(s)) of the message | | Optional | |

Table 13 defines the content of the PutMessage element

The MessageDeclaration element is a container element for XML content describing the construction of the envelope portion of a message. The XML content necessary to describe a basic message should be minimal, with default parameter values supplied by the Test Driver for most message content. If the test developer wishes to “override” the default element and attribute values, they may do so by explicitly declaring those values in the XML markup.

Default values for element and attribute content may come from two sources. Either the Test Driver/MSH generates that value, (for dynamic message content such as for a message Timestamp or a Message Id), or the value is declared in the ConfigurationGroup parameters described in section 4.2.1, which are passed through to the XSL or Xupdate processor for inclusion in the message.

For ebXML Messaging testing, message content parameters required for the ConfigurationGroup content include:

CPAId

Service

Action

SenderParty

ReceiverParty

Dynamic message content parameters defined at “runtime” by the Test Driver include:

ConversationId

MessageId

Timestamp

As an example, a Test Suite ConfigurationGroup element content could be:

```
<ebTest:ConfigurationGroup ebTest:id="mshc_basic">
  <ebTest:CPAId>mshc_basic</ebTest:CPAId>
  <ebTest:Mode>connection</ebTest:Mode>
  <ebTest:SenderParty>urn:oasis:iic:testdriver</ebTest:SenderParty>
  <ebTest:ReceiverParty>urn:oasis:iic:testservice</ebTest:ReceiverParty>
  <ebTest:Service>urn:ebXML:iic:test</ebTest:Service>
  <ebTest:Action>Dummy</ebTest:Action>
  <ebTest:StepDuration>10</ebTest:StepDuration>
  <ebTest:Transport>HTTP</ebTest:Transport>
  <ebTest:Envelope>ebXML</ebTest:Envelope>
</ebTest:ConfigurationGroup>
```

If the message being constructed is an ebXML message used with an HTTP or SMTP transport, and utilizing SOAP messaging, the XML schema defined in Appendix C (and illustrated below) MUST be used to construct a valid Message Declaration:

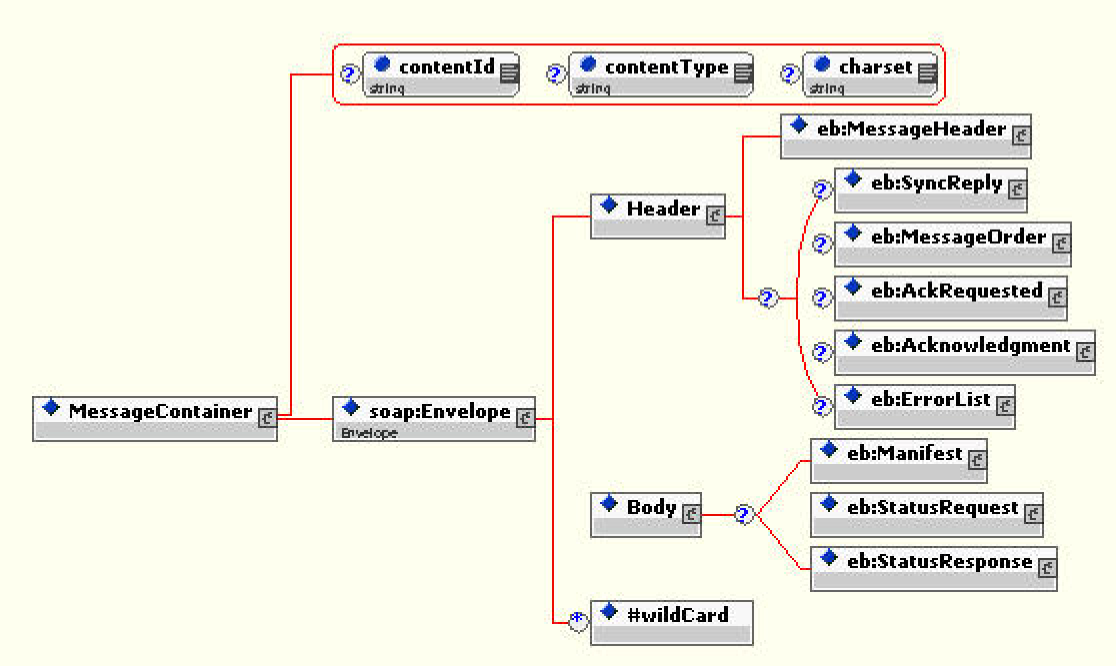

Figure 24 – Image of MessageContainer Content

### 7.1.2.5.1 Schema for ebXML Message Declaration using SOAP

**MIME header data:** MIME headers MUST be created or modified using the declarative syntax described in the mime.xsd schema in Appendix X. Default message MIME header data is illustrated in the message envelope template in section 7.1.6. How the MIME headers are actually constructed is implementation dependent. Test Drivers operating in “service” mode MAY ignore the MIME portion of a Message Declaration, since message MIME manipulation may be unavailable at the application level interface used for a particular ebXML MSH implementation. Test drivers in “connection” mode MUST properly interpret the MIME portion of a Message Declaration and generate the appropriate MIME header information.

**SOAP header and body data:** SOAP message content MUST be created or modified using the MessageDeclaration content syntax described in the soap.xsd schema described in Appendix E. Default message SOAP content is illustrated in the message envelope template in section 7.1.6. Test Drivers operating in “service” mode MAY ignore the SOAP portion of a MessageDeclaration, 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 Message Declaration and generate the appropriate SOAP header information.

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

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

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

```
<eb:TestMessageDeclaration>
 <mime:Message>
  <mime:MessageContainer>
   <soap:Envelope>
    <soap:Header>
     <eb:MessageHeader/>
    </soap:Header>
    <soap:Body/>
   </soap:Envelope>
  </mime:MessageContainer>
 </mime:Message>
</eb:TestMessageDeclaration>
```

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

```
Content-Type: multipart/related; type="text/xml"; boundary="boundaryText";
start=messagepackage@oasis.org

--boundaryText
```

```
Oeh-     m    essaeackeOsing
Oeh-     m·     text/xml·  charset= UTF-8"

<san  En           htt
          xhtt hem-                                   tane"
          httchemsaneaea/
          ehttpssi-                     pogiteese                 sgchemsg
headr0    x           xschemaathttchemsaneae/

httssi-          enoiteese                 schemeexsd

httpssi-         pogiteesesgchemsg                          headr2d

http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2.0.xsd">
<san  Hadr>
   eMssaeHadr         san  mtHrstad        1" ebrs         2.0">
          eFm
                 ePrtd       >urn:oasis:iic:testdriver</Prtd
          </Fm
          eT
                 ePrtd        urn:oasis:iic:testservice</Prtd
          </T
          e              mshc_basic</
          eCrsatd               987654321</Crsatd
          eSrve>        urn:ebXML:iic:test</Srve>
          eAtn         Dummy</Atn
          eMssaeDta>
                 eMssaed        0123456789</Mssaed
                 eTstam         2000-07-25T12:19:05</Tstam          MssaeDta>
   </MssaeHadr>
</ san  Hadr>
</ san  En>
```

Dynamic ebXML message content values (illustrated in red above) are supplied by the Test Driver.

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

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

#### 7.1.2.5.2 Interpreting the MIME portion of the Message Declaration

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

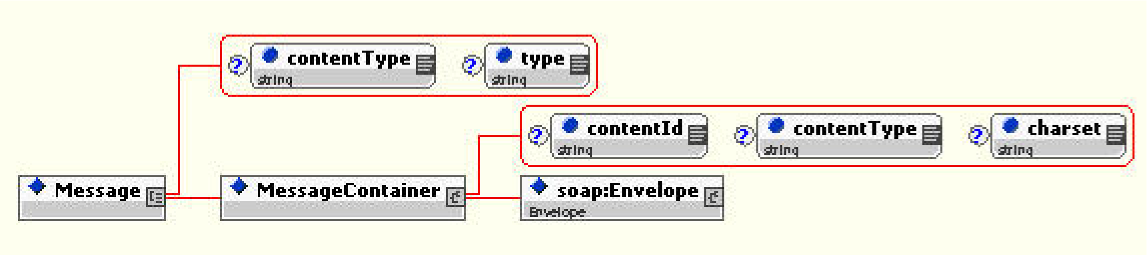

Figure 25 – Graphic representation of expanded view of the MessageDeclaration element

Definition of Content

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

Table 14 defines the MIME message content of the MessageDeclaration element

An Example of Minimal MIME Declaration Content

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

```
<ebMessag          =" httpssiongceh                    ctestg          ">
  <ebMessageName             />
</ ebMessag     >
```

### 7.1.2.5.3 Interpreting the SOAP portion of the ebXML Message Declaration

The XML syntax interpreted by the Test Driver to construct the SOAP message content consists of the declaration of a SOAP Envelope element, which in turn is a container for the SOAP Header, Body and non-SOAP XML content.  Construction of the SOAP Header and Body content is simple for the Test Driver, requiring only the creation of the two container elements with their namespace properly declared, and valid according to the [SOAP]. The Test Driver only constructs the SOAP Body element if it is explicitly declared in the content.

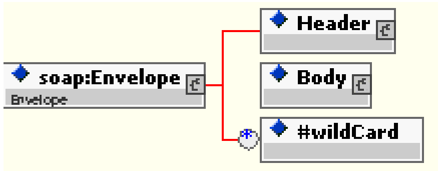

Figure 26 – Graphic representation of expanded view of the soap:Envelope element declaration

Definition of Content

| **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 | Optional |

|  |  |  |  |
|---|---|---|---|
|  |  | time |  |
| #wildCard | Generate “inline” wildcard content inside SOAP Envelope |  | Optional |

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

An Example of Minimal SOAP Declaration Content

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

```
<soap:Envelope >
  <soap:Header />
</soap:Envelope>
```

### 7.1.2.5.4 Interpreting the SOAP Header Extension Element Declaration

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

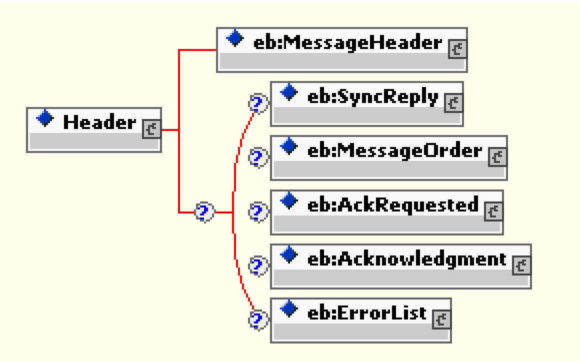

Figure 27 – Graphic representation of expanded view of the soap:Header element declaration

Definition of Content

| **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 |

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

#### 7.1.2.5.5 Interpreting the ebXML MessageHeader Element Declaration

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

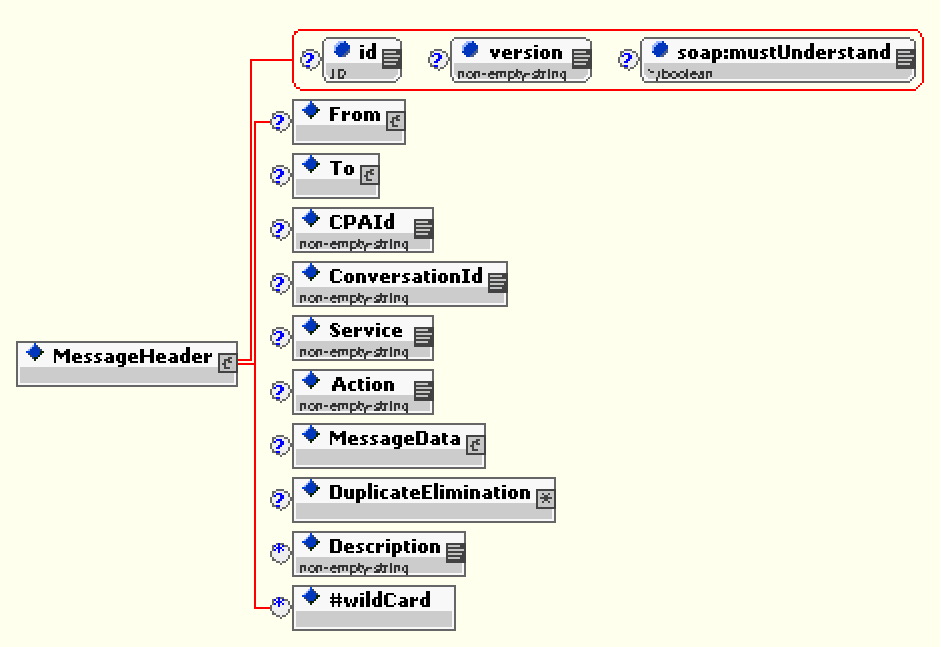

Figure 28 – Graphic representation of expanded view of the ebXML MessageHeader element declaration

Definition of Content

| 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 declared value | | Optional | |
| version | Modify default attribute value | 2.0 | Optional | |
| soap:mustUnderstand | Modify default attribute value | true | Optional | |
| From | Modify default From message element generated by Test Driver | Generated by Test Driver/MSH at run time | Optional | |
| PartyId | Replace default element value with new value | Generated by | Required | |

| | | | | |
|---|---|---|---|---|
| | | Test Driver/MSH at run time, using config value | | |
| 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 Driverat 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 | |

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

An Example of a Minimal ebXML MessageHeader Content Declaration

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

```
<eb:MessageHeader/>
```

#### 7.1.2.5.6 Interpreting the ebXML ErrorList Element Declaration

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

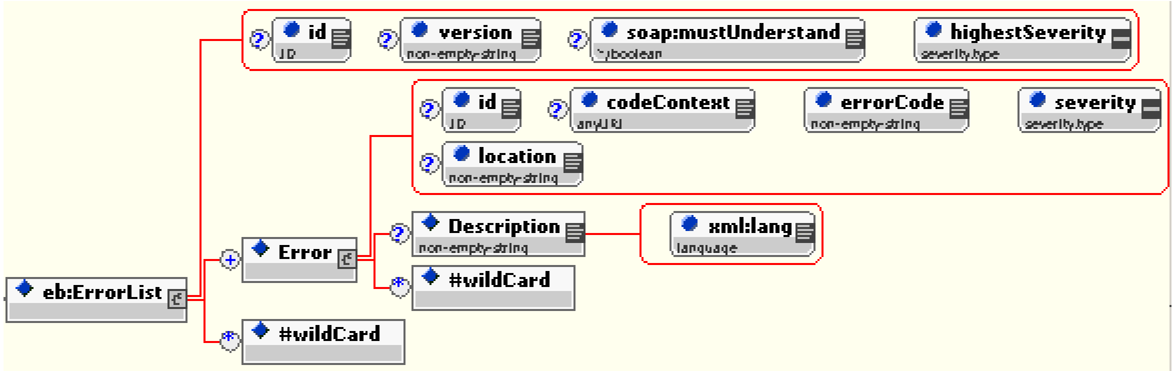

Figure 29 - Graphic representation of expanded view of the ebXML ErrorList element declaration

Definition of Content

| 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 | | | Optional | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | Generate element with declared value |  |  |  |
| #wildCard | Generate content “inline” into message |  | Optional |  |

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

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.

```
ebErroList        ebhigestSeverityErro>
   <ebErro      ebrrorCodeMimeheader”                 ebeverityErro/>
</ebErrorList>
```

### 7.1.2.5.7 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.

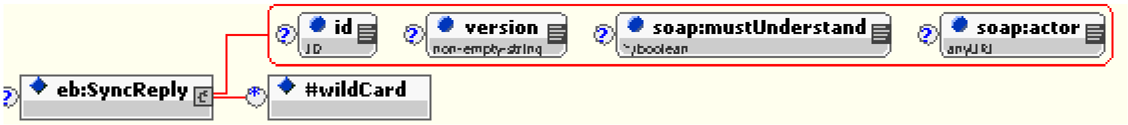

Figure 30 – Graphic representation of expanded view of the ebXML SyncReply element declaration

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

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

An Example of a Minimal ebXML SyncReply Content Declaration

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

```
[illegible]
```

#### 7.1.2.5.8 Interpreting the ebXML AckRequested Element Declaration

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

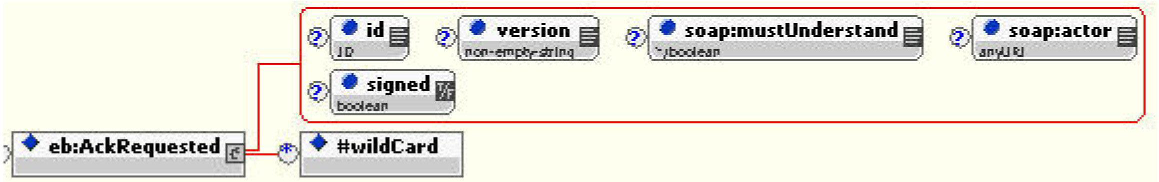

Figure 31 – Graphic representation of expanded view of the ebXML AckRequested element declaration

Definition of Content

| **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 | |

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

An Example of a Minimal ebXML AckRequested Content Declaration

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

```
eARqsted/
```

### 7.1.2.5.9 Interpreting the ebXML Acknowledgment Element Declaration

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

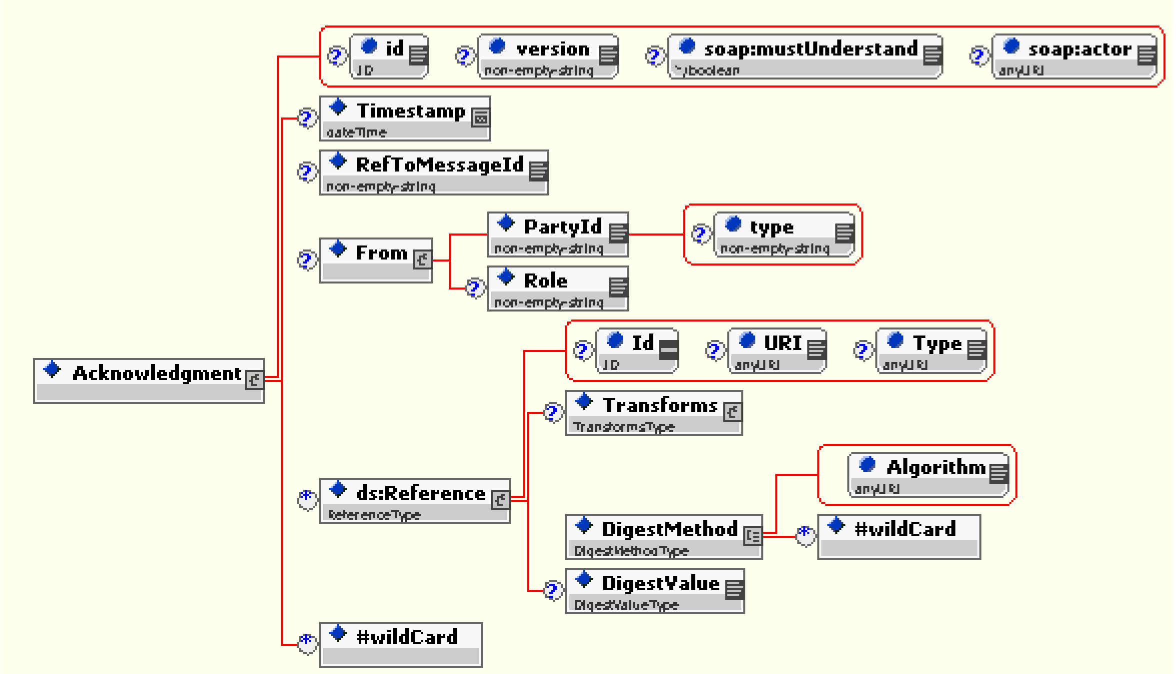

Figure 32 – Graphic representation of expanded view of the ebXML Acknowledgment element declaration

Definition of Content

| 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 run time | Optional | |
| RefToMessageId | Modify default element value | Generated by Test Driver at run time | Optional | |
| From | Modify default container | Generated by Test Driver at run time | Optional | |
| PartyId | Modify default value | urn:ebxml:iic:testdriver | Required | |
| type | Generate type attribute with value | | Optional | |
| Role | Generates a Role element with its value | | Optional | |
| ds:Reference | Generate container element and all default content | | Optional | |
| Id | Generate attribute and its value | | Optional | |
| URI | Modify default attribute value | “” | Required | |
| type | Generate attribute and its value | | Optional | |
| Transforms | Generate container relement | | Optional | |
| Transform | Generate element with its value | | Optional | |
| Algorithm | Modify default attribute value | http://www.w3.org/TR/2001/REC-xml-c14n-20010315 | Required | |
| #wildCard | Generate content “inline” | | Optional | |
| XPath | Generate element with its value | | Optional | |
| DigestMethod | Generate element with its value | | Required | |
| | Modify default attribute | Generated by | Required | |

|  |  |  |  |  |
|---|---|---|---|---|
| Algorithm | value | Test Driver at run time, based upon CPA |  |  |
| #wildCard | Generate content “inline” |  | Optional |  |
| DigestValue | Generate element with its value | Computed by Test Driver at run time | Required |  |
| #wildCard | Generate content “inline” |  | Optional |  |

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

An Example of a Minimal “unsigned” ebXML Acknowledgment Content Declaration

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

```
[illegible]
```

### 7.1.2.5.10 Interpreting the ebXML MessageOrder Element Declaration

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

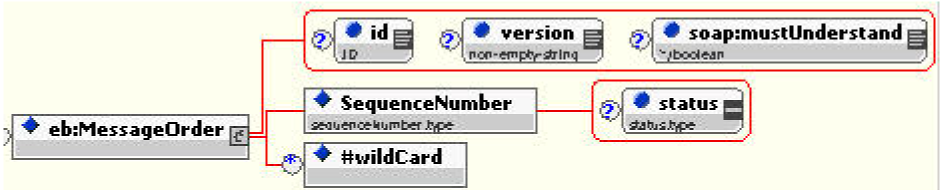

Figure 33 – Graphic representation of expanded view of the ebXML MessageOrder element declaration

Definition of Content

| 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 | |

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

An Example of a Minimal ebXML MessageOrder Content Declaration

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

```
<eMssagOdr>
<SeneNr<#SeneNr>
</eMssagOdr>
```

#### 7.1.2.5.11 Interpreting the SOAP Body Extension Element Declaration

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

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

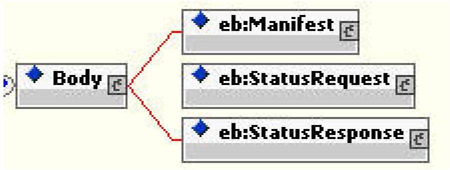

Figure 34 – Graphic representation of expanded view of the soap:Body element declaration

### 7.1.2.5.12 Interpreting the ebXML Manifest Element Declaration

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

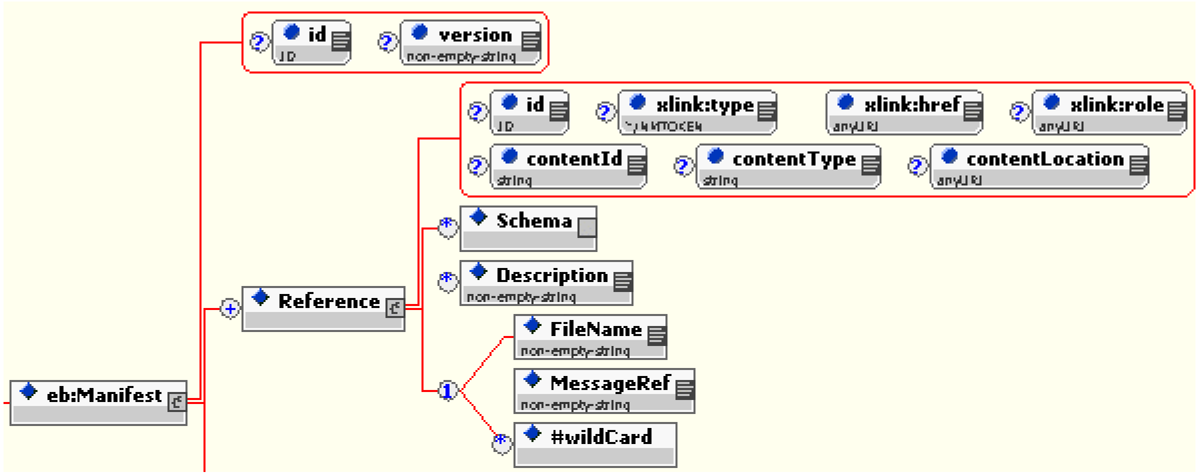

Figure 35 – Graphic representation of expanded view of the ebXML Manifest element declaration

Definition of Content

| 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 value | 2.0 | Optional | |
| id | Modify default attribute value | true | Optional | |
| xlink:type | Generate element with declared value | | Optional | |
| xlink:href | Generate attribute with declared value | | Required | |
| xlink:role | Generate attribute with declared value | | Optional | |
| contentId | Modify the Content-ID MIME header of the payload | | Optional | |
| contentType | Set the 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 | | Required | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | attachment |  |  |  |

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

An Example of a Minimal ebXML Manifest Content Declaration

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

```
eMfst>
eRfrene        href
/Mfst>
```

### 7.1.2.5.13 Interpreting the ebXML StatusRequest Element Declaration

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

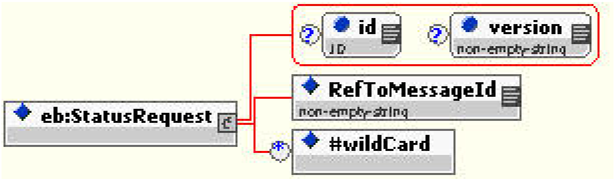

Figure 36 – Graphic representation of expanded view of the ebXML StatusRequest element declaration

Definition of Content

| 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 |  | Required |  |

|  |  |  |  |  |
|---|---|---|---|---|
|  | its value |  |  |  |
| #wildCard | Generate content "inline" |  | Optional |  |

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

An Example of a Minimal ebXML StatusRequest Content Declaration

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

```
<eb:StatusRequest>
<eb:RefToMessageId>[illegible]</eb:RefToMessageId>
</eb:StatusRequest>
```

