Creating A Single Global Electronic Market

ebXML Test Framework

Committee Specification Version 1.1 DRAFT

OASIS ebXML Implementation, Interoperability and Conformance Technical Committee

14 July, 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
Han Kim Ngo, NIST <han.ngo@nist.gov>

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

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

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

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

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

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

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

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

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

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 over the Test Harness. Each Test Case includes: configuration material (CPA data), message material associated with eachoperation and test verification conditions that define 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 received messages. These operations support and automate the execution of Test Cases.

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

3.1 The Test Driver

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

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

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

3.1.1 Functions

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

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

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

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

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

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

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

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

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

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

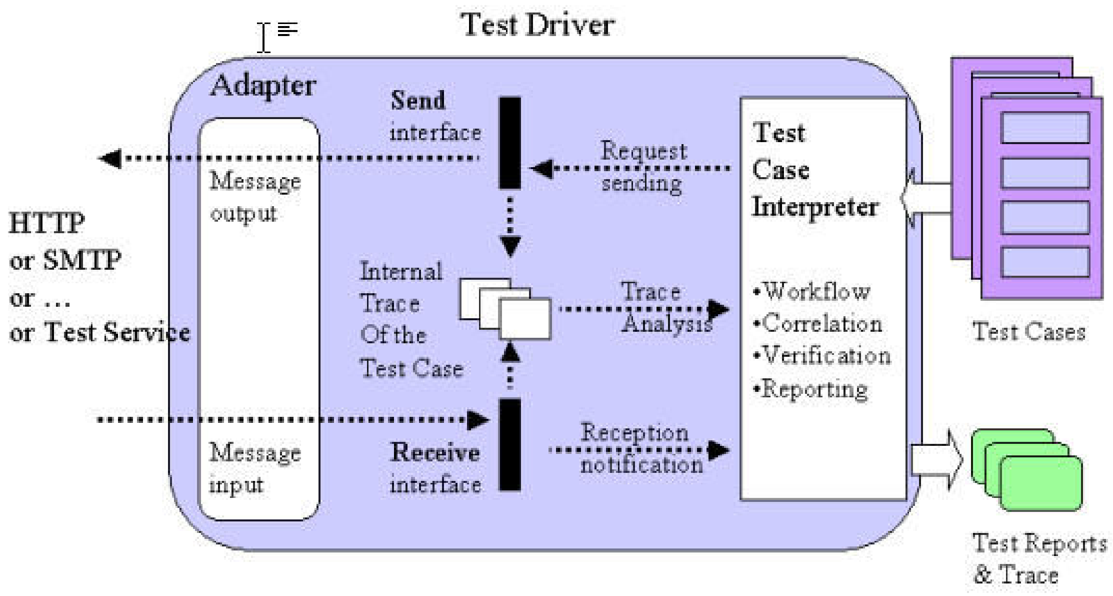


Figure 1- The Test Driver: Functions and Data Flows

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.

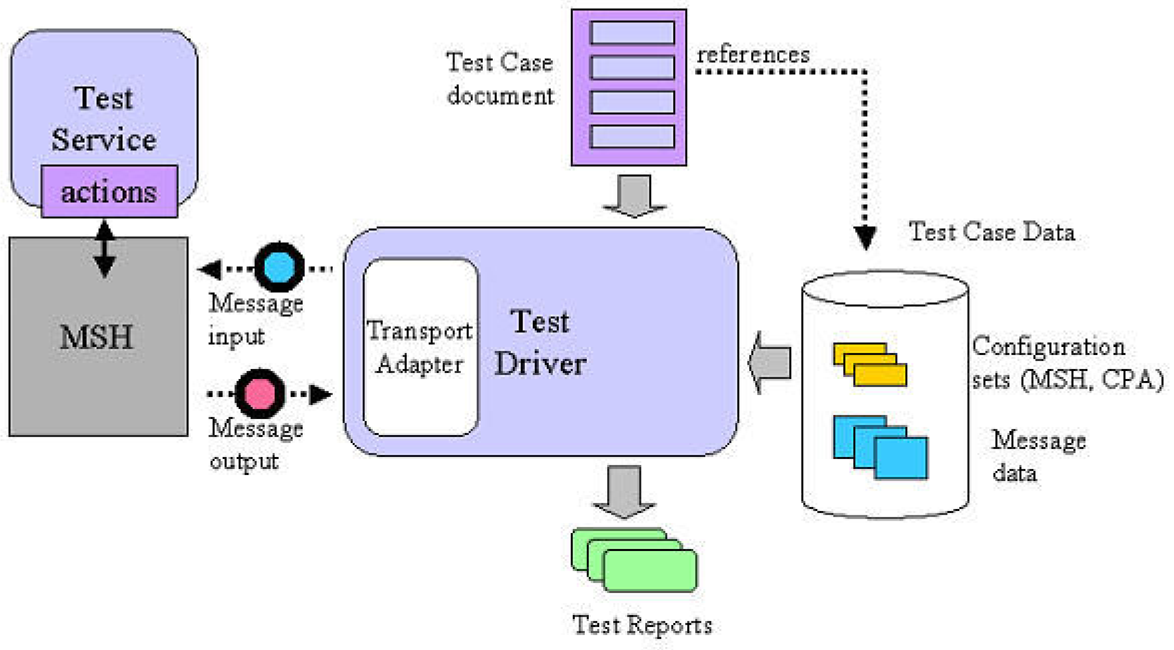


Figure 2- Test Driver: Connection Mode

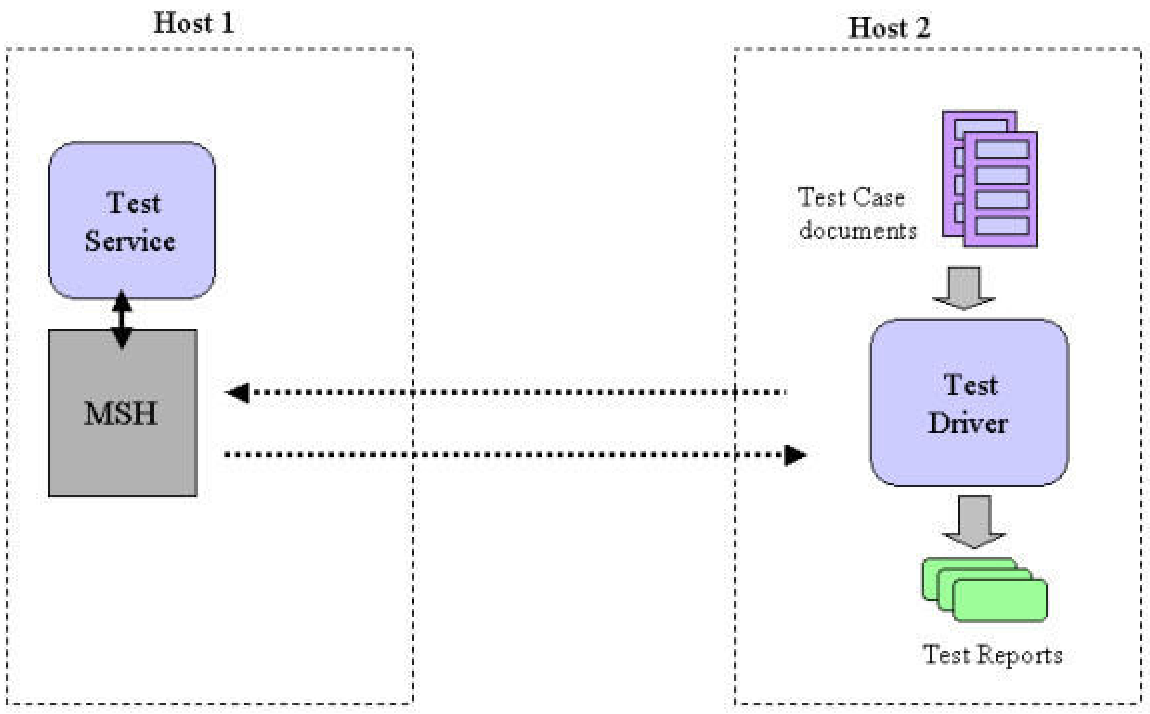


Figure 3 – Test Driver: Remote Connection Mode

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 method of the Test Service(represented by the the "Initiator" scripting element) , instructs the MSH it has been interfaced with to construct and send a message. This method also MUST interface with the Test Service at application level. When invoked by a call that contains message data, the method generates a sending request using that MSH's API.

Receiving: As all actions of the Test Service may participate in the execution of a Test Case (i.e. of its Threads), the Test Driver needs to be aware of their invocation by incoming notification messages provided by the Test Service. Each of these actions notify the Test Driver through its "Receive" interface, passing received message data, as well as response data. In 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. Alternately, the Test Service MAY notify the Test Driver of receive messages and errors via RPC.

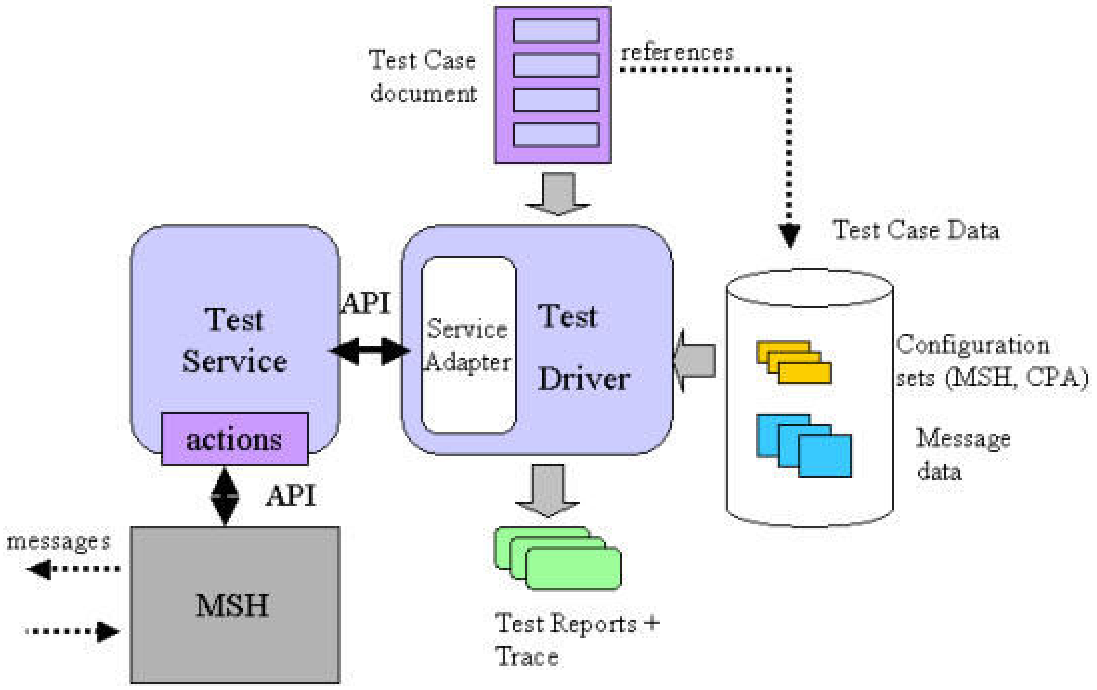


Figure 4 – Test Driver: Service Mode

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.

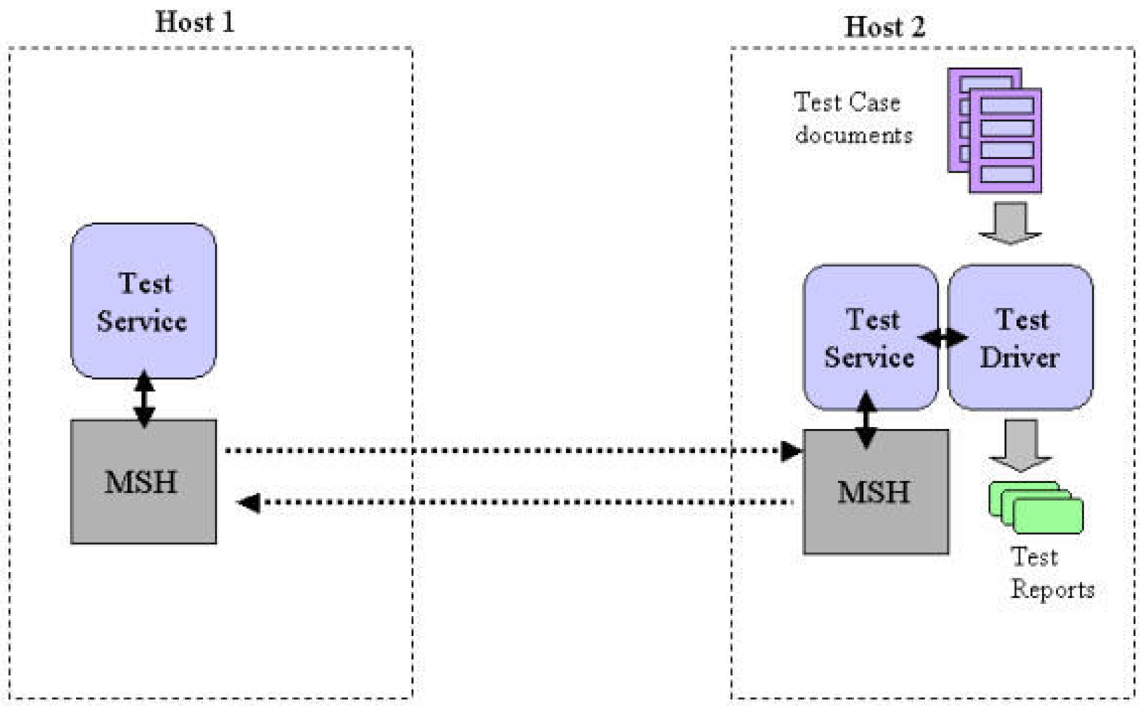


Figure 5 – Test Driver in Service Mode: Point-to-Point Interoperability Testing of Message Handlers

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 messagesto be transmitted. The Test Actions are predefined, and are part of the Test Framework). 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.

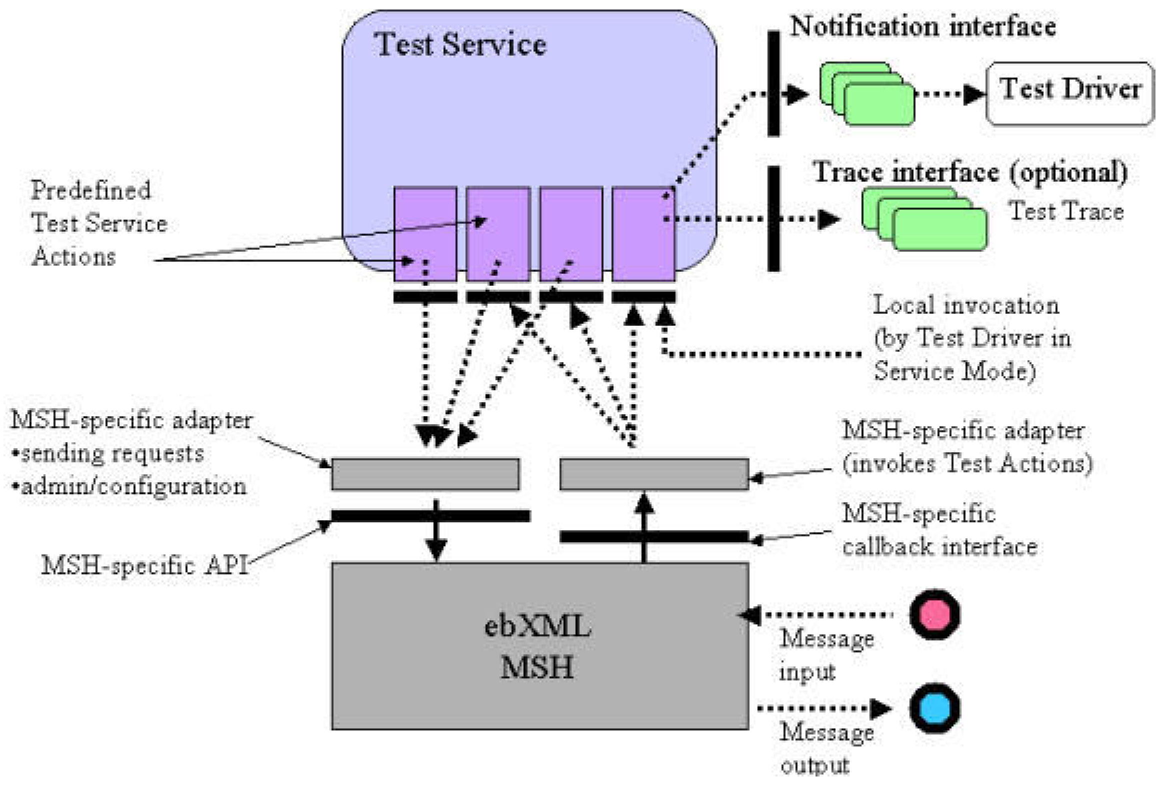

Test Service
Notification interface
Test Driver
Trace interface (optional)
Test Trace
Predefined Test Service Actions
Local invocation (by Test Driver in Service Mode)
MSH-specific adapter
•sending requests
•admin/configuration
MSH-specific adapter (invokes Test Actions)
MSH-specific API
MSH-specific callback interface
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 either locally, or via RPC.

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 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 Test Service 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.

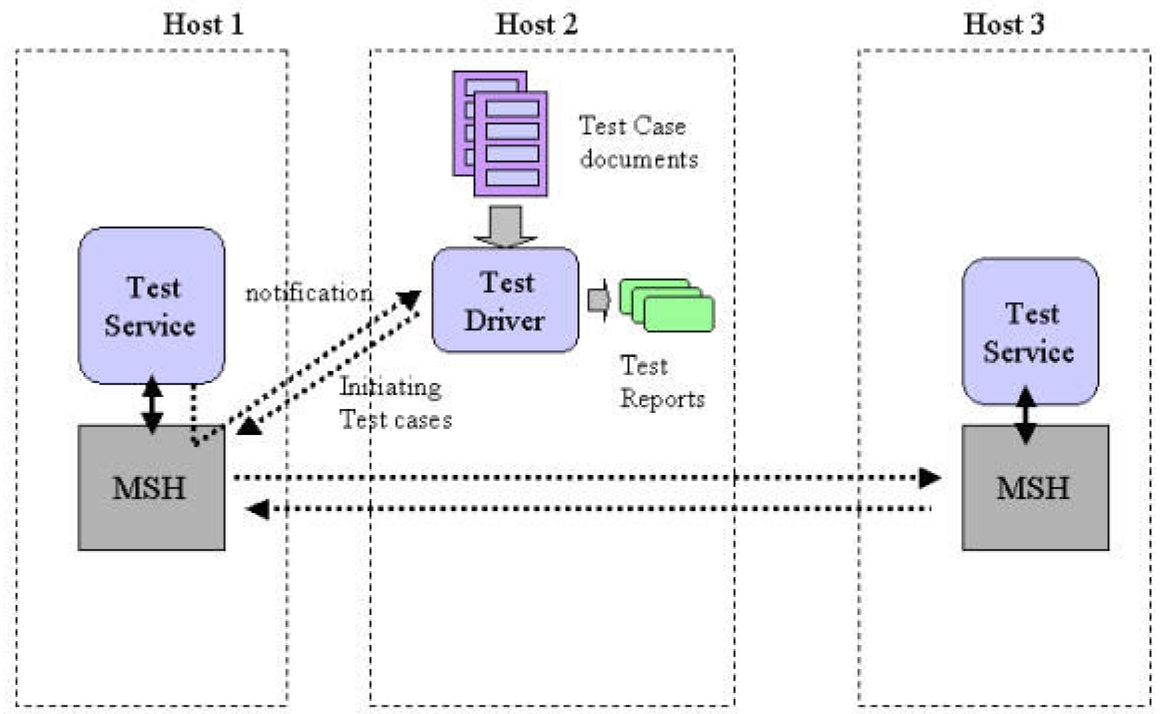


Figure 7 – Example of Remote Reporting Mode : The Interoperability Test Center Model

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 sends the TestServiceConfigurator element content found at the beginning of the TestSuite document to the Test Service via its 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 mode)

ResponseURL (destination for response messages in any mode)

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

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 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 Messaging Services 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 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 an 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 TestServiceConfiguration parameters supplied by the Test Driver. 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 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 both loop and 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 "`urn:ebXML:iic:test`" / **"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 **report** method of the Test Service. The report method notifies the Test Driver of the 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 Test Service message initiation, configuration and error notification. . 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, builds the message envelope, attaches any referenced payloads, and sends the message. The method returns an XML notification document with a "pass/fail" Result element.

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

These two 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 – Its "notify" method accepts incoming notification messages from the Test Service andpasses them to the Test Service for storage in its Message Store. Notification messages include messages received by the Test Service (when the Test Service is in "reporting mode") and application error messages generated by the Test Service and response messages from the Test Service referencing success/failure of received message payload verification.

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

Table 1 – Initiator method description

Semantic Description: The Initiator call instructs the Test Service to generate a new message. The new message content is provided as a argument to the initiator call. Any payload content is provided as attachments in the SOAP message, and have the same content-Id as defined in the message Declaration. The envelope 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 Declaration .

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.