### 7.1.2.5.14 Interpreting the ebXML StatusResponse Element Declaration

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

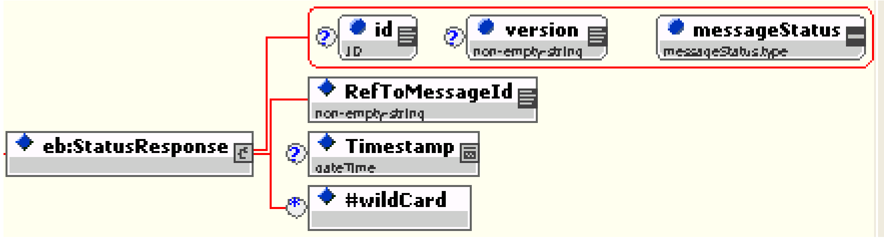

Figure 37 – Graphic representation of expanded view of the ebXML StatusResponse element declaration

Definition of Content

| Name | Declaration Description | Default Value From Test Driver | Required/Optional | Exception Condition |
|---|---|---|---|---|
| eb:StatusResponse |  |  | Optional |  |

|  |  |  |  |  |
|---|---|---|---|---|
|  | Generate container element and all default content |  |  |  |
| 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 |  |

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

An Example of a Minimal ebXML StatusResponse Content Declaration

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

```
<ebStatusRespe          messageStatusProessed/
```

### 7.1.2.6 The SetPayload Operation

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

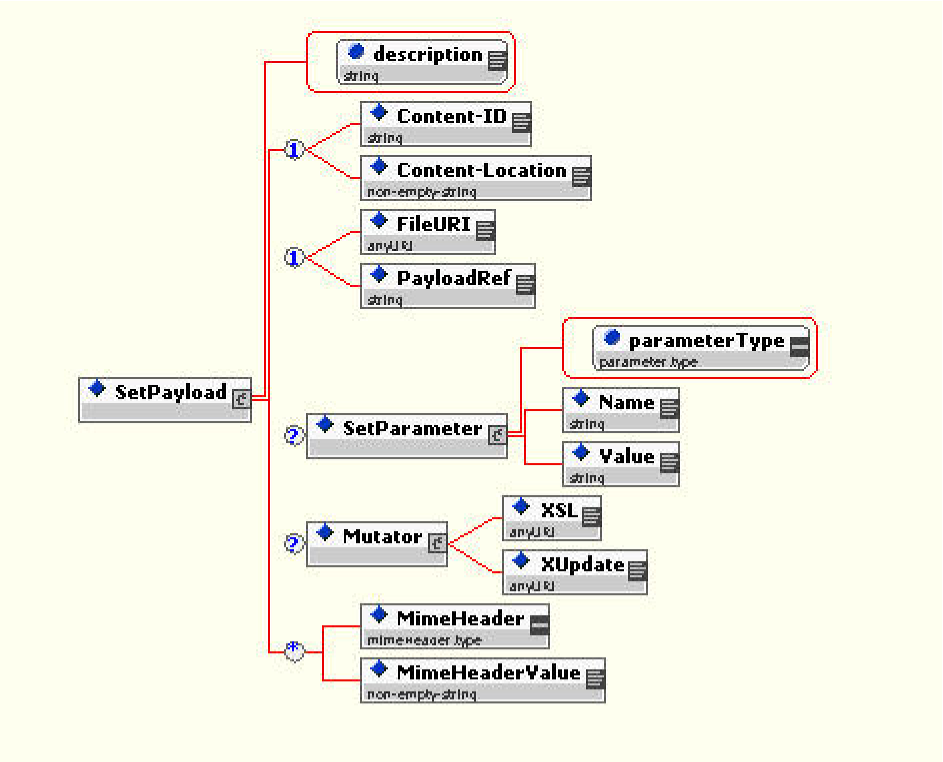

Figure 38 – Graphic representation of expanded view of the SetPayload element

Definition of Content

| Name | Description | Default Value | Required/Optional | Exception Condition |
|---|---|---|---|---|
| description | Metadata describing the nature of the SetPayload operation |  | Required |  |
| Content-ID | Set the Content-Id MIME header of the payload |  | Required |  |
| Content-Location | Set the the MIME Content-Location header of the payload |  | Required |  |

| | | | | |
|---|---|---|---|---|
| FileURI | URI of the file to be loaded as a payload | | Required | File not found |
| PayloadRef | Unique ID of the in memory XML document to be loaded as the payload | | Required | |
| MimeHeader | Set any type of MIME header name | | Optional | |
| MimeHeaderValue | Set corresponding MIME header value | | Optional | |
| SetParameter | Container for user-defined parameter to be made available to other Test Steps | | Optional | |
| Name | Name of new parameter | | Required | |
| Value | String value of parameter | | Required | |
| Mutator | Container Element for reference to either an XSL Stylesheet document or an Xupdate document for payload mutation | | Optional | |
| XSL | URI reference to XSL stylesheet | | Optional | Stylesheet document not found |
| XUpdate | URI reference to Xupdate document | | Optional | Xupdate document not found |

Table 26 defines the content of the SetPayload element

#### 7.1.2.7 The Dsign Operation

The DSign Operation is another sub-operation of PutMessage, and instructs the Test Driver to digitally sign the portion of the message identified by its Reference element content.

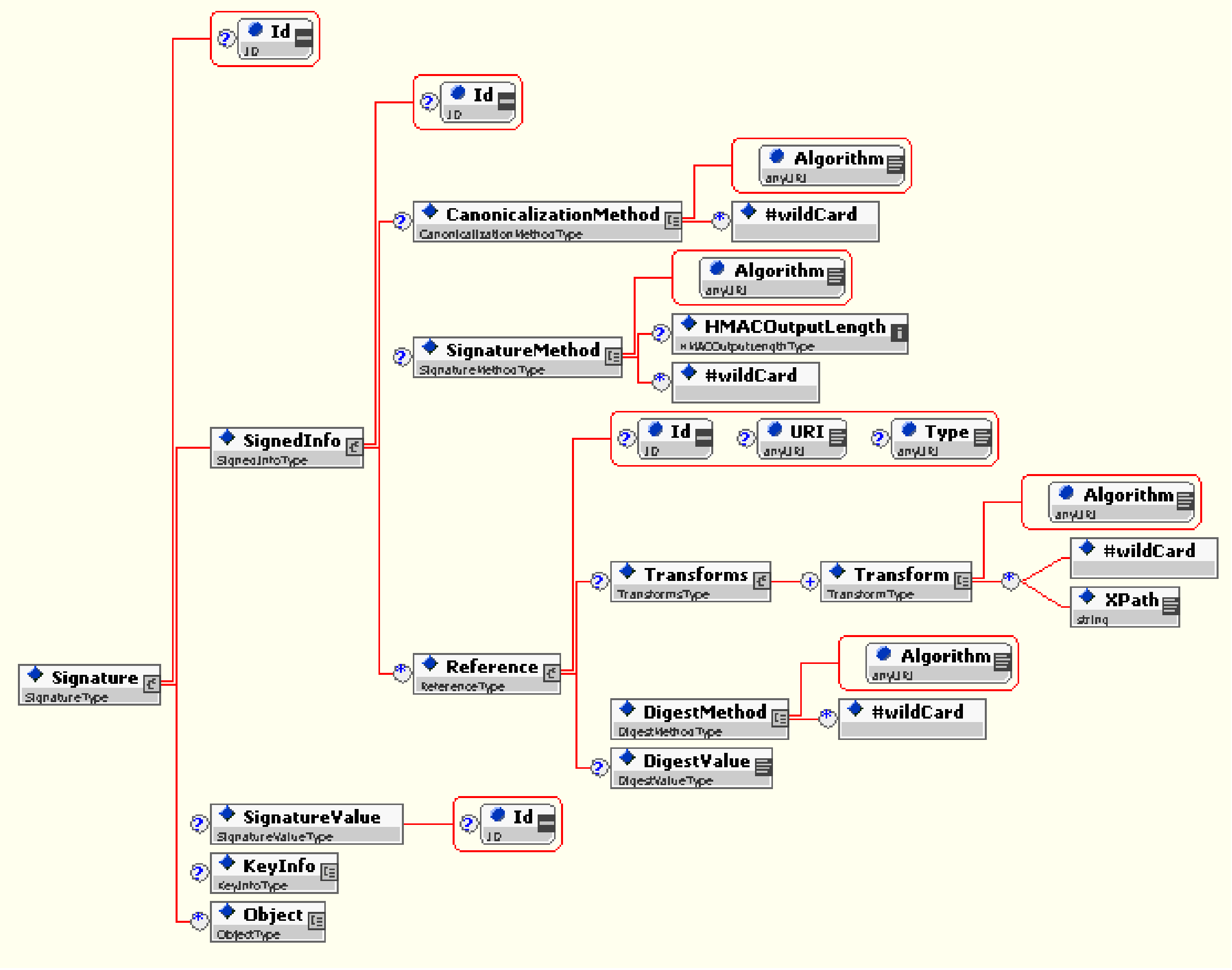

Figure 39 – Graphic representation of expanded view of the DSign element

### Definition of Content

| 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 Canonicalizatoin 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 | http://www.w3.org/TR/2001/REC-xml-c14n-20010315 | 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 | | Optional | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | value |  |  |  |
| 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 | 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 |  | Optional |  |

| | | | | |
|---|---|---|---|---|
| | element | | | |

Table 27 - Content of the Dsign element

#### 7.1.2.8 The Initiator Operation

The Initiator Operation provides a means to initiate a conversation from the candidate MSH. The Test Driver through the “Send” interface of the Test Service performs the Initiator operation. This is accomplished programmatically if the Test Driver is “local” to the Test Service. If this is not the case, then this is accomplished through a remote procedure call (RPC), described in section 3.2.4. The Test Driver passes on the XML content illustrated and described below to the Test Service “initiator” RPC method to construct a message. The type of content in the MessageDeclaration element will vary with the message envelope type (e.g. ebXML, RNIF..etc.). Also, because it is the Test Service that is actually constructing the message (not the Test Driver), message declarations MUST only contain directives that the MSH API can execute. For example MIME and SOAP content is generally not available for manipulation by an ebXML MSH API. Therefore, MIME and SOAP message construction directives SHOULD NOT be present as MessageDeclaration content, or if present, MUST be ignored by the initiator method of the Send interface.

The schema illustrating the MessageDeclaration content for ebXML Messaging Services v2.0 testing can be found in Appendix C.

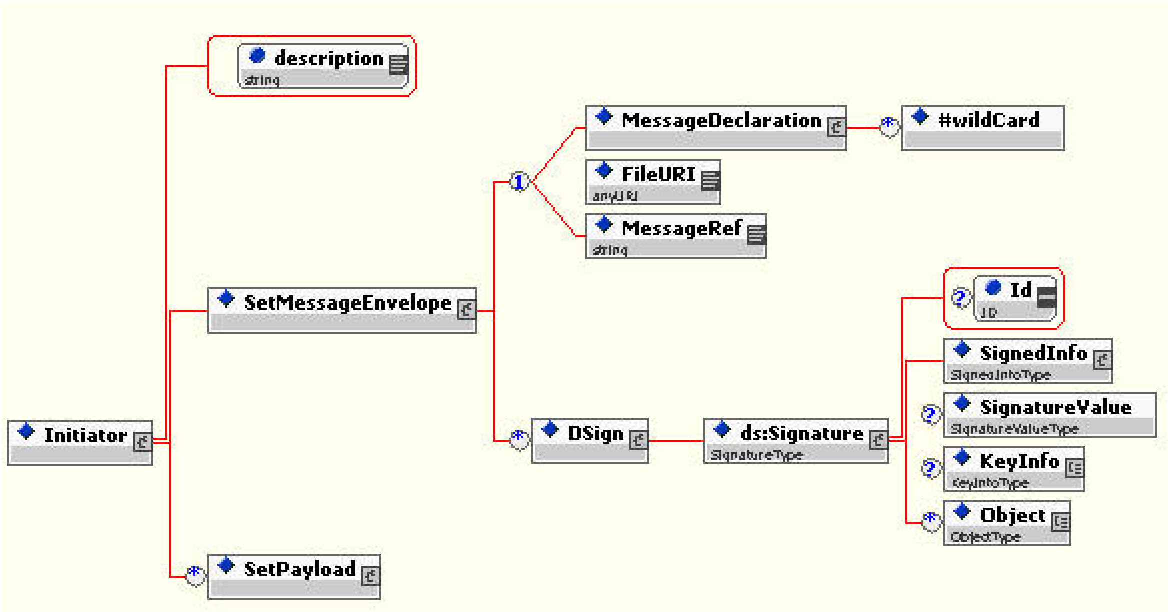

Figure 40 – Graphic representation of expanded view of the generic Initiator element

Definition of Content

| 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 | |
| MessageDeclaration | Message construction directives to be passed to MSH for interpretation and message generation | | 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 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 | |
| SetPayload | Container element for Test Driver directives to add MIME attachments (or Payloads) to message | | Optional | |

Table 28 – Content of the Initiator operation

### 7.1.2.9 The TestServiceConfigurator Operation

The TestServiceConfigurator operation provides a method to remotely reconfigure a Test Service. The Configurator operation is performed by the Test Service via the Configure interface (if the Test Driver is local to the MSH) or via a remote procedure RPC call to the Test Service configurator method (if the Test Service is remote). Details of the Configurator operation are described in section 3.2.4

The XML content illustrated and described below is passed from the Test Driver to the MSH to construct a message that contains all the necessary message information.

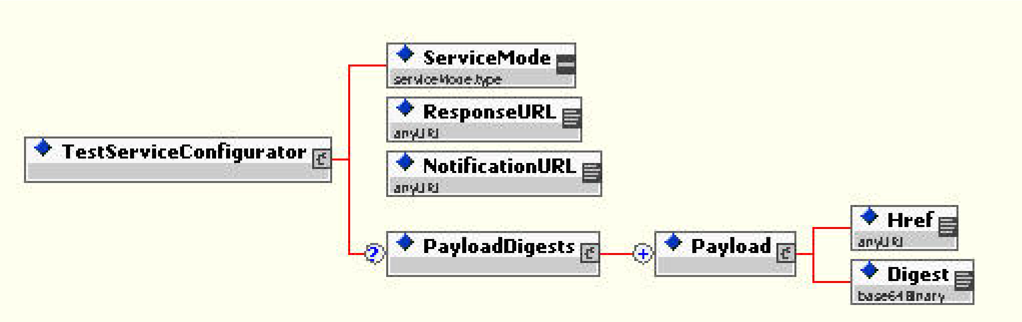

Figure 41 - Graphic representation of expanded view of the Configurator element

Definition of Content

| Name | Description | Default Value From Test Driver | Required/Optional | Exception Condition |
|---|---|---|---|---|
| Configurator | Container Test Driver/MSH configuration data | | Required | |
| OperationMode | One of three types, “local-reporting”, “remote-reporting” or “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 | |
| PayloadDigests | Container for individual payload identifiers, and their corresponding MD5 digest values | | Optional | |

| | | | | |
|---|---|---|---|---|
| Payload | Individual message payload date container | | Required | |
| Href | Identifier for payload in message | | Required | |
| Digest | MD5 digest value of payload | | Required | |

Table 29 – Content description for the TestServiceConfigurator operation

### 7.1.3 The GetMessage Operation

The GetMessage Operation, using its child XPath Filter operation, retrieves a node-list of Messages from the Message Store of the Test Driver. The content of the node-list is dependent upon the XPath Filter provided. The resulting node-list MAY be queried for adherence to a particular test Precondition or Test Assertion. In addition, any XML payload associated with a message MAY be queried in the same manner through the GetPayload sub-operation.

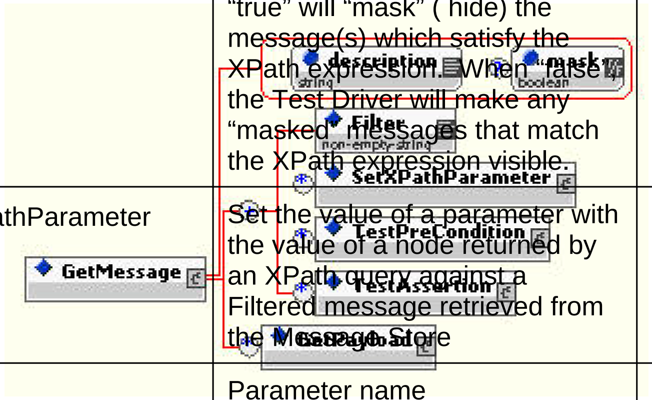

Figure 42 – Graphic representation of expanded view of the GetMessage element

| 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 | |
| Filter | XPath query to select message(s) from MessageStore | | Required | First element in the returned node list is not a Message element; Not a valid XPath or well formed XPath |
| mask | Boolean attribute, when set to “true” will “mask” ( hide) the message(s) which satisfy the XPath expression. When “false”, the Test Driver will make any “masked” messages that match the XPath expression visible. | false | 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 |
| Name | Parameter name | | Required | |
| Value | Parameter value | | Required | Not a valid XPath or well formed XPath |
| Type | Parmeter type (string \| namespace) | | Required | |
| TestPreCondition | Container for verification or validation operation to be performed on message as pre-condition to testing the Assertion | | Optional | Not a valid XPath or well formed XPath |
| TestAssertion | Container for verification or validation operation to be performed to test a conformance or interoperability assertion | | Optional | Not a valid XPath or well formed XPath |
| GetPayload | Retrieve a payload for the current message, and test its content using the same operations available at the GetMessage level | | Optional | Payload not found |

Table 30 defines the content of the GetMessage element

#### 7.1.3.1 Semantics of the GetMessage operation

A fundamental aspect of the GetMessage operation is its behavior and effect over the Message Store. The Message Store is an XML document object created by the Test Driver that contains an XML representation of all synchronous and asynchronously received ebXML messages for a Test Case. The received messages for a particular Test Case MUST persist in the Message Store for the life of the Test Case. Messages in the Message Store MAY contain an XML representation of all MIME, SOAP, ebXML or other types of message content, represented as an XML document (the schema permits any type of XML representation of a messaging envelope, with each representation specified in a "best practice" document for a particular testing community). The particular XML representation of a message in the Message Store is based upon a "best practice" schema for representing a particular message type. If the messages being stored are ebXML messages using HTTP transport and a SOAP envelope, the XML format of the Message Store document MUST validate against the ebXMLMessageStore.xsd schema in appendix D. The scope of message content stored in the Message Store is "global", meaning its content is accessible at any time by any Test Step or Thread (even concurrently executing Threads) during the execution of a Test Case. Message Store content changes dynamically with each received message or notification.

The GetMessage "Filter" operation queries the Message Store document object, and retrieves the XML content that satisfies the XPath expression specified in its Filter child element. As the MessageStore is updated every time a new message comes in, a GetMessage operation will automatically execute as often as needed, until either (1) its Filter selects and returns a non-empty node-list, or (2) the timeout (stepDuration) expires.

The XPath query used as content for a Filter operation MUST yield a node-list of Message elements, as defined by the Message Store schema in Appendix D. Although the content of a message may vary (e.g. ebXML, RNIF, SOAP), all node-list results from a Filter operation MUST contain Message elements. Any subsequent VerifyContent or ValidateContent operations MUST then append this node-list to a FilterResult root element, creating a new document object for further examination. The required structure of the FilterResult document object is defined in the Filter Result schema in Appendix D.

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

In addition to storing message content, the Message Store MAY also store parameter values to be used in the evaluation of subsequent received messages. This is not an implementation requirement, but an option. A schema defining a "ParameterGroup" tree for maintaining the state of parameter definitions within the Test Case object hierarchy is provided in the MessageStore sscheam defined in Appendix

Parameter values are defined in the “default” ConfigurationGroup of the Test Suite, and in any subsequent Test Cases, Threads or Test Steps that reference a ConfigurationGroup via its ID.   A test writer may additionally define  (or redefine) a parameter using the SetParameter element.  The SetParameter element requires three child elements to define a parameter:  Name, Value and Type. Once a parameter is defined, it may subsequently be used via XSLT or Xupdate in the PutMessage Mutator operation or GetMessage Filter operations (as arguments in an XPath expression).

Additionally, parameters may be defined/redefined through the SetXPathParameter operation.  This operation extracts message content from the Message Store and stores it as a parameter value. Whether it is a message header, or an XML message payload being examined, the test writer may assign a parameter name, and an XPath pointing to the content to be stored as a parameter.    XPath parameters are stored in the Message Store, as defined in the Message Store schema in Appendix D. Each parameter value is a string representation of the nodelist content retrieved by the XPath query.

A TestPreCondition and/orTestAssertion sub-operation (which may also contain a single XPath query) query the resulting document object constructed from the node-list generated by a GetMessage Filter operation. That document object to be examined by a TestPreCondition of TestAssertion action MUST have a root element with a name of test:FilterResult (as defined in the FilterResult schema in Appendix D).   If the Test Driver is unable to create a FilterResult document object because the resulting node-list contains any XML content other than Message elements (such as an attribute list), then the Test Driver MUST generate an exception, and terminate the Test Case with a result of "undetermined".  (i.e. if the XPath query does not return XML content that can be built into a document object, then it cannot be further examined with XPath in the TestPreCondition or TestAssertion operations)

**Message Masking:**

All the message items available for querying are children of the MessageStore element. The Xpath expression in the Filter will typically select all the /MessageStore/Message elements that satisfy the filter, and these will be consolidated as children of a FilterResult element, available for further querying, by the TestAssertion operation.

The messages that have been selected by a GetMessage operation are "invisible" to future Getmessage operations in the same test case, if the "mask" attribute is set to "true" (the default is "false"). A GetMessage operation will query all messages in the message store for this test case, except those messages that have been masked by a previous GetMessage operation.

How message masking takes place in a Test Driver is implementation specific.

### 7.1.4 The TestPreCondition Operation

The TestPreCondition Operation examines a message or messages in a GetMessage node-list by testing the content of the node-list against the VerifyContent (content value comparison) or ValidateContent (content integrity evaluation) operation.  The TestPreCondition operation is semantically significant, in that a “failure” to verify or validate the content of a message results in an exit from the execution of the Test Step with an exit condition of “undetermined” (meaning, because the precondition could not be verified or validated, the ultimate result of the Test Case could not be determined). If no further branching is possible within the Test Case workflow, the Test Service will halt with a final Test Case result of "undetermined". The TestPreCondition element should only be used to verify a condition this is beyond the control of the Test Harness (e.g. a particular optional feature on a candidate implementation under test MAY NOT be implemented). Usage of this element in scripting a Test Case should be done with care to avoid ambiguous or "false" test results (for example defining a "required", testable feature of a candidate implementation as a "precondition).

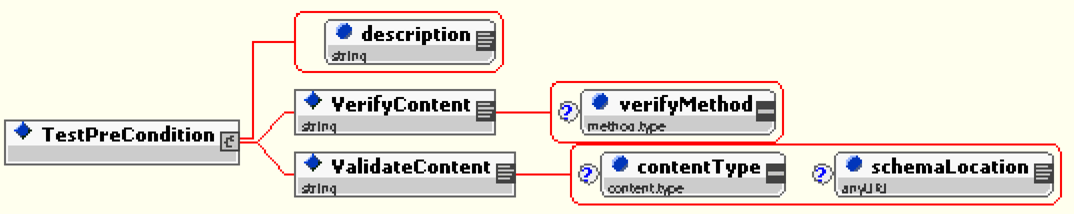

Figure 43 – Graphic representation of expanded view of the TestPreCondition element

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 | Contains 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 | | Optional | Invalid XPath expression |
| contentType | An enumerated list of XML, URI, dateTime, or Signature validation descriptors | XML | Optional | |
| schemaLocation | URI pointing to location of schema used to validate a content type of XML | | Optional | Schema not found |

Table 31 defines the content of the TestPreCondition element

#### 7.1.4.1 Semantics of the TestPreCondition operation

The TestPreCondition operation MUST return either a boolean true or false result (or semantically a pass/fail result) to the Test Driver. Based upon that result, execution of the Test Case will either proceed (if the result is “true”) or halt (if no logical branching in the workflow is possible) from this Test Step, with the final state of the Test Case set to “undetermined”.

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

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

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

- if the contentType attribute value is XML, as defined in [XML] , the operation evaluates to true if the content at the specified XPath validates according to the schema defined in the “schemaLocation” attribute
- if the contentType is URI, as defined in [XMLSCHEMA], the operation evaluates to true if the content at the specified XPath is a valid URI
- if the contentType is dateTime, as defined in [XMLSCHEMA], the operation evaluates to true if the content af the specified XPath is a valid dateTime
- if the contentType is signature, as defined in [XMLDSIG], the operation evaluates to true if the content at the specified XPath is a valid signature.

### 7.1.5 The TestAssertion Operation

The TestAssertion Operation examines a message or messages in a node-list by testing the content against an XPath expression in the TestPreCondition text.   If the XPath expression returns a node-list with one or more nodes, the ConformanceCondition is “true”, else it is “false”.  Within a TestAssertion Operation, content of the node-list can be further examined through the VerifyContent (content evaluation) or ValidateContent (content format evaluation). The TestAssertion operation is semantically significant, in that a “failure” to verify or validate the content of a message results in a Boolean result of “false” for the TestAssertion, and therefore the TestStep object.  If there is no logical workflow branching from that TestStep, or from any of that TestStep’s parent Threads, then the Test Driver ends the Test Case with a final Test Case result of of “fail” (meaning, because the assertion could not be verified or validated, the ultimate result of the Test Case is “failure.  Conversely, if the TestAssertion returns a result of “true”, the Test Driver continues with the TestCase workflow.

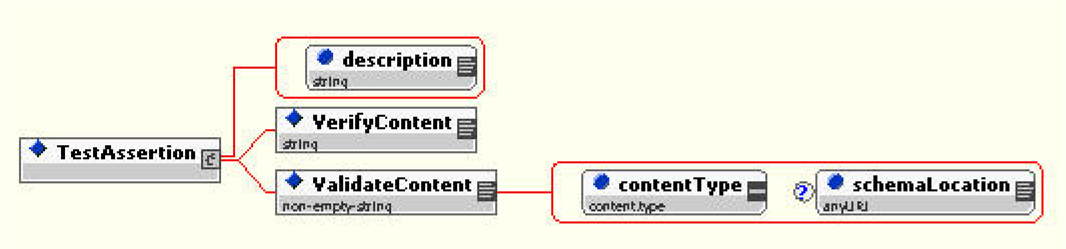

Figure 44 – Graphic representation of expanded view of the TestAssertion element

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 |

Table 32 defines the content of the TestAssertion element

## 7.1.6 Semantics of the TestAssertion operation

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

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

- a returned node-set object evaluates to true if and only if it is non-empty
- a returned boolean object evaluates to true if it evaluates to "true" and false if it evaluates to "false"
- a returned number object evaluates to true if and only if it is neither positive or negative zero nor NaN
- a returned string object evaluates to true if and only if its length is non-zero

The VerifyContent operation MAY also be used to compare an MD5 digest value of a message payload (computed by the Test Driver) with a content value contained in the VerifyContent element. If the "verifyMethod" attribute value of the VerifyContent element is "MD5", then a VerifyContent operation will return a value of "true" if the digest value of current message payload returned by a GetPayload operation is equal to the value contained in the VerifyContent element. Conversely, the operation will return a result of "false" if the values do not match.

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

- if the contentType attribute value is XMLSchema, as defined in [XML] , the operation evaluates to true if the content at the specified XPath validates according to the schema defined in the "schemaLocation" attribute
- if the contentType is URI, as defined in [XMLSCHEMA],  the operation evaluates to true if the content at the specified XPath is a valid URI
- if the contentType is dateTime, as defined in [XMLSCHEMA],  the operation evaluates to true if the content af the specified XPath is a valid dateTime
- if the contentType is signature, as defined in [XMLDSIG], the operation evaluates to true if the content at the specified XPath is a valid signature.

### 7.1.7  The GetPayload Operation

The GetPayload operation fetches the message payload from the current message retrieved with the GetMessage operation.  The message payload is retrieved based upon the required Content-ID, Content-Location or Index child element value.  As with the MessageHeader, both PreCondition and TestAssertion operations can be performed on the message payload.  Payload content can be verified (using the VerifyContent operation described above ) using an XPath expression. If the payload is an XML document, the entire document can be validated (using the ValidateContent operation described above) against the provided XML schema, or a discreet element or attribute value can be validated as a URI or dateTime.

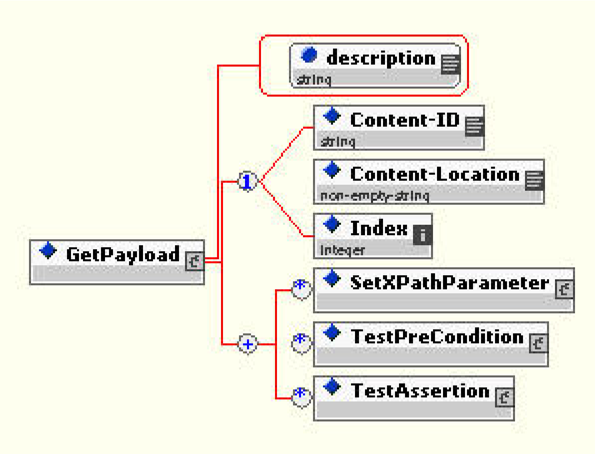

Figure 46 – Graphic representation of expanded view of the GetPayload element

Definition of Content

| Name | Description | Default Value from Test Driver | Required/Optional | Exception Condition |
|---|---|---|---|---|
| GetPayload | Container element for operations to verify and validate message content |  | Optional | More than one message currently in the FilterResult |
| description | Data describing the nature of the GetPaylod operation |  | Required |  |
| ContentId | Retrieve the payload using the MIME Content-ID header value | false | Optional | Payload not found |
| ContentLocation | Retrieve the payload using the MIME Content-Location value |  | Optional | Payload not found |
| Index | Retrieve the payload as the Nth attachment after the message envelope |  | Required | Payloat not found |
| SetXPathParameter | Set the value of a parameter with the value of a node returned by an XPath |  | Optional | Invalid XPath syntax in Expression element |

| | | | | |
|---|---|---|---|---|
| | query against a Filtered message retrieved from the Message Store | | | |
| Name | Parameter name | | Required | |
| Value | Parameter value | | Required | |
| Type | Parameter Type (string or namespace) | | Required | |
| TestPreCondition | Container for verification or validation operation to be performed on message as an optional pre-condition to testing the Assertion | | Optional | |
| TestAssertion | Container for verification or validation operation to be performed on message as a test of the Assertion | | Optional | |

Table 34 defines the content of the GetPayload element

Semantics of the GetPayload operation

Although message payloads are not stored in the MessageStore, the Test Driver MUST be able to retrieve them for verification or validation through their corresponding Content-ID, Content-Type or Index.

How the Test Driver stores the message payloads is implementation dependent.  Unlike the GetMessage operation, which can evaluate multiple messages, the GetPayload operation can only perform a TestPreCondition or TestAssertion operation on a single payload, in a single message.  That message is the "current" message retrieved by the parent GetMessage operation.  If more than one message is retrieved using the GetMessage operation, the GetPayload operation will generate an exception, and the GetPayload operation will return an "undetermined" result for the Test Case.

All other rules regarding the TestPreCondition or TestAssertion operations previously described apply to these operations when applied to a GetPayload operation as well.

### 7.1.8 Message Store Schema

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

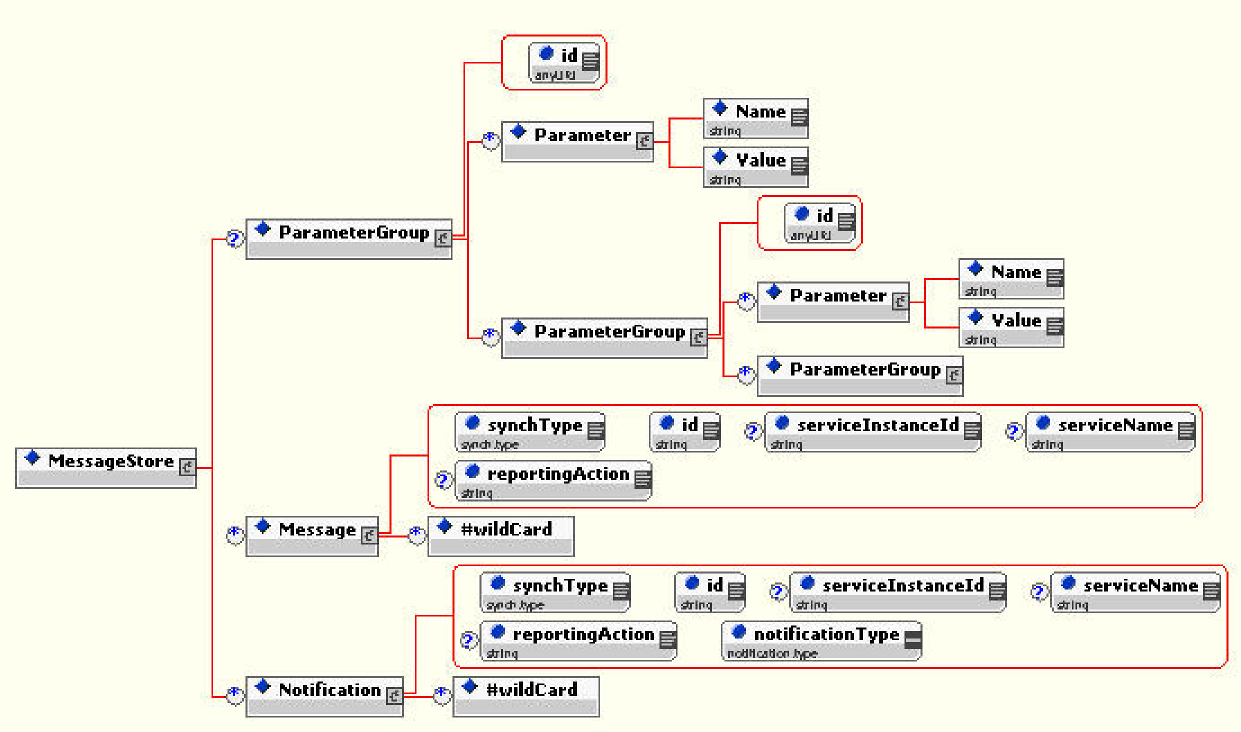

Figure 47 – Graphic representation of expanded view of the generic Test Driver MessageStore schema

Description of Content

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

|  |  |  |  |  |
|---|---|---|---|---|
| ParameterGroup | Container for hierarchical representation of all parameters of the Test Case instance.  The ParamaterGroup element hierarchy begins with TestCase, then Thread, then Test Step  etc… |  | Optional (storage and retrieval of parameter MAY be done in any way that benefits Test Driver implementation) |  |
| Message | Generic container for any type of message, with certain required and optional data associated with each message |  | 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 |  |
| #wildcard | Wildcard element used to represent “any” XML content that may be used to represent any type of message |  | Required |  |
| Notification | Container for a message or error passed from the Test Servcie (in local or remote-reporting mode) to the Test Driver |  | 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 | |
| #wildcard | Wildcard element used to represent “any” XML content that may be used to represent any type of message | | Required | |

#### 7.1.8.1 Semantics of the Message Store

As mentioned earlier, the ParameterGroup schema is a structure intended to assist those who wish to store run-time parameters in the Message Store. The benefit of doing so lies in the ability to perform XPath queries where the parameter values can easily be referenced from within the same Message Store document object. There is no requirement, however, to use the Message Store as a repository for run-time parameters.

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

This is not the case for Notification messages. Notification messages are received via an alternate interface from the Test Service. Because the messaging system being tested cannot be trusted to provide the notifications, notifications are either passed locally (via the Test Service Notification interface) or remotely (via RPC) between Test Service and Test Driver. As a result, message content is restricted to what part of the message was exposed to the Test Service. Therefore the representation or received messages passed via notification is less complete than message content directly received by the Test Driver (for example, MIME content may not be exposed to a Test Service application, therefore MIME headers are not represented in the Notifcation message). For all other purposes however, the format of the Notification message is identical to that of a message directly received by the Test Driver.

### 7.1.8.2 ebXML Specific Message Store Schema

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

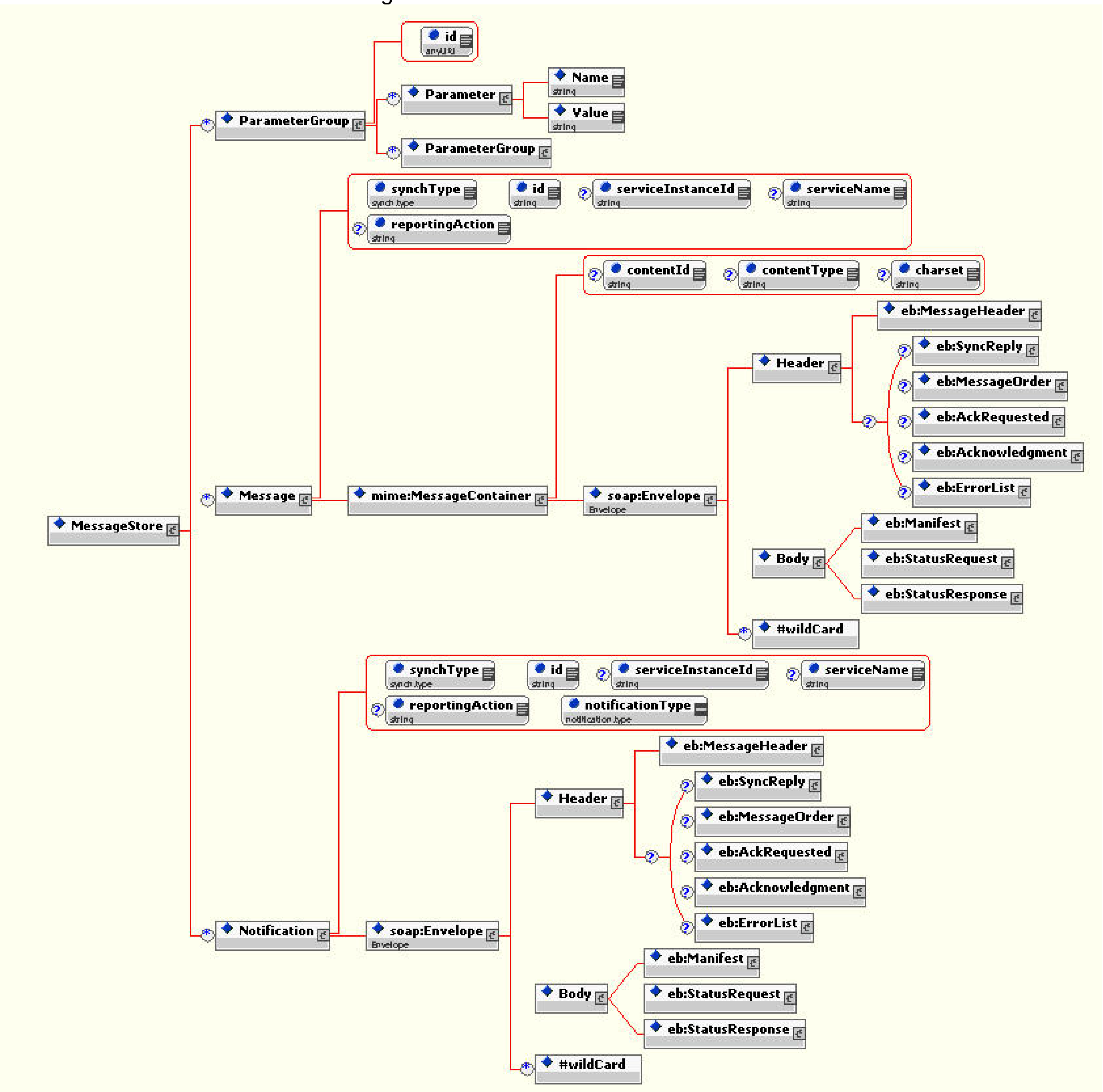

Figure 48 – Graphic representation of expanded view of Message Store content model, specifically for ebXML/SOAP messaging services

Definition of Content

| Name | Declaration Description | Default Value From Test Driver | Required/Optional |
|---|---|---|---|
| MessageStore | Container for XML representation of all messages received by Test Driver for a given Test Case | | Required |
| ParameterGroup | Test Case specific container for all parameters and their cooresponding values | | Required |
| Parameter | Container of name/value pairs | | Optional |
| Name | Parameter name | | Required |
| Value | Parameter Value | | Required |
| Type | Parameter type (parameter, namespace) | | Required |
| Message | Container for MIME, SOAP and ebXML message content | | Optional |
| contentType | MIME message ‘Content-Type’ header | | Optional |
| type | MIME message ‘type’ header | | Optional |
| serviceInstanceId | Unique identifier for instance of Test Service that reports the message | | Required |
| reportingAction | Action name that received the message on reporting service | | Required |
| id | Unique identifier for this message | | Required |
| syncType | Classifier of “synchronous” or “asynchronous” | | Required |
| serviceName | Name of service that received the message | | Required |

|  |  |  |  |
|---|---|---|---|
| MessageContainer | MIME SOAP messagecontainer |  | Required |
| contentId | SOAP container ‘Content-ID’ header |  | Optional |
| contentType | SOAP message package ‘Content-Type’ header |  | Optional |
| charset | SOAP message package character set |  | Optional |
| soap:Envelope | Generates container for SOAP message |  | Required |
| Notification | Message received from Test Service via the Test Driver Receive interface’s “Notify” method |  | Optional |

Table 35 defines the content of the MessageStore element

### 7.1.8.3 ebXML Specific Filter Result Schema

The ebXML MS v2.0 Filter Result Schema (Appendix D) defines the structure of an individual ebXML MS version 2.0.  message received over HTTP.  This schema MUST be used to define the message structure of ebXML messages within the Filter Result.

Like the Message Store, the Filter Result is a document object that can be queried for content testing and verification.  Unlike the MessageStore,  the FilterResult document object only needs to exist for the lifecycle of a single TestStep. The Filter Result document is identical (in structure) to the MessageStore document, with one exception.  The root node of the Filter Result document is a FilterResult element, not a MessageStore element.  The content of the Filter Result document is any messages that satisfy the filter query.

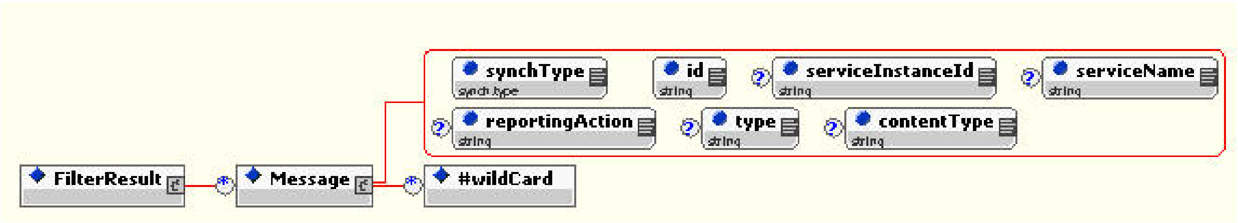

Figure 49 – Generic Filter Result schema

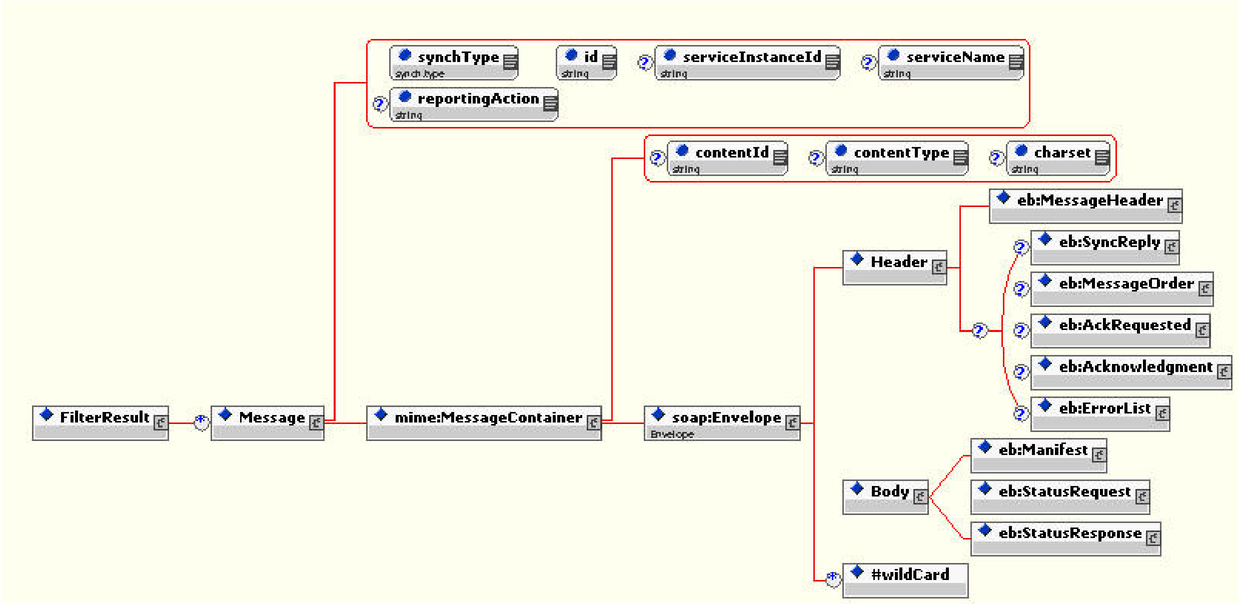

Figure 50 – ebXML Filter Result schema

Definition of Content

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

| | | | |
|---|---|---|---|
| synchType | Classifier of “synchronous” or “asynchronous” | | Required |
| serviceName | Name of service that received the message | | Required |
| MessageContainer | MIME SOAP messagecontainer | | Required |
| contentId | SOAP container ‘Content-ID’ header | | Optional |
| contentType | SOAP message package ‘Content-Type’ header | | Optional |
| charset | SOAP message package character set | | Optional |
| soap:Envelope | Generates container for SOAP message | | Required |

Table 36 defines the content of the MessageStore element

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

## 7.3 Configurator, Initiator, and Notification Message Formats

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

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

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

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

.

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

.

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

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

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

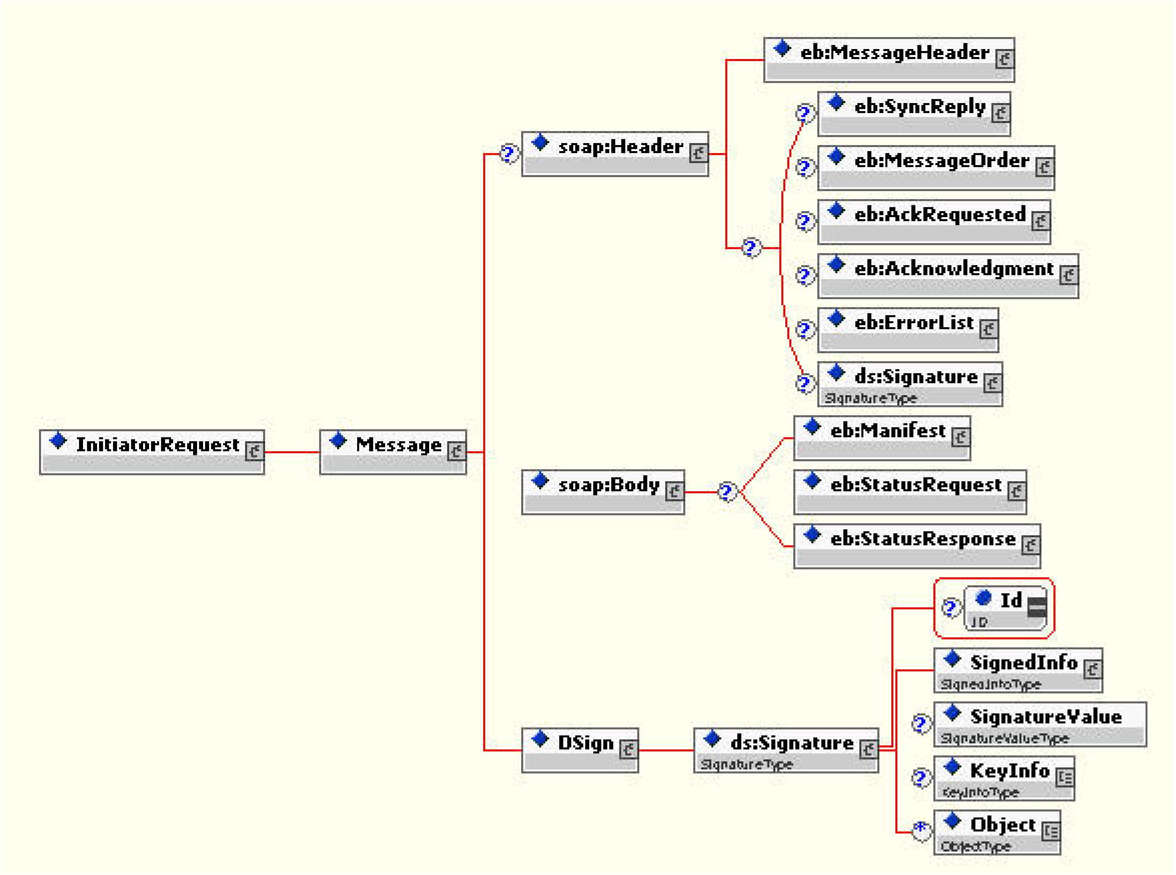

Figure 51 – Initiator request content

Definition of Content

| **Name** | **Declaration Description** | **Default Value From Test Driver** | **Required/Optional** | **Exception Condition** |
|---|---|---|---|---|
| InitiatorRequest | Container for message declaration | | Required | |
| Message | Container for message component declarations | | Required | |
| soap:Header | MIME message 'Content-Type' header | | Optional | |
| Soap:Body | MIME message 'type' header | | Optional | |
| DSign | Instruction to Test Service to digitally sign the appropriate portion of the | | Optional | |

|  |  |  |  |  |
|---|---|---|---|---|
|  | message envelope |  |  |  |

Table 37 – Describes the content of the InitiatorRequest element

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

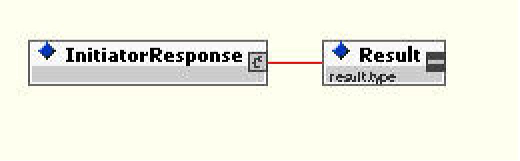

Figure 52 – Graphical representation of the InitiatorResponse schema

Definition of Content

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

Table 38 – Describes the content of the InitiatorResponse element

**ConfiguratorRequest** – XML message content passed to a candidate MSH Test Service, to be interpreted by the configurator method call. Content consists of three required parameter names and their corresponding values and types. If the Test Driver is in "service" mode, the configuration parameters are passed to the Test Service Configuration interface via the configurator method call. If the

Test Driver is in “loop” mode, the parameters are passed to the Test Service via RPC call to the configurator method.