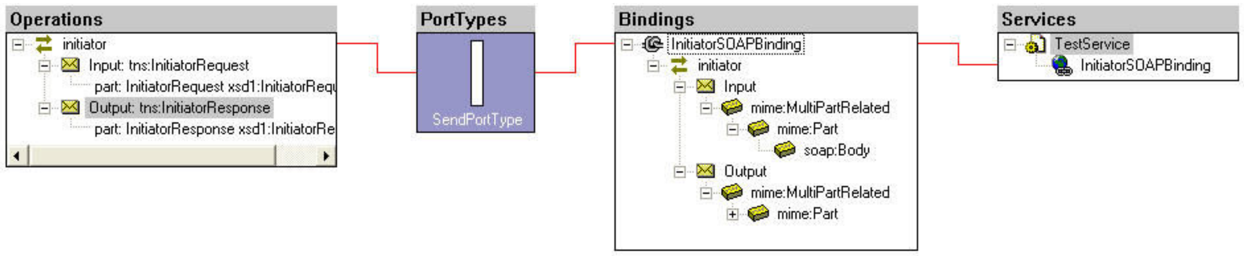


Figure 8 – WSDL diagram of the Initiator SOAP method

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 (an XML document containing a boolean result element)	configurator (ConfigurationList list) Passes the configuration parameters to the Test Service	Test Service fails to configure properly

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 payload verification payload integrity verification.

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

3.2.5.3.2 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.

3.2.5.3.3

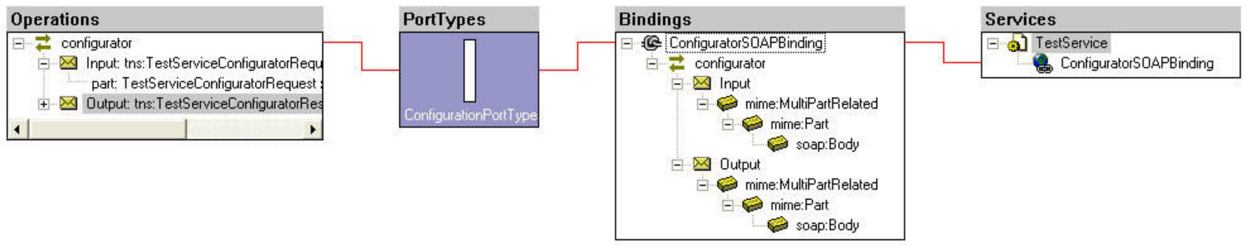


Figure 9 – WSDL diagram of the configurator SOAP method

3.2.5.4 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 (NotificationMessage message, MessagePayloadList messagePayloads) Passes the, received notification message envelope and any encapsulated message payloads to the Test Driver	Test Driver fails to accept the notification message

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.

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

An **application error notification message** 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.

Application Error Notification Message Format:

Error notification messages 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.

An **MSH Error notification message** captures "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.

MSH Error Notification Message Format:

Error notification messages 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.

Received Message notifications 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 notify method is a "copy" of the received message envelope and an encapsulated list of any attachments provided with the message. The message contains.

Received Message Notification Format:

All notification messages generated by the report method will have the same characteristics a normal message (i.e. have a MessageHeader with refToMessageId, ConversationId, CPAId). Additionally, an encapsulated list of message attachments that were a part of the received message is passed to the Test Driver. The 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 (notify), synch type (synchronous or asynchronous), and id.

Payload verification notification messages inform the Test Driver of the result of the PayloadVerify action of the Test Service. A notification message consists of a message envelope with the same characteristics a normal response message (i.e. have a MessageHeader with refToMessageId, ConversationId, CPAId corresponding to that of the incoming 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 (payloadVerify), synch type (synchronous or asynchronous), and id.

Received Payload Verification Format:

All payload verification messages will have the same characteristics a normal message (i.e. have a MessageHeader with refToMessageId, ConversationId, CPAId). Additionally, the notify method will pass to the Test Driver an XML document describing the result of the payload verification. 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.

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

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 at the application level. Only message envelopes, and accompanying message payloads are passed on to the Test Driver's "Receive" interface.

3.2.5.4.1 WSDL representation of the Test Driver 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 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.

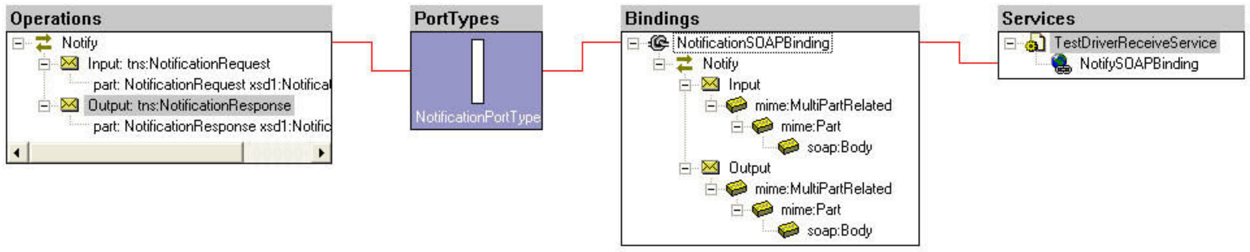


Figure 10 – WSDL diagram of the Test Driver notify SOAP method

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 ebXML (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 if necessary.

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 testing requirements profile.

Execution of Test Cases against the specified testing requirements 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

:

A Test Case is a workflow of Test Threads. 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. In addition, a Thread may test an assertion of expected message content from a received message. A Thread 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 sequence of test operations,Two instances of the same Test Case will be distinguished by distinct ConversationId and MessageId values in the generated messages (referred to as the message "context"). An example of a sequence of Threads associated with an MS Conformance Test Case is:

Thread 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.

Thread 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 Declarations

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

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 Testing 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, and define the collaborations that execute on these implementations.

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

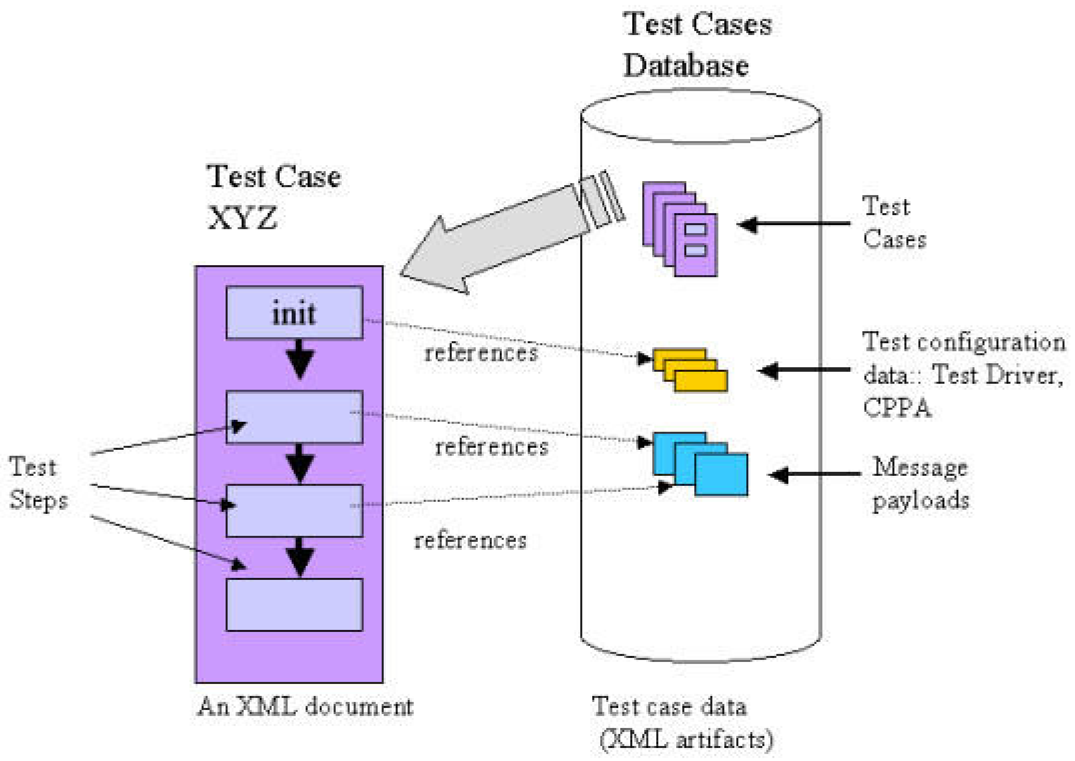


Figure 11 – Test Case Document and Database

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 Threads. Each Thread 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.

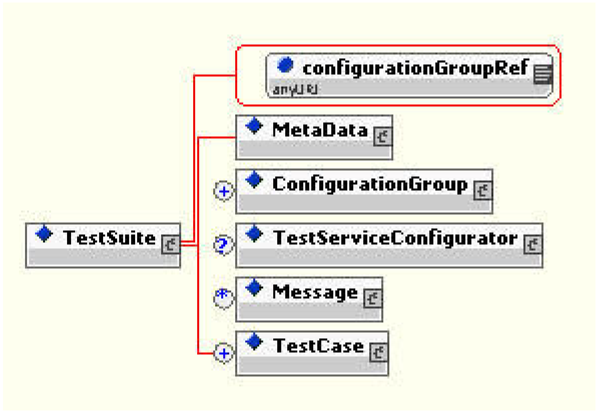


Figure 12 – Graphic representation of basic view of TestSuite schema

Definition of Content

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

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

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.

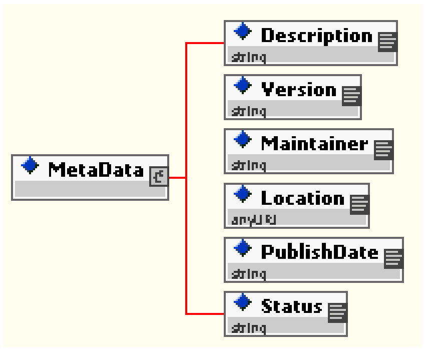


Figure 13 – Graphic representation of expanded view of the Metadata element

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 element contains configuration data for both 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 and Thread 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 object hierarchy, then the ConfigurationGroup defined at that level again takes precedence over any defined at a lower level by a descendent test object.

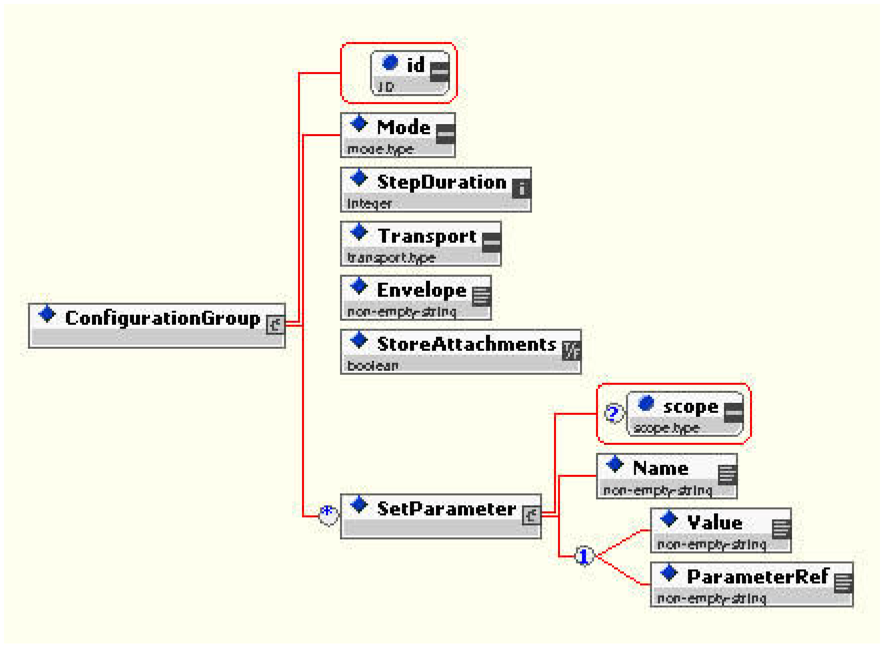


Figure 14 – Graphic representation of expanded view of the ConfigurationGroup element

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	
Mode	One of two types for the Test Driver, (service | connection)		Required	
StepDuration	Timeout (in seconds) of a message send or receiver operation		Required	
Transport	Directs the Test Driver as to which transport protocol to use to carry messages.		Required	
Envelope	Directs the Test Driver as to which Messaging envelope type it is constructing		Required	
StoreAttachments	Toggle switch directing Test Driver to ignore (false) or store (true) incoming message attachments		Required	
SetParameter	Container for ”ad-hoc”name/value pair used by the Test Driver for configuration or possibly for message payload content construction		Optional	
Name	Name for the ConfigurationItem		Required	
Value	Value of the ConfigurationItem		Optional	
ParameterRef	Name of previously defined parameter, whose value is substituted for the value of this parameter		Optional	

Table 7 provides a list of ConfigurationGroup element and attribute content

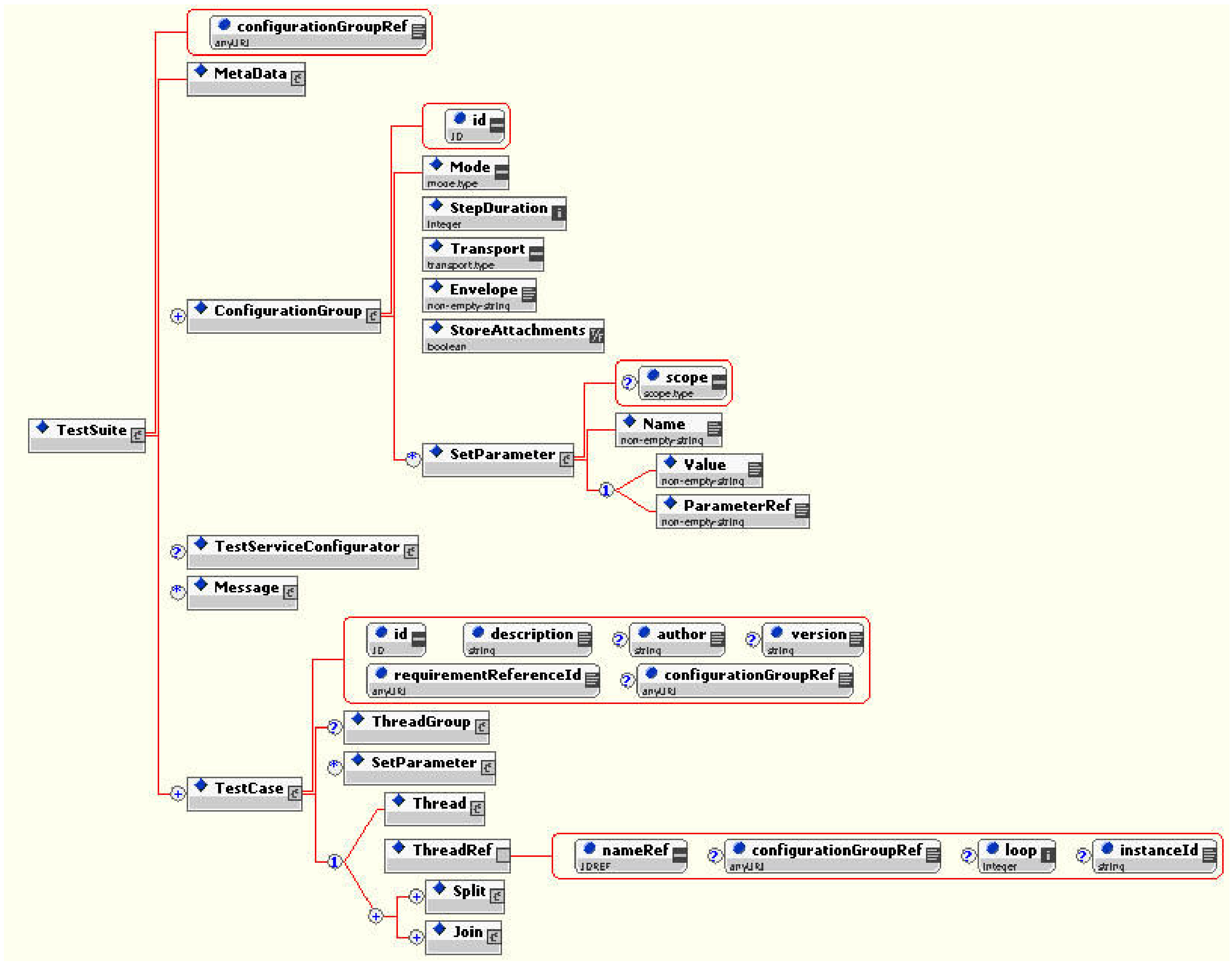


Figure 15 – Graphic representation of hierarchical use of the ConfigurationGroup via reference at TestSuite, TestCase and ThreadRef Thread levels in the test object hierarchy

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 run time generated parameters:

ConversationId – Unique to each new Test Case

MessageId – Unique to each message generated by the PutMessage instruction

Timestamp - Unique to each message generated by the PutMessage instruction

These run time parameters MUST have the names specified above (case sensitive).

Additional message content MAY be provided through parameter definitions in the current ConfigurationGroup, or through a SetParameter or SetXPathParameter operation within a Thread. This includes message content such as:

CPA Id

Service

Action

Sender Party Id

Receiver Party Id

The parameters listed above canbe given any parameter name the test writer chooses. However, the test writer MUST reference the parameter in XSLT mutator stylesheets, or in XPath expressions using the identical name (case sensitive) with which it was defined using the SetParameter instruction.

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

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

Parameters MAY be used by an XSL stylesheet or XUpdate document to “mutate” a Declaration into a valid message. They are passed to the XSL or XUpdate processor via name reference.

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.

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 TestCase or) Thread Thread 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 Thread that is intended).

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

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

4.2.2.1.1 Exception Conditions

A Test Driver MUST generate an exception and terminate the Test Case with a result of “undetermined” if it cannot mutate 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 mode of the Test Driver.

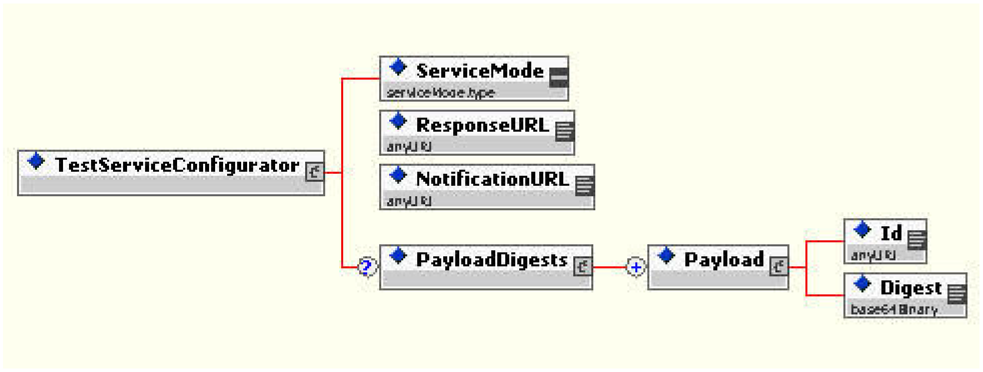


Figure 16 – Graphic representation of the TestServiceConfigurator content

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	
Id	Id 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 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.

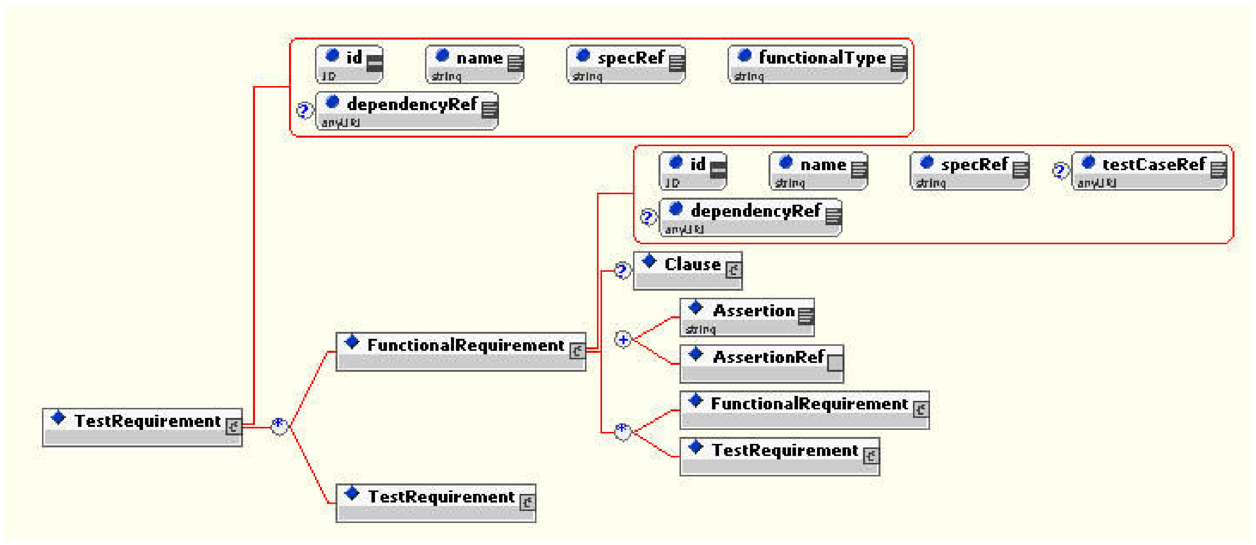


Figure 17 – Graphic representation of ebXMLTestRequirements.xsd schema

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 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. <u>When the tests requirements are about conformance</u>: 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.