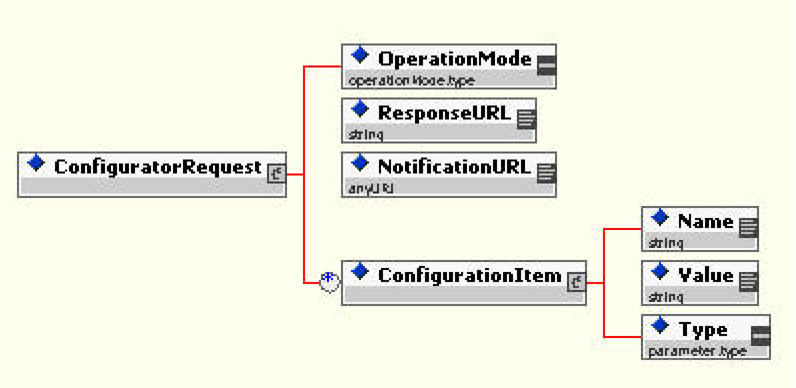

Figure 53 – A Graphical representation of the ConfiguratorRequest content schema

Definition of Content

| 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 | |

Table 39 – Describes the content of the ConfigurationRequest element

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

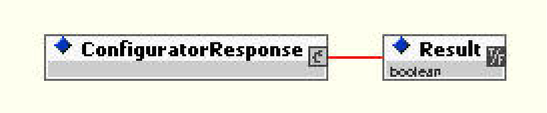

Figure 54 - A graphical representation of the ConfiguratorResponse content schema

Definition of Content

| **Name** | **Declaration Description** | **Default Value From Test Service** | **Required/Optional** | **Exception Condition** |
|---|---|---|---|---|
| ConfiguratorResponse | Container for response from Test Service | | Required | |
| Result | Boolean result (true \| false) for Test Service configuration | | Required | |

Table 40 – Description of content for the ConfiguratoreResponse element

**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 identical for the errorURLNotify, errorAppNotify and messageNotify methods of the Test Service Notification schema, 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 “message” notification, with the exception that the message was generated by the Test Driver in response to an erroneous incoming message.  As a result, it has a SenderParty ID of the Test Service, and an Action of “Notify” in its message header.  It also contains the error message(s) generated by the MSH under test.  The same error format used by the messaging service under test is used to construct the error list.

A Notification message has the same attributes of an “errorURL” message.  The only difference is that the errors contained in the message are application-level errors.  The same error format used by the messaging service under test is used to construct the error list.

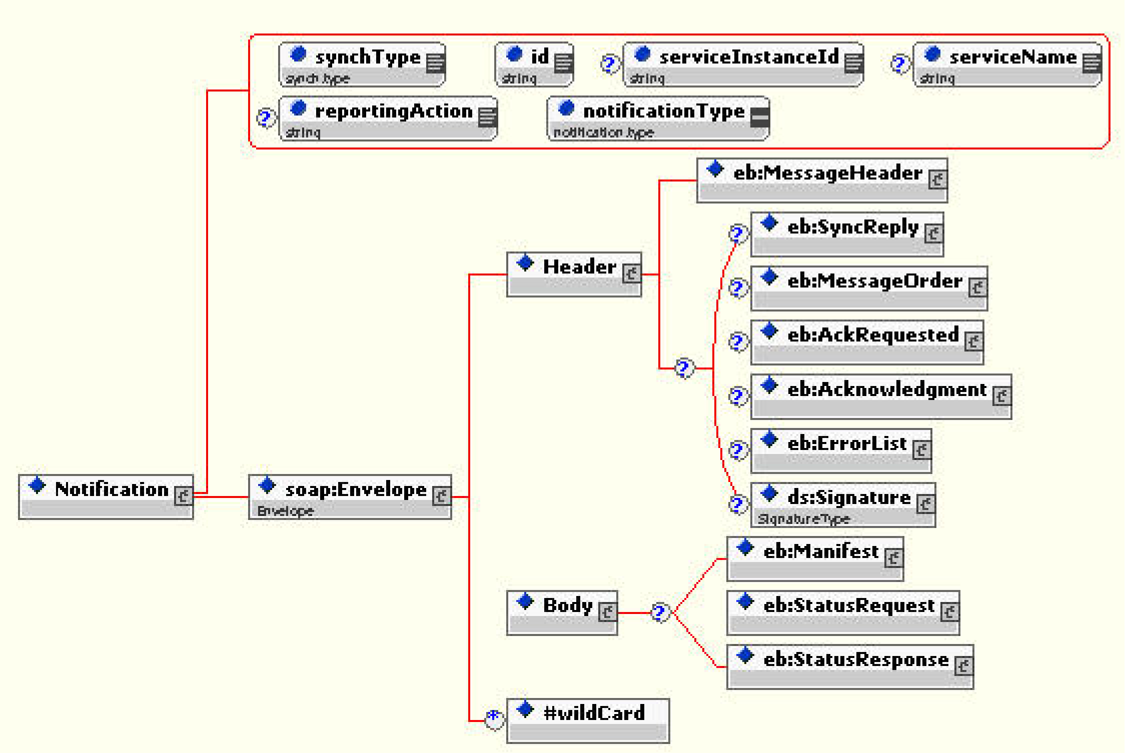

Figure 55 – Graphical representation of the Notification element content schema

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 | |
| Soap:Header | SOAP Header declaration and container for ebXML ebXML Header Extension Element declarations | | Required | |
| eb:MessageHeader | ebXML MessageHeader element with namespace declaration | | Required | |
| eb:ErrorList | ebXML ErrorList element | | Optional | |
| eb:SyncReply | ebXML SyncReply element | | Optional | |
| eb:MessageOrder | ebXML MessageOrder element | | Optional | |
| eb:AckRequested | ebXML AckRequested element | | Optional | |
| eb:Acknowledgment | ebXML Acknowledgment element | | Optional | |

| | | | | |
|---|---|---|---|---|
| ds:Signature | XML Signature element and its content | | Optional | |
| soap:Body | Container for allowable ebXML content within a SOAP body element | | Required | |
| eb:Manifest | Container for references to attachments in message | | Optional | |
| eb:StatusRequest | ebXML StatusRequest element | | Optional | |
| eb:StatusResponse | ebXML StatusResponse element | | Optional | |

Table 41 – Description of  MessageNotification element content

**NotificationResponse** – XML message content to be interpreted by the errorAppNotify, errorURLNotify or messageNotify methods of the Test Service Notification interface.  The response is passed to Test Service through its Notification interface (if Test Service is in local-reporting mode) or sent to the Test Driver Receive RPC Service (if Test Service is in remote-reporting mode. and other information.

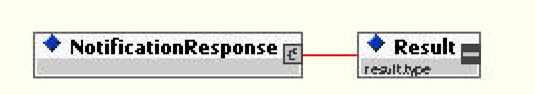

Figure 56 -  A graphical representation of the NotificationResponse content schema

Definition of Content

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

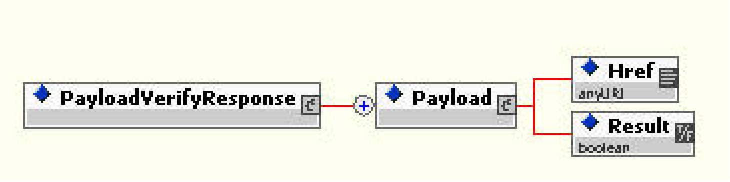

Figure 57 -  A graphical representation of the PayloadVerifyResponse content schema

Definition of Content

Definition of Content

| | | | | |
|---|---|---|---|---|
| | | Default | | |

| **Name** | Declaration Description | 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 | |
| Href | ID of the payload | | Required | |
| Result | Boolean comparison result | | Required | |

Table 42 – Description of  PayloadVerifyResponse content

## 7.4 Test Report Schema

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

The Test Report is essentially a “full trace” of the Test Case.  All XML content in the XML Test Case is available in the Test Report.  Additionally, a “Result” element is appended to each Test Case, Thread, Test Step, PutMessage, GetMessage, TestAssertion, TestPreCondition, VerifContent and ValidateContent operation.  The “result” attribute in all test objects has a value of “pass”, “fail” or “undtermined”. The Test Report schema is too large to graphically display on this page.  Please consult Appendix G if you wish to examine the schema.

# 8 Test Material

Test material necessary to support the ebXML Testing Framework includes:

A Testing Profile XML document

A Test Requirements XML document

A Test Suite XML document

A "Basic CPA" from which variants are derived for particular tests

### 8.1.1 Testing Profile Document

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

### 8.1.2 Test Requirements Document

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

### 8.1.3 Test Suite Document

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

When the message material is to be sent or analyzed at by a Test Driver in Service Mode (i.e. the Test Driver acts as an application component), MIME header and SOAP content SHOULD NOT be declared (in a PutMessage operation) or queried (in a GetMessage operation). However, for the sake of uniform scripting, a Test Driver that conforms to this specification MUST accept MIME message envelope and SOAP header material defined in the declaration of the PutMessage operation (it will then ignore superfluous elements when passing the message to the Test Service). “Accepting” means: (1) when sending (e.g. via PutMessage), MIME and SOAP envelope material will be ignored when invoking the Initiator service action, (2) when receiving, any filtering condition or reference to MIME and SOAP material will be ignored, e.g. removed from the set of conditions used in a GetMessage step. In addition, a Test Driver that conforms to this specification MUST accept XPath query expressions that reference MIME and SOAP message content, even though such content MAY not be included in the MessageStore representation of a message.

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

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

### 8.1.4 Base CPA and derived CPAs

Both conformance and interoperability testing require the existence of a “base CPA” configuration that describes 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. How the base CPA is represented to the applications is implementation specific, however the base CPA configuration MUST be semantically equivalent to the CPA defined in Conformance or Interoperability Test Suite Specification.

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

# 9 Test Material Examples

This section includes example test material to illustrate

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

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

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

## 9.1 Example Test Requirements

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

### 9.1.1 Conformance Test Requirements

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

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns="http://www.oasis-open.org/...   ...conformance/req"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/...   ...conformance/req/
ebXMLTestRequirements.xsd">
<MetaData>
  <Description>Master Requirements File: ebXML Messaging Services 2.0</Description>
  <Version>1.0</Version>
  <Maintainer>Michael Kass<Michael.kass@nist.gov></Maintainer>
  <Location>http://www.oasis-open.org/commitees/ebxml-iic/ebmsg/requirements1.0.xml</Location>
  <PublishDate>20 Feb 2003</PublishDate>
  <Status>DRAFT</Status>
  </MetaData>
<!-- Main Test Requirement for message packaging -->
<TestRequirement id="req_id_2" name="PackagingSpecification" specRef="ebMS-2#2.1" functionalType="packaging">
<!-- Add first sub requirement to begin packaging testing -->
<FunctionalRequirement id="funreq_id_2" name="GenerateConformantSOAPWithAttachmentsMIMEHeaders" specRef="ebMS-2#2.1">
<Clause>
<!-- Set first condition of the message is of type "multipart/mime" -->
  <Condition id="condition_id_1" requirementType="required">For each generated mesage, if it is multipart MIME</Condition>
  <Or/>
<!-- Set alternate condition that the message is not "text/xml" -->
  <Condition id="condition_id_305" requirementType="required">if it is not text/xml</Condition>
  </Clause>
<!-- Add the Assertion that the first part of message is a SOAP message -->
  <Assertion id="assert_id_1" requirementType="required">The primary SOAP message is carried in the root body part of the message.</Assertion>
  </FunctionalRequirement>
<!-- Add a second sub requirement to begin packaging testing -->
<FunctionalRequirement id="funreq_id_3" name="GenerateCorrectMessagePackageContent-Type" specRef="ebMS-2#2.1.2">
<Clause>
<!-- Add condition that the candidate MSH generates a message -->
  <Condition id="condition_id_4" requirementType="required">For each generated message</Condition>
  </Clause>
<!-- Add the Assertion that the Content-Type of MIME header of that message is "text/xml" -->
  <Assertion id="assert_id_2" requirementType="required">The Content-Type MIME header in the Message Package contains a type attribute of "text/xml"</Assertion>
  </FunctionalRequirement>
</TestRequirement>
<!-- Add a new Test Requirement for the Core Extension Elements of message -->
<TestRequirement id="req_id_3" name="CoreExtensionElements" specRef="ebMS-2#3.1.1" functionalType="packaging">
<!-- Add a sub requirement to test the core extension elements -->
<FunctionalRequirement id="funreq_id_4" name="ReportFailedExtension" specRef="ebMS-2#3.1.1">
<Clause>
<!-- First, set condition of a candidate MSH receiving a message with an unresolvable CPAId -->
  <Condition id="condition_id_5" requirementType="required">For each received message, if value of the CPAId element on an inbound message cannot be resolved</Condition>
  </Clause>
<!-- Next, define the Assertion that the candidate MSH MUST ( since requirement type is "required") respond with an Error -->
  <Assertion id="assert_id_3" requirementType="required">The MSH responds with an error (ValueNotRecognized/Error).</Assertion>
  </FunctionalRequirement>
<!-- Add a sub requirement to test conversation id message generation -->
<FunctionalRequirement id="funreq_id_5" name="ProvideConversationId" specRef="ebMS-2#3.1.3">
<Clause>
<!-- First, set condition of all messages generated by a candidate MSH pertaining to a conversation -->
  <Condition id="condition_id_6" requirementType="required">For each generated message within the context of the specified CPAId</Condition>
  </Clause>
<!-- Next, define the Assertion that a ConversationId element is always present -->
  <Assertion id="assert_id_4" requirementType="required">The generated ConversationId will be present in all messages pertaining to the given conversation.</Assertion>
  </FunctionalRequirement>
```

```
</TestRequirement>
</Requirements>
```

## 9.1.2 Interoperability Test Requirements

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

```
<?xml version="1.0" encoding="UTF-8"?>
<Requirements xmlns="http://www.oasis-open.org/tc/ebxml-iic/interop/requirements"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/interop/requirements
ebTestRequirements.xsd">
<MetaData>
  <Description>Interoperability Requirements File: ebXML Messaging Services
2.0</Description>
  <Version>1.0</Version>
  <Maintainer>Michael Kass <michael.kass@nist.gov></Maintainer>
  <Location>http://www.oasis-open.org/commitees/ebxml-
iic/ebmsg/ms_2.0_interop_requirements1.0.xml</Location>
  <PublishDate>11 Feb 2003</PublishDate>
  <Status>DRAFT</Status>
  </MetaData>
<!-- Main Test Requirement for basic interoperability testing -->
<TestRequirement id="req_id_1" name="BasicInteroperability" PROFILE" specRef="MS_2.0_BIP
A" functionType="basic_interoperability">
<!-- First sub requirement to test basic testing send a no payload
message -->
<FunctionalRequirement id="funreq_id_1" name="BasicExchangeNoPayload" specRef="ebMS_2.0
#" >
<Clause>
<!-- Test set condition for a candidate MSH receiving a message with no payload -->
  <Condition id="cond_id_1" requirementType="required">For each received ebXML
message with no payload, received by the “Dummy” action</Condition>
  </Clause>
<!-- Test definition of the Assertion of expected behavior of the MSH Action -->
  <Assertion id="assert_id_1" requirementType="required">The message is received and
processed, and a simple response message is returned</Assertion>
  </FunctionalRequirement>
<!-- Second sub requirement to test basic testing send a one payload
message -->

<FunctionalRequirement id="funreq_id_2" name="BasicExchangeOnePayload" specRef="ebMS_2.0
#" >
<Clause>
<!-- Test condition for a candidate MSH receiving a message with one payload -->
  <Condition id="cond_id_2" requirementType="required">For each received ebXML
message with one payload, received by the “Reflector” action </Condition>
  </Clause>
```

```
<!-- the Assertion of expected behavior of the Reflector Action -->
  <Assertion id="assertion_2" requirementType="required">The message is received and
processed, and a simple response message with the identical  payload is
returned</Assertion>
  </FunctionalRequirement>
<!-- third sub requirement to the basic testing sending a "three payload"
message -->
<FunctionalRequirement id="fr_3" name="BasicExchangeThreePayloads" specRef="eb[illegible]
[illegible]">
<Clause>
<!-- Set condition of a candidate MSH receiving a message with three payloads -->
  <Condition id="cond_3" requirementType="required">For each received ebXML
message with three payloads, received by the "Reflector" action</Condition>
  </Clause>
<!-- the Assertion of expected behavior of the Reflector Action -->
  <Assertion id="assertion_3" requirementType="required">The message is received and
processed, and a simple response message with the identical three payloads are
returned</Assertion>
  </FunctionalRequirement>
<!-- third sub requirement to the basic testing generation of Error messages -->
<FunctionalRequirement id="fr_4" name="BasicExchangeGenerateError" specRef="eb[illegible]
[illegible]">
<Clause>
<!-- Set condition of a candidate MSH receiving an erroneous message -->
  <Condition id="cond_4" requirementType="required">For each received basic
ebXML message that should generate  an Error </Condition>
  </Clause>
<!-- the Assertion of expected behavior of the candidate MSH -->
  <Assertion id="assertion_4" requirementType="required">The message is received and,
the MSH returns a message to the originating party with an ErrorList and appropriate
Error message </Assertion>
  </FunctionalRequirement>
</TestRequirement>
</Requirements>
```

## 9.2 Example Test Profiles

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

### 9.2.1 Conformance Test Profile Example

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<TestProfile xmlns="http[illegible]test-                    [illegible]  "
xmlns:xsi="htt[illegible]hem[illegible]tance              " xsi:schemaLocation="http[illegible]-
[illegible]       [illegible]test-   [illegible]    http[illegible]-       [illegible]      [illegible]test-
[illegible]     est[illegible]xsd      " requirementsLocation="[illegible]         c[illegible]ce_req[illegible]      "
name="[illegible]   [illegible]  [illegible]   [illegible]ce      [illegible]est  [illegible]ents      " description="[illegible]e  c[illegible]ce      test[illegible]
[illegible]
  <TestRequirementRef id="funreq_id_2" />  [illegible]cute      al  [illegible]est  [illegible]sses  that  re[illegible]ce   the
[illegible]
  <TestRequirementRef id="funreq_id_4" />  [illegible]cute      al  [illegible]est  [illegible]ses  that  re[illegible]ce   [illegible]ssag
[illegible]
  <TestRequirementRef id="req_id_2" />  [illegible]cu      al  [illegible]est  [illegible]ses  that  re[illegible]ce   al
[illegible]
  </TestProfile>
```

### 9.2.3 Interoperability Test Profile

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<TestProfile xmlns="http[illegible]test-                    [illegible]  "
xmlns:xsi="htt[illegible]hem[illegible]tance              " xsi:schemaLocation="http[illegible]-
[illegible]       [illegible]test-   [illegible]    http[illegible]-       [illegible]      [illegible]test-
[illegible]     est[illegible]xsd      " requirementsLocation="[illegible]         c[illegible]ce_req[illegible]      "
name="[illegible]   [illegible]  [illegible]   [illegible]ce      [illegible]est  [illegible]ents      " description="[illegible]e  c[illegible]ce      test[illegible]
[illegible]
  <TestRequirementRef id="funreq_id_1.1" />  [illegible]cute      al  [illegible]est  [illegible]sses  that  re[illegible]ce   the
[illegible]
  <TestRequirementRef id="funreq_id_1.2" />  [illegible]cute      al  [illegible]est  [illegible]sses  that  re[illegible]ce   the
[illegible]
  </TestProfile>
```

## 9.3 Example Test Suites

Below are two XML documents illustrating how Test Cases are constructed, in this case for testing an ebXML MS v2.0 implementation.  Each Test Case has a required “requirementReferenceId” attribute, pointing to a Functional Requirement in the Test Requirements document.   A Test Driver executes all

Test Cases in this document that have a requirementReferenceId value matching the particular Semantic Test Requirement being exercised.

### 9.3.1 Conformance Test Suite

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

Send a message and elicit a response message that is verified as a SOAP message

Verifies that an elicited response message content type is “text/xml”

Verifies that an ebXML Error is returned in a response message when an unresolvable CPAId is received

Verifies that the ConversationId element is present in a simple response message

```
<?xml version = "1.0" encoding = "UTF-16"?>

<!--
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.
-->
<?xml-stylesheet type="text/xsl" href="xslt/strip_att_namespace.xsl?>
<ebTest:TestSuite configurationGroupRef = "mshc_Basic" xmlns:ebTest = "http://www.oasis-
open.org/tc/ebxml-iic/tests" xmlns:xpath = "http://www.oasis-open.org/tc/ebxml-iic/xpath"
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:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/tns" xmlns:xlink =
"http://www.w3.org/1999/xlink" xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns:ds
= "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig" xsi:schemaLocation = "http://www.oasis-
open.org/tc/ebxml-iic/tests/schemas/ebTest.xsd schemas/ebTest.xsd">
    <ebTest:MetaData>
              <ebTest:Description>Executable Conformance Test Suite File: ebXML Messaging Services
2.0</ebTest:Description>
              <ebTest:Version>1.0</ebTest:Version>
              <ebTest:Maintainer>Michael Kass &lt;michael.kass@nist.gov></ebTest:Maintainer>
              <ebTest:Location>http://www.oasis-open.org/commitees/ebxml-
iic/ebmsg/requirements1.0.xml</ebTest:Location>
```

```
            <ebTest:PublishDate>10 October  2003</ebTest:PublishDate>
            <ebTest:Status>DRAFT</ebTest:Status>
    </ebTest:MetaData>
    <ebTest:ConfigurationGroup id = "mshc_Basic">
            <ebTest:Mode>local-service</ebTest:Mode>
            <ebTest:SenderParty>TestService</ebTest:SenderParty>
            <ebTest:ReceiverParty>TestService</ebTest:ReceiverParty>
            <ebTest:Service>urn:ebxml:iic:test</ebTest:Service>
            <ebTest:Action>Dummy</ebTest:Action>
            <ebTest:StepDuration>0</ebTest:StepDuration>
            <ebTest:Transport>HTTP</ebTest:Transport>
            <ebTest:Envelope>ebXML</ebTest:Envelope>
            <ebTest:StoreAttachments>true</ebTest:StoreAttachments>
    </ebTest:ConfigurationGroup>
    <ebTest:TestServiceConfigurator>
            <ebTest:ServiceMode>loop</ebTest:ServiceMode>
            <ebTest:ResponseURL>http://myTestDriver.SOAPEndpoint.com</ebTest:ResponseURL>

    <ebTest:NotificationURL>http://myTestDriver.SOAPEndpoint.com</ebTest:NotificationURL>
    </ebTest:TestServiceConfigurator>
    <ebTest:TestCase requirementReferenceId = "funreq_id_36" id = "testcase_36" description =
"ConversationId is always present">
            <ebTest:TestStep id = "TC43TS1">
                    <ebTest:PutMessage description = "Send basic Dummy message header">
                            <ebTest:MessageDeclaration>
                                    <mime:Message>
                                            <mime:MessageContainer>
                                                    <soap:Envelope>
                                                            <soap:Header>
                                                                    <eb:MessageHeader/>
                                                            </soap:Header>
                                                            <soap:Body/>
                                                    </soap:Envelope>
                                            </mime:MessageContainer>
                                    </mime:Message>
                            </ebTest:MessageDeclaration>
                    </ebTest:PutMessage>
            </ebTest:TestStep>
            <ebTest:TestStep id = "TC43TS2">
                    <ebTest:GetMessage description = "Correlate returned messages based upon
CPAId, ConversationId and Action">

    <ebTest:Filter>/TEST:MessageStore/mime:Message[mime:Container[1]/soap:Envelope/soap:Header
/eb:MessageHeader[eb:CPAId='mshc_Basic' and
eb:ConversationId=$ConversationId and eb:Action='Mute' ]] </ebTest:Filter>
                            <ebTest:TestAssertion description = "Verify that ConversationId is
present">