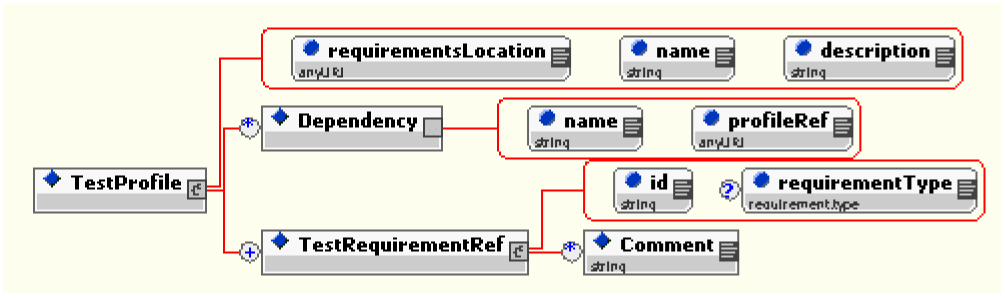


Figure 18 – Graphic representation of ebXMLTestProfile.xsd schema

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	

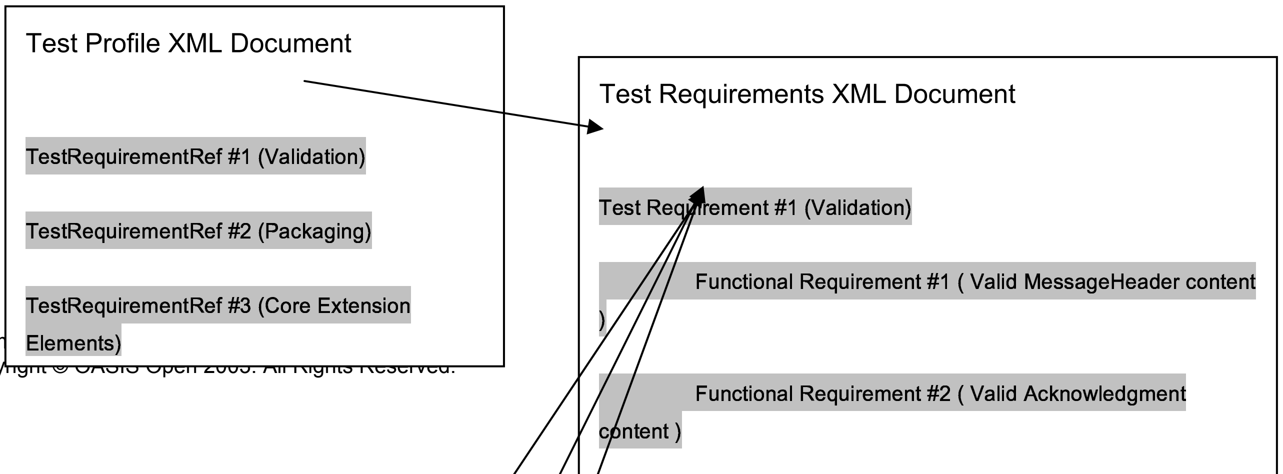
Table 9 provides a list of TestProfile element and attribute content

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)
Test Requirements XML Document
Test Requirement #1 (Validation)
Functional Requirement #1 (Valid MessageHeader content)
Functional Requirement #2 (Valid Acknowledgment content)

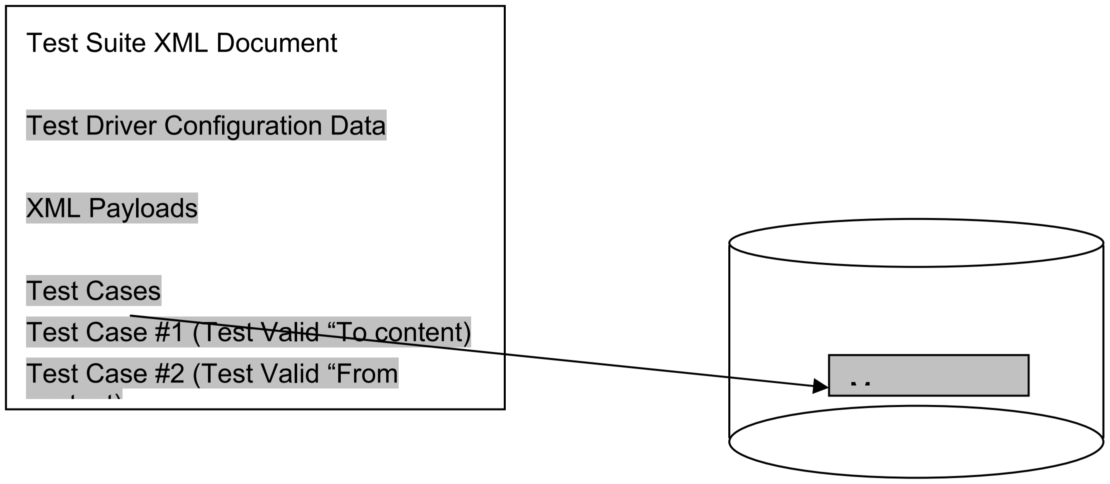

Test Suite XML Document
Test Driver Configuration Data
XML Payloads
Test Cases
Test Case #1 (Test Valid "To content)
Test Case #2 (Test Valid "From

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 Threads

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

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 for each Test Requirement.

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 and logical branches. Ultimately, a Test Case result is determined by the state set 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 sets the Test Case state to "pass", and the workflow executes to completion without any exception conditions.

A Test Case has a final state of "fail" if:

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

A Test Case has a final state of "undetermined" if:

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

OR

A system 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.

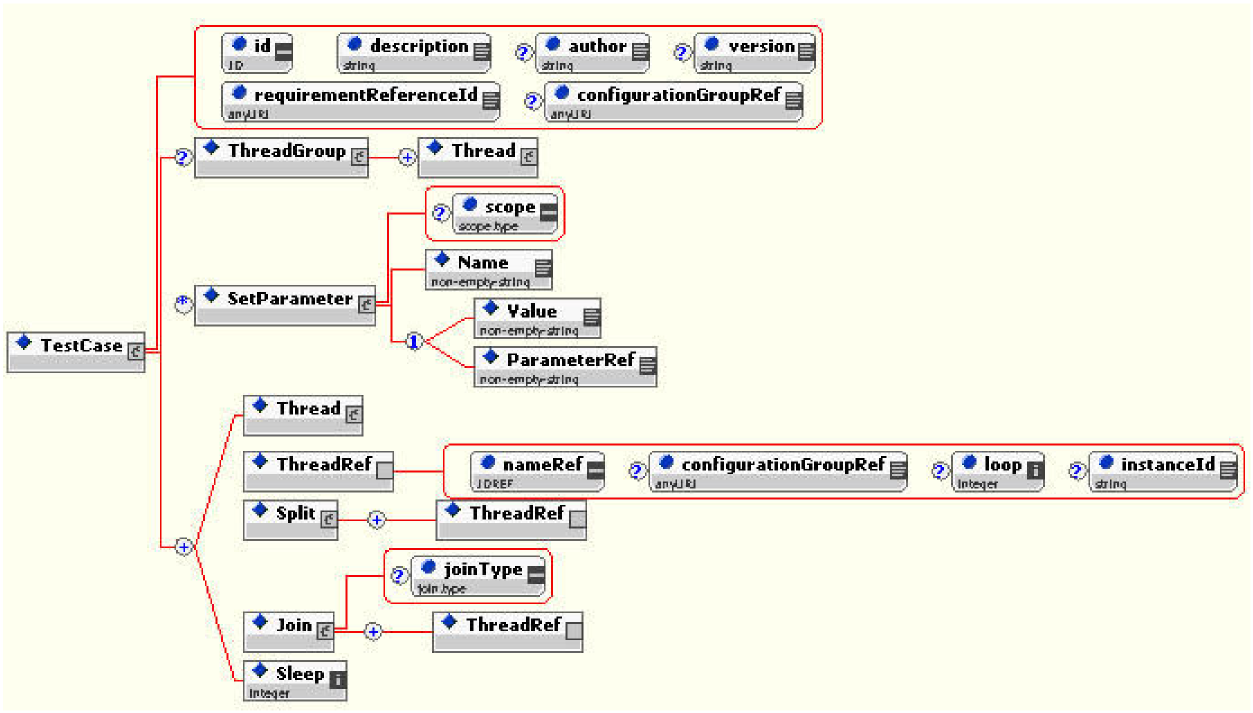


Figure 20 – Graphic representation of expanded view of the TestCase element

Definition of Content

Name	Description	Default Value From Test Driver	Required/Optional	Exception Condition
TestCase	Container element for all test case content		Optional	
id	Unique identifier for this Test Case		Required	
description	Short description of TestCase		Optional	
author	Name of person(s) creating the Test Case		Optional	
version	Version number of Test Case		Optional	
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 operations and/or Threads that may be forked synchronously or asynchronously		Required	
SetParameter	Contains name/value pair to be used by subsequent Threads in this Test Case		Optional	
TestServiceConfigurator	Container of configuration content for Test Service when Test Driver is in “service” mode		Optional	Unable to configure Test Service
ThreadRef	Name of the Thread to be executed in this TestCase		Optional	Thread not found
Split	Parallel execution of referenced sub-threads inside of the Split element		Optional	
Join	Evaluation of results of named threads (as “andjoin” or “orjoin”) permits execution of operations that follow the Join element		Optional	
Sleep	Instruction to the Test Driver		Optional	

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

Table 10 provides a list of TestCase element and attribute content

7.1.1 Test Threads

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

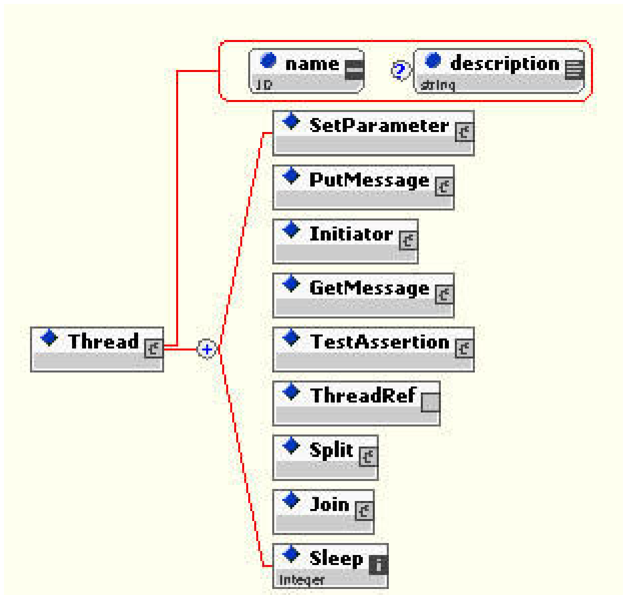
Sub-threads MAY be executed in parallel if they are the child of a Split element.

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.

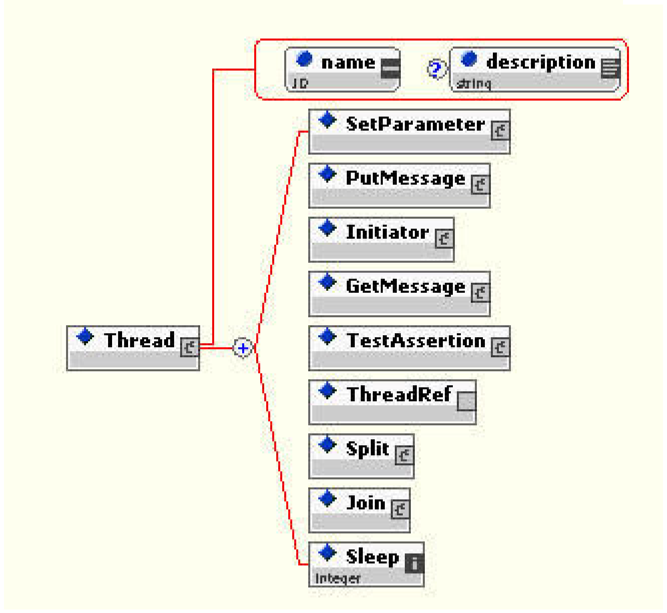
A Join operation “synchs” the execution of the Test Case, waiting until one (orJoin) or all (andJoin) Threads defined as children within the Join complete execution Concurrent Threads MUST be “joined” anywhere in the scripting AFTER the Split but within the same Thread in which they were invoked.

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

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



1621



1622

1623 Figure 21 – The Thread content diagram

1624

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

Initiator	Instruction to Test Driver to pass a message declaration to the Test Service for sending		Optional	Message could not be initiated by Test Service
GetMessage	Instruction to Test Driver to retrieve message(s) from the Message STore		Optional	Protocol error occured
TestAssertion	Instruction to the Test Driver to perform an evaluation		Optional	
ThreadRef	Reference via name to Thread to execute serially		Optional	Thread not found
Split	Directive to run the referenced Thread(s) enclosed in the Split element in parallel		Optional	Thread not found
Join	Directive to evaluate the boolean result of the enclosed referenced Thread(s) in a previous Split		Optional	Thread not found
Sleep	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 Thread Operations

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 behavior in various modes (if applicable) in more detail.

DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclaration

Thread

Table 12 – Expanded list of SetParameter element content

7.1.2.1 SetParameter: Setting Parameter values

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

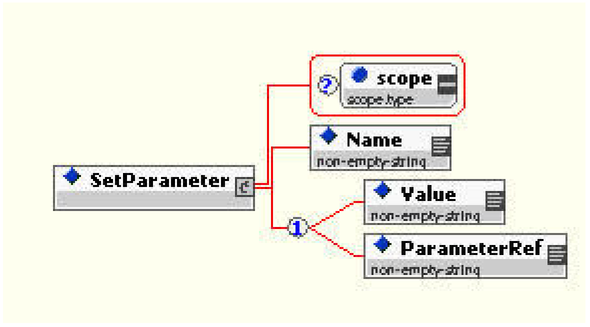


Figure 22 – Graphic representatin of expanded view of SetParameter element

Thread

Definition of Content

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

Semantics of the SetParameter operation:

Parameters that could be used to manipulate sent message content, or to evaluate received message content, can be assigned for use by Thread operations in three ways:

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

Using SetParameter at the beginning of a Thread

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

7.1.2.1.1 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, and Thread 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 exposed to descendent TestCaseor Thread. If a parameter is "redefined" 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 back to the current 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 or Thread) 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 Threads.. If it is defined within a Thread, then its definition exists for the lifecycle of that Thread. When the workflow execution moves to a "higher" level (i.e. to the parent Thread containing the Thread) then that parameter **a)** ceases to exist if it was not already defined at a higher level in the workflow hierarchy or **b)** if defined at a higher level, takes the previously value defined at the next highest level in the workflow hierarchy.

7.1.2.1.2 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 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.

7.1.2.1 **PutMessage:** Message Construction and Transmission

The "PutMessage" directive instructs the Test Driver to construct a message and transmit it to the designated party. The PutMessage element contains a 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 PutMessage operation instructs the Test Driver to build and send a messageDeclaration.A minimal Declaration (contained within its child Declaration 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).

DeclarationDeclarationDeclarationDeclaration

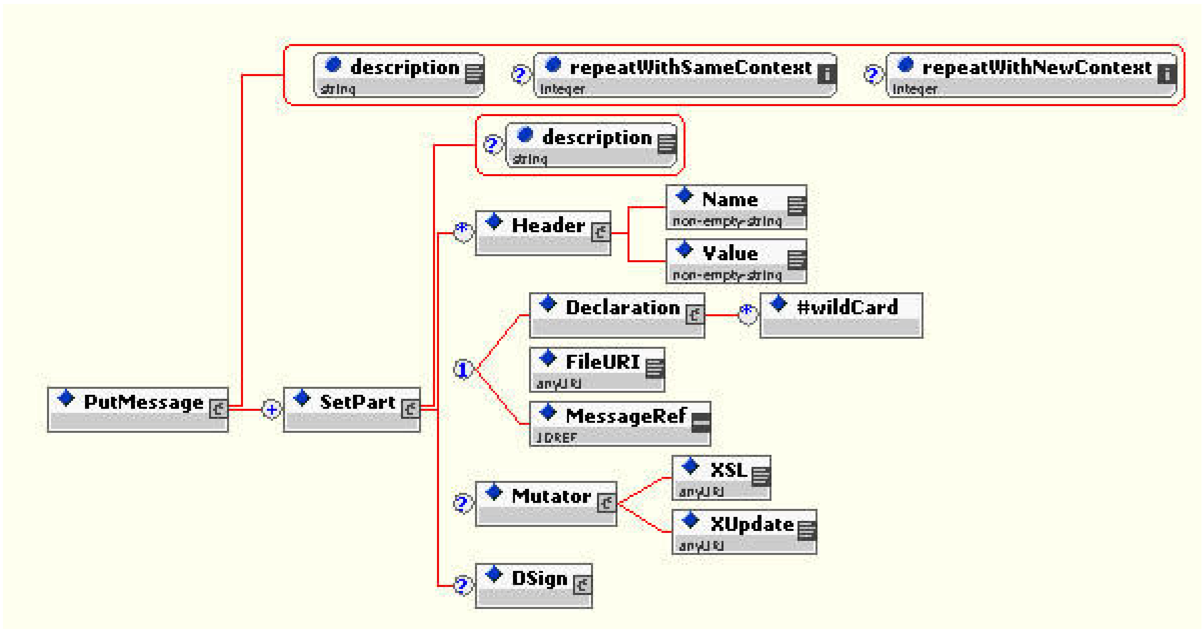


Figure 23 – Graphic representation of expanded view of the PutMessage element

7.1.2.2

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

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

Table 13 defines the content of the PutMessage element

Semantics of the PutMessage operation:

7.1.2.2.1 The Declaration

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

DeclarationAs a best practice, the XML content necessary to describe a basic message should be minimal, with default parameter values supplied by the Test Driver for most common and reuseable message content (such as Conversatoinld, CPAId, Sending Party Id..etc) . 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 message content are typically set using the Test Suite ConfigurationGroup parameters. Setting parameter values at the Test Suite level makes them "global" for use by any Test Case in the Test Suite. Parameters such as CPAId, ConversationId, Service, Action, ToPartyId and FromPartyId (or their equivalent) would typically be set globally for a messaging Test Suite. They could be optionally "overridden" locally within each Test Case by use of an individual "SetParameter" instruction in the Test Case scripting.

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

Two parameters (using the exact names specified below) are generated by the Test Driver, and CAN NOT be overridden using parameter definitions. They are MessageId and Timestamp. These two values can however, be explicitly defined in a Declaration if the test writer wishes to subsitute an explicit value for that supplied by the Test Driver in their mutator transformation.

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

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

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

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

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

7.1.2.2.3 Mutator: Turning a Declaration into an actual Message

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

7.1.2.2.4 DSign: Applying an XML Signature to the message

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

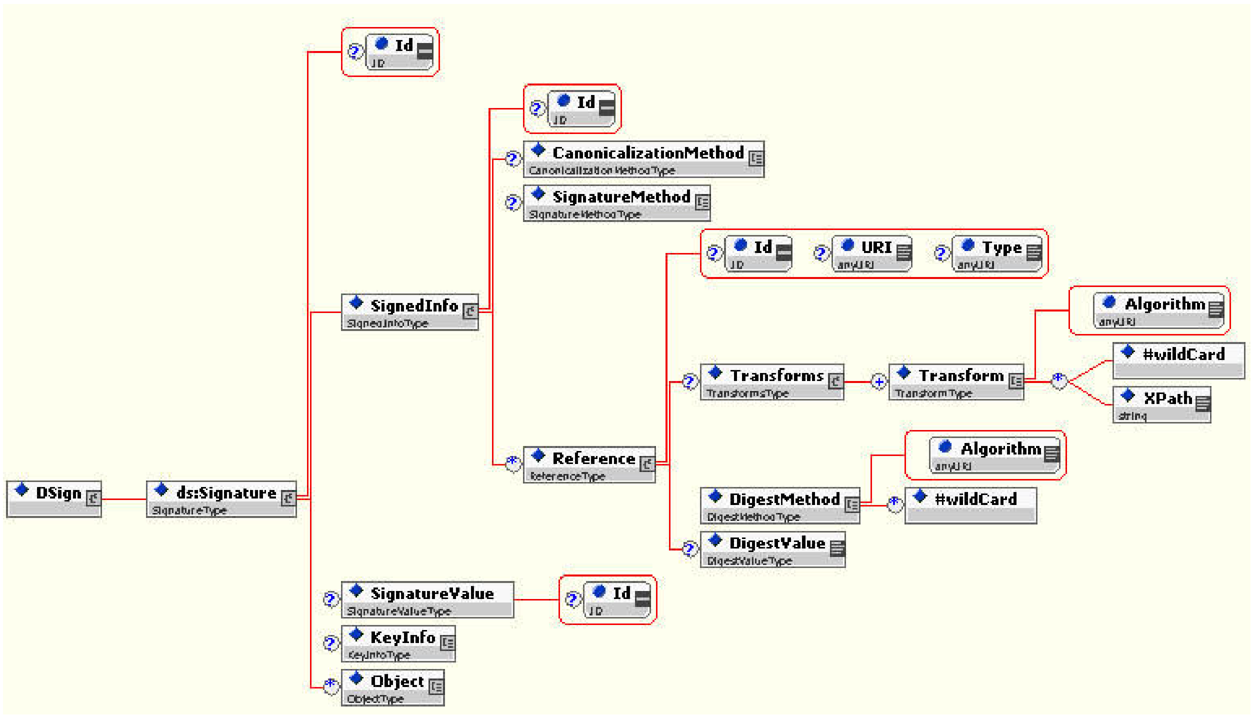


Figure 39 – Graphic representation of expanded view of the DSign element

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 Canonicalization and Signature algorithms and References		Required	
CanonicalizationMethod	Modify default container element	Container auto-generated by Test Driver	Optional	Method not supported

				by Test Driver
Algorithm	Modify default attribute and value	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 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	Algorithm not supported by Test Driver
#wildCard	Generate content "inline"		Optional	
XPath	Generate element with its value		Optional	Invalid XPath expression

DigestMethod	Generate element with its value		Required	Method not supported by Test Driver
Algorithm	Generate attribute and value		Required	Algorithm not supported by Test Driver
#wildCard	Generate content “inline”		Optional	
DigestValue	Generate element with its value	Set by Test Driver, based upon URI value	Optional	
#wildCard	Generate content “inline”		Optional	
SignatureValue	Generate element and its value	Set by Test Driver at run time	Optional	
Id	Generate attribute and its value		Optional	
KeyInfo	Generate container Element	All required and optional content, as described in [XMLDSIG] MUST be explicitly declared (no auto-generation by Test Driver)	Optional	Invalid Key data
Object	Generate container element		Optional	

Table 27 - Content of the Dsign element

Declaration

7.1.2.3 Initiator: Passing message construction directives to the Test Service

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

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

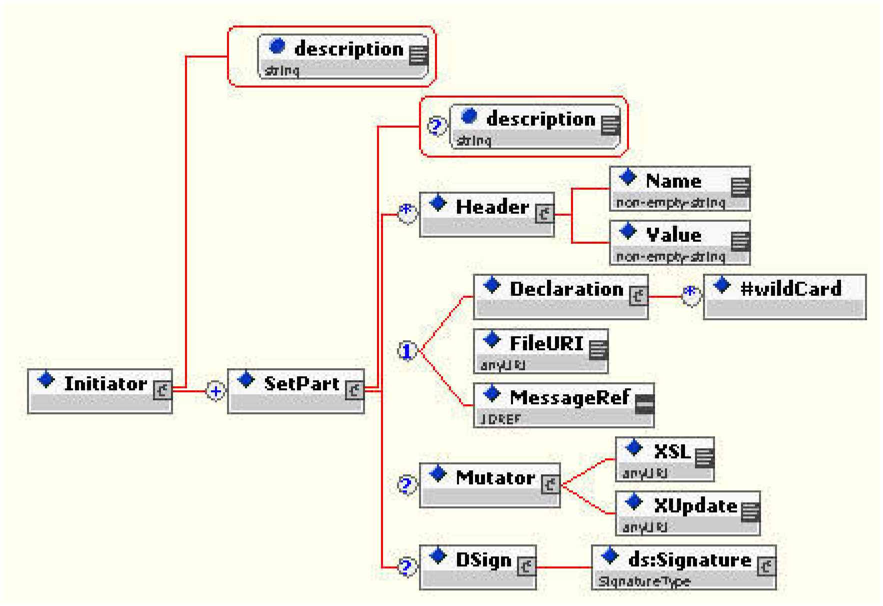


Figure 24 – Graphic representation of expanded view of the Initiator element

Definition of Content

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

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

Table 14 defines the content of the Initiator element

.

```
DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclaration
DeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDeclarationDec
larationDeclarationDeclaration
DeclarationDeclaration
Declaration
```

```
Declaration
```

7.1.2.4 GetMessage: Message Retrieval

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

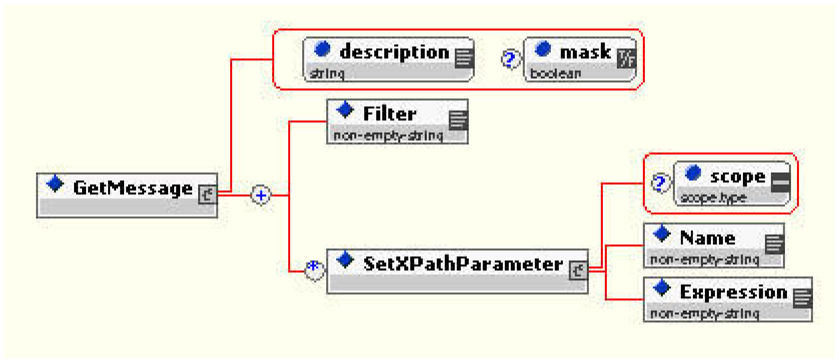


Figure 38 – Graphic representation of expanded view of the GetMessage element

Definition of Content

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

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

Table 26 defines the content of the GetMessage element

7.1.2.5 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 Declaration 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 Declaration content, or if present, MUST be ignored by the initiator method of the Send interface.

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

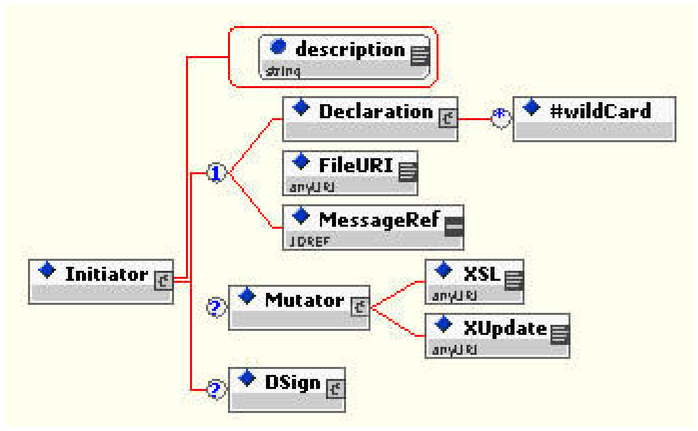


Figure 40 – Graphic representation of expanded view of the generic Initiator element

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	
Declaration	Message construction directives to be passed to MSH for interpretation and message generation		Optional	
FileURI	Reference to message declaration contained in a		Optional	File not found

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

Table 28 – Content of the Initiator operation

7.1.2.6 The GetMessage Operation

The GetMessage Operation, using its child XPath Filter instruction, 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 then be queried for adherence to a particular Test Assertion Additionally, parameter values that may be used later in the Test Case script can be assigned using the SetXPathParameter instruction.

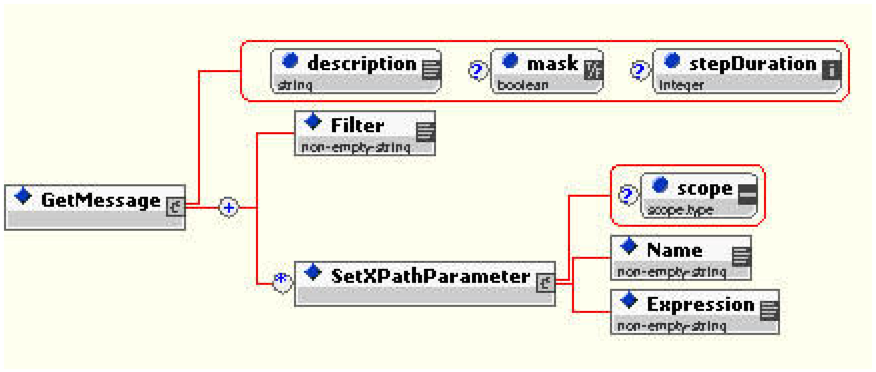


Figure 41 – Graphic representation of expanded view of the GetMessage element

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

Table 29 defines the content of the GetMessage element

7.1.2.6.1 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 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 XPath Filteris satisfied (evaluates to “true”), or (2) the timeout (stepDuration) expires.

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

Message Masking:

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

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

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

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 however.

As in the case of the SetParameter operation described in 4.2.2.1 , parameters may also 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. Each parameter value is a string representation of the nodelist content retrieved by the XPath query.

•

7.1.2.7 The TestAssertion Operation

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

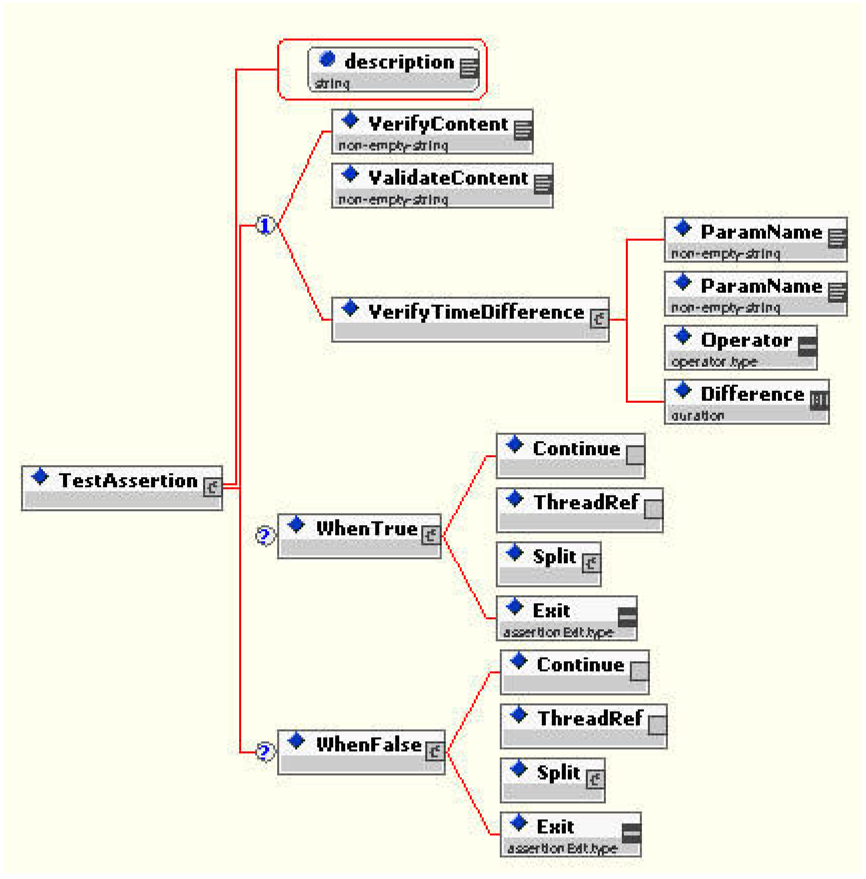


Figure 44 – Graphic representation of expanded view of the TestAssertion element

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
VerifyTimeDifference	Instruction to Test Driver to compute the time difference between two parameters and determine if the difference is less than equal or greater to an expected value		Optional	
ParamName	Parameter used in computation of time difference		Required	
Operator	(lessThen|lessThanOrEqual|equal|greaterThan|greaterThanOrEqual)		Required	
Difference	Expected value		Required	
WhenTrue	Branching instruction based upon boolean result of the TestAssertion operation		Optional	
WhenFalse	Branching instruction based upon boolean result of the TestAssertion operation		Optional	

Table 32 defines the content of the TestAssertion element

7.1.2.7.1 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

If the TestAssertion sub-operation is ValidateContent, then the content pointed to by the XPath expression contained in the text content MUST validate according to its contentType attribute . 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.

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

7.1.3 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.

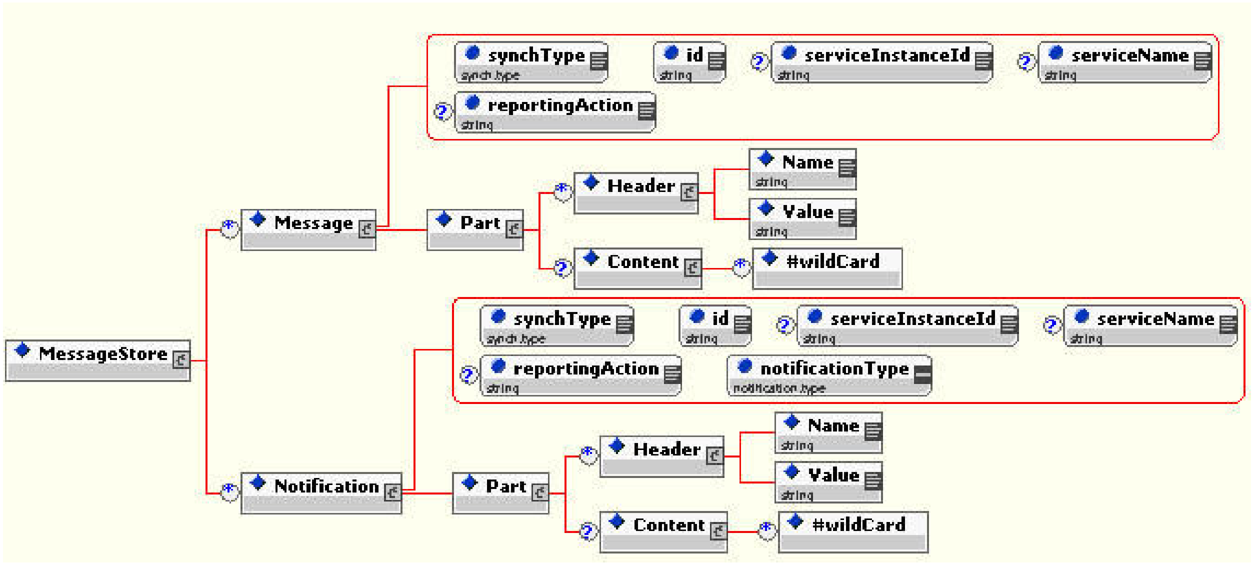


Figure 47 – Graphic representation of expanded view of the generic Test Driver MessageStore schema

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		Required	

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

7.1.3.1 Semantics of the Message Store

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.

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

Test Driver "Receive" interface. As a result, message content is restricted to what part of the message was exposed to the Test Service application layer. 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 content is identical to that of a message directly received by the Test Driver.

7.1.3.2 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.

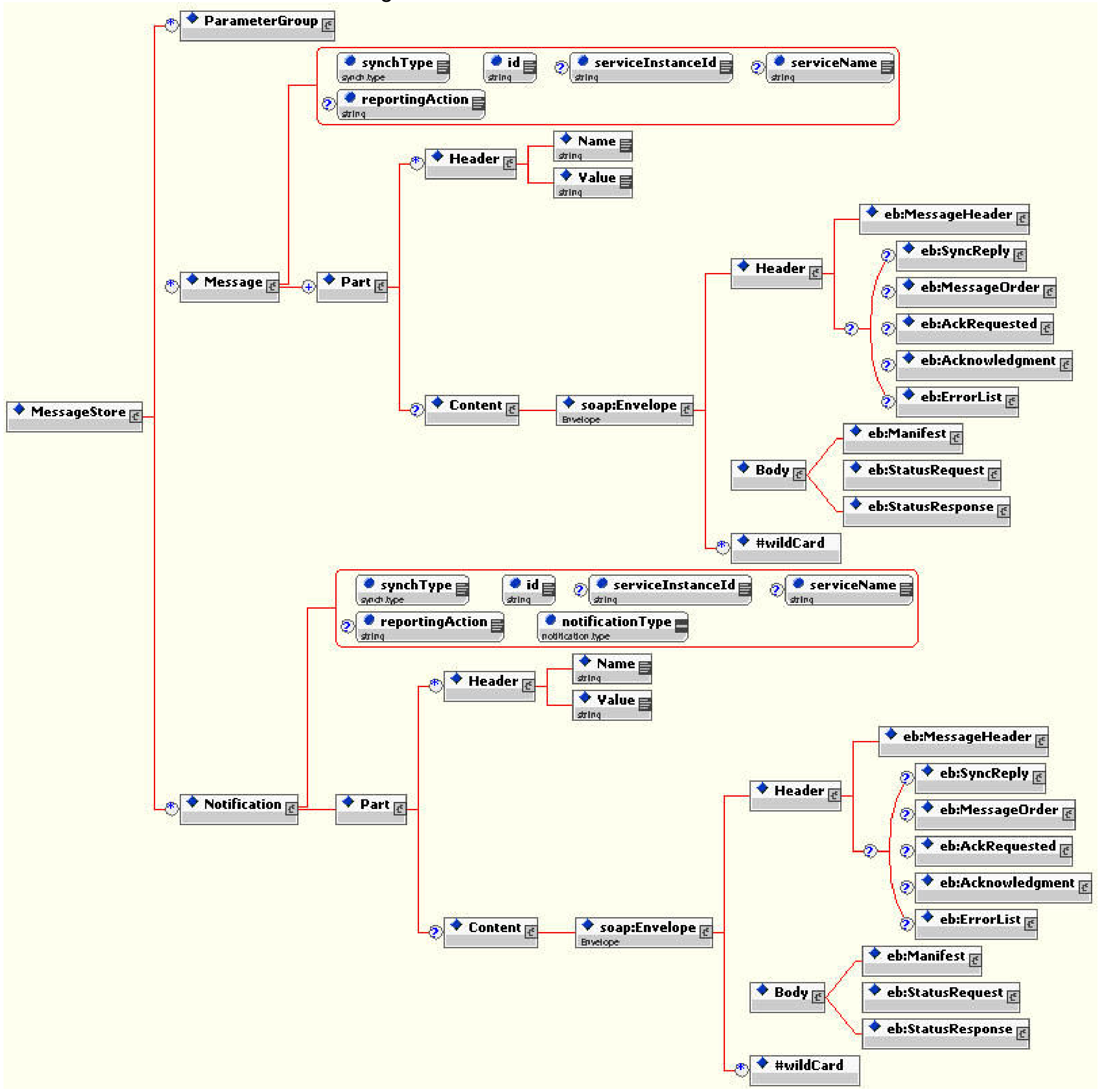


Figure 48 – Graphic representation of expanded view of Message Store content model, specifically for ebXML/SOAP messaging services

Definition of Content

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

Table 35 defines the content of the MessageStore element

7.1.3.3 Filter Result Schema

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 Thread. 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 MUST be a node list object whose node(s) are XML elements. This means that any Filter XPath expression MUST always query for elements within the Message Store. Doing so means that the Test Driver will be able to construct a document object from the Filter node list, and use it for subsequent VerifyContent and ValidateContent operations.

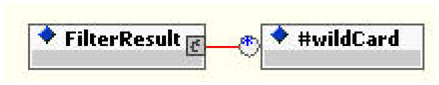


Figure 49 – Generic Filter Result schema, permitting any Message Store element content

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		Required

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

Table 36 defines the content of the FilterResult element

7.3 Test Service 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.

InitiatorRequest – 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.

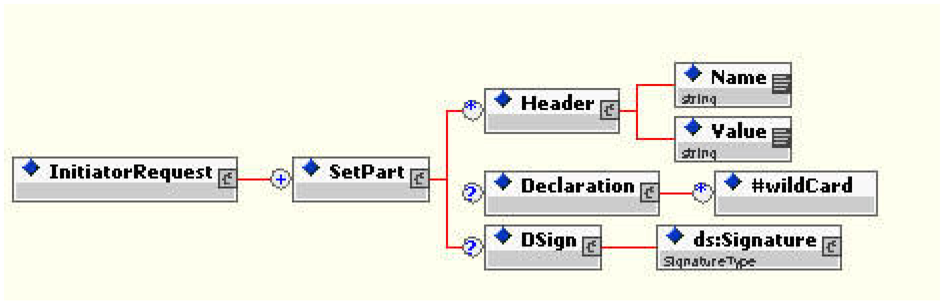


Figure 51 – Initiator request content

Definition of Content

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

Table 37 – Describes the content of the InitiatorRequest element

InitiatorResponse – 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

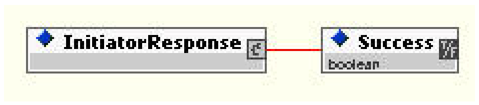


Figure 52 – Graphical representation of the InitiatorResponse schema

Definition of Content

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

Table 38 – Describes the content of the InitiatorResponse element

TesetServiceConfiguratorRequest – 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.

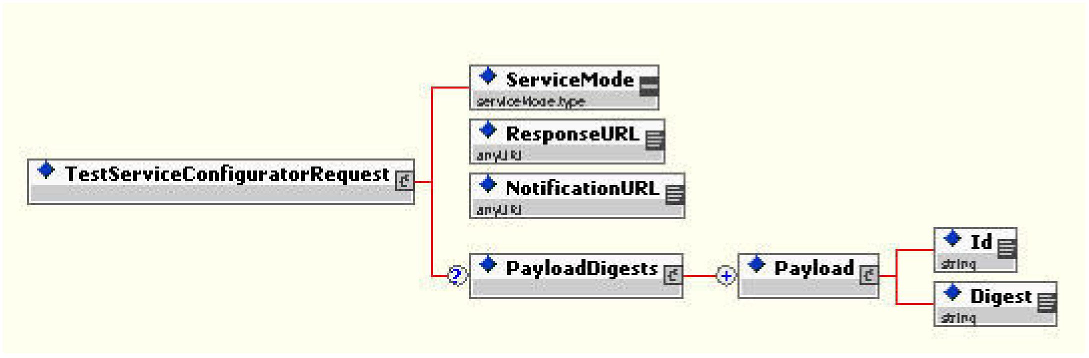


Figure 53 – A Graphical representation of the ConfiguratorRequest content schema

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

TestServiceConfiguratorResponse – 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.

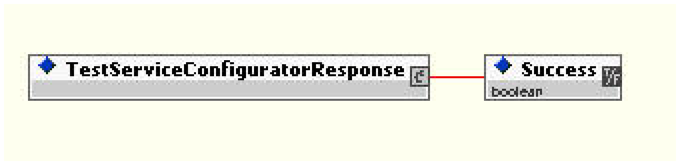


Figure 54 - A graphical representation of the ConfiguratorResponse content schema

Definition of Content

Name	Declaration Description	Default Value From Test Service	**Required/Optional**	**Exception Condition**
TestServiceConfiguratorResponse	Container for response from Test Service		Required	
Success	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 stored the same way in the MessageStore, there are important differences for each type of notification.