    <ebTest:VerifyContent>/mime:Message[mime:MessageContainer[1]/soap:Envelope/soap:Header/eb
:MessageHeader/eb:ConversationId] </ebTest:VerifyContent>
                            </ebTest:TestAssertion>
                    </ebTest:GetMessage>
            </ebTest:TestStep>
    </ebTest:TestCase>
</ebTest:TestSuite>
```

## 9.3.2 Interoperability Test Suite

In the example below, a series of four Test Cases make up an Interoperability Test Suite.   A Test Driver executing conformance Test Cases operates in “service” mode, meaning it is interfaced to a MSH.  Each

Test Case exercises a Functional Requirement listed in section 10.2 The Test Cases below do the following:

Perform a basic message exchange with no message payload

Verify integrity of 1 payload in round-trip message transmission

Verify integrity of 3 payloads in round-trip message transmission

Perform a basic message exchange with a returned Error message

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

<!--
    [illegible] Message Service 2.0 Interop Test Suite Instance [illegible]
    Michael Kass <michael.kass@nist.gov>
    Date: [illegible]
    This file was created by the National Institute of Standards and Technology
    This software can be redistributed and/or modified freely provided that any
derivative works bear some notice that they are derived from it, and any modified
versions bear some notice that they have been modified.
  -->
<?xml-stylesheet type="text/xsl" href="[illegible]"?>
<ebTest:TestSuite configurationRef="[illegible]" = "[illegible]Basic" xmlns:ebTest = "http://www.oasis-
open.org/tc/ebxml-iic/tests" xmlns:xpath = "http://www.oasis-open.org/tc/ebxml-
iic/xpath" xmlns:[illegible] = "http://www.oasis-open.org/tc/ebxml-iic/tests/[illegible]" 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:[illegible] = "http://www.oasis-open.org/tc/ebxml-
iic/tests/[illegible]" xmlns:[illegible] = "http://[illegible]" xmlns:xsi =
"http://[illegible]/schema-instance" xmlns:[illegible] = "http://www.oasis-
open.org/tc/ebxml-iic/tests/[illegible]" xsi:schemaLocation = "http://www.oasis-
open.org/tc/ebxml-iic/tests/schemas/ebTest.xsd schemas/ebTest.xsd">
    <ebTest:MetaData>
        <ebTest:Description>Interoperability Test Suite: ebXML Message Service
</ebTest:Description>
        <ebTest:Version>[illegible]</ebTest:Version>
        <ebTest:Maintainer>Michael Kass <michael.kass@nist.gov></ebTest:Maintainer>
        <ebTest:Location>http://www.oasis-open.org/committees/ebxml-
iic/[illegible]</ebTest:Location>
        <ebTest:PublishDate>[illegible] March [illegible]</ebTest:PublishDate>
        <ebTest:Status>[illegible]</ebTest:Status>
    </ebTest:MetaData>
    <ebTest:ConfigurationGroup id = "[illegible]">
        <ebTest:[illegible]</ebTest:[illegible]>
        <ebTest:Mode>[illegible]service</ebTest:Mode>
        <ebTest:SenderParty>TestService</ebTest:SenderParty>
        <ebTest:ReceiverParty>TestService</ebTest:ReceiverParty>
        <ebTest:Service>[illegible]test</ebTest:Service>
        <ebTest:Action>[illegible]</ebTest:Action>
        <ebTest:StepDuration>[illegible]</ebTest:StepDuration>
        <ebTest:Transport>[illegible]</ebTest:Transport>
        <ebTest:[illegible]>[illegible]</ebTest:[illegible]>
        <ebTest:StoreAttachments>true</ebTest:StoreAttachments>
    </ebTest:ConfigurationGroup>
    <ebTest:TestServiceConfigurator>
        <ebTest:ServiceMode>[illegible]</ebTest:ServiceMode>
        <ebTest:ResponseURL>[illegible]</ebTest:ResponseURL>

    <ebTest:NotificationURL>[illegible]</ebTest:NotificationURL>
        <ebTest:PayloadDigests>
            <ebTest:Payload>
                [illegible]
                [illegible]</ebTest:[illegible]>
```

```
                  </ebTest:Payload>
                  <ebTest:Payload>
                         <ebTest:Href>[illegible]</ebTest:Href>
                         <ebTest:Digest>[illegible]</ebTest:Digest>
                  </ebTest:Payload>
                  <ebTest:Payload>
                         <ebTest:Href>[illegible]</ebTest:Href>
                         <ebTest:Digest>[illegible]</ebTest:Digest>
                  </ebTest:Payload>
             </ebTest:PayloadDigests>
        </ebTest:TestServiceConfigurator>
        <ebTest:TestCase requirementReferenceId="req_1" id="testcase_1"
description="Basic exchange, no payload">
             <ebTest:TestStep id="step_1" [illegible]="http[illegible]at"
xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests">
                  <ebTest:PutMessage description="Send basic message header with
default configuration Action and no [illegible]
                       <ebTest:MessageDeclaration>
                            <mime:Message>
                                 <mime:MessageContainer>
                                      <soap:Envelope>
                                           <soap:Header>
                                                <eb:MessageHeader/>
                                           </soap:Header>
                                           <soap:Body/>
                                      </soap:Envelope>
                                 </mime:MessageContainer>
                            </mime:Message>
                       </ebTest:MessageDeclaration>
                  </ebTest:PutMessage>
             </ebTest:TestStep>
             <ebTest:TestStep id="step_2" [illegible]="http[illegible]at"
xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests">
                  <ebTest:GetMessage description="Correlate returned message">

        <ebTest:Filter>/MessageStore/Message[MessageContainer[1                    ]/soap:Envelope/soap:H
eader/eb:MessageHeader/eb:[illegible]                          and
eb:ConversationId=$ConversationId                    and eb:Action=[illegible]</ebTest:Filter>
                       <ebTest:TestAssertion description="Verify that an ebXML
message is returned">

        <ebTest:VerifyContent>/MessageStore/Message[MessageContai              ner[1]/soap:Envelope/soap:Head
er/eb:MessageHeader]          </ebTest:VerifyContent>
                       </ebTest:TestAssertion>
                  </ebTest:GetMessage>
             </ebTest:TestStep>
        </ebTest:TestCase>
        <ebTest:TestCase requirementReferenceId="req_2" id="testcase_2"
description="Basic asynchronous exchange with one payload">
             <ebTest:TestStep id="step_1" [illegible]="http[illegible]at"
xmlns:ebTest="http://www.oasis-open.org/tc/ebxml-iic/tests">
                  <ebTest:PutMessage description="Send basic message header to the
Reflector Action">
                       <ebTest:MessageDeclaration>
                            <mime:Message>
                                 <mime:MessageContainer>
                                      <soap:Envelope>
                                           <soap:Header>
                                                <eb:MessageHeader>

        <eb:Action>Reflector</eb:Action>
                                                </eb:MessageHeader>
                                           </soap:Header>
                                           <soap:Body>
                                                <eb:Manifest>
                                                     <eb:Reference href
= "cid:[illegible]">
                                                </eb:Manifest>
                                           </soap:Body>
```

```
                                                </soap>
                                           </MessageContainer>
                                      </Message>
                                 </TestMessageDeclaration>
                                 <TestStepPayload              description          = "A    content-    id  and  payload      to
the     message">
                                      <TestContentPayload>TestContent-                                               B
                                      <TestPayload>TestPayload
                                 </TestStepPayload>
                            </TestMessage>
                       </TestStep>
                       <TestStep          id   = "3"          to          = "http://...at"
hostTest           = "http://service/case"                   .../tests">
                            <TestGetMessage          description        = "Create       returned    messages">

  <TestFilterMessageStore MessageContainer[                                        ]/soap:Body/soap:
eader/MessageHeader/...1                     and
eConversation/Conversation                 and  eAt...</TestFilter>
                                 <TestTestAssertion          description        = "Check    of    returned
...">

  <TestVerifyContentMessage[MessageContai                                        ner[1]/Test/
Reference/Reference[1]                         </TestVerifyContent>
                                 </TestTestAssertion>
                                 </TestStepPayload          description          = "A      payload      in   message">
                                      <TestContentPayload>TestContent-                                               B
                                      <TestTestAssertion          description        = "Verify       returned
payload      checks">
                                           <TestVerifyContent/>
                                      </TestTestAssertion>
                                 </TestStepPayload>
                            </TestGetMessage>
                       </TestStep>
                  </TestTestCase>
                  <TestTestCase        requirementReference            = "reqID"            id   = "testcase3"
description        = "Basic      exchange     with    three  payloads">
                       <TestTestStep          id   = "1"          to          = "http://...at"
hostTest           = "http://service/case"                   .../tests">
                            <TestPutMessage          description        = "Send     basic    message    header  to  the
Reflector      Action">
                                 <TestMessageDeclaration>
                                      <Message>
                                           <MessageContainer>
                                                <soap>
                                                     <soapHeader>
                                                          <MessageHeader>

  <Action>ReflectorAction</Action>
                                                          </MessageHeader>
                                                     </soapHeader>
                                                     <soap
                                                          <Manifest>
                                                               <Reference        href
= "cid:...">

                                                               <Reference        href
= "cid:...">

                                                               <Reference        href
= "cid:...">

                                                          </Manifest>
                                                     </soap
                                                </soap>
                                           </MessageContainer>
                                      </Message>
                                 </TestMessageDeclaration>
                                 <TestStepPayload              description          = "A    content-    id  and  payload      to
the     message">
                                      <TestContentPayload>TestContent-                                               B
                                      <TestPayload>TestPayload
                                 <TestStepPayload
```

```
                    <tst:SetPayload          description     = "A    content-   id  and  payload     to
the    message">
                         <tst:Content-                                                 
                         <tst:                                 
                    </tst:SetPayload>
                    <tst:SetPayload          description     = "A    content-   id  and  payload     to
the    message">
                         <tst:Content-                                      
                         <tst:                                 
                    </tst:SetPayload>
               </tst:PutMessage>
          </tst:TestStep>
          <tst:TestStep       id  = "        "       = "http         at"
        st         = "http                                 /tests">
               <tst:GetMessage         description     = "Create     returned   message">

   <tst:Filter                                                       ]/soap  /soap
eader/MessageHeader/                                 and
e   rsat   rsat                                  and  e        </tst:Filter>
                    <tst:TestAssertion          description     = "Check     returned
   ">

   <tst:Verify                               ai                         ner         ste
Reference  Ref               </tst:Verify>
                    </tst:TestAssertion>
                    <tst:SetPayload          description     = "       first   payload     
message">
                         <tst:Content-                                                 
                         <tst:TestAssertion          description     = "Verify     returned
payload     contents">
                              </tst:Verify>
                         </tst:TestAssertion>
                    </tst:SetPayload>
                    <tst:SetPayload          description     = "       second   payload     
message">
                         <tst:Content-                                                 
                         <tst:TestAssertion          description     = "Verify     returned
payload     contents">
                              </tst:Verify>
                         </tst:TestAssertion>
                    </tst:SetPayload>
                    <tst:SetPayload          description     = "       third   payload     
message">
                         <tst:Content-                                                 
                         <tst:TestAssertion          description     = "Verify     returned
payload     contents">
                              </tst:Verify>
                         </tst:TestAssertion>
                    </tst:SetPayload>
               </tst:GetMessage>
          </tst:TestStep>
   </tst:TestCase>
   <tst:TestCase       requirementReferenceId          = "re          id  = "testcase  
description     = "       exchange    with   Error   Message">
          <tst:TestStep       id  = "        "       = "http         at"
        st         = "http                                 /tests">
               <tst:PutMessage         description     = "Send    MessageHeader     with   messa
Action    name    ">
                    <tst:MessageDeclaration
                         <Message>
                              <MessageContainer>
                                   <soap:    >
                                        <soap:Header>
                                             <eb:MessageHeader>

   <eb:Action

                                             </eb:MessageHeader>
                                        </soap:Header>
                                        <soap:
                                                </[illegible]>
                                         </mime:MessageContainer>
                                    </mime:Message>
                               </ebTest:MessageDeclaration>
                          </ebTest:PutMessage>
                     </ebTest:TestStep>
                     <ebTest:TestStep      id = "3"          [illegible]      = "http://[illegible]at"
[illegible]Test          = "http://[illegible]                          [illegible]/tests">
                          <ebTest:GetMessage        description       = "Correlate      returned    messages">

   <ebTest:Filter>MessageStore/Message[Container[1[                                        ]/soap:Envelope/soap:H
eader/eb:MessageHeader/eb:[illegible]                  and
eb:ConversationId[illegible]                          and   eb:[illegible]</ebTest:Filter>
                               <ebTest:TestAssertion         description       = "Test    for   NotRecognized"
Error    is   generated">

   <ebTest:VerifyContent>MessageStore/Message/MessageContainer                               [illegible]/soap:Envelope/soap
:Body/[illegible]NotRecognized                                 ]    </ebTest:VerifyContent>
                               </ebTest:TestAssertion>
                          </ebTest:GetMessage>
                     </ebTest:TestStep>
   </ebTest:TestCase>
</ebTest:TestSuite>
```

# Appendix A (Normative) The ebXML Test Profile Schema

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

```
<?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-                  ml"
    xmlns:tp="http://www.oasis-open.org/tc/ebxml-iic/test-                  ml"
   >
   <!--
Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema
  -->


   <!--
 $Id: TestProfile.xsd      ?   0        $       att   $   $
  -->

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

```
</schema>
```

# Appendix B (Normative) The ebXML Test Requirements Schema

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].

```
<?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/[illegible]ic/committees"
    xmlns = "http://www.oasis-open.org/[illegible]ic/committees"
   >
   <group name = "[illegible]RequirementGroup">
           <sequence>
                  <element     ref = "[illegible]Requirement"/>
           </sequence>
   </group>

   <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January [illegible] 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/2001/XMLSchema-
>


   <!--    [illegible]              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: [illegible]
           Version:    [illegible]
           - >


   <!--    [illegible]
          Version:    [illegible]   [illegible])
                 - add   attributes    requirement[illegible]         and  name    to level
                 - add   other  to [illegible]                enumeration
   - >
```

```
<element name = "TestRequirement">
          <complexType>
                    <sequence>
                              <element ref = "tns:Clause" minOccurs = "0"/>
                              <choice maxOccurs = "unbounded">
                                        <element ref = "tns:Assertion"/>
                                        <element ref = "tns:AssertionRef"/>
                              </choice>
                              <choice minOccurs = "0" maxOccurs = "unbounded">
                                        <element ref = "tns:FunctionalRequirement"/>
                                        <element ref = "tns:TestRequirement"/>
                              </choice>
                    </sequence>
                    <attribute name = "id" use = "required" type = "ID"/>
                    <attribute name = "name" use = "required" type = "string"/>
                    <attribute name = "specRef" use = "required" type = "string"/>
                    <attribute name = "functionalType" use = "required" type = "string"/>
                    <attribute name = "headerRef" use = "optional" type = "anyURI"/>
          </complexType>
</element>
<element name = "FunctionalRequirement">
          <complexType>
                    <sequence>
                              <element ref = "tns:Clause" minOccurs = "0"/>
                              <choice maxOccurs = "unbounded">
                                        <element ref = "tns:Assertion"/>
                                        <element ref = "tns:AssertionRef"/>
                              </choice>
                              <choice minOccurs = "0" maxOccurs = "unbounded">
                                        <element ref = "tns:FunctionalRequirement"/>
                                        <element ref = "tns:TestRequirement"/>
                              </choice>
                    </sequence>
                    <attribute name = "id" use = "required" type = "ID"/>
                    <attribute name = "name" use = "required" type = "string"/>
                    <attribute name = "specRef" use = "required" type = "string"/>
                    <attribute name = "testCaseRef" use = "optional" type = "anyURI"/>
                    <attribute name = "headerRef" use = "optional" type = "anyURI"/>
          </complexType>
</element>
<element name = "Clause">
          <complexType>
                    <sequence>
                              <choice>
                                        <element ref = "tns:Clause"/>
                                        <choice>
                                                  <element ref = "tns:Condition"/>
                                                  <element ref = "tns:ConditionRef"/>
                                        </choice>
                              </choice>
                              <sequence minOccurs = "0" maxOccurs = "unbounded">
                                        <choice>
                                                  <element ref = "tns:And"/>
                                                  <element ref = "tns:Or"/>
                                        </choice>
                                        <choice>
                                                  <element ref = "tns:Clause"/>
                                                  <choice>
                                                            <element ref = "tns:Condition"/>
                                                            <element ref = "tns:ConditionRef"/>
                                                  </choice>
                                        </choice>
                              </sequence>
                    </sequence>
          </complexType>
</element>
<element name = "And">
          <complexType>
                    <sequence>
```

```
                        <extension base = "string">
                             <attribute name = "[illegible]" use = "required" type = "[illegible]"/>
                             <attribute name = "requirement" use = "optional"
type = "tns:requirementType"/>
                        </extension>
                    </simpleContent>
               </complexType>
     </element>
     <element name = "[illegible]">
          <complexType>
               <attribute name = "[illegible]" use = "required" type = "[illegible]"/>
          </complexType>
     </element>
     <element name = "[illegible]" type = "string"/>
     <element name = "[illegible]" type = "string"/>
     <element name = "[illegible]">
          <complexType>
               <simpleContent>
                    <extension base = "string">
                         <attribute name = "requirement" use = "required"
type = "tns:requirementType"/>
                         <attribute name = "[illegible]" use = "required" type = "[illegible]"/>
                    </extension>
               </simpleContent>
          </complexType>
     </element>
     <element name = "MetaData">
          <complexType>
               <sequence>
                    <element ref = "tns:Description"/>
                    <element ref = "tns:Version"/>
                    <element ref = "tns:Maintainer"/>
                    <element ref = "tns:[illegible]"/>
                    <element ref = "tns:[illegible]"/>
                    <element ref = "tns:Status"/>
               </sequence>
          </complexType>
     </element>
     <element name = "Description" type = "string"/>
     <element name = "Version" type = "string"/>
     <element name = "[illegible]" type = "string"/>
     <element name = "Maintainer" type = "string"/>
     <element name = "[illegible]" type = "[illegible]"/>
     <element name = "[illegible]" type = "string"/>
     <element name = "Status" type = "tns:statusType"/>
     <simpleType name = "statusType">
          <restriction base = "string">
               <enumeration value = "[illegible]"/>
               <enumeration value = "[illegible]"/>
               <enumeration value = "[illegible]"/>
          </restriction>
     </simpleType>
     <simpleType name = "requirementType">
          <restriction base = "string">
               <enumeration value = "required"/>
               <enumeration value = "strongly recommended"/>
               <enumeration value = "recommended"/>
               <enumeration value = "[illegible]"/>
          </restriction>
     </simpleType>
     <simpleType name = "[illegible]">
          <restriction base = "string">
               <enumeration value = "[illegible]"/>
               <enumeration value = "[illegible]"/>
               <enumeration value = "[illegible]"/>
               <enumeration value = "[illegible]"/>
          </restriction>
     </simpleType>
     <simpleType name = "[illegible]">
```

```
          <restriction base="string">
                 <enumeration value="Security"/>
                 <enumeration value="Reliable Messaging"/>
                 <enumeration value="Acks"/>
                 <enumeration value="Other"/>
          </restriction>
   </simpleType>
   <simpleType name="Priority">
          <list itemType="string"/>
   </simpleType>
   <complexType name="Requirements">
          <sequence>
                 <choice>
                        <element ref="ts:MetaData"/>
                        <element ref="ts:TestRequirement" maxOccurs="unbounded"/>
                 </choice>
          </sequence>
   </complexType>
   <complexType name="Assertion">
          <sequence>
                 <attribute name="id" use="required" type="ID"/>
          </sequence>
   </complexType>
</schema>
```

# Appendix C (Normative) The ebXML Test Suite 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].

MIME Portion of the ebXML Message Declaration Schema

```
<?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://[illegible]ictestsuite"
    xmlns:[illegible] = "http://[illegible]ictestsuite"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:[illegible] = "http://[illegible]ictestsuite/[illegible]"
  <import namespace = "http://[illegible]ictestsuite/soap"
schemaLocation = "[illegible]"/>
  <!--
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 = "Message">
        <complexType mixed = "true">
              <choice>
                    <element ref = "ns:MessageContainer"/>
              </choice>
              <attribute name = "contentType" default = "multipart/related" type =
"string"/>
              <attribute name = "id" default = "[illegible]" type = "string"/>
        </complexType>
  </element>
  <element name = "MessageContainer">
        <complexType>
              <sequence>
                    <element ref = "[illegible]"/>
```

```
                </sequence>
                <attribute name = "[illegible]"      [illegible]se = "[illegible]"         type = "string"/>
                <attribute name = "[illegible]"         default = "text/xml"         type = "string"/>
                <attribute name = "charset"   default = [illegible]         type = "string"/>
            </complexType>
   </element>
</schema>
```

SOAP Portion of the ebXML MessageDeclaration Schema

```
<?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.[illegible]/[illegible]      [illegible]/testsuite/[illegible]"
    xmlns:tns = "http://www.[illegible]/[illegible]      [illegible]/testsuite/[illegible]"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:eb = "http://www.[illegible]/[illegible]      [illegible]/testsuite/[illegible]"
    xmlns:ds = "http://[illegible]">
   <import namespace = "http://www.[illegible]/[illegible]      [illegible]/testsuite/[illegible]      schemaLocation
= "[illegible]"/>
   <import namespace = "http://[illegible]"      schemaLocation =
"http://www.[illegible]-[illegible]/committees/ebxml-iic/[illegible]/schema/[illegible]-      [illegible]schema.xsd"/>
   <group name = "[illegible]s">
          <all minOccurs = "0">
                <element ref = "[illegible]" minOccurs = "0"/>
                <element ref = "[illegible]MessageHeader"  minOccurs = "0"/>
                <element ref = "[illegible]AckRequested"  minOccurs = "0"/>
                <element ref = "[illegible]"   minOccurs = "0"/>
                <element ref = "[illegible]Acknowledgment"  minOccurs = "0"/>
                <element ref = "[illegible]Signature"  minOccurs = "0"/>
          </all>
   </group>
   <attributeGroup name = "[illegible]">
          <attribute name = "[illegible]"      type = "[illegible]"/>
   </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 2001 [illegible]

     Changes made are the following:
     - [illegible] Header and [illegible] element children [illegible]  constrained  to [illegible]  eb[illegible]  children


     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
-->


   <!-- Envelope, header and body -->

   <element name="Envelope" type="tns:Envelope"/>
   <complexType name="Envelope">
          <sequence>
                 <element ref="tns:Header"/>
                 <element ref="tns:Body"/>
                 <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
          </sequence>
          <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>
   <element name="Header">
          <complexType>
                 <sequence>
                        <element ref="eb:MessageHeader"/>
                        <group ref="tns:[illegible]"/>
                 </sequence>
          </complexType>
   </element>
   <complexType name="Header">
          <sequence>
                 <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
          </sequence>
          <anyAttribute namespace="##other" processContents="lax"/>
   </complexType>
   <element name="Body">
          <complexType>
                 <choice minOccurs="0">
                        <element ref="eb:Manifest"/>
                        <element ref="eb:StatusRequest"/>
                        <element ref="eb:StatusResponse"/>
                 </choice>
          </complexType>
   </element>
   <complexType name="Body">
          <annotation>
                 <documentation>
                   Prose in the spec does not specify that attributes are allowed on the Body
element
          </documentation>
          </annotation>
          <sequence>
                 <any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
          </sequence>
          <anyAttribute namespace="##any" processContents="lax"/>
   </complexType>

   <!-- Global Attributes. The following attributes are intended to be usable via
qualified attribute names on any complex type referencing them. -->

   <attribute name="mustUnderstand" default="0">
          <simpleType>
                 <restriction base="boolean">
                        <pattern value="0|1"/>
                 </restriction>
          </simpleType>
   </attribute>
   <attribute name="actor" type="anyURI"/>
   <simpleType name="encodingStyle">
```