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

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

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

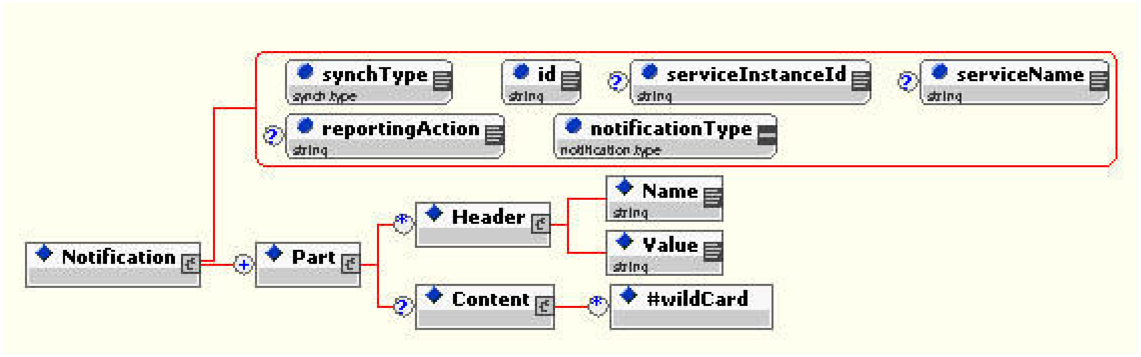


Figure 55 – Graphical representation of the Notification element content schema

Definition of Content

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

Table 41 – Description of MessageNotification element content

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

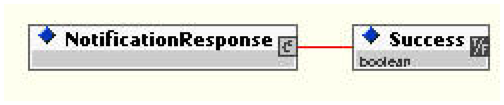


Figure 56 - A graphical representation of the NotificationResponse content schema

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.

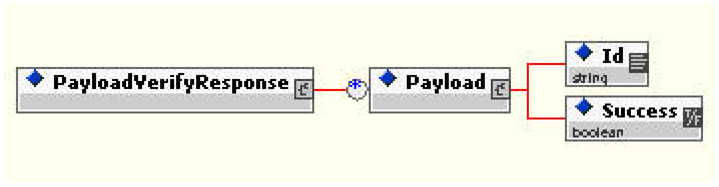


Figure 57 - A graphical representation of the PayloadVerifyResponse content schema

Definition of Content

Definition of Content

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

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 Threads.

The Test Report is 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 appendedto certain operation elements in the trace, to provide dignaostic information. The "result" attribute MUST have a value of "pass", "fail" or "undetermined". The Test Report schema is too large to graphically display on this page. Please consult Appendix G if you wish to examine the normative schema.

8 Test Material

Test material to support the ebXML Testing Framework includes:

A Testing Profile XML document
A Test Requirements XML document
A Test Suite XML document
Message Declaration Mutator document
Collaboration Agreement document (if needed to configure an MSH)

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 low-level message content (such as HTTP or MIME header 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:

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

8.1.4 Mutator documents

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

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

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

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

8.1.5 CPAs

For ebXML Messaging Services (MS) testing), both conformance and interoperability testing require the existence of a “base” CPA configuration that describes the “bootstrap” configuration of the candidate MSH for conformance and interoperability testing. Additional CPAs MAY be needed if testing requires different configurations of the candidate MSH. All CPA configurations MUST be uniquely defined (via a CPA ID) and documented in the Conformance or Interoperability Test Suite Specification document accompanying the Executable Test Suite. How the CPA configuration is presented to the candidate MSH imiplementation isnot defined in this specification.

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
A Mutator XSL Stylesheet

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 Functional Requirements “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/tc/ebxml-iic/conformance/reqs"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs/
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">
```

```
<!—Define first sub-requirement to fulfill packaging testing→
<FunctionalRequirement id="funreq_id_2"
name="GenerateConformantSOAPWithAttachMIMEHeaders" specRef="ebMS-2#2.1.2">
<Clause>
<!—Set first condition of the message is of type "multipart-mime" →
  <Condition id="condition_id_2" 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>
<!—Define the Assertion that the first part of message is a SOAP message →
  <Assertion id="assert_id_2" requirementType="required">The primary SOAP message is
carried in the root body part of the message.</Assertion>
  </FunctionalRequirement>
<!—Define a second sub-requirement to fulfill packaging testing→
<FunctionalRequirement id="funreq_id_4" name="GenerateCorrectMessagePackageContent-Type"
specRef="ebMS-2#2.1.2">
<Clause>
<!—Define condition that the candidate MSH generates a message →
  <Condition id="condition_id_4" requirementType="required">For each generated
message</Condition>
  </Clause>
<!—Define the Assertion that the Content-Type of MIME header of that message is
"text/xml" →
  <Assertion id="assert_id_4" requirementType="required">The Content-Type MIME header in
the Message Package contains a type attribute of "text/xml".</Assertion>
  </FunctionalRequirement>
</TestRequirement>
<!—Define a new Test Requirement, for  the Core Extension Elements of messaging→
<TestRequirement id="req_id_3" name="CoreExtensionElements" specRef="ebMS-2#3.1.1"
functionalType="packaging">
<!—Define a sub-requirement to test the CPAId extension element→
<FunctionalRequirement id="funreq_id_35" name="ReportFailedCPAIDResolution"
specRef="ebMS-2#3.1.2">
<Clause>
<!—First , set condition of a candidate MSH  receiving a message with an unresolvable
CPAId→
  <Condition id="condition_id_40" 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 requirementType is
"required")  respond with an Error→
  <Assertion id="assert_id_35" requirementType="required">The MSH responds with an error
(ValueNotRecognized/Error).</Assertion>
  </FunctionalRequirement>
<!—Define a sub-requirement to test continuity in message ConversationId→
<FunctionalRequirement id="funreq_id_36" name="ProvideConversationIdIntegrity"
specRef="ebMS-2#3.1.3">
<Clause>
<!—First , set condition of all messages generated by a Candidate Implementation
pertaining to a single CPAId→
  <Condition id="condition_id_41" 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_36" 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/reqs"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.oasis-open.org/tc/ebxml-iic/interop/reqs
ebXMLTestRequirements.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="Basic Interoperability Profile" specRef="MS 2.0 BIP
0.8" functionalType="basic interoperability">
<!—Define first sub-requirement to fulfill basic testing, sending a “no payload”
message→
<FunctionalRequirement id="funreq_id_1" name="BasicExchangeNoPayload" specRef="ebMS 2.0
BIP#3.2.1">
<Clause>
<!—First , set condition of a candidate MSH  receiving a message with no payload→
  <Condition id="condition_id_1" requirementType="required">For each received ebXML
message with no payload, received by the “Dummy” action</Condition>
  </Clause>
<!—Next , define the Assertion of expected behavior for the Dummy Action→
  <Assertion id="assert_id_1" requirementType="required">The message is received and
processed, and a simple response message is returned</Assertion>
  </FunctionalRequirement>
<!—Define second sub-requirement to fulfill basic testing, sending a “one payload”
message→

<FunctionalRequirement id="funreq_id_2" name="BasicExchangeOnePayload" specRef="ebMS 2.0
BIP#3.2.2">
<Clause>
<!—Set condition of a candidate MSH  receiving a message with one payload→
  <Condition id="condition_id_2" requirementType="required">For each received ebXML
message with one payload, received by the “Reflector” action </Condition>
  </Clause>
<!—Define the Assertion of expected behavior for the Reflector Action→
  <Assertion id="assert_id_2" requirementType="required">The message is received and
processed, and a simple response message with the identical  payload is
returned</Assertion>
  </FunctionalRequirement>
<!—Define third sub-requirement to fulfill basic testing, sending a “three payload”
message→
<FunctionalRequirement id="funreq_id_3" name="BasicExchangeThreePayloads" specRef="ebMS
2.0 BIP#3.2.3">
<Clause>
<!—Set condition of a candidate MSH  receiving a message with three payloads→
```

```
  <Condition id="condition_id_3" requirementType="required">For each received ebXML
message with three payloads, received by the "Reflector" action</Condition>
  </Clause>
```

```
<!—Define the Assertion of expected behavior for the Reflector Action→
  <Assertion id="assert_id_3" requirementType="required">The message is received and
processed, and a simple response message with the identical three payloads are
returned</Assertion>
  </FunctionalRequirement>
<!—Define third sub-requirement to fulfill basic testing, generating Error messages→
<FunctionalRequirement id="funreq_id_4" name="BasicExchangeGenerateError" specRef="ebMS
2.0 BIP#3.2.4">
<Clause>
<!—Set condition of a candidate MSH  receiving an erroneous message→
  <Condition id="condition_id_4" requirementType="required">For each received basic
ebXML message that should generate  an Error </Condition>
  </Clause>
<!—Define the Assertion of expected behavior for the candidate MSH  →
  <Assertion id="assert_id_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://www.oasis-open.org/tc/ebxml-iic/test-profile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-
profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml"
name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing
profile for ebXML MS v2.0 implementations">
```

```
  <TestRequirementRef id="funreq_id_2" />  <!—Execute all Test Casses that reference the 
Basic SOAP message structure Functional Requirement→
  <TestRequirementRef id="funreq_id_4" /> <!—Execute all Test Cases that reference Message 
Packaeg Content Type Functional Requirement→
  <TestRequirementRef id="req_id_2" />  <!—Execut all Test Cases that reference all 
Functional Requirements within the  Core Extension Elements Test Requirement→
  </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://www.oasis-open.org/tc/ebxml-iic/test-profile" 
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.oasis-
open.org/tc/ebxml-iic/test-profile http://www.oasis-open.org/tc/ebxml-iic/test-
profile/test-profile.xsd" requirementsLocation="ebxml-iic-msg-v20-conformance_reqs.xml" 
name="ebXML MS v2.0 Conformance Test Requirements" description="Core conformance testing 
profile for ebXML MS v2.0 implementations”>
  <TestRequirementRef id="funreq_id_1.1" /> <!—Execute all Test Casses that reference the 
“Basic Exchange, No Payload” Functional Requirement→
  <TestRequirementRef id="funreq_id_1.2" /> <!—Execute all Test Casses that reference the 
“Basic Exchange, One Payload” Functional Requirement→
  </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

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

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. The Test Case exercises a Functional Requirement listed in section 10.1   The Test Case below verifies that a ConversationId element is present in an ebXML response message

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

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

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

```
                                        <ebTest:SetPart>
                                                <ebTest:Declaration>
                                                                <soap:Envelope>
                                                                        <soap:Header>

    <eb:MessageHeader>

    <eb:Action>Dummy</eb:Action>

    </eb:MessageHeader>
                                                                        </soap:Header>
                                                                </soap:Envelope>
                                                </ebTest:Declaration>
                                            <ebTest:Mutator>ebXMLEnvelope.xsl</ebTest:Mutator>
                                        </ebTest:SetPart>
                                </ebTest:PutMessage>
                                <ebTest:GetMessage ebTest:description = "Retrieve response message ">

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

    <ebTest:VerifyContent>/FilterResult/Message/soap:Envelope/soap:Header/eb:MessgeHeader/eb:Co
nversationId</ebTest:VerifyContent>
                                </ebTest:TestAssertion>
                        </ebTest:Thread>
                </ebTest:ThreadGroup>
                <ebTest:Thread ebTest:name = "main">
                        <ebTest:ThreadRef ebTest:nameRef = "thread_01"/>
                </ebTest:Thread>
    </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. The Test Case exercises a Functional Interoperability Requirement. The Test Case below performs a basic message exchange with no message payload. The complete ebXML Basic Interoperability Profile Test Suite is available online at the OASIS IIC Technical Commite web site.

```

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

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

```

```
<ebTest:TestSuite xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
ebTest:configurationGroupRef = "base" xmlns:ds = "http://www.oasis-open.org/tc/ebxml-
iic/tests/xmldsig" xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime" xmlns:soap =
"http://www.oasis-open.org/tc/ebxml-iic/tests/soap" xmlns:eb = "http://www.oasis-open.org/tc/ebxml-
iic/tests/eb" xmlns:xlink = "http://www.w3.org/1999/xlink" xmlns:xsi =
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation = "http://www.oasis-
open.org/tc/ebxml-iic/tests schemas\ebTest.xsd">
    <ebTest:MetaData>
            <ebTest:Description>ebXML MS Interoperabilty  Test Suite </ebTest:Description>
            <ebTest:Version>0.1</ebTest:Version>
            <ebTest:Maintainer>Michael Kass</ebTest:Maintainer>
            <ebTest:Location>ScriptingTestSuite.xml</ebTest:Location>
            <ebTest:PublishDate>05/20/2004</ebTest:PublishDate>
            <ebTest:Status>DRAFT</ebTest:Status>
    </ebTest:MetaData>
    <ebTest:ConfigurationGroup ebTest:id = "mshc_basic">
            <ebTest:Mode>connection</ebTest:Mode>
            <ebTest:StepDuration>300</ebTest:StepDuration>
            <ebTest:Transport>HTTP</ebTest:Transport>
            <ebTest:Envelope>ebXML</ebTest:Envelope>
            <ebTest:StoreAttachments>false</ebTest:StoreAttachments>
            <ebTest:SetParameter>
                    <ebTest:Name>SenderParty</ebTest:Name>
                    <ebTest:Value>TestDriver</ebTest:Value>
            </ebTest:SetParameter>
            <ebTest:SetParameter>
                    <ebTest:Name>ReceiverParty</ebTest:Name>
                    <ebTest:Value>TestService</ebTest:Value>
            </ebTest:SetParameter>
            <ebTest:SetParameter>
                    <ebTest:Name>Service</ebTest:Name>
                    <ebTest:Value>urn:ebxml:iic:test</ebTest:Value>
            </ebTest:SetParameter>
            <ebTest:SetParameter>
                    <ebTest:Name>Action</ebTest:Name>
                    <ebTest:Value>Dummy</ebTest:Value>
            </ebTest:SetParameter>
    </ebTest:ConfigurationGroup>
    <ebTest:TestCase ebTest:id = "testcase_1" ebTest:description = "Basic request/response test"
ebTest:requirementReferenceId = " funreq_id_1.1">
            <ebTest:ThreadGroup>
                    <ebTest:Thread ebTest:name = "thread_01">
                            <ebTest:PutMessage ebTest:description = "Send a message to the
Dummy action">
                                    <ebTest:SetPart>
                                            <ebTest:Declaration>
                                                                    <soap:Envelope>
                                                                            <soap:Header>

    <eb:MessageHeader>

    <eb:Action>Dummy</eb:Action>

    </eb:MessageHeader>
                                                                            </soap:Header>
                                                                    </soap:Envelope>
                                            </ebTest:Declaration>
                                        <ebTest:Mutator>ebXMLEnvelope.xsl</ebTest:Mutator>
                                    </ebTest:SetPart>
                            </ebTest:PutMessage>
                            <ebTest:GetMessage ebTest:description = "Retrieve response message ">

    <ebTest:Filter>/TEST:MessageStore/mime:Message[mime:Container[1]/soap:Envelope/soap:Header
/eb:MessageHeader[eb:CPAId='mshc_Basic' and eb:MessageData/eb:RefToMessageId=$MessageId and
eb:Action='Mute']]</ebTest:Filter>
```

```
                                </ebTest:GetMessage>
                                <ebTest:TestAssertion ebTest:description = "Verify that an ebXML
Message was received">

    <ebTest:VerifyContent>/FilterResult/Message/soap:Envelope/soap:Header/eb:MessgeHeader
</ebTest:VerifyContent>
                                </ebTest:TestAssertion>
                        </ebTest:Thread>
                </ebTest:ThreadGroup>
                <ebTest:Thread ebTest:name = "main">
                        <ebTest:ThreadRef ebTest:nameRef = "thread_01"/>
                </ebTest:Thread>
    </ebTest:TestCase>
</ebTest:TestSuite>
```

9.3.3 A sample Mutator XSL Document

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

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

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

```
To Be added
```

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


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

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

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/tc/ebxml-iic/conformance/reqs"
    xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/conformance/reqs"
   >
   <group name = "FunctionalRequirementGroup">
          <sequence>
                 <element ref = "tns:FunctionalRequirement"/>
          </sequence>
   </group>

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


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


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

          Author: Michael Kass
         Organization: NIST

          Author: Matthew MacKenzie
          Organization: XML Global

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


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

   <element name = "TestRequirement">
```

```
		<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 = "dependencyRef" 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 = "dependencyRef" 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 = "Condition">
		<complexType>
			<simpleContent>
				<extension base = "string">
					<attribute name = "id" use = "required" type = "ID"/>
```

```
                                 <attribute name = "requirementType" use = "optional"
type = "tns:requirement.type"/>
                          </extension>
                   </simpleContent>
            </complexType>
     </element>
     <element name = "ConditionRef">
            <complexType>
                   <attribute name = "id" use = "required" type = "IDREF"/>
            </complexType>
     </element>
     <element name = "And" type = "string"/>
     <element name = "Or" type = "string"/>
     <element name = "Assertion">
            <complexType>
                   <simpleContent>
                          <extension base = "string">
                                 <attribute name = "requirementType" use = "required"
type = "tns:requirement.type"/>
                                 <attribute name = "id" use = "required" type = "ID"/>
                          </extension>
                   </simpleContent>
            </complexType>
     </element>
     <element name = "MetaData">
            <complexType>
                   <sequence>
                          <element ref = "tns:Description"/>
                          <element ref = "tns:Version"/>
                          <element ref = "tns:Maintainer"/>
                          <element ref = "tns:Location"/>
                          <element ref = "tns:PublishDate"/>
                          <element ref = "tns:Status"/>
                   </sequence>
            </complexType>
     </element>
     <element name = "Description" type = "string"/>
     <element name = "Version" type = "string"/>
     <element name = "SourceControlInfo" type = "string"/>
     <element name = "Maintainer" type = "string"/>
     <element name = "Location" type = "anyURI"/>
     <element name = "PublishDate" type = "string"/>
     <element name = "Status" type = "tns:pubStatus.type"/>
     <simpleType name = "pubStatus.type">
            <restriction base = "string">
                   <enumeration value = "DRAFT"/>
                   <enumeration value = "FINAL"/>
                   <enumeration value = "RETIRED"/>
            </restriction>
     </simpleType>
     <simpleType name = "requirement.type">
            <restriction base = "string">
                   <enumeration value = "required"/>
                   <enumeration value = "strongly recommended"/>
                   <enumeration value = "recommended"/>
                   <enumeration value = "optional"/>
            </restriction>
     </simpleType>
     <simpleType name = "testLevel.type">
            <restriction base = "string">
                   <enumeration value = "full"/>
                   <enumeration value = "most"/>
                   <enumeration value = "partial"/>
                   <enumeration value = "none"/>
            </restriction>
     </simpleType>
     <simpleType name = "functional.type">
            <restriction base = "string">
                   <enumeration value = "security"/>
                   <enumeration value = "reliable messaging"/>
                   <enumeration value = "packaging"/>
```

```
                  <enumeration value = "other"/>
          </restriction>
   </simpleType>
   <simpleType name = "layerList">
          <list itemType = "string"/>
   </simpleType>
   <element name = "Requirements">
          <complexType>
                 <sequence>
                        <element ref = "tns:MetaData"/>
                        <element ref = "tns:TestRequirement" maxOccurs = "unbounded"/>
                 </sequence>
          </complexType>
   </element>
   <element name = "AssertionRef">
          <complexType>
                 <attribute name = "id" use = "required" type = "IDREF"/>
          </complexType>
   </element>
</schema>
```

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

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

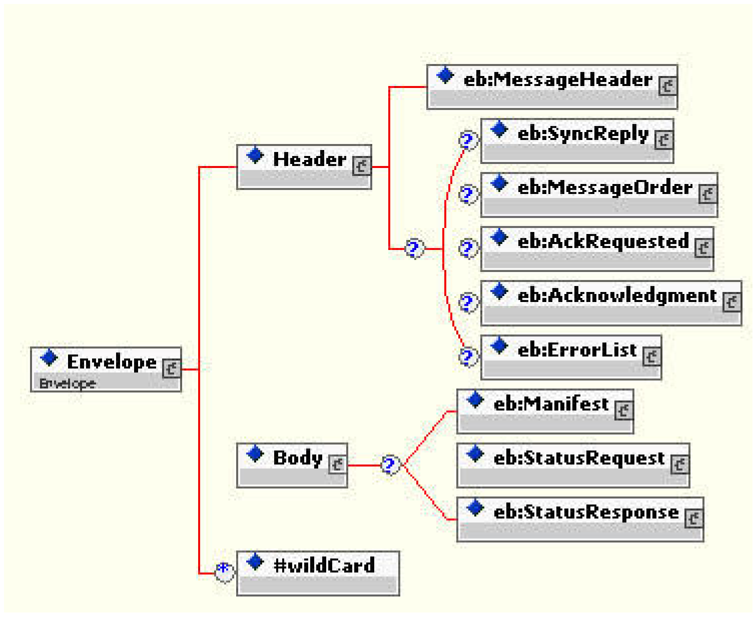


Figure 24 – Image of Message Envelope Declaration

9.3.3.1.1 Schema for ebXML Declaration using SOAP

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

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

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

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 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 Declaration. The Test Driver mutates the 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 Declaration.

```
<ebTest:Declaration>
   <soap:Envelope>
    <soap:Header>
     <eb:MessageHeader/>
    </soap:Header>
    <soap:Body />
   </soap:Envelope>
</ebTest:Declaration>
```

For illustrative purposes, the resulting message can be represented by the example message below. The Test Driver, after parsing the simple 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

Content-ID: <messagepackage@oasis.org>
Content-Type: text/xml;  charset="UTF-8"

<soap:Envelope    xmlns:xlink="http://www.w3.org/1999/xlink"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:eb="http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"          xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/

http://www.oasis-open.org/committees/ebxml-msg/schema/envelope.xsd
```

```
http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-2_0.xsd

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

Dynamic ebXML message content values (highlighted 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 Declaration content in order to be conformant to this specification for ebXML MS testing.

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.

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

9.3.3.1.2 Interpreting the SOAP portion of the ebXML 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.

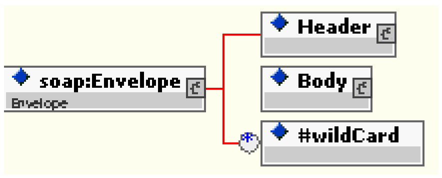


Figure 26 – Graphic representation of expanded view of the soap:Envelope element declaration

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

Table 15 defines the SOAP message content of the Declaration 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>
```

9.3.3.1.3 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.

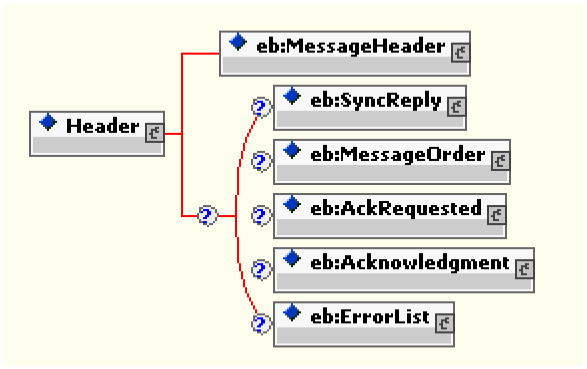


Figure 27 – Graphic representation of expanded view of the soap:Header element declaration

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

9.3.3.1.4 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 Declaration. The figure below illustrates the schema for an ebXML Message Header declaration to be interpreted by the Test Driver.

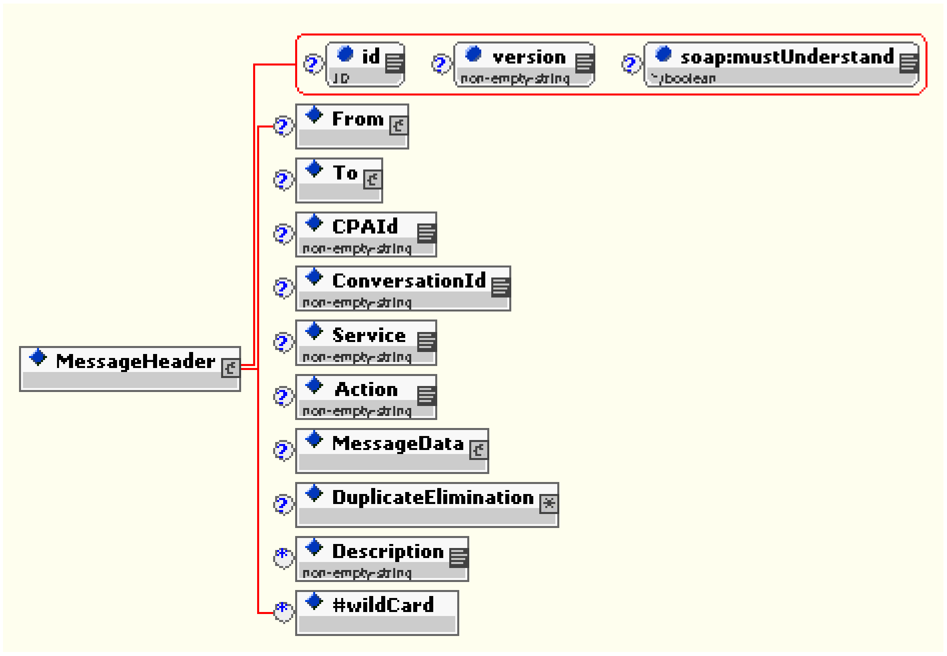


Figure 28 – Graphic representation of expanded view of the ebXML MessageHeader element declaration

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		Optional	

	declared value			
version	Modify default attribute value	2.0	Optional	
soap:mustUnderstand	Modify default attribute value	true	Optional	
From	Modify default From message element generated by Test Driver	Generated by Test Driver/MSH at run time	Optional	
PartyId	Replace default element value with new value	Generated by Test Driver/MSH at run time, using config value	Required	
type	Generate a type attribute with value		Optional	
Role	Generates a Role element with its value		Optional	
To	Modify default To message element generated by Test Driver	Generated by Test Driver at run time	Optional	
PartyId	Replace default element value with new value	Generated by Test Driver/MSH at run time, using config value	Required	
type	Generate type attribute with value		Optional	
Role	Generates a Role element with its value		Optional	
CPAId	Generate element with its value	Generated by Test Driver/MSH at run time, using config value	Optional	
ConversationId	Modify default value provided by Test Driver	Generated by Test Driver at run time	Optional	
Service	Modify default value generated by Test Driver	Generated by Test Driver/MSH at run time, using config value	Optional	
Action	Replace default value with specified Action name	Generated by Test Driver/MSH at run time, using config value	Optional	

MessageData	Modify default container generated by Test Driver	Generated by Test 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/>
```

9.3.3.1.5 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.

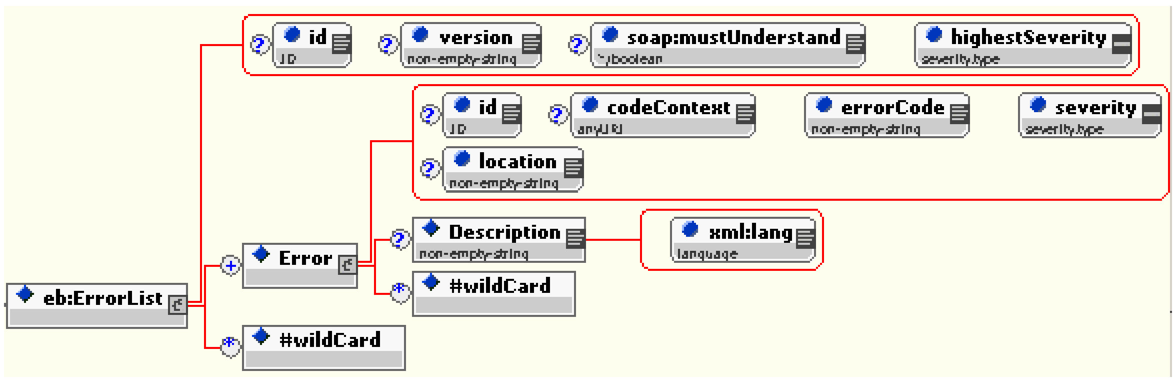


Figure 29 - Graphic representation of expanded view of the ebXML ErrorList element declaration

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	Generate element with declared value		Optional	
#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.

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

9.3.3.1.6 Interpreting the ebXML SyncReply Element Declaration

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

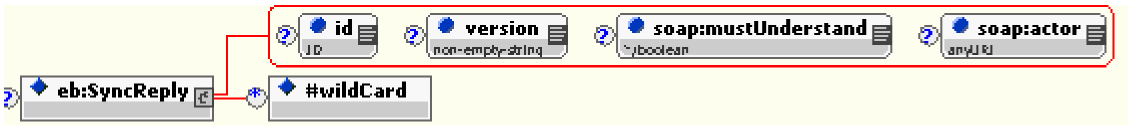


Figure 30 – Graphic representation of expanded view of the ebXML SyncReply element declaration

Definition of Content

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

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

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.

```
<eb:SyncReply/>
```

9.3.3.1.7 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.

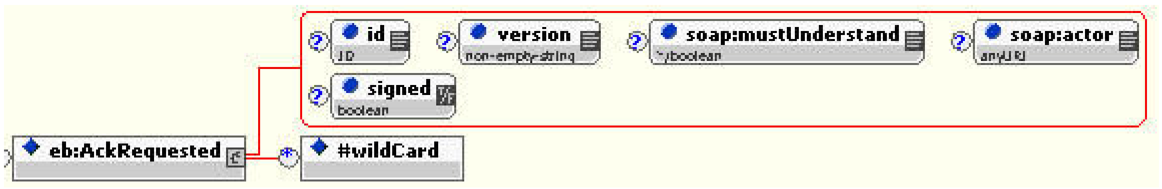


Figure 31 – Graphic representation of expanded view of the ebXML AckRequested element declaration

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.

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

9.3.3.1.8 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.

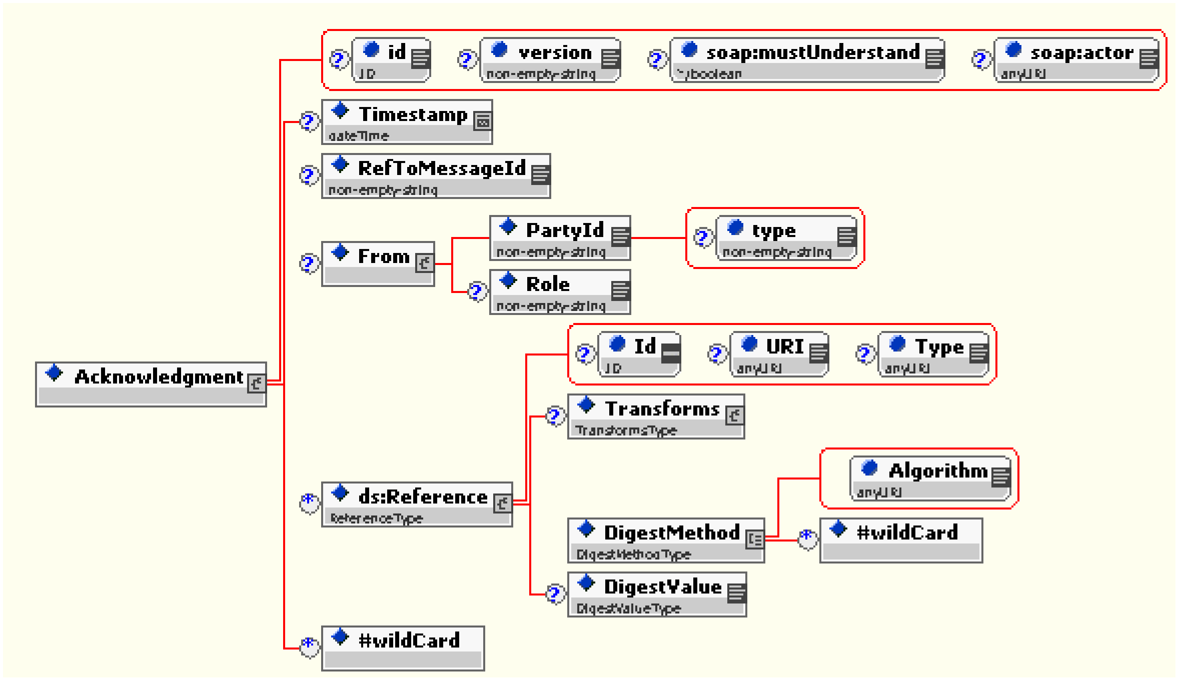


Figure 32 – Graphic representation of expanded view of the ebXML Acknowledgment element declaration

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	
Algorithm	Modify default attribute value	Generated by Test Driver at run time, based upon CPA	Required	
#wildCard	Generate content "inline"		Optional	
DigestValue	Generate element with its value	Computed by Test Driver at run time	Required	
#wildCard	Generate content "inline"		Optional	

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.

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

9.3.3.1.9 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.

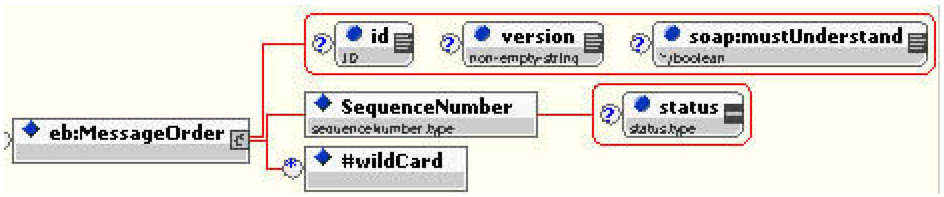


Figure 33 – Graphic representation of expanded view of the ebXML MessageOrder element declaration

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.

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

9.3.3.1.10 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.

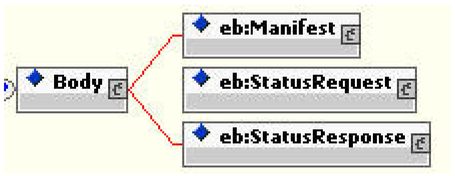


Figure 34 – Graphic representation of expanded view of the soap:Body element declaration

9.3.3.1.11 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

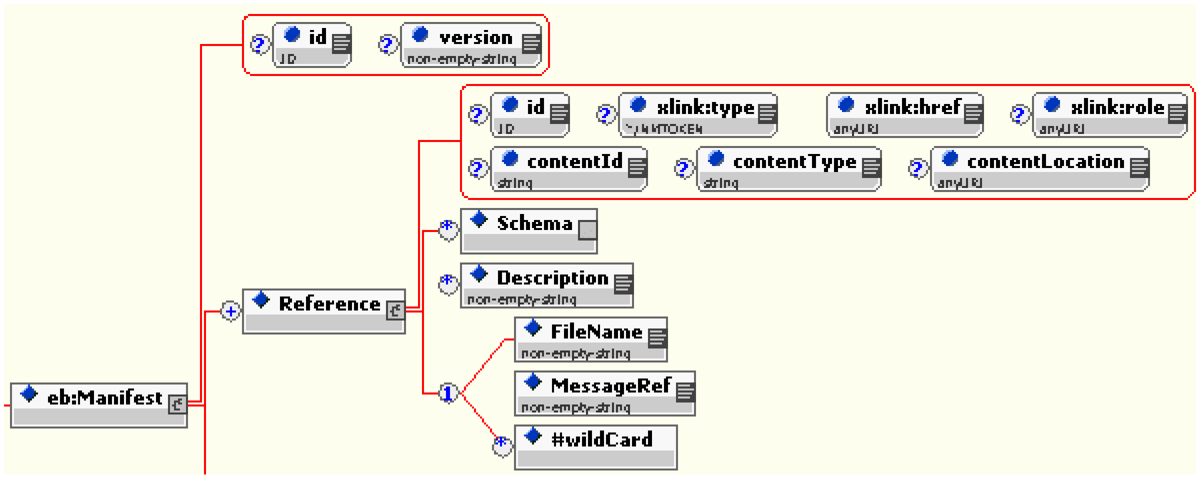


Figure 35 – Graphic representation of expanded view of the ebXML Manifest element declaration

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	2.0	Optional	

	value			
id	Modify default attribute value	true	Optional	
xlink:type	Generate element with declared value		Optional	
xlink:href	Generate attribute with declared value		Required	
xlink:role	Generate attribute with declared value		Optional	
contentId	Modify the Content-ID MIME header of the payload		Optional	
contentType	Set the the Content-Type MIME header of the payload		Optional	
contentLocation	Set the the Content-Location MIME header of the payload		Optional	
Schema	Generate schema container element		Optional	
location	Generate URI attribute and value of schema location		Required	
version	Generate schema version attribute and value		Optional	
Description	Generate description element and value		Optional	
xml:lang	Generate description language attribute and value		Required	
PayloadLocation	Load specified file as a MIME attachment to message		Required	File not found
MessageRef	Load designated XML document via IDREF as a MIME attachment to message		Required	
PayloadDeclaration	"Inline" the XML content of this element as a MIME message attachment		Required	

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.

```
<eb:Manifest>
<eb:Reference xlink:href=”cid:payload_1”/>
</eb:Manifest>
```

9.3.3.1.12 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

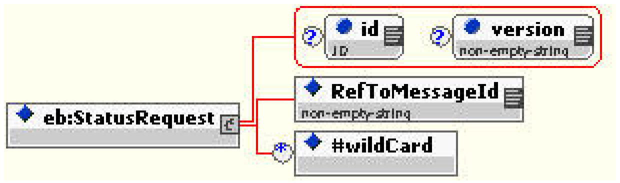


Figure 36 – Graphic representation of expanded view of the ebXML StatusRequest element declaration

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

9.3.3.1.13 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.

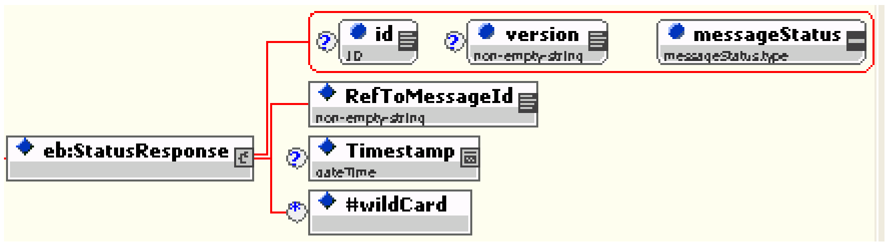


Figure 37 – Graphic representation of expanded view of the ebXML StatusResponse element declaration

Definition of Content

Name	Declaration Description	Default Value From Test Driver	Required/Optional	**Exception Condition**
eb:StatusResponse	Generate container element and all default content		Optional	
id	Generate attribute and its value		Optional	
version	Modify default attribute value	2.0	Optional	
messageStatus	Generate attribute and its value		Optional	
RefToMessageId	Generate element and its value		Required	
Timestamp	Modify default value	Generated by Test Driver at run time	Optional	
#wildCard	Generate content “inline”		Optional	

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.

```
<eb:StatusResponse messageStatus=”Processed”/>
```

SOAP Portion of the ebXML 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://www.oasis-open.org/tc/ebxml-iic/tests/soap"
    xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:eb = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
    xmlns:ds = "http://www.w3.org/2000/09/xmldsig#">
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb" schemaLocation
= "eb.xsd"/>
   <import namespace = "http://www.w3.org/2000/09/xmldsig#" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xmldsig-core-schema.xsd"/>
   <group name = "optionElements">
          <all minOccurs = "0">
                 <element ref = "eb:SyncReply" minOccurs = "0"/>
                 <element ref = "eb:MessageOrder" minOccurs = "0"/>
                 <element ref = "eb:AckRequested" minOccurs = "0"/>
                 <element ref = "eb:Acknowledgment" minOccurs = "0"/>
                 <element ref = "eb:ErrorList" minOccurs = "0"/>
                 <element ref = "ds:Signature" minOccurs = "0"/>
          </all>
   </group>
   <attributeGroup name = "encodingStyle">
          <attribute name = "encodingStyle" type = "tns:encodingStyle"/>
   </attributeGroup>

   <!-- Schema for the SOAP/1.1 envelope

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

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

     Copyright 2001 Martin Gudgin, Developmentor.