```
            <annotation>
                <documentation>
      'encodingStyle' indicates any canonicalization conventions followed in the
contents of the containing element. For example, the value
"http://schemas.xmlsoap.org/soap/encoding/" indicates the pattern described in SOAP
specification
                </documentation>
            </annotation>
            <list itemType="anyURI"/>
   </simpleType>
   <complexType name="Fault" final="extension">
            <annotation>
                <documentation>
      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="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>
```

ebMS portion of the ebXML Message Declaration Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-[illegible] -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.oasis-open.org/committees/ebxml-iic/tests/[illegible]"
    xmlns:tns="http://www.oasis-open.org/committees/ebxml-iic/tests/[illegible]"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xlink="http://www.oasis-open.org/committees/ebxml-iic/tests/[illegible]"
    xmlns:soap="http://www.oasis-open.org/committees/ebxml-iic/tests/soap[illegible]"

    version="[illegible]"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">
   <import namespace="http://www.w3.org/2000/09/xmldsig#" schemaLocation="
http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd"/>
   <import namespace="http://www.oasis-open.org/committees/ebxml-iic/tests/[illegible]"
schemaLocation="[illegible]"/>
   <import namespace="http://www.oasis-open.org/committees/ebxml-iic/tests/soap"
schemaLocation="[illegible]"/>
   <import namespace="http://www.w3.org/XML/1998/namespace" schemaLocation="
http://www.w3.org/2001/03/xml.xsd"/>
   <attributeGroup name="headerExtension.grp">
            <attribute ref="tns:id"/>
            <attribute ref="tns:version" use="required"/>
            <attribute ref="soap:mustUnderstand" use="required"/>
   </attributeGroup>
   <attributeGroup name="bodyExtension.grp">
            <attribute ref="tns:id"/>
            <attribute ref="tns:version" 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.
The limited permissions granted above are perpetual and will not be revoked by OASIS or
its successors or assigns.
-->


   <!-- MANIFEST, for use in soap:Body element -->

   <element name="Manifest">
          <complexType>
                 <sequence>
                        <element ref="tns:Reference" maxOccurs="unbounded"/>
                        <any namespace="##other" processContents="lax" minOccurs=
"0" maxOccurs="unbounded"/>
                 </sequence>
                 <attributeGroup ref="tns:bodyExtension.grp"/>
          </complexType>
   </element>
   <element name="Reference">
          <complexType>
                 <sequence>
                        <element ref="tns:Schema" minOccurs="0" maxOccurs=
"unbounded"/>
                        <element ref="tns:Description" minOccurs="0" maxOccurs=
"unbounded"/>
                        <choice>
                               <element ref="tns:FileName"/>
                               <element ref="tns:MessageRef"/>
                               <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
                        </choice>
                 </sequence>
                 <attribute ref="tns:id"/>
                 <attribute ref="xlink:type" fixed="simple"/>
                 <attribute ref="xlink:href" use="required"/>
                 <attribute ref="xlink:role"/>
                 <attribute name="contentId" use="optional" type="string"/>
                 <attribute name="contentType" use="optional" type="string"/>
                 <attribute name="contentLocation" use="optional" type="anyURI"/>
          </complexType>
   </element>
   <element name="Schema">
          <complexType>
                 <attribute name="location" use="required" type="anyURI"/>
                 <attribute name="version" type="tns:non-empty-string"/>
          </complexType>
   </element>

   <!-- MESSAGEHEADER, for use in soap:Header element -->

   <element name="MessageHeader">
          <complexType>
                 <sequence>
                        <element ref="tns:From" minOccurs="1"/>
                        <element ref="tns:To" minOccurs="1"/>
                        <element ref="tns:CPAId" minOccurs="1"/>
```

```
                  <element ref="tns:ConversationId" minOccurs="1"/>
                  <element ref="tns:Service" minOccurs="1"/>
                  <element ref="tns:Action" minOccurs="1"/>
                  <element ref="tns:MessageData" minOccurs="1"/>
                  <element ref="tns:DuplicateElimination" minOccurs="0"/>
                  <element ref="tns:Description" minOccurs="0" maxOccurs="
unbounded"/>
                  <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
              </sequence>
              <attributeGroup ref="tns:headerExtension.grp"/>
          </complexType>
  </element>
  <element name="CPAId" type="tns:non-empty-string"/>
  <element name="ConversationId" type="tns:non-empty-string"/>
  <element name="Service">
          <complexType>
              <simpleContent>
                  <extension base="tns:non-empty-string">
                      <attribute name="type" type="tns:non-empty-
string"/>
                  </extension>
              </simpleContent>
          </complexType>
  </element>
  <element name="Action" type="tns:non-empty-string"/>
  <element name="MessageData">
          <complexType>
              <sequence>
                  <element ref="tns:MessageId" minOccurs="1"/>
                  <element ref="tns:Timestamp" minOccurs="1"/>
                  <element ref="tns:RefToMessageId" minOccurs="0"/>
                  <element ref="tns:TimeToLive" minOccurs="0"/>
              </sequence>
          </complexType>
  </element>
  <element name="MessageId" type="tns:non-empty-string"/>
  <element name="TimeToLive" type="dateTime"/>
  <element name="DuplicateElimination"/>

  <!-- SYNC REPLY, for use in soap:Header element -->

  <element name="SyncReply">
          <complexType>
              <sequence>
                  <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
              </sequence>
              <attributeGroup ref="tns:headerExtension.grp"/>
              <attribute ref="soap:actor" default="urn:oasis:names:tc:ebxml-
msg:actor:toPartyMSH"/>
          </complexType>
  </element>

  <!-- MESSAGE ORDER, for use in soap:Header element -->

  <element name="MessageOrder">
          <complexType>
              <sequence>
                  <element ref="tns:SequenceNumber"/>
                  <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
              </sequence>
              <attributeGroup ref="tns:headerExtension.grp"/>
          </complexType>
  </element>
  <element name="SequenceNumber" type="tns:sequenceNumber.type"/>

  <!-- ACK REQUESTED, for use in soap:Header element -->
```

```
    <element name="AckRequested">
            <complexType>
                    <sequence>
                            <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
                    </sequence>
                    <attributeGroup ref="tns:headerExtension.grp"/>
                    <attribute ref="soap:actor"/>
                    <attribute name="signed" use="required" type="boolean"/>
            </complexType>
    </element>

    <!-- ACKNOWLEDGMENT, for use in soap:Header element -->

    <element name="Acknowledgment">
            <complexType>
                    <sequence>
                            <element ref="tns:Timestamp" minOccurs="0"/>
                            <element ref="tns:RefToMessageId" minOccurs="0"/>
                            <element ref="tns:From" minOccurs="0"/>
                            <element name="Reference" minOccurs="0" maxOccurs="
unbounded"/>
                            <any namespace="##other" processContents="lax" minOccurs="
0"/>
                            <element ref="ds:Reference" minOccurs="0" maxOccurs="
unbounded"/>
                    </sequence>
                    <attributeGroup ref="tns:headerExtension.grp"/>
                    <attribute ref="soap:actor" default="urn:oasis:names:tc:ebxml-
msg:actor:toPartyMSH"/>
            </complexType>
    </element>

    <!-- ERROR LIST, for use in soap:Header element -->

    <element name="ErrorList">
            <complexType>
                    <sequence>
                            <element ref="tns:Error" maxOccurs="unbounded"/>
                            <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
                    </sequence>
                    <attributeGroup ref="tns:headerExtension.grp"/>
                    <attribute name="highestSeverity" use="required" type="
tns:severity.type"/>
            </complexType>
    </element>
    <element name="Error">
            <complexType>
                    <sequence>
                            <element ref="tns:Description" minOccurs="0"/>
                            <any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
                    </sequence>
                    <attribute ref="tns:id"/>
                    <attribute name="codeContext" default="urn:oasis:names:tc:ebxml-
msg:service:errors" type="anyURI"/>
                    <attribute name="errorCode" use="required" type="tns:non-empty-
string"/>
                    <attribute name="severity" use="required" type="
tns:severity.type"/>
                    <attribute name="location" type="tns:non-empty-string"/>
            </complexType>
    </element>

    <!-- STATUS RESPONSE, for use in soap:Body element -->

    <element name="StatusResponse">
```

```
           <complexType>
                   <sequence>
                           <element ref="tns:RefToMessageId"/>
                           <element ref="tns:Timestamp" minOccurs="0"/>
                           <any namespace="##other" processContents="lax" minOccurs =
"0" maxOccurs="unbounded"/>
                   </sequence>
                   <attributeGroup ref="tns:headerExtension.grp"/>
                   <attribute name="messageStatus" use="required" type =
"tns:messageStatus.type"/>
           </complexType>
   </element>

   <!-- STATUS REQUEST, for use in soap:Body element -->

   <element name="StatusRequest">
           <complexType>
                   <sequence>
                           <element ref="tns:RefToMessageId"/>
                           <any namespace="##other" processContents="lax" minOccurs =
"0" maxOccurs="unbounded"/>
                   </sequence>
                   <attributeGroup ref="tns:headerExtension.grp"/>
           </complexType>
   </element>

   <!-- COMMON TYPES -->

   <complexType name="sequenceNumber.type">
           <simpleContent>
                   <extension base="nonNegativeInteger">
                           <attribute name="status" default="Continue" type =
"tns:status.type"/>
                   </extension>
           </simpleContent>
   </complexType>
   <simpleType name="status.type">
           <restriction base="NMTOKEN">
                   <enumeration value="Reset"/>
                   <enumeration value="Continue"/>
           </restriction>
   </simpleType>
   <simpleType name="messageStatus.type">
           <restriction base="NMTOKEN">
                   <enumeration value="UnAuthorized"/>
                   <enumeration value="NotRecognized"/>
                   <enumeration value="Received"/>
                   <enumeration value="Processed"/>
                   <enumeration value="Forwarded"/>
           </restriction>
   </simpleType>
   <simpleType name="non-empty-string">
           <restriction base="string">
                   <minLength value="1"/>
           </restriction>
   </simpleType>
   <simpleType name="severity.type">
           <restriction base="NMTOKEN">
                   <enumeration value="Warning"/>
                   <enumeration value="Error"/>
           </restriction>
   </simpleType>

   <!-- COMMON ATTRIBUTES and ATTRIBUTE GROUPS -->

   <attribute name="id" type="ID"/>
   <attribute name="version" type="tns:non-empty-string"/>

   <!-- COMMON ELEMENTS -->
   <element     name   = "Part">
        <complexType>
            <simpleContent>
                <extension      base  = "tns:nonEmptyString">
                    <attribute     name   = "type"    type   = "tns:nonEmpty
String"    >
                </extension>
            </simpleContent>
        </complexType>
   </element>
   <element     name   = "[illegible]">
        <complexType>
            <sequence>
                <element     ref = "tns:Part"/>
                <element     name   = "[illegible]"     type   = "tns:nonEmpty          String"    minOccurs
= "0"/>
            </sequence>
        </complexType>
   </element>
   <element     name   = "[illegible]">
        <complexType>
            <sequence>
                <element     ref = "tns:Part"/>
                <element     name   = "[illegible]"     type   = "tns:nonEmpty          String"    minOccurs
= "0"/>
            </sequence>
        </complexType>
   </element>
   <element     name   = "Description">
        <complexType>
            <simpleContent>
                <extension      base  = "tns:nonEmpty           String">
                    <attribute     ref = "xml:lang"          use   = "required"/>
                </extension>
            </simpleContent>
        </complexType>
   </element>
   <element     name   = "MessageId"           type   = "tns:nonEmpty           String"/>
   <element     name   = "Status"       type   = "[illegible]"/>
   <element     name   = "Name"       type   = "tns:nonEmpty           String"/>
   <element     name   = "MessageRef"      type   = "tns:nonEmpty           String"/>
</schema>
```

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by [illegible] to [illegible] http[illegible]schema- [illegible] -->
<xs:schema xmlns="http[illegible]test[illegible]messageStore"
    targetNamespace="http[illegible]test[illegible]messageStore"
    xmlns:[illegible]="http[illegible]test[illegible]"
    xmlns:soap="http[illegible]test[illegible]"
    xmlns:xs="http[illegible]schema">
  <xs:import namespace="http[illegible]test[illegible]"
schemaLocation="[illegible]Test[illegible]chemas[illegible]essageStore[illegible].xsd"/>
  <xs:import namespace="http[illegible]test[illegible]soap"
schemaLocation="[illegible]Test[illegible]chemas[illegible]essageStore[illegible]soap.xsd"/>
  <xs:element name="MessageStore">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ParameterGroup" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="Message" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="Notification" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ParameterGroup">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Parameter" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="ParameterGroup" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" use="required" type="xs:ID"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Parameter">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="Value" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="GenericMessage" type="GenericMessageType"/>
  <xs:simpleType name="SynchType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Synchronous"/>
      <xs:enumeration value="Asynchronous"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ParameterType">
    <xs:restriction base="xs:[illegible]">
      <xs:enumeration value="String"/>
```

```
                <xsd:enumeration value="Message"/>
            </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="NotificationType">
            <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="Error"/>
                <xsd:enumeration value="ErrorURL"/>
                <xsd:enumeration value="Message"/>
            </xsd:restriction>
    </xsd:simpleType>
    <xsd:element name="Message">
            <xsd:complexType>
                <xsd:complexContent>
                    <xsd:extension base="GenericMessageType">
                        <xsd:sequence>
                            <xsd:element ref="MessageSummary"/>
                        </xsd:sequence>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="GenericMessageType">
            <xsd:attribute name="Source" use="required" type="SourceType"/>
            <xsd:attribute name="Id" use="required" type="xsd:string"/>
            <xsd:attribute name="ServiceInstanceId" use="optional" type=
"xsd:string"/>
            <xsd:attribute name="ServiceName" use="optional" type="xsd:string"/>
            <xsd:attribute name="ParentAction" use="optional" type="xsd:string"/>
            <xsd:anyAttribute namespace="##other" processContents="strict"/>
    </xsd:complexType>
    <xsd:element name="Notification">
            <xsd:complexType>
                <xsd:complexContent>
                    <xsd:extension base="GenericMessageType">
                        <xsd:sequence>
                            <xsd:element ref="Summary"/>
                        </xsd:sequence>
                        <xsd:attribute name="NotificationType" use=
"required" type="NotificationType"/>
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

MIME Portion of ebXML-Specific MessageStore Schema

```
<?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/mime"
	 xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
	 xmlns:xs = "http://www.w3.org/2001/XMLSchema"
	 xmlns:soap = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap">
	<import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
schemaLocation = "messageStoreSoap.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.
-->

	<element name = "Message">
		<complexType>
			<sequence>
				<element ref = "tns:MessagePart"/>
			</sequence>
			<attribute name = "id" use = "optional" type = "string"/>
			<attribute name = "contentType" use = "optional" type = "string"/>
			<attribute name = "syncType" use = "required" type =
"tns:syncType"/>
			<attribute name = "id" use = "required" type = "string"/>
			<attribute name = "serviceInstanceId" use = "optional" type =
"string"/>
			<attribute name = "serviceName" use = "optional" type = "string"/>
			<attribute name = "contentLength" use = "optional" type = "string"/>
		</complexType>
	</element>
	<element name = "MessagePart">
		<complexType>
			<sequence>
				<element ref = "soap:Envelope"/>
			</sequence>
			<attribute name = "contentId" use = "optional" type = "string"/>
			<attribute name = "contentType" use = "optional" type = "string"/>
			<attribute name = "charset" use = "optional" type = "string"/>
		</complexType>
	</element>
	<simpleType name = "syncType">
		<restriction base = "[illegible]
```

```
                  <enumeration        [illegible]    = "[illegible]"/>
                  <enumeration        [illegible]    = "[illegible]"/>
          </restriction>
   </[illegible]>
   <element      name    = "Message[illegible]">
          <[illegible]>
                  <sequence>
                          <element      ref = "[illegible]"/>
                  </sequence>
          </[illegible]>
   </element>
   <element      name    = "[illegible]"          type    = "[illegible]"/>
</schema>
```

SOAP Portion of ebXML-Specific Message Store Schema

```
<?xml     version    = "[illegible]"     encoding      = "[illegible]"?>

<!--   Generated   by  [illegible]   Authority        [illegible]       to [illegible]    http[illegible]schema-                              -->
<schema    [illegible]      = "http[illegible]schema"
    targetNamespace      = "http[illegible]"                              [illegible]test[illegible]"
    [illegible]         = "http[illegible]"                         [illegible]test[illegible]"
    [illegible]        = "http[illegible]schema"
    [illegible]eb        = "http[illegible]-            [illegible]tees[illegible]                 [illegible]schema[illegible]
header[illegible]        ">
   <import       namespace     = "http[illegible]-            [illegible]tees[illegible]                 [illegible]schema[illegible]
header [illegible]         schemaLocation       = "http[illegible]-          [illegible]tees[illegible]
[illegible]schema/    [illegible]header[illegible]/>
   <[illegible]      name    = "[illegible]">
          <all    minOccurs       = "[illegible]">
                  <element      ref = "[illegible]"            minOccurs       = "[illegible]"/>
                  <element      ref = "[illegible]Message[illegible]"        minOccurs       = "[illegible]"/>
                  <element      ref = "[illegible]Requested"        minOccurs       = "[illegible]"/>
                  <element      ref = "[illegible]"             minOccurs       = "[illegible]"/>
                  <element      ref = "[illegible]"        minOccurs       = "[illegible]"/>
          </all>
   </[illegible]>
   <attributeGroup         name    = "[illegible]">
          <attribute      name    = "[illegible]"            type    = "[illegible]"/>
   </attributeGroup>

   <!--     Schema    for    the  [illegible]         envelope

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

     http[illegible]                            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

     Original       copyright:

     Copyright        2001     W3C   (Massachusetts     Institute      of   Technology
     Institut      National       de  Recherche  en  Informatique          et  en  Automatique
     Keio    University)         All   Rights     Reserved
     http[illegible]Consortium[illegible]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
-->


  <!-- Envelope, header and body -->

  <element name="Envelope" type="tns:Envelope"/>
  <complexType name="Envelope">
    <sequence>
      <element ref="tns:Header"/>
      <element ref="tns:Body"/>
      <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>
  <element name="Header">
    <complexType>
      <sequence>
        <element ref="eb:MessageHeader"/>
        <any ref="tns:[illegible]"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="Header">
    <sequence>
      <any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>
  <element name="Body">
    <complexType>
      <choice>
        <element ref="eb:Manifest"/>
        <element ref="eb:StatusRequest"/>
        <element ref="eb:StatusResponse"/>
      </choice>
    </complexType>
  </element>
  <complexType name="Body">
    <annotation>
      <documentation>
        Prose in the spec does not specify that attributes are allowed on the Body
element
      </documentation>
    </annotation>
    <sequence>
      <any namespace="##any" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>

  <!-- Global Attributes. The following attributes are intended to be usable via qualified
attribute names on any complex type referencing them. -->

  <attribute name="mustUnderstand" default="0">
    <simpleType>
      <restriction base="boolean">
        <pattern value="0|1"/>
      </restriction>
    </simpleType>
  </attribute>
  <attribute name="actor" type="anyURI"/>
  <simpleType name="encodingStyle">
    <annotation>
```

```
                  [illegible]
        [illegible]'         [illegible]dates     [illegible]  ca[illegible]          c[illegible]       [illegible]       [illegible]  the
c[illegible]s    [illegible]  the c[illegible]        e[illegible].     [illegible]   ex[illegible],     the [illegible]
http[illegible]schemas[illegible]                         [illegible]dates     the pattern  d[illegible]scr[illegible]d    [illegible]  [illegible]
spec[illegible]
     [illegible]
           [illegible]
           [illegible]      [illegible]        = [illegible]
   [illegible]>
   [illegible]            [illegible]    = [illegible]"
            [illegible]      = [illegible]
           [illegible]
                  [illegible]
       [illegible]     [illegible]      [illegible]
     [illegible]
           [illegible]
           [illegible]>
                  [illegible]      [illegible]    = [illegible]"       [illegible]    = [illegible]
                  [illegible]      [illegible]    = [illegible]         [illegible]    = [illegible]
                  [illegible]      [illegible]    = [illegible]"       [illegible]    = [illegible]       [illegible]       = [illegible]
                  [illegible]      [illegible]    = [illegible]        [illegible]    = [illegible]       [illegible]       = [illegible]
           [illegible]>
   [illegible]>
   [illegible]            [illegible]    = [illegible]
           [illegible]>
                  [illegible]   [illegible]space     = [illegible]      p[illegible]                = [illegible]    [illegible]       = [illegible]
[illegible]      = [illegible]
           [illegible]>
           [illegible]         [illegible]     = [illegible]      [illegible]           = [illegible]
   [illegible]>
[illegible]schema>
```

ebMS Portion of  ebXML-Specific Message Store Schema

```
[illegible]     [illegible]     = [illegible]     e[illegible]      = [illegible]
[illegible]   [illegible]erated   [illegible]  [illegible]   [illegible]       [illegible]        to [illegible]   htt[illegible]hem[illegible]-                    >
<schem[illegible]   [illegible]      = "htt[illegible]schem[illegible]"
    targ[illegible]space      = "http[illegible]ss[illegible]-           [illegible]tees[illegible]               [illegible]schem[illegible]
head[illegible]- [illegible]
    [illegible]       = "http[illegible]ss[illegible]-           [illegible]tees[illegible]               [illegible]schem[illegible]
head[illegible]        [illegible]
    [illegible]           = "htt[illegible]
    [illegible]        = "htt[illegible]
    [illegible]        = "htt[illegible]schemas[illegible]
    
    [illegible]     = [illegible]
    [illegible]            = [illegible]
    attr[illegible]           = [illegible]
   [illegible]t      [illegible]space     = "http[illegible]                       schem[illegible]         =
"http[illegible]ss[illegible]-        [illegible]tees[illegible]                 [illegible]schem[illegible]
   [illegible]t      [illegible]space     = "http[illegible]                          schem[illegible]        =
"http[illegible]ss[illegible]-        [illegible]tees[illegible]schem[illegible]-                 schem[illegible]
   [illegible]t      [illegible]space     = "http[illegible]schemas[illegible]"                 schem[illegible]        =
"http[illegible]ss[illegible]-        [illegible]tees[illegible]                 [illegible]schem[illegible]
   [illegible]t      [illegible]space     = "http[illegible]space"                schem[illegible]        =
"http[illegible]ss[illegible]-        [illegible]tees[illegible]                 [illegible]schem[illegible]
   [illegible]        [illegible]    = "[illegible]
           [illegible]     ref = "[illegible]
           [illegible]     ref = "[illegible]        [illegible]  = "[illegible]
   [illegible]
   [illegible]        [illegible]    = "Head[illegible]
           [illegible]     ref = "[illegible]
```

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


	<!-- MANIFEST, for use in soap:Body element -->

	<element name="Manifest">
		<complexType>
			<sequence>
				<element ref="tns:Reference" maxOccurs="unbounded"/>
				<any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
			</sequence>
			<attributeGroup ref="tns:bodyExtension.grp"/>
		</complexType>
	</element>
	<element name="Reference">
		<complexType>
			<sequence>
				<element ref="tns:Schema" minOccurs="0" maxOccurs="
unbounded"/>
				<element ref="tns:Description" minOccurs="0" maxOccurs="
unbounded"/>
				<any namespace="##other" processContents="lax" minOccurs="
0" maxOccurs="unbounded"/>
			</sequence>
			<attribute ref="tns:id"/>
			<attribute ref="xlink:type" fixed="simple"/>
			<attribute ref="xlink:href" use="required"/>
			<attribute ref="xlink:role"/>
		</complexType>
	</element>
	<element name="Schema">
		<complexType>
			<attribute name="location" type="anyURI" use="required"/>
			<attribute name="version" type="tns:non-empty-string"/>
		</complexType>
	</element>

	<!-- MESSAGEHEADER, for use in soap:Header element -->

	<element name="MessageHeader">
		<complexType>
			<sequence>
				<element ref="tns:From"/>
				<element ref="tns:To"/>
				<element ref="tns:CPAId"/>
				<element ref="tns:ConversationId"/>
				<element ref="tns:Service"/>
				<element ref="tns:Action"/>
				<element ref="tns:MessageData"/>
```

```
                        eemn     ref = "nateAtn                  Acns      = 0
                        eemn     ref = "tsDescrpn          Acns      = 0   mxAcns     =
Hx
                        ay   amspace    = ther"      poessnets        = ax    Acns      =
0   mxAcns      = Hx
                <eqne>
                attrheGn         ref = "nheadrKenmx
            <emm>
    <emn>
    <emn      am    = A        tm    = "nemy           strm
    <emn      am    = Arsatn           tm    = "nemy           strm
    <emn      am    = Arve>
            <emm>
                <mnen>
                        <xenn       hse   = "nemy           strm
                                attrbe      am    = tp"     tp    = tse       epy
strn      >
                        <xenn
                <mnen>
            <emm>
    <emn>
    <emn      am    = Atn       tm    = "nemy           strm
    <emn      am    = MssaeDta>
            <emm>
                <eqne>
                        <emn      ref = "nMssaen
                        <emn      ref = "nmsta>
                        <emn      ref = "nRMssaen             Acns      = 0
                        <emn      ref = "nm"             Acns      = 0
                <eqne>
            <emm>
    <emn>
    <emn      am    = Mssaen       tm    = "nemy           strm
    <emn      am    = m"           tm    = Atem>
    <emn      am    = nateAtn

    <    M    M       n    ne   m  seHadr       eemn     - >

    <emn      am    = ARn
            <emm>
                <eqne>
                        ay   amspace    = ther"      poessnets        = ax    Acns      =
0   mxAcns      = Hx
                <eqne>
                attrheGn         ref = "nheadrKenmx
                attrbe      ref = Senctn"      ne   = Fenex
            <emm>
    <emn>

    <    M       M       n    ne   m  seHadr       eemn     - >

    <emn      am    = MssaeAdr>
            <emm>
                <eqne>
                        <emn      ref = "nAneMr>
                        ay   amspace    = ther"      poessnets        = ax    Acns      =
0   mxAcns      = Hx
                <eqne>
                attrheGn         ref = "nheadrKenmx
            <emm>
    <emn>
    <emn      am    = AneMr"          tm    = "nseneMrtm>

    <    M   M           n    ne   m  seHadr       eemn     - >

    <emn      am    = ARestex
            <emm>
                seqne>
```

```
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                    </sequence>
                    <attributeGroup ref = "tns:headerExtension.grp"/>
                    <attribute ref = "soap:actor"/>
                    <attribute name = "signed" use = "required" type = "boolean"/>
            </complexType>
    </element>

    <!-- ACKNOWLEDGMENT, for use in soap:Header element -->

    <element name = "Acknowledgment">
            <complexType>
                    <sequence>
                        <element ref = "tns:Timestamp"/>
                        <element ref = "tns:RefToMessageId"/>
                        <element ref = "tns:From" minOccurs = "0"/>
                        <element ref = "ds:Reference" minOccurs = "0" maxOccurs =
"unbounded"/>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                    </sequence>
                    <attributeGroup ref = "tns:headerExtension.grp"/>
                    <attribute ref = "soap:actor"/>
            </complexType>
    </element>

    <!-- ERROR LIST, for use in soap:Header element -->

    <element name = "ErrorList">
            <complexType>
                    <sequence>
                        <element ref = "tns:Error" maxOccurs = "unbounded"/>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                    </sequence>
                    <attributeGroup ref = "tns:headerExtension.grp"/>
                    <attribute name = "highestSeverity" use = "required" type =
"tns:severity.type"/>
            </complexType>
    </element>
    <element name = "Error">
            <complexType>
                    <sequence>
                        <element ref = "tns:Description" minOccurs = "0"/>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                    </sequence>
                    <attribute ref = "tns:id"/>
                    <attribute name = "codeContext" default = "urn:oasis:names:tc:ebxml-
msg:service:errors" type = "anyURI"/>
                    <attribute name = "errorCode" use = "required" type = "tns:non-empty-
string"/>
                    <attribute name = "severity" use = "required" type =
"tns:severity.type"/>
                    <attribute name = "location" type = "tns:non-empty-string"/>
            </complexType>
    </element>

    <!-- STATUS RESPONSE, for use in soap:Body element -->

    <element name = "StatusResponse">
            <complexType>
                    <sequence>
                        <element ref = "tns:RefToMessageId"/>
                        <element ref = "tns:Timestamp" minOccurs = "0"/>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                    </sequence>
```

```
                  <attributeGroup ref = "tns:commonAttributes"/>
                  <attribute name = "messageStatus" use = "required" type =
"tns:messageStatus.type"/>
            </complexType>
      </element>

      <!-- STATUS REQUEST for use in StatusRequest element -->

      <element name = "StatusRequest">
            <complexType>
                  <sequence>
                        <element ref = "tns:RefToMessageId"/>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                  </sequence>
                  <attributeGroup ref = "tns:commonAttributes"/>
            </complexType>
      </element>

      <!-- COMMON TYPES -->

      <complexType name = "sequenceNumber.type">
            <simpleContent>
                  <extension base = "nonNegativeInteger">
                        <attribute name = "status" default = "Continue" type =
"tns:status.type"/>
                  </extension>
            </simpleContent>
      </complexType>
      <simpleType name = "status.type">
            <restriction base = "NMTOKEN">
                  <enumeration value = "Reset"/>
                  <enumeration value = "Continue"/>
            </restriction>
      </simpleType>
      <simpleType name = "messageStatus.type">
            <restriction base = "NMTOKEN">
                  <enumeration value = "UnAuthorized"/>
                  <enumeration value = "NotRecognized"/>
                  <enumeration value = "Received"/>
                  <enumeration value = "Processed"/>
                  <enumeration value = "Forwarded"/>
            </restriction>
      </simpleType>
      <simpleType name = "non-empty-string">
            <restriction base = "string">
                  <minLength value = "1"/>
            </restriction>
      </simpleType>
      <simpleType name = "service.type">
            <restriction base = "NMTOKEN">
                  <enumeration value = "[illegible]"/>
                  <enumeration value = "[illegible]"/>
            </restriction>
      </simpleType>

      <!-- COMMON ATTRIBUTES and ELEMENTS -->

      <attribute name = "id" type = "ID"/>
      <attribute name = "version" type = "tns:non-empty-string"/>

      <!-- COMMON ELEMENTS -->

      <element name = "Part">
            <complexType>
                  <simpleContent>
                        <extension base = "tns:non-empty-string">
                              <attribute name = "type" type = "tns:non-empty-
string"/>
```

```
                        </extension>
                    </complexContent>
                </complexType>
    </element>
    <element name="[illegible]">
                <complexType>
                    <sequence>
                        <element ref="tns:Part[illegible]" maxOccurs="[illegible]"/>
                        <element name="[illegible]" type="tns:non-empty-string" minOccurs
= "0"/>
                    </sequence>
                </complexType>
    </element>
    <element name="[illegible]">
                <complexType>
                    <sequence>
                        <element ref="tns:Part[illegible]" maxOccurs="[illegible]"/>
                        <element name="[illegible]" type="tns:non-empty-string" minOccurs
= "0"/>
                    </sequence>
                </complexType>
    </element>
    <element name="Description">
                <complexType>
                    <simpleContent>
                        <extension base="tns:non-empty-string">
                            <attribute ref="xml:lang" use="required"/>
                        </extension>
                    </simpleContent>
                </complexType>
    </element>
    <element name="Message" type="tns:non-empty-string"/>
    <element name="Status" type="[illegible]"/>
</schema>
```

Generic FilterResult Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns="http://www.oasis-open.org/[illegible]/test/MessageStore"
    targetNamespace="http://www.oasis-open.org/[illegible]/test/MessageStore"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by Michael Kass (NIST) -->


   <!-- edited with XML Spy v4.2 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="[illegible]">
          <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="SynchronicityType">
          <xsd:restriction base="xsd:string">
                 <xsd:enumeration value="synchronous"/>
                 <xsd:enumeration value="asynchronous"/>
          </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="NotificationType">
          <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">
          <xsd:complexType>
                 <xsd:complexContent>
                        <xsd:extension base="GenericMessageType">
                               <xsd:attribute name="type" use="optional" type="xsd:string"/>
                               <xsd:attribute name="contentType" use="optional" type="xsd:string"/>
                        </xsd:extension>
                 </xsd:complexContent>
          </xsd:complexType>
   </xsd:element>
   <xsd:complexType name="GenericMessageType">
          <xsd:sequence>
                 <xsd:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
          <xsd:attribute name="syncType" use="required" type="SynchronicityType"/>
          <xsd:attribute name="id" use="required" type="xsd:string"/>
          <xsd:attribute name="serviceInstanceId" use="optional" type="xsd:string"/>
          <xsd:attribute name="serviceName" use="optional" type="xsd:string"/>
          <xsd:attribute name="[illegible]" use="optional" type="xsd:string"/>
          <xsd:anyAttribute namespace="##other" processContents="strict"/>
   </xsd:complexType>
   <xsd:element name="Notification">
          <xsd:complexType>
                 <xsd:complexContent>
                        <xsd:extension base="GenericMessageType">
                               <xsd:attribute name="notificationType" use="required" type="NotificationType"/>
                        </xsd:extension>
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

## ebXML Specific Filter Result Schema

```
<?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/tests/messageStore"
    targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messageStore"
    xmlns:eb="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
    xmlns:soap="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
schemaLocation =
"file:///C:/MoneTst/schemas/messageStore/eb.xsd"/>
   <xsd:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
schemaLocation =
"file:///C:/MoneTst/schemas/messageStore/soap.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) -->


   <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2003. 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="ErrorMessage" type="ErrorMessageType"/>
   <xsd:simpleType name="SynchType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="synchronous"/>
            <xsd:enumeration value="asynchronous"/>
        </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name="NotificationType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="Error"/>
            <xsd:enumeration value="ErrorApp"/>
```

```
                  <xs:enumeration          value    = "Message"/>
              </xs:restriction>
      </xs:simpleType>
      <xs:element          name    = "Message">
              <xs:complexType>
                  <xs:complexContent>
                      <xs:extension          base   = "GenericMessageType">
                          <xs:sequence>
                              <xs:element          ref = "MessageSummary"/>
                          </xs:sequence>
                          <xs:attribute          name    = "type"    use   = "optional"          type    =
"xs:string"/>
                          <xs:attribute          name    = "[illegible]"          use   = "optional"
type    = "xs:string"/>
                      </xs:extension>
                  </xs:complexContent>
              </xs:complexType>
      </xs:element>
      <xs:complexType              name    = "GenericMessageType">
              <xs:attribute          name    = "[illegible]"          use   = "required"        type    = "[illegible]"/>
              <xs:attribute          name    = "id"     use   = "required"        type    = "xs:string"/>
              <xs:attribute          name    = "ServiceInstanceId"          use   = "optional"          type    =
"xs:string"/>
              <xs:attribute          name    = "ServiceName"        use   = "optional"          type    = "xs:string"/>
              <xs:attribute          name    = "[illegible]"             use   = "optional"          type    = "xs:string"/>
              <xs:anyAttribute            namespace     = "##other"       processContents        = "strict"/>
      </xs:complexType>
      <xs:element          name    = "Notification">
              <xs:complexType>
                  <xs:complexContent>
                      <xs:extension          base   = "GenericMessageType">
                          <xs:sequence>
                              <xs:element          ref = "[illegible]"/>
                          </xs:sequence>
                          <xs:attribute          name    = "[illegible]"              use   =
"required"       type    = "[illegible]"/>
                      </xs:extension>
                  </xs:complexContent>
              </xs:complexType>
      </xs:element>
</xs:schema>
```

# Appendix E (Normative) The  Test Report Schema

```
<?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/testreport"
    xmlns:testReport="http://www.oasis-open.org/tc/ebxml-iic/testreport"
    xmlns:ds="http://www.oasis-open.org/tc/ebxml-iic/testsuite"
    xmlns="http://www.oasis-open.org/tc/ebxml-iic/testsuite"
    version="1.0"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/testsuite"
schemaLocation="[illegible]"/>
   <import namespace="http://www.oasis-open.org/tc/ebxml-iic/testsuite"
schemaLocation="[illegible]"/>
   <!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by Michael Kass (NIST) -->


   <!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by Michael Kass (NIST) -->


   <!-- edited with XML Spy v4.2 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="TestReport">
          <complexType>
                 <sequence>
                        <element ref="testReport:MetaData"/>
                        <element ref="testReport:TestCase"          maxOccurs="unbounded"/>
                 </sequence>
          </complexType>
   </element>
   <element name="MetaData">
          <complexType>
                 <sequence>
                        <element ref="testReport:Description"/>
                        <element ref="testReport:Version"/>
                        <element ref="testReport:Maintainer"/>
                        <element ref="testReport:Location"/>
                        <element ref="testReport:PublishDate"/>
                        <element ref="testReport:Status"/>
                 </sequence>
          </complexType>
   </element>
   <element name="Description"        type="testReport:nonemptyString"/>
   <elemn      am    = #rsn       tn    = "estRntemy             strn
   <elemn      am    = Mnanr"        tn    = "estRntemy              strn
   <elemn      am    = Natn        tn    = 3n
   <elemn      am    = RhPte"          tn    = "estRntemy              strn
   <elemn      am    = Satn"     tn    = "estRntemy             strn
   <elemn      am    = #stRse>
          <cmn>
                 <eqne>
                        <elemn      ref = "estRntRsn>
                        <chie     mnOcns      = HH
                               <elemn      ref = "estRntRtPramter>
                               <elemn      ref = "estRntTstSrvenatn>
                               <elemn      ref = "estRntTread
                               <elemn      ref = "estRntTstSen
                               <elemn      ref = "estRnt>
                               <elemn      ref = "estRntR>
                               <elemn      ref = "estRntn
                               <elemn      ref = "estRntRen
                        </chie>
                 </eqne>
                 <attrbe      am    = n     se   = "emen       tn    = n
                 <attrbe      am    = Hscrnn          se   = HnT          tn    = Strn
                 <attrbe      am    = 3nhn"     se   = HnT          tn    = Strn
                 <attrbe      am    = #rsn       se   = HnT          tn    = Strn
                 attrbe      am    = "eqemeRReened             se   = "eqed       tp    =
3n
                 <attrbe      am    = "ememn"          se   = "emen       tn    = Strn
                 <attrbe      am    = SacRT      se   = HnT          tn    = Strn
                 attrbe      am    = dn"     se   = "eqed       tp    = testRpte
emy   strn
                 <attrbe      am    = "esn"     se   = "emen       tn    = "esntn>
          </cmn>
   </elemn>
   <elemn      am    = natn
          <cmn>
                 <eqne>
                        <elemn      ref = "estRntn              nOcns       = n
                        <elemn      ref = "estRntn>
                        <elemn      ref = "estRntRdrRrt>
                        <elemn      ref = "estRntRceirRrt>
                        <elemn      ref = "estRntRrie>
                        <elemn      ref = "estRntAtn
                        <elemn      ref = "estRntSeRatn
                        <elemn      ref = "estRntRsen              nOcns       = n
                        <elemn      ref = "estRntnfatn              nOcns       = n
                        <elemn      ref = "estRntTaRnt>
                        <elemn      ref = "estRntnn>
                        elemn      ref = testRptgatnen                  nOcns       = 0
mnOcns       = HH
                 </eqne>
                 <attrbe      am    = n     se   = "emen       tn    = n
          </cmn>
   </elemn>
   <elemn      am    = n         tn    = "estRntemy             strn
   <elemn      am    = n"       tn    = "estRntntn>
   <elemn      am    = RdrRrtV         tn    = 3n
   <elemn      am    = RceirRrtV         tn    = 3n
   <elemn      am    = Rrie"       tn    = 3n
   <elemn      am    = Atn        tn    = "estRntemy             strn
   <elemn      am    = SeRatn           tn    = near>
   <elemn      am    = TaRnt"       tn    = "estRntemy             strn
   <elemn      am    = nn"         tn    = "estRntemy             strn
   <smn          am    = ntn>
          <estrctn      bse   = n
                 <enratn        vn     = nal     serie>
                 <enratn        vn     = "eme-    serie>
                 <enratn        vn     = Hactn
          </estrctn
   </smp>
```

```
<simpleType         name    = "[illegible]Header[illegible]Type">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "MessageContent-                Type"/>
            <enumeration          value     = "MessageStart"/>
            <enumeration          value     = "Content-       Type"/>
            <enumeration          value     = "Start"/>
            <enumeration          value     = "Charset"/>
            <enumeration          value     = "[illegible]"/>
            <enumeration          value     = "[illegible]ar[illegible]"/>
        </restriction>
</simpleType>
<simpleType         name    = "ContentType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "[illegible]"/>
            <enumeration          value     = "[illegible]te"/>
            <enumeration          value     = "[illegible]"/>
            <enumeration          value     = "S[illegible]tne"/>
            <enumeration          value     = "[illegible]er"/>
        </restriction>
</simpleType>
<simpleType         name    = "[illegible]th[illegible]Type">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "[illegible]th"/>
            <enumeration          value     = "[illegible]"/>
        </restriction>
</simpleType>
<simpleType         name    = "MessageContextType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "Fr[illegible]"/>
            <enumeration          value     = "[illegible]e"/>
        </restriction>
</simpleType>
<simpleType         name    = "Re[illegible]ementType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "Re[illegible]e[illegible]"/>
            <enumeration          value     = "Str[illegible]ecm[illegible]d[illegible]"/>
            <enumeration          value     = "Recm[illegible]d[illegible]"/>
            <enumeration          value     = "[illegible]"/>
        </restriction>
</simpleType>
<simpleType         name    = "[illegible]y            str[illegible]">
        <restriction      base   = "Str[illegible]">
            <[illegible]h          value     = ""/>
        </restriction>
</simpleType>
<simpleType         name    = "C[illegible]t[illegible]Type">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "[illegible]r"/>
            <enumeration          value     = "Re[illegible]ce"/>
        </restriction>
</simpleType>
<simpleType         name    = "ActionType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "Reset"/>
            <enumeration          value     = "[illegible]"/>
        </restriction>
</simpleType>
<simpleType         name    = "C[illegible]e[illegible]Type">
        <restriction      base   = "[illegible]">
</simpleType>
<simpleType         name    = "ParameterType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "[illegible]space"/>
            <enumeration          value     = "Str[illegible]"/>
        </restriction>
</simpleType>
<simpleType         name    = "ConnectivePredicateType">
        <restriction      base   = "[illegible]">
            <enumeration          value     = "a[illegible]"/>
```

```
                <xs:enumeration value="[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="threadType">
            <xs:restriction base="xs:string">
                <xs:enumeration value="synchronous"/>
                <xs:enumeration value="asynchronous"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="matchResultType">
            <xs:restriction base="xs:string">
                <xs:enumeration value="pass"/>
                <xs:enumeration value="fail"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
            <xs:restriction base="xs:string">
                <xs:enumeration value="[illegible]"/>
                <xs:enumeration value="[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
            <xs:restriction base="xs:string">
                <xs:enumeration value="[illegible]"/>
                <xs:enumeration value="[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
            <xs:restriction base="xs:string">
                <xs:enumeration value="[illegible]"/>
                <xs:enumeration value="[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="ServiceModeType">
            <xs:restriction base="xs:string">
                <xs:enumeration value="loop"/>
                <xs:enumeration value="local-reporting"/>
                <xs:enumeration value="remote-reporting"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]ResultType">
            <xs:restriction base="xs:boolean">
                <xs:enumeration value="true"/>
                <xs:enumeration value="false"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
            <xs:restriction base="xs:string">
                <xs:enumeration value="[illegible]Test"/>
                <xs:enumeration value="AssertionTest"/>
                <xs:enumeration value="Alter[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
            <xs:restriction base="xs:string">
                <xs:enumeration value="pass"/>
                <xs:enumeration value="fail"/>
                <xs:enumeration value="Alter[illegible]"/>
                <xs:enumeration value="[illegible]"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:element name="MessageAccess">
            <xs:complexType>
                <xs:sequence>
                        <xs:element ref="testRootErrorMessage"/>
                </xs:sequence>
            </xs:complexType>
    </xs:element>
    <xs:element    name="ErrorMessage"    type="testReport[illegible]"             string
```

```
   <element     name   = "Message">
         <complexType>
               <sequence>
                     <element     ref = "testRpt:StRsh"/>
                     <choice>
                           <element     ref = "testRpt:MessageDeclaration"/>
                           <element     ref = "testRpt:FM"/>
                           <element     ref = "testRpt:MessageR"/>
                     </choice>
                     <element     ref = "testRpt:Mutator"         minOccurs      = "0"/>
                     <element     ref = "testRpt:g"            minOccurs      = "0"   maxOccurs     =
"unbounded"/>
                     <element     ref = "testRpt:StPayload"            minOccurs      = "0"
maxOccurs      = "unbounded"/>
               </sequence>
               <attribute     name    = "description"         use   = "required"      type    = "string"/>
               <attribute     name    = "repeatWithSameContext"            use   = "optional"         type    =
"integer"/>
               <attribute     name    = "repeatWithNewContext"            use   = "optional"         type    =
"integer"/>
               <attribute     name    = "result"    use   = "required"      type    = "resultType"/>
         </complexType>
   </element>
   <element     name   = "GetPayload">
         <complexType>
               <sequence>
                     <choice>
                           <element     ref = "testRpt:Get-          "/>
                           <element     ref = "testRpt:Get-          Data"/>
                           <element     ref = "testRpt:M"/>
                     </choice>
                     <element     ref = "testRpt:SetParameterWithParameter"
minOccurs      = "0"   maxOccurs      = "unbounded"/>
                     <element     ref = "testRpt:TestPreCondition"                minOccurs      = "0"
maxOccurs      = "unbounded"/>
                     <element     ref = "testRpt:TestAssertion"           minOccurs      = "0"
maxOccurs      = "unbounded"/>
                     <element     name   = "If"     minOccurs      = "0"   maxOccurs      = "unbounded">
                           <complexType>
                                 <sequence>
                                       <choice>
                                             <element     ref =
"testRpt:TestPreCondition"/>
                                             <element     ref =
"testRpt:TestAssertion"/>
                                       </choice>
                                       <element     ref = "testRpt:Then"/>
                                       <element     name    = "Else"       minOccurs      = "0"
maxOccurs      = "unbounded">
                                             <complexType>
                                                   <sequence     maxOccurs     =
"unbounded">
                                                         <choice>
                                                               <element
ref = "testRpt:TestPreCondition"/>
                                                               <element
ref = "testRpt:TestAssertion"/>
                                                         </choice>
                                                         <element     ref =
"testRpt:Then"/>
                                                   </sequence>
                                             </complexType>
                                       </element>
                                       <element     ref = "testRpt:Continue"
minOccurs      = "0"/>
                                 </sequence>
                                 <attribute     name    = "id"        use   = "required"       type
= "testRpt:forType"/>
                           </complexType>
```

```
                          </element>
                      </sequence>
                      <attribute    name    = "description"         use   = "required"       type    = "string"/>
                      <attribute    name    = "result"    use   = "required"      type    = "resultType"/>
                  </complexType>
          </element>
          <element      name    = "GetMessage">
                  <complexType>
                      <sequence>
                          <element      ref = "testResult"/>
                          <element      ref = "testRnt:Filter"/>
                          <element      ref = "testRptStParameterWthParameter"
minOccurs       = "0"   maxOccurs       = "unbounded"/>
                          <element      ref = "testRptTestPreCondition"                  minOccurs      = "0"
maxOccurs      = "unbounded"/>
                          <element      ref = "testRptTestAssertion"            minOccurs      = "0"
maxOccurs      = "unbounded"/>
                          <element      name    = "r"     minOccurs       = "0"   maxOccurs      = "unbounded">
                              <complexType>
                                  <sequence>
                                      <choice>
                                          <element      ref =
"testRnt:TestPreCondition"/>
                                          <element      ref =
"testRnt:TestAssertion"/>
                                      </choice>
                                      <element      ref = "testRnt:Message"/>
                                      <element      name    = "Ref"       minOccurs      = "0"
maxOccurs      = "unbounded">

                                          <complexType>
                                              <sequence     maxOccurs     =
"unbounded">
                                                  <choice>
                                                      <element

ref = "testRnt:TestPreCondition"/>
                                                      <element

ref = "testRnt:TestAssertion"/>
                                                  </choice>
                                                  <element      ref =
"testRnt:Message"/>
                                              </sequence>
                                          </complexType>
                                      </element>
                                      <element      ref = "testRptMe"
minOccurs       = "0"/>
                                  </sequence>
                                  <attribute     name    = "p"       use   = "required"       type
= "testRnt:formy"/>
                              </complexType>
                          </element>
                          <element      ref = "testRptStatus"            minOccurs      = "0"
maxOccurs      = "unbounded"/>
                      </sequence>
                      <attribute    name    = "description"         use   = "required"       type    = "string"/>
                      <attribute    name    = "testService"         use   = "optional"        type    = "b"/>
                      <attribute    name    = "mask"     use   = "optional"          type    = "mask"/>
                      <attribute    name    = "result"    use   = "required"      type    = "resultType"/>
                  </complexType>
          </element>
          <element      name    = "Filter"      type    = "testRnt:anyType"            string"/>
          <element      name    = "SetPayload"
                  <complexType>
                      <sequence>
                          <element      ref = "testRnt:Result"/>
                          <choice>
                              <element      ref = "testRnt:Message-          "/>
                              <element      ref = "testRnt:Message-          Data"/>
                          </choice>
                          <choice>
```

```
                                    <element      ref = "testRntPubDcaratn"/>
                                    <element      ref = "testRntSM"/>
                                    <element      ref = "testRntPR"/>
                              </choice>
                              <element      ref = "testRntMatn"          minOccurs       = "0"
                              <sequence      minOccurs       = "0"    maxOccurs      = "unbounded">
                                    <element      ref = "testRntMHeadr"/>
                                    <element      ref = "testRntMHeadrVal"/>
                              </sequence>
                        </sequence>
                        <attribute      name    = "description"         use   = "required"       type    = "string"/>
                        <attribute      name    = "result"     use   = "required"       type    = "resultType"/>
                  </complexType>
            </element>
            <element      name    = "TestPreamble">
                  <complexType>
                        <choice>
                              <element      ref = "testRntVerifyContent"/>
                              <element      ref = "testRntValidateContent"/>
                        </choice>
                        <attribute      name    = "description"         use   = "required"       type    = "string"/>
                        <attribute      name    = "result"     use   = "required"       type    =
"testRntPassResultType"/>
                  </complexType>
            </element>
            <element      name    = "TestAssertion">
                  <complexType>
                        <choice>
                              <element      ref = "testRntVerifyContent"/>
                              <element      ref = "testRntValidateContent"/>
                        </choice>
                        <attribute      name    = "description"         use   = "required"       type    = "string"/>
                  </complexType>
            </element>
            <element      name    = "MessageHeader"        type    = "testRntMessageHeaderType"/>
            <element      name    = "MessageHeaderValue"           type    = "testRntanyURI            string"/>
            <element      name    = "Non-      path"       type    = "testRntanyURI             string"/>
            <element      name    = "Query"       type    = "Query"/>
            <element      name    = "PM"          type    = "xM"/>
            <element      name    = "PayloadRef"          type    = "string"/>
            <element      name    = "Attribute"        type    = "BooleanType"/>
            <element      name    = "Non-      M"    type    = "string"/>
            <element      name    = "MessageDeclaration"/>
                  <complexType>
                        <sequence>
                              <any    namespace     = "##other"      processContents        = "lax"     minOccurs       =
"0"    maxOccurs      = "unbounded"/>
                        </sequence>
                  </complexType>
            </element>
            <element      name    = "ValidateContent"/>
                  <complexType          mixed     = "true">
                        <choice>
                              <element      ref = "testRntResult"/>
                        </choice>
                        <attribute      name    = "contentType"         use   = "required"       type    =
"testRntcontentType"/>
                        <attribute      name    = "SchemaPath"          use   = "optional"         type    = "xM"/>
                        <attribute      name    = "result"     use   = "required"       type    =
"testRntPassResultType"/>
                  </complexType>
            </element>
            <element      name    = "VerifyContent"/>
                  <complexType          mixed     = "true">
                        <choice>
                              <element      ref = "testRntResult"/>
                        </choice>
                        <attribute      name    = "result"     use   = "required"       type    =
"testRntPassResultType"/>
```

```
            </xs:complexType>
    </xs:element>
    <xs:element name="Message">
            <xs:complexType>
                    <xs:sequence>
                            <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="id" use="required" type="xs:ID"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="SetParameter">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element name="Name" type="xs:string"/>
                            <xs:element name="Value" type="xs:string"/>
                            <xs:element name="[illegible]" type="[illegible]"/>
                    </xs:sequence>
            </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]">
            <xs:complexType>
                    <xs:choice>
                            <xs:element ref="[illegible]"/>
                            <xs:element ref="[illegible]"/>
                    </xs:choice>
            </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]" type="[illegible]"/>
    <xs:element name="[illegible]" type="[illegible]"/>
    <xs:element name="[illegible]">
            <xs:complexType>
                    <xs:attribute name="[illegible]date" use="required" type=
"[illegible]"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="Parameter" type="xs:string"/>
    <xs:element name="[illegible]"/>
            <xs:complexType/>
    </xs:element>
    <xs:element name="[illegible]">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="[illegible]"/>
                    </xs:sequence>
            </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]">
            <xs:complexType>
                    <xs:sequence>
                            <xs:element ref="[illegible]"/>
                            <xs:element ref="[illegible]Parameter" minOccurs="0"
maxOccurs="unbounded"/>
                            <xs:choice>
                                    <xs:element ref="[illegible]Message"/>
                                    <xs:element ref="[illegible]Message"/>
                                    <xs:element ref="[illegible]"/>
                            </xs:choice>
                    </xs:sequence>
                    <xs:attribute name="[illegible]" use="optional" type="xs:string"/>
                    <xs:attribute name="id" use="required" type="xs:ID"/>
                    <xs:attribute name="[illegible]" use="optional" type="[illegible]"/>
                    <xs:attribute name="[illegible]" use="optional" type="[illegible]"/>
                    <xs:attribute name="[illegible]" use="required" type=
"[illegible]"/>
            </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]">
            <xs:complexType>
```

```
                  <sequence>
                          <any namespace = "##other" processContents = "lax" minOccurs =
0 maxOccurs = [illegible]/>
                  </sequence>
              </complexType>
   </element>
   <element name = [illegible]>
              <complexType>
                  <sequence>
                          <element ref = [illegible] minOccurs = [illegible]/>
                  </sequence>
              </complexType>
   </element>
   <element name = "Thread">
              <complexType>
                  <choice minOccurs = [illegible]>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                  </choice>
                  <attribute name = [illegible] use = "required" type = [illegible]/>
                  <attribute name = [illegible] use = [illegible] type = [illegible]/>
                  <attribute name = "result" use = "required" type =
[illegible]/>
              </complexType>
   </element>
   <element name = [illegible]>
              <complexType>
                  <attribute name = [illegible] use = "required" type = [illegible]/>
              </complexType>
   </element>
   <element name = [illegible] type = [illegible]/>
   <element name = [illegible]>
              <complexType/>
   </element>
   <element name = [illegible]>
              <complexType/>
   </element>
   <element name = [illegible]>
              <complexType>
                  <choice minOccurs = [illegible]>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                  </choice>
              </complexType>
   </element>
   <element name = [illegible]>
              <complexType>
                  <choice minOccurs = [illegible]>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                          <element ref = [illegible]/>
                  </choice>
```

```
            </complexType>
    </element>
    <element      name    = "[illegible]">
            <complexType>
                    <choice>
                            <element      ref = "testRpt:[illegible]"/>
                            <element      ref = "testRpt:Thread[illegible]"/>
                            <element      ref = "testRpt:Thread"/>
                    </choice>
            </complexType>
    </element>
    <element      name    = "Thread[illegible]">
            <complexType>
                    <sequence>
                            <element      ref = "testRpt:Thread"          minOccurs      = "0"   maxOccurs      =
"unbounded"/>
                    </sequence>
            </complexType>
    </element>
    <element      name    = "[illegible]">
            <complexType>
                    <sequence>
                            <choice     maxOccurs      = "unbounded">
                                    <element      ref = "testRpt:SetParameter"/>
                                    <element      ref = "testRpt:Thread"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                            </choice>
                            <element      ref = "testRpt:[illegible]"/>
                            <element      ref = "testRpt:[illegible]"          minOccurs      = "0"   maxOccurs      =
"unbounded"/>
                            <element      ref = "testRpt:[illegible]"         minOccurs      = "0"/>
                    </sequence>
                    <attribute      name    = "[illegible]"       use   = "required"      type    =
"testRpt:[illegible]"/>
            </complexType>
    </element>
    <element      name    = "[illegible]"     type    = "string"/>
    <element      name    = "[illegible]"      type    = "[illegible]"/>
    <element      name    = "[illegible]">
            <complexType>
                    <sequence>
                            <choice     maxOccurs      = "unbounded">
                                    <element      ref = "testRpt:SetParameter"/>
                                    <element      ref = "testRpt:Thread"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                                    <element      ref = "testRpt:[illegible]"/>
                            </choice>
                            <element      ref = "testRpt:[illegible]"/>
                    </sequence>
                    <attribute      name    = "[illegible]"       use   = "required"      type    =
"testRpt:[illegible]"/>
            </complexType>
    </element>
    <element      name    = "[illegible]">
            <complexType>
                    <sequence     maxOccurs      = "unbounded">
                            <element      ref = "testRpt:Thread[illegible]"/>
                    </sequence>
            </complexType>
    </element>
    <element      name    = "[illegible]">
            <complexType>
                    <sequence     maxOccurs      = "unbounded">
                            <element      ref = "testRpt:Thread"/>
```