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


     Copyright 2003 OASIS

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


     Original copyright:

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

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

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

```
-->


   <!-- 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:optionElements"/>
                   </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 = "tns:detail" minOccurs = "0"/>
           </sequence>
   </complexType>
   <complexType name = "detail">
           <sequence>
                  <any namespace = "##any" processContents = "lax" minOccurs = "0"
maxOccurs = "unbounded"/>
           </sequence>
           <anyAttribute namespace = "##any" processContents = "lax"/>
   </complexType>
</schema>
```

ebMS portion of the ebXML 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://www.oasis-open.org/tc/ebxml-iic/tests/eb"
    xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/eb"
    xmlns:xlink = "http://www.w3.org/1999/xlink"
    xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
    xmlns:soap = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"

    version = "1.0"
    elementFormDefault = "qualified"
    attributeFormDefault = "qualified">
   <import namespace = "http://www.w3.org/1999/xlink" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xlink.xsd"/>
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/soap"
schemaLocation = "soap.xsd"/>
   <import namespace = "http://www.w3.org/XML/1998/namespace" schemaLocation =
"http://www.oasis-open.org/committees/ebxml-msg/schema/xml_lang.xsd"/>
   <attributeGroup name = "headerExtension.grp">
           <attribute ref = "tns:id"/>
           <attribute ref = "tns:version" use = "optional"/>
           <attribute ref = "soap:mustUnderstand" use = "optional"/>
   </attributeGroup>
   <attributeGroup name = "bodyExtension.grp">
           <attribute ref = "tns:id"/>
           <attribute ref = "tns:version" use = "optional"/>
   </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 = "0"/>
                        <element ref = "tns:To" minOccurs = "0"/>
                        <element ref = "tns:CPAId" minOccurs = "0"/>
                        <element ref = "tns:ConversationId" minOccurs = "0"/>
                        <element ref = "tns:Service" minOccurs = "0"/>
                        <element ref = "tns:Action" minOccurs = "0"/>
                        <element ref = "tns:MessageData" minOccurs = "0"/>
                        <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 = "0"/>
                        <element ref = "tns:Timestamp" minOccurs = "0"/>
                        <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 = "optional" 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:bodyExtension.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:bodyExtension.grp"/>
          </complexType>
   </element>

   <!-- COMMON TYPES -->

   <complexType name = "sequenceNumber.type">
          <simpleContent>
                 <extension base = "positiveInteger">
                        <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 = "PartyId">
          <complexType>
                 <simpleContent>
                        <extension base = "tns:non-empty-string">
                               <attribute name = "type" type = "tns:non-empty-
string"/>
                        </extension>
                 </simpleContent>
          </complexType>
   </element>
   <element name = "To">
          <complexType>
```

```
                  <sequence>
                         <element ref = "tns:PartyId"/>
                         <element name = "Role" type = "tns:non-empty-string" minOccurs
= "0"/>
                  </sequence>
           </complexType>
   </element>
   <element name = "From">
           <complexType>
                  <sequence>
                         <element ref = "tns:PartyId"/>
                         <element name = "Role" 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 = "RefToMessageId" type = "tns:non-empty-string"/>
   <element name = "Timestamp" type = "dateTime"/>
   <element name = "FileName" type = "tns:non-empty-string"/>
   <element name = "MessageRef" type = "tns:non-empty-string"/>
</schema>
```

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

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

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

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
    targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/tests"
    xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
    xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
    xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
    version = "1.0"
    elementFormDefault = "unqualified"
    attributeFormDefault = "unqualified">
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting_poc_07_13_04/schemas/mime.xsd"/>
   <!-- <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/> -->


   <!-- <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting_poc_06_27_04/schemas/mime.xsd"/> -->


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


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


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


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

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

   <element name = "TestSuite">
          <complexType>
                  <sequence>
                         <element ref = "ebTest:MetaData"/>
                         <element ref = "ebTest:ConfigurationGroup" maxOccurs =
"unbounded"/>
                         <element ref = "ebTest:TestServiceConfigurator" minOccurs =
"0"/>
                         <element ref = "ebTest:Message" minOccurs = "0" maxOccurs =
"unbounded"/>
                         <element ref = "ebTest:TestCase" maxOccurs = "unbounded"/>
                  </sequence>
                  <attribute name = "configurationGroupRef" use = "required" type =
"anyURI"/>
          </complexType>
   </element>
   <element name = "MetaData">
          <complexType>
                  <sequence>
                         <element ref = "ebTest:Description"/>
                         <element ref = "ebTest:Version"/>
                         <element ref = "ebTest:Maintainer"/>
                         <element ref = "ebTest:Location"/>
                         <element ref = "ebTest:PublishDate"/>
                         <element ref = "ebTest:Status"/>
                  </sequence>
          </complexType>
   </element>
   <element name = "Description" type = "ebTest:non-empty-string"/>
   <element name = "Version" type = "ebTest:non-empty-string"/>
   <element name = "Maintainer" type = "ebTest:non-empty-string"/>
   <element name = "Location" type = "anyURI"/>
   <element name = "PublishDate" type = "ebTest:non-empty-string"/>
   <element name = "Status" type = "ebTest:non-empty-string"/>
   <element name = "TestCase">
          <complexType>
                  <sequence>
                         <element ref = "ebTest:ThreadGroup" minOccurs = "0"/>
                         <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
= "unbounded"/>
                         <choice maxOccurs = "unbounded">
                                <element ref = "ebTest:Thread"/>
                                <element ref = "ebTest:ThreadRef"/>
                                <element ref = "ebTest:Split"/>
                                <element ref = "ebTest:Join"/>
                                <element ref = "ebTest:Sleep"/>
                         </choice>
                  </sequence>
                  <attribute name = "id" use = "required" type = "ID"/>
                  <attribute name = "description" use = "required" type = "string"/>
                  <attribute name = "author" use = "optional" type = "string"/>
                  <attribute name = "version" use = "optional" type = "string"/>
                  <attribute name = "requirementReferenceId" use = "required" type =
"anyURI"/>
                  <attribute name = "configurationGroupRef" use = "optional" type =
"anyURI"/>
          </complexType>
   </element>
   <element name = "ConfigurationGroup">
          <complexType>
                  <sequence>
                         <element ref = "ebTest:Mode"/>
                         <element ref = "ebTest:StepDuration"/>
                         <element ref = "ebTest:Transport"/>
                         <element ref = "ebTest:Envelope"/>
                         <element ref = "ebTest:StoreAttachments"/>
```

```
                        <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
= "unbounded"/>
                </sequence>
                <attribute name = "id" use = "required" type = "ID"/>
        </complexType>
   </element>
   <element name = "CPAId" type = "ebTest:non-empty-string"/>
   <element name = "Mode" type = "ebTest:mode.type"/>
   <element name = "SenderParty" type = "anyURI"/>
   <element name = "ReceiverParty" type = "anyURI"/>
   <element name = "Service" type = "anyURI"/>
   <element name = "Action" type = "ebTest:non-empty-string"/>
   <element name = "StepDuration" type = "integer"/>
   <element name = "Transport" type = "ebTest:transport.type"/>
   <element name = "Envelope" type = "ebTest:non-empty-string"/>
   <simpleType name = "mode.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "local-service"/>
                <enumeration value = "remote-service"/>
                <enumeration value = "connection"/>
        </restriction>
   </simpleType>
   <simpleType name = "mimeHeader.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "MIMEMessageContent-Type"/>
                <enumeration value = "MIMEMessageStart"/>
                <enumeration value = "Content-Type"/>
                <enumeration value = "start"/>
                <enumeration value = "charset"/>
                <enumeration value = "type"/>
                <enumeration value = "wildcard"/>
        </restriction>
   </simpleType>
   <simpleType name = "content.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "XML"/>
                <enumeration value = "date"/>
                <enumeration value = "URI"/>
                <enumeration value = "signature"/>
                <enumeration value = "XPointer"/>
        </restriction>
   </simpleType>
   <simpleType name = "method.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "xpath"/>
                <enumeration value = "md5"/>
        </restriction>
   </simpleType>
   <simpleType name = "messageContext.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "true"/>
                <enumeration value = "false"/>
        </restriction>
   </simpleType>
   <simpleType name = "requirement.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "required"/>
                <enumeration value = "stronglyrecommended"/>
                <enumeration value = "recommended"/>
                <enumeration value = "optional"/>
        </restriction>
   </simpleType>
   <simpleType name = "non-empty-string">
        <restriction base = "string">
                <minLength value = "1"/>
        </restriction>
   </simpleType>
   <simpleType name = "configAction.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "query"/>
                <enumeration value = "replace"/>
        </restriction>
    </simpleType>
    <simpleType name = "action.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "reset"/>
            <enumeration value = "modify"/>
        </restriction>
    </simpleType>
    <simpleType name = "configItem.type">
        <restriction base = "NMTOKEN"/>
    </simpleType>
    <simpleType name = "parameter.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "string"/>
            <enumeration value = "parameter"/>
        </restriction>
    </simpleType>
    <simpleType name = "connectivePredicate.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "and"/>
            <enumeration value = "or"/>
        </restriction>
    </simpleType>
    <simpleType name = "thread.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "synchronous"/>
            <enumeration value = "asynchronous"/>
        </restriction>
    </simpleType>
    <simpleType name = "matchResult.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "pass"/>
            <enumeration value = "fail"/>
        </restriction>
    </simpleType>
    <simpleType name = "if.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "andif"/>
            <enumeration value = "orif"/>
        </restriction>
    </simpleType>
    <simpleType name = "split.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "andsplit"/>
            <enumeration value = "orsplit"/>
        </restriction>
    </simpleType>
    <simpleType name = "join.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "andjoin"/>
            <enumeration value = "orjoin"/>
        </restriction>
    </simpleType>
    <simpleType name = "serviceMode.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "loop"/>
            <enumeration value = "local-reporting"/>
            <enumeration value = "remote-reporting"/>
        </restriction>
    </simpleType>
    <simpleType name = "time.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "timeToAcknowlegeReceipt"/>
            <enumeration value = "timeToAcknowledgeAcceptance"/>
            <enumeration value = "timeToPerform"/>
            <enumeration value = "other"/>
        </restriction>
    </simpleType>
    <simpleType name = "operator.type">
        <restriction base = "NMTOKEN">
            <enumeration value = "equal"/>
```

```
                        <enumeration value = "lessThanl"/>
                        <enumeration value = "lessThanOrEqual"/>
                        <enumeration value = "greaterThan"/>
                        <enumeration value = "greaterThanOrEqual"/>
                </restriction>
        </simpleType>
        <simpleType name = "assertionExit.type">
                <restriction base = "NMTOKEN">
                        <enumeration value = "pass"/>
                        <enumeration value = "fail"/>
                        <enumeration value = "undetermined"/>
                </restriction>
        </simpleType>
        <simpleType name = "preconditionExit.type">
                <restriction base = "NMTOKEN">
                        <enumeration value = "undetermined"/>
                </restriction>
        </simpleType>
        <simpleType name = "scope.type">
                <restriction base = "NMTOKEN">
                        <enumeration value = "self"/>
                        <enumeration value = "selfAndDescendents"/>
                </restriction>
        </simpleType>
        <simpleType name = "transport.type">
                <restriction base = "NMTOKEN">
                        <enumeration value = "FTP"/>
                        <enumeration value = "SMTP"/>
                        <enumeration value = "HTTP"/>
                </restriction>
        </simpleType>
        <element name = "MessageExpression">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:ErrorMessage"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "ErrorMessage" type = "ebTest:non-empty-string"/>
        <element name = "PutMessage">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:SetPart" maxOccurs = "unbounded"/>
                        </sequence>
                        <attribute name = "description" use = "required" type = "string"/>
                        <attribute name = "repeatWithSameContext" use = "optional" type =
"integer"/>
                        <attribute name = "repeatWithNewContext" use = "optional" type =
"integer"/>
                </complexType>
        </element>
        <element name = "GetPayload">
                <complexType>
                        <sequence>
                                <choice>
                                        <element ref = "ebTest:Content-ID"/>
                                        <element ref = "ebTest:Content-Location"/>
                                        <element ref = "ebTest:Index"/>
                                </choice>
                                <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
maxOccurs = "unbounded"/>
                        </sequence>
                        <attribute name = "description" use = "required" type = "string"/>
                </complexType>
        </element>
        <element name = "GetMessage">
                <complexType>
                        <sequence maxOccurs = "unbounded">
                                <element ref = "ebTest:Filter"/>
                                <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
maxOccurs = "unbounded"/>
```

```
                        </sequence>
                        <attribute name = "description" use = "required" type = "string"/>
                        <attribute name = "mask" use = "optional" type = "boolean"/>
                </complexType>
        </element>
        <element name = "Filter">
                <complexType>
                        <simpleContent>
                                <extension base = "ebTest:non-empty-string">
                                        <attribute name = "stepDuration" use = "optional" type
= "integer"/>
                                </extension>
                        </simpleContent>
                </complexType>
        </element>
        <element name = "SetPart">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:Header" minOccurs = "0" maxOccurs =
"unbounded"/>
                                <choice>
                                        <element ref = "ebTest:Declaration"/>
                                        <element ref = "ebTest:FileURI"/>
                                        <element ref = "ebTest:MessageRef"/>
                                </choice>
                                <element ref = "ebTest:Mutator" minOccurs = "0"/>
                                <element ref = "ebTest:DSign" minOccurs = "0"/>
                        </sequence>
                        <attribute name = "description" use = "optional" type = "string"/>
                </complexType>
        </element>
        <element name = "TestAssertion">
                <complexType>
                        <sequence>
                                <choice>
                                        <element ref = "ebTest:VerifyContent"/>
                                        <element ref = "ebTest:ValidateContent"/>
                                        <element ref = "ebTest:VerifyTimeDifference"/>
                                </choice>
                                <element name = "WhenTrue" minOccurs = "0">
                                        <complexType>
                                                <choice>
                                                        <element ref = "ebTest:Continue"/>
                                                        <element ref = "ebTest:ThreadRef"/>
                                                        <element ref = "ebTest:Split"/>
                                                        <element name = "Exit" type =
"ebTest:assertionExit.type"/>
                                                </choice>
                                        </complexType>
                                </element>
                                <element name = "WhenFalse" minOccurs = "0">
                                        <complexType>
                                                <choice>
                                                        <element ref = "ebTest:Continue"/>
                                                        <element ref = "ebTest:ThreadRef"/>
                                                        <element ref = "ebTest:Split"/>
                                                        <element name = "Exit" type =
"ebTest:assertionExit.type"/>
                                                </choice>
                                        </complexType>
                                </element>
                        </sequence>
                        <attribute name = "description" use = "required" type = "string"/>
                </complexType>
        </element>
        <element name = "MimeHeader" type = "ebTest:mimeHeader.type"/>
        <element name = "MimeHeaderValue" type = "ebTest:non-empty-string"/>
        <element name = "Content-Location" type = "ebTest:non-empty-string"/>
        <element name = "Index" type = "integer"/>
        <element name = "FileURI" type = "anyURI"/>
        <element name = "PayloadRef" type = "ebTest:non-empty-string"/>
```

```
   <element name = "Signature" type = "base64Binary"/>
   <element name = "Content-ID" type = "ebTest:non-empty-string"/>
   <element name = "MessageDeclaration">
          <complexType>
                 <sequence>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                 </sequence>
          </complexType>
   </element>
   <element name = "ValidateContent">
          <complexType>
                 <simpleContent>
                        <extension base = "ebTest:non-empty-string">
                               <attribute name = "contentType" use = "required" type =
"ebTest:content.type"/>
                               <attribute name = "schemaLocation" use = "optional"
type = "anyURI"/>
                        </extension>
                 </simpleContent>
          </complexType>
   </element>
   <element name = "VerifyContent" type = "ebTest:non-empty-string"/>
   <element name = "Message">
          <complexType>
                 <sequence>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                 </sequence>
                 <attribute name = "id" use = "required" type = "ID"/>
          </complexType>
   </element>
   <element name = "SetParameter">
          <complexType>
                 <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <choice>
                               <element name = "Value" type = "ebTest:non-empty-
string"/>
                               <element name = "ParameterRef" type = "ebTest:non-
empty-string"/>
                        </choice>
                 </sequence>
                 <attribute name = "scope" use = "optional" type =
"ebTest:scope.type"/>
          </complexType>
   </element>
   <element name = "Mutator">
          <complexType>
                 <choice>
                        <element ref = "ebTest:XSL"/>
                        <element ref = "ebTest:XUpdate"/>
                 </choice>
          </complexType>
   </element>
   <element name = "XSL" type = "anyURI"/>
   <element name = "XUpdate" type = "anyURI"/>
   <element name = "BooleanClause">
          <complexType>
                 <attribute name = "booleanPredicate" use = "required" type =
"boolean"/>
          </complexType>
   </element>
   <element name = "DSign">
          <complexType>
                 <sequence>
                        <element ref = "ds:Signature"/>
                 </sequence>
          </complexType>
   </element>
   <element name = "Declaration">
```

```
          <complexType>
                  <sequence>
                          <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                  </sequence>
          </complexType>
  </element>
  <element name = "Thread">
          <complexType>
                  <choice maxOccurs = "unbounded">
                          <element ref = "ebTest:SetParameter"/>
                          <element ref = "ebTest:PutMessage"/>
                          <element ref = "ebTest:Initiator"/>
                          <element ref = "ebTest:GetMessage"/>
                          <element ref = "ebTest:TestAssertion"/>
                          <element ref = "ebTest:ThreadRef"/>
                          <element ref = "ebTest:Split"/>
                          <element ref = "ebTest:Join"/>
                          <element ref = "ebTest:Sleep"/>
                  </choice>
                  <attribute name = "name" use = "required" type = "ID"/>
                  <attribute name = "description" use = "optional" type = "string"/>
          </complexType>
  </element>
  <element name = "ThreadRef">
          <complexType>
                  <attribute name = "nameRef" use = "required" type = "IDREF"/>
                  <attribute name = "configurationGroupRef" use = "optional" type =
"anyURI"/>
                  <attribute name = "loop" use = "optional" type = "integer"/>
                  <attribute name = "instanceId" use = "optional" type = "string"/>
          </complexType>
  </element>
  <element name = "Pass">
          <complexType/>
  </element>
  <element name = "Fail">
          <complexType/>
  </element>
  <element name = "ThreadGroup">
          <complexType>
                  <sequence>
                          <element ref = "ebTest:Thread" maxOccurs = "unbounded"/>
                  </sequence>
          </complexType>
  </element>
  <element name = "Sleep" type = "integer"/>
  <element name = "Split">
          <complexType>
                  <sequence maxOccurs = "unbounded">
                          <element ref = "ebTest:ThreadRef"/>
                  </sequence>
          </complexType>
  </element>
  <element name = "Join">
          <complexType>
                  <sequence maxOccurs = "unbounded">
                          <element ref = "ebTest:ThreadRef"/>
                  </sequence>
                  <attribute name = "joinType" use = "optional" type =
"ebTest:join.type"/>
          </complexType>
  </element>
  <element name = "Initiator">
          <complexType>
                  <sequence>
                          <choice>
                                  <element ref = "ebTest:Declaration"/>
                                  <element ref = "ebTest:FileURI"/>
                                  <element ref = "ebTest:MessageRef"/>
                          </choice>
```

```
                        <element ref = "ebTest:Mutator" minOccurs = "0"/>
                        <element ref = "ebTest:DSign" minOccurs = "0"/>
                </sequence>
                <attribute name = "description" use = "required" type = "string"/>
        </complexType>
   </element>
   <element name = "TestServiceConfigurator">
        <complexType>
                <sequence>
                        <element ref = "ebTest:ServiceMode"/>
                        <element ref = "ebTest:ResponseURL"/>
                        <element ref = "ebTest:NotificationURL"/>
                        <element ref = "ebTest:PayloadDigests" minOccurs = "0"/>
                </sequence>
        </complexType>
   </element>
   <element name = "MessageRef" type = "IDREF"/>
   <element name = "ConfigurationItem">
        <complexType>
                <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <element name = "Value" type = "ebTest:non-empty-string"/>
                </sequence>
        </complexType>
   </element>
   <element name = "ErrorURL" type = "anyURI"/>
   <element name = "NotificationURL" type = "anyURI"/>
   <element name = "SetXPathParameter">
        <complexType>
                <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <element name = "Expression" type = "ebTest:non-empty-
string"/>
                </sequence>
                <attribute name = "scope" use = "optional" type =
"ebTest:scope.type"/>
        </complexType>
   </element>
   <element name = "ResponseURL" type = "anyURI"/>
   <element name = "StoreAttachments" type = "boolean"/>
   <element name = "OperationMode" type = "string"/>
   <element name = "PayloadDigests">
        <complexType>
                <sequence>
                        <element name = "Payload" maxOccurs = "unbounded">
                                <complexType>
                                        <sequence>
                                                <element name = "Id" type = "anyURI"/>
                                                <element name = "Digest" type =
"base64Binary"/>
                                        </sequence>
                                </complexType>
                        </element>
                </sequence>
        </complexType>
   </element>
   <element name = "ServiceMode" type = "ebTest:serviceMode.type"/>
   <element name = "Transaction">
        <complexType>
                <sequence maxOccurs = "unbounded">
                        <choice maxOccurs = "unbounded">
                                <element ref = "ebTest:PutMessage"/>
                                <element ref = "ebTest:Initiator"/>
                        </choice>
                        <element ref = "ebTest:GetMessage" minOccurs = "0" maxOccurs =
"unbounded"/>
                </sequence>
                <attribute name = "timeToPerform" use = "optional" type = "duration"/>
        </complexType>
   </element>
   <element name = "VerifyTimeDifference">
```

```
		<complexType>
			<sequence>
				<element ref = "ebTest:ParamName"/>
				<element ref = "ebTest:ParamName"/>
				<element ref = "ebTest:Operator"/>
				<element ref = "ebTest:Difference"/>
			</sequence>
		</complexType>
	</element>
	<element name = "TimeToAcknowledgeReceipt">
		<complexType>
			<sequence>
				<element ref = "ebTest:XPathExpression"/>
			</sequence>
		</complexType>
	</element>
	<element name = "TimeToAcknowledgeAcceptance">
		<complexType>
			<sequence>
				<element ref = "ebTest:XPathExpression"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Difference" type = "duration"/>
	<element name = "Operator" type = "ebTest:operator.type"/>
	<element name = "XPathExpression" type = "ebTest:non-empty-string"/>
	<element name = "Continue">
		<complexType/>
	</element>
	<element name = "ParamName" type = "ebTest:non-empty-string"/>
	<element name = "VerifyTimeToPerform">
		<complexType>
			<sequence>
				<element ref = "ebTest:ThreadName" maxOccurs = "unbounded"/>
			</sequence>
			<attribute name = "maxTime" use = "required" type = "duration"/>
		</complexType>
	</element>
	<element name = "ThreadName" type = "IDREF"/>
	<element name = "Header">
		<complexType>
			<sequence>
				<element ref = "ebTest:Name"/>
				<element ref = "ebTest:Value"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Name" type = "ebTest:non-empty-string"/>
	<element name = "Value" type = "ebTest:non-empty-string"/>
```
</schema>

SOAP Portion of ebXML-Specific Message Store 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/testing/soap"
    xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
    xmlns:xs = "http://www.w3.org/2001/XMLSchema"
    xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd">
   <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"/>
   <group name = "optionElements">
          <all minOccurs = "0">
                 <element ref = "eb:SyncReply" minOccurs = "0"/>
                 <element ref = "eb:MessageOrder" minOccurs = "0"/>
                 <element ref = "eb:AckRequested" minOccurs = "0"/>
                 <element ref = "eb:Acknowledgment" minOccurs = "0"/>
                 <element ref = "eb:ErrorList" minOccurs = "0"/>
          </all>
   </group>
   <attributeGroup name = "encodingStyle">
          <attribute name = "encodingStyle" type = "tns:encodingStyle"/>
   </attributeGroup>

   <!-- Schema for the SOAP/1.1 envelope

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

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

     Copyright 2001 Martin Gudgin, Developmentor.