```
                  </eane>
                  <attrbe     am   = "i"         se  = "reqed"      tp   =
"testRnt:ny"/>
          </cm>
   <cmn>
   <cmn     am   = "Matn">
          <cm>
                  <eane>
                          <emn     ref = "tstRsn"/>
                          <emn     am   = "StMssaen"/>
                                  <cm>
                                          <eane>
                                                  <hie>
                                                          <emn     ref =
"testRnt:MssaeDcaratn"/>
                                                          <emn     ref =
"testRnt:E"/>
                                                          <emn     ref =
"testRnt:MssaeR"/>
                                                  </hie>
                                          </eane>
                                  </cm>
                          </cmn>
                          <emn     ref = "testRnt:"          mcs      = "0"   mxcs     =
"u"/>
                          <emn     ref = "testRnt:Srd"           mcs      = "0"
mxcs     = "u"/>
                  </eane>
                  <attrbe     am   = "Dscrn"         se  = "ren"      tn   = "Strn"
                  <attrbe     am   = "res"    se  = "reqed"      tp   =
"testRnt:aRsntn"/>
          </cm>
   <cmn>
   <cmn     am   = "tstSrvieMatn">
          <cm>
                  <eane>
                          <emn     am   = "Srvie"         tp   =
"testRnt:servieMtn"/>
                          <emn     ref = "testRnt:Rsae"/>
                          <emn     ref = "testRnt:Mfat"/>
                          <emn     ref = "testRnt:"/>
                          <emn     ref = "testRnt:Srvie"/>
                          <emn     ref = "testRnt:At"/>
                          <emn     ref = "testRnt:SdrPrt"/>
                          <emn     ref = "testRnt:RceivrPrt"/>
                          <emn     ref = "testRnt:atiem"               mcs      = "0"
mxcs     = "u"/>
                  </eane>
                  <attrbe     am   = "res"    se  = "reqed"      tp   =
"testRnt:aRsntn"/>
          </cm>
   <cmn>
   <cmn     am   = "MssaeT"       tn   = "Strn"
   <cmn     am   = "atien"
          <cm>
                  <eane>
                          <emn     am   = "m"     tn   = "S"
                          <emn     am   = "h"      tn   = "testRntny           strn
                          <emn     am   = "t"      tn   = "testRntcten"/>
                  </eane>
          </cm>
   <cmn>
   <cmn     am   = "Frn"       tn   = "S"
   <cmn     am   = "Mfat"              tn   = "S"
   <cmn     am   = "StPramterthPramter">
          </cm>
                  <eane>
                          <emn     am   = "m"     tn   = "Strn"
                          <emn     am   = "essn"        tp   = "Strn"
                          <element name="[illegible]" type="testParameterType"/>
                  </sequence>
          </complexType>
   </element>
   <element name="Response" type="[illegible]"/>
   <element name="Result">
          <complexType>
                  <attribute name="[illegible]" use="optional" type="testPointe
[illegible]
                  <attribute name="[illegible]" use="[illegible]" type="[illegible]"/>
                  <attribute name="[illegible]" use="optional" type=
"[illegible]"/>
          </complexType>
   </element>
</schema>
```

# Appendix F (Normative) ebXML Test Service Message Schema

```
<?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/[illegible]/tests/messages"
    xmlns:tns="http://www.oasis-open.org/[illegible]/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/[illegible]/tests/[illegible]"
    xmlns:soap="http://www.oasis-open.org/[illegible]/tests/soap"
    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/[illegible]/tests/[illegible]"
schemaLocation="[illegible]"/>
   <import namespace="http://www.oasis-open.org/[illegible]/tests/soap"
schemaLocation="[illegible]/schemas/soap[illegible]"/>
   <!--
Copyright (C) The Organization for the Advancement of Structured Information Standards
[OASIS]
January 2003. 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 2003. 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 = "Message">
         <complexType mixed = "true">
               <choice>
                     <element ref = "tns:MessageContainer"/>
               </choice>
               <attribute name = "contentType" default = "multipart/related" type =
"string"/>
               <attribute name = "type" default = "text/xml" type = "string"/>
```

```
                </complexType>
        </element>
        <element name = "[illegible]">
                <complexType>
                        <sequence>
                                <element ref = "[illegible]"/>
                        </sequence>
                        <attribute name = "[illegible]" use = "[illegible]" type = "string"/>
                        <attribute name = "[illegible]" default = "[illegible]" type = "string"/>
                        <attribute name = "charset" default = "[illegible]" type = "string"/>
                </complexType>
        </element>
        <element name = "[illegible]Request">
                <complexType>
                        <sequence>
                                <element name = "[illegible]">
                                        <complexType>
                                                <sequence>
                                                        <element ref = "soap:Header" minOccurs = "0"/>
                                                        <element ref = "[illegible]"/>
                                                        <element ref = "[illegible]"/>
                                                </sequence>
                                        </complexType>
                                </element>
                        </sequence>
                </complexType>
        </element>
        <element name = "[illegible]Response">
                <complexType>
                        <sequence>
                                <element name = "[illegible]" type = "[illegible]"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "[illegible]Response">
                <complexType>
                        <sequence>
                                <element name = "[illegible]" type = "[illegible]"/>
                        </sequence>
                </complexType>
        </element>
        <complexType name = "[illegible]">
                <attribute name = "[illegible]" use = "[illegible]" type = "[illegible]"/>
                <attribute name = "[illegible]" use = "[illegible]" type = "string"/>
                <attribute name = "[illegible]" use = "[illegible]" type = "string"/>
                <attribute name = "[illegible]" use = "[illegible]" type = "string"/>
                <attribute name = "[illegible]" use = "[illegible]" type = "string"/>
                <anyAttribute namespace = "[illegible]" processContents = "strict"/>
        </complexType>
        <element name = "[illegible]">
                <complexType>
                        <complexContent>
                                <extension base = "[illegible]">
                                        <sequence>
                                                <element ref = "[illegible]"/>
                                        </sequence>
                                        <attribute name = "[illegible]" use = "[illegible]"
type = "[illegible]"/>
                                </extension>
                        </complexContent>
                </complexType>
        </element>
        <element name = "[illegible]Request">
                <complexType>
                        <sequence>
                                <element ref = "[illegible]"/>
                        </sequence>
                </complexType>
```

```
    </xs:element>
    <xs:element name="[illegible]">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="[illegible]" type="tns:[illegible]"/>
                <xs:element ref="tns:[illegible]"/>
                <xs:element ref="tns:[illegible]"/>
                <xs:element ref="tns:[illegible]sts" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]" type="[illegible]"/>
    <xs:element name="[illegible]Response">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="[illegible]" type="[illegible]"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]">
        <xs:complexType>
            <xs:sequence>
                <xs:any namespace="##other" processContents="lax" minOccurs=
"0" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="[illegible]" type="[illegible]"/>
    <xs:element name="[illegible]" type="tns:nonEmptyString"/>
    <xs:element name="[illegible]" type="tns:nonEmptyString"/>
    <xs:element name="[illegible]" type="tns:nonEmptyString"/>
    <xs:element name="[illegible]" type="tns:nonEmptyString"/>
    <xs:simpleType name="nonEmptyString">
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
        <xs:restriction base="[illegible]">
            <xs:enumeration value="[illegible]"/>
            <xs:enumeration value="[illegible]"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
        <xs:restriction base="[illegible]">
            <xs:enumeration value="[illegible]"/>
            <xs:enumeration value="[illegible]"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
        <xs:restriction base="[illegible]">
            <xs:enumeration value="[illegible]"/>
            <xs:enumeration value="[illegible]"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="parameterType">
        <xs:restriction base="[illegible]">
            <xs:enumeration value="parameter"/>
            <xs:enumeration value="string"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]Type">
        <xs:restriction base="[illegible]">
            <xs:enumeration value="[illegible]"/>
            <xs:enumeration value="[illegible]"/>
            <xs:enumeration value="[illegible]"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="[illegible]">
```

```
            <restriction base = [illegible]
                    <enumeration value = [illegible]
                    <enumeration value = [illegible]
            </restriction>
    </simpleType>
    <simpleType name = [illegible]
            <restriction base = [illegible]
                    <enumeration value = [illegible]
                    <enumeration value = [illegible]
                    <enumeration value = [illegible]
            </restriction>
    </simpleType>
    <simpleType name = [illegible]
            <restriction base = [illegible]
                    <enumeration value = [illegible]
                    <enumeration value = [illegible]
            </restriction>
    </simpleType>
    <element name = [illegible]"          type = [illegible]
    <element name = [illegible]             type = [illegible]
    <element name = [illegible]
            <complexType>
                    <sequence>
                            <element name = [illegible]"    type = [illegible]
                            <element name = [illegible]"     type = [illegible]
                            <element name = [illegible]"    type = [illegible]
                    </sequence>
            </complexType>
    </element>
    <element name = [illegible]
            <complexType>
                    <sequence>
                            <element ref = [illegible]
                    </sequence>
            </complexType>
    </element>
    <element name = [illegible]
            <complexType>
                    <sequence>
                            <element name = [illegible]       minOccurs = [illegible]
                                    <complexType>
                                            <sequence>
                                                    <element name = [illegible]    type = [illegible]
                                                    <element name = [illegible]"     type =
[illegible]
                                            </sequence>
                                    </complexType>
                            </element>
                    </sequence>
            </complexType>
    </element>
    <element name = [illegible]
            <complexType>
                    <sequence>
                            <element name = [illegible]       minOccurs = [illegible]
                                    <complexType>
                                            <sequence>
                                                    <element name = [illegible]    type = [illegible]
                                                    <element name = [illegible]"     type =
[illegible]
                                            </sequence>
                                    </complexType>
                            </element>
                    </sequence>
            </complexType>
    </element>
</schema>
```

# Appendix G  ConfigurationGroup Schema

```
<?  ersn  = "1  enn  = "
<!  Generated  b  X  Ahorty  frm  to  @  http  hem-  >
<schem  n  = "http  hem"
    targetNmsace  = "http  sservice  ntests"
    xmlnsTst  = "http  sservice  ntests"

    ersn  = "1
    elementFormDefault  = "  "
    attributeFormDefault  = "  "
  >
<!--
Copyright  (C)  The  Organization  for  the  Advancement  of  Structured  Information  Standards
[OASIS]
January  20  All  Rights  Reserved
This  document  and  translation  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.
- >

  <simpleType  name  = "HeaderType">
        <restriction  base  = "  ">
              <enumeration  value  = "MessageHeader-  Type"/>
              <enumeration  value  = "MessagePartType"/>
              <enumeration  value  = "Header-  Type"/>
              <enumeration  value  = "StartType"/>
              <enumeration  value  = "CharsetType"/>
              <enumeration  value  = "Type"/>
              <enumeration  value  = "Param"/>
        </restriction>
  </simpleType>
  <simpleType  name  = "ContentType">
        <restriction  base  = "  ">
              <enumeration  value  = "  "/>
              <enumeration  value  = "Hte"/>
              <enumeration  value  = "  "/>
              <enumeration  value  = "Signature"/>
              <enumeration  value  = "Sadd"/>
              <enumeration  value  = "SchemaType"/>
              <enumeration  value  = "Other"/>
        </restriction>
  </simpleType>
  <simpleType  name  = "MethodType">
        <restriction  base  = "  ">
              <enumeration  value  = "Methd"/>
              <enumeration  value  = "Sha"/>
        </restriction>
  </simpleType>
  <simpleType  name  = "RoleType">
        <restriction  base  = "  ">
              <enumeration  value  = "Mal  service"/>
              <enumeration  value  = "Remote-  service"/>
              <enumeration  value  = "Reactor"/>
        </restriction>
  </simpleType>
```

```
    <simpleType         name    = "MessageContextType">
            <restriction      base   = [illegible]
                    <enumeration        value     = "true"/>
                    <enumeration        value     = "false"/>
            </restriction>
    </simpleType>
    <simpleType         name    = "RequirementType">
            <restriction      base   = [illegible]
                    <enumeration        value     = "Required"/>
                    <enumeration        value     = "StronglyRecommended"/>
                    <enumeration        value     = "Recommended"/>
                    <enumeration        value     = [illegible]/>
            </restriction>
    </simpleType>
    <simpleType         name    = "non-empty-          string">
            <restriction      base   = "string">
                    <minLength        value     = "1"/>
            </restriction>
    </simpleType>
    <simpleType         name    = [illegible]>
            <restriction      base   = [illegible]
                    <enumeration        value     = [illegible]/>
                    <enumeration        value     = "Replace"/>
            </restriction>
    </simpleType>
    <simpleType         name    = "ActionType">
            <restriction      base   = [illegible]
                    <enumeration        value     = "Reset"/>
                    <enumeration        value     = [illegible]
            </restriction>
    </simpleType>
    <simpleType         name    = [illegible]>
            <restriction      base   = [illegible]
                    <enumeration        value     = "Parameter"/>
                    <enumeration        value     = "Namespace"/>
            </restriction>
    </simpleType>
    <simpleType         name    = "ParameterType">
            <restriction      base   = [illegible]
                    <enumeration        value     = "Path"/>
                    <enumeration        value     = "String"/>
                    <enumeration        value     = [illegible]/>
            </restriction>
    </simpleType>
    <element      name    = [illegible]>
            <complexType>
                    <sequence>
                            <element      name    = [illegible]        type    = [illegible]            string"
minOccurs       = "0"/>
                            <element      ref  = [illegible]/>
                            <element      name    = [illegible]        type    = [illegible]
                            <element      name    = [illegible]        type    = [illegible]
                            <element      name    = [illegible]      type    = [illegible]
                            <element      name    = [illegible]       type    = [illegible]            string"/>
                            <element      name    = [illegible]          type    = [illegible]/>
                            <element      name    = [illegible]       type    = [illegible]            string"/>
                            <element      name    = [illegible]        type    = [illegible]            string"/>
                            <element      name    = [illegible]                minOccurs       = "0"    maxOccurs      =
[illegible]
                                    <complexType>
                                            <sequence>
                                                    <element      name    = [illegible]    type    =
[illegible]        empty    string"/>
                                                    <element      name    = [illegible]     type    =
[illegible]
                                                    <element      name    = [illegible]     type    =
[illegible]/>
                                            </sequence>
                                    </complexType>
```

```
                        </element>
                    </sequence>
                    <attribute     name   = "[illegible]"    use  = "required"      type   = "[illegible]
                </complexType>
            </element>
            <element      name    = "[illegible]"      type    = "[illegible]st[illegible]"/>
            <element      name    = "[illegible]at[illegible]"/>
                <complexType>
                    <sequence>
                        <element     ref = "[illegible]st[illegible]at[illegible]"                    maxOccurs     =
"[illegible]"
                    </sequence>
                </complexType>
            </element>
</schema>
```

# Appendix H WSDL Definitions for Test Service

Below is  the WSDL definition file for Test Service notification method

```
[illegible]    ersi[illegible]         en[illegible]
<[illegible]  e[illegible]ed   [illegible]h  [illegible]     [illegible]    rel  3 U (http[illegible])              [illegible]  [illegible]   [illegible]ss  [illegible]rsa[illegible])
>
[illegible]        [illegible]http[illegible]chem[illegible]s[illegible]ap[illegible]
[illegible]soa[illegible]http[illegible]chem[illegible]s[illegible]ap[illegible]soap[illegible]                  [illegible]ts[illegible]asi[illegible]stce[illegible]
i[illegible]testservi[illegible]ew[illegible]           [illegible]http[illegible]chem[illegible]s[illegible]ap[illegible]
[illegible]http[illegible]si-               po[illegible]ce[illegible]         [illegible]testsm[illegible]ssags"
targ[illegible]Nespace[illegible]asi[illegible]stce[illegible]testservi[illegible]ew[illegible]
[illegible]tr[illegible]rvi[illegible]e>
   [illegible]pt        [illegible]spce[illegible]http[illegible]s[illegible]po[illegible]ce[illegible]                [illegible]testsm[illegible]ssags"
[illegible]at[illegible]chem[illegible]s[illegible]rvi[illegible]essa[illegible]s[illegible]
   [illegible]ssae        [illegible]fat[illegible]
          [illegible]rt       [illegible]fat[illegible]            e[illegible]fat[illegible]
   [illegible]ssae>
   [illegible]ssae        [illegible]fat[illegible]spe[illegible]
          [illegible]rt       [illegible]fat[illegible]spe"              e[illegible]fat[illegible]spe[illegible]
   [illegible]ssae>
   [illegible]ssae        [illegible]ssae[illegible]
   [illegible]t[illegible]         [illegible]fat[illegible]t[illegible]
          [illegible]t[illegible]          to the [illegible]fat[illegible]       [illegible]erf[illegible]ce   [illegible]  [illegible]st  F[illegible]k
spec[illegible]at[illegible]
          [illegible]rat[illegible]         [illegible]ssae[illegible]fat[illegible]
                 [illegible]          [illegible]ssae[illegible]fat[illegible]
                 [illegible]          [illegible]ssae[illegible]fat[illegible]spe[illegible]
          [illegible]rat[illegible]
   [illegible]t[illegible]>
   [illegible]        [illegible]fat[illegible]                    t[illegible]fat[illegible]t[illegible]
          s[illegible]          st[illegible]"
tra[illegible]tt[illegible]chem[illegible]s[illegible]tt[illegible]
          [illegible]rat[illegible]         [illegible]ssae[illegible]fat[illegible]
                 [illegible]rat[illegible]       s[illegible]ssae[illegible]fat[illegible]
                 [illegible]>
                        [illegible]        [illegible]era[illegible]
                 [illegible]>
                 [illegible]>
                        [illegible]rt[illegible]te[illegible]
                               [illegible]rt>
                                      [illegible]      [illegible]rts[illegible]fat[illegible]         [illegible]era[illegible]
                               [illegible]rt>
                               [illegible]rt[illegible]
                        [illegible]rt[illegible]te[illegible]
                 [illegible]>
          [illegible]rat[illegible]
   [illegible]
   [illegible]erv[illegible]e        [illegible]st[illegible]rvi[illegible]e[illegible]
          [illegible]t[illegible]e         [illegible]fat[illegible]       servi[illegible]e   [illegible]  [illegible]    e[illegible]    [illegible]st  F[illegible]k    [illegible]ersi[illegible]
[illegible]at[illegible]
          [illegible]pt       [illegible]fat[illegible]
[illegible]fat[illegible]
                 [illegible]es[illegible]ess      [illegible]at[illegible]tt[illegible]fat[illegible]rvi[illegible]e[illegible]
          [illegible]t>
   [illegible]erv[illegible]e>
   [illegible]t[illegible]s         [illegible]  the the [illegible]at[illegible]    abstract  [illegible]   servi[illegible]e  [illegible]        [illegible]  the
[illegible]    e[illegible]   [illegible]st  [illegible]rvi[illegible]e[illegible]at[illegible]
[illegible]>
```

Test Driver Initiator

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Mike Kass (Personal)
-->
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://www.oasis-open.org/tc/ebxml-
iic/testservice/wsdl" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:xsd="http://www.oasis-open.org/committees/ebxml-iic/tests/messages"
targetNamespace="http://www.oasis-open.org/tc/ebxml-iic/testservice/wsdl"
name="InitiatorDriver">
   <wsdl:import namespace="http://www.oasis-open.org/committees/ebxml-iic/tests/messages"
location="schemas/DriverMessages.xsd"/>
   <wsdl:message name="InitiatorRequest">
         <wsdl:part name="InitiatorRequest" element="xsd:InitiatorRequest"/>
   </wsdl:message>
   <wsdl:message name="InitiatorResponse">
         <wsdl:part name="InitiatorResponse" element="xsd:InitiatorResponse"/>
   </wsdl:message>
   <wsdl:portType name="InitiatorPort">
         <wsdl:documentation>Maps to the Initiator interface of Test Framework
spec</wsdl:documentation>
         <wsdl:operation name="Initiator">
               <wsdl:input message="tns:InitiatorRequest"/>
               <wsdl:output message="tns:InitiatorResponse"/>
         </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="InitiatorBinding" type="tns:InitiatorPort">
         <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
         <wsdl:operation name="Initiator">
               <soap:operation
soapAction="http://www.oasis-open.org/tc/ebxml-iic/testservice/Initiator"/>
               <wsdl:input>
                     <mime:multipartRelated>
                           <mime:part>
                                 <soap:body parts="InitiatorRequest"
use="literal"/>
                           </mime:part>
                           <mime:part/>
                     </mime:multipartRelated>
               </wsdl:input>
               <wsdl:output>
                     <soap:body use="literal"/>
               </wsdl:output>
         </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="TestDriver">
         <wsdl:documentation>Query Manager service of the ebXML Test Framework version 1.0
</wsdl:documentation>
         <wsdl:port name="InitiatorPort" binding="tns:InitiatorBinding">
               <soap:address
location="http://[illegible]/InitiatorDriver"/>
         </wsdl:port>
   </wsdl:service>
   <wsdl:documentation>This is the the concrete and abstract WSDL service definition of the
ebXML Test Driver</wsdl:documentation>
</wsdl:definitions>
```

Test Service configure

```
<?      ersi     = "1      end      = "

<!    eded   wh   X        @      rel  3 U (http/)                  y  X     Xss  (ersa)
>
wdfi                 n      = "http/chemsoapg/                          nsap      =
"http/chemsoapgdap/                          nts       = "oasisnstcen
ctestse   rviewd             nsd        = "http/chemsoapg/                           nsd       =
"http/si-            porg/cen            ctestsmessags"        targtNmspace      =
"oasisnstcentestservewd                                        am    = "PetrServe">
   spt            amspace     = "http/siporg/cen                           ctestsmessags"
lcatn      = "schemse/ServeMssagesx"
   wmssae          am    = "atnRest">
          wprt         am    = "atnRest"              eemn      = "atnRest">
   wmssae>
   wmssae          am    = "atnRspe">
          wprt         am    = "gatoRspe"               eemn      =
"atnRspe">
   wmssae>
   wrtp              am    = "atnrtp">
          dcmtatip              to the gato          nerface     o   Test   Famwk
spcdmntatn
          wratn           am    = "atn">
                  wi          mssae    = "atnRest">
                  wn           mssae    = "atnRspe">
          wratn
   wrtp>
   wn              am    = "atnn                        tp    = "natntp">
          sag            sty    = "dcmt"       transpt     =
"http/chemsoapgsoaphttp"
          wratn           am    = "atn">
                  soapratn          soapctn      =
"oasisstestservnatnatn">
                  wi>
                         sab          se   = "lera">
                  wi>
                  wn>
                         sab          se   = "lera">
                  wn>
          wratn
   wn
   wserve          am    = "TestServe">
          dcmtatie             Orgager        serve    o  X      eX     Test   Famwk     ersn
dcmntatn
          wpt          am    = "gaton                        g        =
"natnn">
                  soapdess        latn      =
"http/Rotatserve">
          wpt>
   wserve>
   dcmtatis               s  the the oatw        abstract  W     serve    din         o    the
X      eX     Test  ServedmntatN
wdfi>
```

Test Driver notify

```
<    eded   wh   X      @     rel  3 U (ttpp)                  y  H    Kss  (rsa)
>
wdfi                 a="ttpschemsbapgd
asaptttpschemsbapgdap/                               atstasiamstcem
iitestservewsd             awsttpschemsbapgd
axsttpssi-                    porgcem            i/testsmessags"
targtNmspcetasiamstcemtestservewsd
amesitrarive>
   wpt           amspcettpssiporgcem                            i/testsmessags"
oatschemsemrivemssagsxd
   wmssae         amNfat8
           wart        amNfatn                emnsNfat8
   /mssae>
   wmssae         amNfatmspe>
           wart        amNfatmspe"                    emnsNfatmspe>
   wmtm           amNfatmtm>
           dmatip               to the Nfatn         merface    o  Test  Famwk
secdmatm
           wratn            amN
                     wn           mssaetsNfat8
                     wmn           mssaetsNfatmspe>
           /wratn
   /wmtm>
   wm             amNfatm                           tmtsNfatmtm>
           sapg              styeon"
transrttschemsbapgdttm
           wratn            amN
                     saprato
seAtmasiemtestserveRceiNfat8
                     wm>
                              mhartRated
                                        mart>
                                                  sab         artsNfatn             sera>
                                        /mart>
                                        mart>
                              /mhartRated
                     /wm>
                     wmn>
                              sab        sera>
                     /wmn>
           /wratn
   /wm
   wderve         amstDrRceiSrive>
           dmatie               Rcev    serve   o  A      eN     Test  Famwk      ersn
dmatm
           wt        amN                           MsNfatm
                     saddess       oattpmRDRceiSrive>
           /wt>
   /derve>
   dmatis              s  the the matv       abtract  W    serve   dfn          o   the
A      eN    Test  Srivedmatm
/dfn>
```

# Appendix I Terminology

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

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

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

# Appendix J References

## J.1 Normative References

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

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

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

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

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

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

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

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

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

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

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

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

[SOAP] W3C-Draft-Simple Object Access Protocol (SOAP) v1.1, Don Box, DevelopMentor; David Ehnebuske, IBM; Gopal Kakivaya, Andrew Layman, Henrik Frystyk Nielsen, Satish Thatte, Microsoft; Noah Mendelsohn, Lotus Development Corp.; Dave Winer, UserLand Software, Inc.; W3C Note 08 May 2000,

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

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

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

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

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

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

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

## J.2 Non-Normative References

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

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

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

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

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

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

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

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

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

# Appendix K Acknowledgments

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

## K.1 IIC Committee Members

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

# Appendix L Revision History

| 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) |

# Appendix M Notices

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

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

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