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

     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:optionElements"/>
                  </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>
                  <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 = "tns:detail" minOccurs = "0"/>
           </sequence>
    </complexType>
    <complexType name = "detail">
           <sequence>
                  <any namespace = "##any" processContents = "lax" minOccurs = "0"
maxOccurs = "unbounded"/>
           </sequence>
           <anyAttribute namespace = "##any" processContents = "lax"/>
    </complexType>
</schema>
```

ebMS Portion of ebXML-Specific Message Store 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/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
    xmlns:tns = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
    xmlns:xlink = "http://www.w3.org/1999/xlink"
    xmlns:ds = "http://www.w3.org/2000/09/xmldsig#"
    xmlns:soap = "http://schemas.xmlsoap.org/soap/envelope/"

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

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

```
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" 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"/>
                        <element ref = "tns:To"/>
                        <element ref = "tns:CPAId"/>
                        <element ref = "tns:ConversationId"/>
                        <element ref = "tns:Service"/>
                        <element ref = "tns:Action"/>
                        <element ref = "tns:MessageData"/>
                        <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"/>
				<element ref = "tns:Timestamp"/>
				<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" use = "required"/>
		</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"/>
				<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:bodyExtension.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:bodyExtension.grp"/>
		</complexType>
	</element>

	<!-- COMMON TYPES -->

	<complexType name = "sequenceNumber.type">
		<simpleContent>
			<extension base = "positiveInteger">
```

```
                         <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 = "PartyId">
          <complexType>
                  <simpleContent>
                         <extension base = "tns:non-empty-string">
                                 <attribute name = "type" type = "tns:non-empty-
string"/>
                         </extension>
                  </simpleContent>
          </complexType>
   </element>
   <element name = "To">
          <complexType>
                  <sequence>
                         <element ref = "tns:PartyId" maxOccurs = "unbounded"/>
                         <element name = "Role" type = "tns:non-empty-string" minOccurs
= "0"/>
                  </sequence>
          </complexType>
   </element>
   <element name = "From">
          <complexType>
                  <sequence>
                         <element ref = "tns:PartyId" maxOccurs = "unbounded"/>
                         <element name = "Role" 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 = "RefToMessageId" type = "tns:non-empty-string"/>
   <element name = "Timestamp" type = "dateTime"/>
</schema>
```

Generic FilterResult 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/testing/messageStore"
    targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->


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


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

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

```
		<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 = "synchType" use = "required" type = "synch.type"/>
		<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 = "reportingAction" use = "optional" type =
"xsd:string"/>
		<xsd:anyAttribute namespace = "##any" processContents = "strict"/>
	</xsd:complexType>
	<xsd:element name = "Notification">
		<xsd:complexType>
			<xsd:complexContent>
				<xsd:extension base = "GenericMessageType">
					<xsd:attribute name = "notificationType" use =
"required" type = "notification.type"/>
				</xsd:extension>
			</xsd:complexContent>
		</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/testing/messageStore"
    targetNamespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/messageStore"
    xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
    xmlns:soap = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <xsd:import namespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/mime"
schemaLocation =
"file:///E:/ebXML_MS_20_Conformance_Testing_1.0/schemas/messagestore_mime.xsd"/>
   <xsd:import namespace = "http://www.oasis-open.org/tc/ebxml-iic/testing/soap"
schemaLocation =
"file:///E:/ebXML_MS_20_Conformance_Testing_1.0/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 2002. All Rights Reserved.
```

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

   <xsd:element name = "FilterResult">
          <xsd:complexType>
                  <xsd:choice>
                         <xsd:element ref = "Message" minOccurs = "0" maxOccurs =
"unbounded"/>
                         <xsd:element ref = "Notification" minOccurs = "0" maxOccurs =
"unbounded"/>
                  </xsd:choice>
          </xsd:complexType>
   </xsd:element>
   <xsd:element name = "GenericMessage" type = "GenericMessageType"/>
   <xsd:simpleType name = "synch.type">
          <xsd:restriction base = "xsd:string">
                  <xsd:enumeration value = "synchronous"/>
                  <xsd:enumeration value = "asynchronous"/>
          </xsd:restriction>
   </xsd:simpleType>
   <xsd:simpleType name = "notification.type">
          <xsd:restriction base = "xsd:NMTOKEN">
                  <xsd:enumeration value = "errURL"/>
                  <xsd:enumeration value = "errorApp"/>
                  <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 = "mime:MessageContainer"/>
                                 </xsd:sequence>
                                 <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:attribute name = "synchType" use = "required" type = "synch.type"/>
          <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 = "reportingAction" use = "optional" type =
"xsd:string"/>
          <xsd:anyAttribute namespace = "##any" processContents = "strict"/>
   </xsd:complexType>
   <xsd:element name = "Notification">
          <xsd:complexType>
                  <xsd:complexContent>
                         <xsd:extension base = "GenericMessageType">
                                 <xsd:sequence>
                                        <xsd:element ref = "soap:Envelope"/>
                                 </xsd:sequence>
                                 <xsd:attribute name = "notificationType" use =
"required" type = "notification.type"/>
```

```
                         </xsd:extension>
                  </xsd:complexContent>
           </xsd:complexType>
    </xsd:element>
</xsd: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/tests"
    xmlns:ebTest = "http://www.oasis-open.org/tc/ebxml-iic/tests"
    xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
    xmlns:mime = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
    version = "1.0"
    elementFormDefault = "unqualified"
    attributeFormDefault = "unqualified">
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/mime"
schemaLocation = "file:///C:/scripting_poc_07_13_04/schemas/mime.xsd"/>
   <!-- edited with XML Spy v4.3 U (http://www.xmlspy.com) by Michael Kass (NIST) -->


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


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


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

   <element name = "TestSuite">
          <complexType>
                 <sequence>
                        <element ref = "ebTest:MetaData"/>
                        <element ref = "ebTest:ConfigurationGroup" maxOccurs =
"unbounded"/>
                        <element ref = "ebTest:TestServiceConfigurator" minOccurs =
"0"/>
                        <element ref = "ebTest:Message" minOccurs = "0" maxOccurs =
"unbounded"/>
                        <element ref = "ebTest:TestCase" maxOccurs = "unbounded"/>
                 </sequence>
                 <attribute name = "configurationGroupRef" use = "required" type =
"anyURI"/>
          </complexType>
   </element>
   <element name = "MetaData">
          <complexType>
                 <sequence>
                        <element ref = "ebTest:Description"/>
                        <element ref = "ebTest:Version"/>
                        <element ref = "ebTest:Maintainer"/>
                        <element ref = "ebTest:Location"/>
                        <element ref = "ebTest:PublishDate"/>
                        <element ref = "ebTest:Status"/>
                </sequence>
        </complexType>
</element>
<element name = "Description" type = "ebTest:non-empty-string"/>
<element name = "Version" type = "ebTest:non-empty-string"/>
<element name = "Maintainer" type = "ebTest:non-empty-string"/>
<element name = "Location" type = "anyURI"/>
<element name = "PublishDate" type = "ebTest:non-empty-string"/>
<element name = "Status" type = "ebTest:non-empty-string"/>
<element name = "TestCase">
        <complexType>
                <sequence>
                        <element ref = "ebTest:ThreadGroup" minOccurs = "0"/>
                        <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
= "unbounded"/>
                        <choice>
                                <element ref = "ebTest:Thread"/>
                                <element ref = "ebTest:ThreadRef"/>
                                <sequence maxOccurs = "unbounded">
                                        <element ref = "ebTest:Split" maxOccurs =
"unbounded"/>
                                        <element ref = "ebTest:Join" maxOccurs =
"unbounded"/>
                                </sequence>
                        </choice>
                </sequence>
                <attribute name = "id" use = "required" type = "ID"/>
                <attribute name = "description" use = "required" type = "string"/>
                <attribute name = "author" use = "optional" type = "string"/>
                <attribute name = "version" use = "optional" type = "string"/>
                <attribute name = "requirementReferenceId" use = "required" type =
"anyURI"/>
                <attribute name = "configurationGroupRef" use = "optional" type =
"anyURI"/>
                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
        </complexType>
</element>
<element name = "ConfigurationGroup">
        <complexType>
                <sequence>
                        <element ref = "ebTest:Mode"/>
                        <element ref = "ebTest:StepDuration"/>
                        <element ref = "ebTest:Transport"/>
                        <element ref = "ebTest:Envelope"/>
                        <element ref = "ebTest:StoreAttachments"/>
                        <element ref = "ebTest:SetParameter" minOccurs = "0" maxOccurs
= "unbounded"/>
                </sequence>
                <attribute name = "id" use = "required" type = "ID"/>
        </complexType>
</element>
<element name = "CPAId" type = "ebTest:non-empty-string"/>
<element name = "Mode" type = "ebTest:mode.type"/>
<element name = "SenderParty" type = "anyURI"/>
<element name = "ReceiverParty" type = "anyURI"/>
<element name = "Service" type = "anyURI"/>
<element name = "Action" type = "ebTest:non-empty-string"/>
<element name = "StepDuration" type = "integer"/>
<element name = "Transport" type = "ebTest:transport.type"/>
<element name = "Envelope" type = "ebTest:non-empty-string"/>
<simpleType name = "mode.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "local-service"/>
                <enumeration value = "remote-service"/>
                <enumeration value = "connection"/>
        </restriction>
```

```
</simpleType>
<simpleType name = "mimeHeader.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "MIMEMessageContent-Type"/>
                <enumeration value = "MIMEMessageStart"/>
                <enumeration value = "Content-Type"/>
                <enumeration value = "start"/>
                <enumeration value = "charset"/>
                <enumeration value = "type"/>
                <enumeration value = "wildcard"/>
        </restriction>
</simpleType>
<simpleType name = "content.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "XML"/>
                <enumeration value = "date"/>
                <enumeration value = "URI"/>
                <enumeration value = "signature"/>
                <enumeration value = "XPointer"/>
        </restriction>
</simpleType>
<simpleType name = "method.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "xpath"/>
                <enumeration value = "md5"/>
        </restriction>
</simpleType>
<simpleType name = "messageContext.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "true"/>
                <enumeration value = "false"/>
        </restriction>
</simpleType>
<simpleType name = "requirement.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "required"/>
                <enumeration value = "stronglyrecommended"/>
                <enumeration value = "recommended"/>
                <enumeration value = "optional"/>
        </restriction>
</simpleType>
<simpleType name = "non-empty-string">
        <restriction base = "string">
                <minLength value = "1"/>
        </restriction>
</simpleType>
<simpleType name = "configAction.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "query"/>
                <enumeration value = "replace"/>
        </restriction>
</simpleType>
<simpleType name = "action.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "reset"/>
                <enumeration value = "modify"/>
        </restriction>
</simpleType>
<simpleType name = "configItem.type">
        <restriction base = "NMTOKEN"/>
</simpleType>
<simpleType name = "parameter.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "string"/>
                <enumeration value = "parameter"/>
        </restriction>
</simpleType>
<simpleType name = "connectivePredicate.type">
        <restriction base = "NMTOKEN">
                <enumeration value = "and"/>
                <enumeration value = "or"/>
```

```
		</restriction>
	</simpleType>
	<simpleType name = "thread.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "synchronous"/>
			<enumeration value = "asynchronous"/>
		</restriction>
	</simpleType>
	<simpleType name = "matchResult.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "pass"/>
			<enumeration value = "fail"/>
		</restriction>
	</simpleType>
	<simpleType name = "if.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "andif"/>
			<enumeration value = "orif"/>
		</restriction>
	</simpleType>
	<simpleType name = "split.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "andsplit"/>
			<enumeration value = "orsplit"/>
		</restriction>
	</simpleType>
	<simpleType name = "join.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "andjoin"/>
			<enumeration value = "orjoin"/>
		</restriction>
	</simpleType>
	<simpleType name = "serviceMode.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "loop"/>
			<enumeration value = "local-reporting"/>
			<enumeration value = "remote-reporting"/>
		</restriction>
	</simpleType>
	<simpleType name = "time.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "timeToAcknowlegeReceipt"/>
			<enumeration value = "timeToAcknowledgeAcceptance"/>
			<enumeration value = "timeToPerform"/>
			<enumeration value = "other"/>
		</restriction>
	</simpleType>
	<simpleType name = "operator.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "equal"/>
			<enumeration value = "lessThanl"/>
			<enumeration value = "lessThanOrEqual"/>
			<enumeration value = "greaterThan"/>
			<enumeration value = "greaterThanOrEqual"/>
		</restriction>
	</simpleType>
	<simpleType name = "assertionExit.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "pass"/>
			<enumeration value = "fail"/>
			<enumeration value = "undetermined"/>
		</restriction>
	</simpleType>
	<simpleType name = "preconditionExit.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "undetermined"/>
		</restriction>
	</simpleType>
	<simpleType name = "scope.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "self"/>
```

```
                    <enumeration value = "selfAndDescendents"/>
            </restriction>
    </simpleType>
    <simpleType name = "transport.type">
            <restriction base = "NMTOKEN">
                    <enumeration value = "FTP"/>
                    <enumeration value = "SMTP"/>
                    <enumeration value = "HTTP"/>
            </restriction>
    </simpleType>
    <simpleType name = "result.type">
            <restriction base = "NMTOKEN">
                    <enumeration value = "pass"/>
                    <enumeration value = "fail"/>
                    <enumeration value = "undetermined"/>
            </restriction>
    </simpleType>
    <simpleType name = "exception.type">
            <restriction base = "NMTOKEN">
                    <enumeration value = "undetermined"/>
            </restriction>
    </simpleType>
    <element name = "MessageExpression">
            <complexType>
                    <sequence>
                            <element ref = "ebTest:ErrorMessage"/>
                    </sequence>
            </complexType>
    </element>
    <element name = "ErrorMessage" type = "ebTest:non-empty-string"/>
    <element name = "PutMessage">
            <complexType>
                    <sequence>
                            <element ref = "ebTest:SetPart" maxOccurs = "unbounded"/>
                    </sequence>
                    <attribute name = "description" use = "required" type = "string"/>
                    <attribute name = "repeatWithSameContext" use = "optional" type =
"integer"/>
                    <attribute name = "repeatWithNewContext" use = "optional" type =
"integer"/>
            </complexType>
    </element>
    <element name = "GetMessage">
            <complexType>
                    <sequence maxOccurs = "unbounded">
                            <element ref = "ebTest:Filter"/>
                            <element ref = "ebTest:SetXPathParameter" minOccurs = "0"
maxOccurs = "unbounded"/>
                    </sequence>
                    <attribute name = "description" use = "required" type = "string"/>
                    <attribute name = "mask" use = "optional" type = "boolean"/>
            </complexType>
    </element>
    <element name = "Filter">
            <complexType>
                    <simpleContent>
                            <extension base = "ebTest:non-empty-string">
                                    <attribute name = "stepDuration" use = "optional" type
= "integer"/>
                                    <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
                            </extension>
                    </simpleContent>
            </complexType>
    </element>
    <element name = "SetPart">
            <complexType>
                    <sequence>
                            <element ref = "ebTest:Header" minOccurs = "0" maxOccurs =
"unbounded"/>
                            <choice>
```

```
                                   <element ref = "ebTest:Declaration"/>
                                   <element ref = "ebTest:FileURI"/>
                                   <element ref = "ebTest:MessageRef"/>
                           </choice>
                           <element ref = "ebTest:Mutator" minOccurs = "0"/>
                           <element ref = "ebTest:DSign" minOccurs = "0"/>
                   </sequence>
                   <attribute name = "description" use = "optional" type = "string"/>
                   <attribute name = "result" use = "required" type = 
"ebTest:result.type"/>
           </complexType>
   </element>
   <element name = "TestAssertion">
           <complexType>
                   <sequence>
                           <choice>
                                   <element ref = "ebTest:VerifyContent"/>
                                   <element ref = "ebTest:ValidateContent"/>
                                   <element ref = "ebTest:VerifyTimeDifference"/>
                           </choice>
                           <element name = "WhenTrue" minOccurs = "0">
                                   <complexType>
                                           <choice>
                                                   <element ref = "ebTest:Continue"/>
                                                   <element ref = "ebTest:ThreadRef"/>
                                                   <element ref = "ebTest:Split"/>
                                                   <element name = "Exit" type = 
"ebTest:assertionExit.type"/>
                                           </choice>
                                   </complexType>
                           </element>
                           <element name = "WhenFalse" minOccurs = "0">
                                   <complexType>
                                           <choice>
                                                   <element ref = "ebTest:Continue"/>
                                                   <element ref = "ebTest:ThreadRef"/>
                                                   <element ref = "ebTest:Split"/>
                                                   <element name = "Exit" type = 
"ebTest:assertionExit.type"/>
                                           </choice>
                                   </complexType>
                           </element>
                   </sequence>
                   <attribute name = "description" use = "required" type = "string"/>
                   <attribute name = "result" use = "required" type = 
"ebTest:result.type"/>
           </complexType>
   </element>
   <element name = "MimeHeader" type = "ebTest:mimeHeader.type"/>
   <element name = "MimeHeaderValue" type = "ebTest:non-empty-string"/>
   <element name = "Content-Location" type = "ebTest:non-empty-string"/>
   <element name = "Index" type = "integer"/>
   <element name = "FileURI" type = "anyURI"/>
   <element name = "PayloadRef" type = "ebTest:non-empty-string"/>
   <element name = "Signature" type = "base64Binary"/>
   <element name = "Content-ID" type = "ebTest:non-empty-string"/>
   <element name = "MessageDeclaration">
           <complexType>
                   <sequence>
                           <any namespace = "##other" processContents = "lax" minOccurs = 
"0" maxOccurs = "unbounded"/>
                   </sequence>
           </complexType>
   </element>
   <element name = "ValidateContent">
           <complexType>
                   <simpleContent>
                           <extension base = "ebTest:non-empty-string">
                                   <attribute name = "contentType" use = "required" type = 
"ebTest:content.type"/>
```

```
                                <attribute name = "schemaLocation" use = "optional"
type = "anyURI"/>
                                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
                        </extension>
                </simpleContent>
        </complexType>
   </element>
   <element name = "VerifyContent">
        <complexType>
                <simpleContent>
                        <extension base = "ebTest:non-empty-string">
                                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
                        </extension>
                </simpleContent>
        </complexType>
   </element>
   <element name = "Message">
        <complexType>
                <sequence>
                        <any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
                </sequence>
                <attribute name = "id" use = "required" type = "ID"/>
        </complexType>
   </element>
   <element name = "SetParameter">
        <complexType>
                <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <choice>
                                <element name = "Value" type = "ebTest:non-empty-
string"/>
                                <element name = "ParameterRef" type = "ebTest:non-
empty-string"/>
                        </choice>
                </sequence>
                <attribute name = "scope" use = "optional" type =
"ebTest:scope.type"/>
                <attribute name = "result" use = "optional" type =
"ebTest:exception.type"/>
        </complexType>
   </element>
   <element name = "Mutator">
        <complexType>
                <choice>
                        <element ref = "ebTest:XSL"/>
                        <element ref = "ebTest:XUpdate"/>
                </choice>
                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
        </complexType>
   </element>
   <element name = "XSL" type = "anyURI"/>
   <element name = "XUpdate" type = "anyURI"/>
   <element name = "BooleanClause">
        <complexType>
                <attribute name = "booleanPredicate" use = "required" type =
"boolean"/>
        </complexType>
   </element>
   <element name = "DSign">
        <complexType>
                <sequence>
                        <element ref = "ds:Signature"/>
                </sequence>
                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
        </complexType>
   </element>
```

```
	<element name = "Declaration">
		<complexType>
			<sequence>
				<any namespace = "##other" processContents = "lax" minOccurs = 
"0" maxOccurs = "unbounded"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Thread">
		<complexType>
			<choice maxOccurs = "unbounded">
				<element ref = "ebTest:SetParameter"/>
				<element ref = "ebTest:PutMessage"/>
				<element ref = "ebTest:Initiator"/>
				<element ref = "ebTest:GetMessage"/>
				<element ref = "ebTest:TestAssertion"/>
				<element ref = "ebTest:ThreadRef"/>
				<element ref = "ebTest:Split"/>
				<element ref = "ebTest:Join"/>
				<element ref = "ebTest:Sleep"/>
			</choice>
			<attribute name = "name" use = "required" type = "ID"/>
			<attribute name = "description" use = "optional" type = "string"/>
		</complexType>
	</element>
	<element name = "ThreadRef">
		<complexType>
			<attribute name = "nameRef" use = "required" type = "IDREF"/>
			<attribute name = "configurationGroupRef" use = "optional" type = 
"anyURI"/>
			<attribute name = "loop" use = "optional" type = "integer"/>
			<attribute name = "instanceId" use = "optional" type = "string"/>
			<attribute name = "result" use = "required" type = 
"ebTest:result.type"/>
		</complexType>
	</element>
	<element name = "Pass">
		<complexType/>
	</element>
	<element name = "Fail">
		<complexType/>
	</element>
	<element name = "ThreadGroup">
		<complexType>
			<sequence>
				<element ref = "ebTest:Thread" maxOccurs = "unbounded"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Sleep" type = "integer"/>
	<element name = "Split">
		<complexType>
			<sequence maxOccurs = "unbounded">
				<element ref = "ebTest:ThreadRef"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Join">
		<complexType>
			<sequence maxOccurs = "unbounded">
				<element ref = "ebTest:ThreadRef"/>
			</sequence>
			<attribute name = "joinType" use = "optional" type = 
"ebTest:join.type"/>
		</complexType>
	</element>
	<element name = "Initiator">
		<complexType>
			<sequence>
				<choice>
					<element ref = "ebTest:Declaration"/>
```

```
                                <element ref = "ebTest:FileURI"/>
                                <element ref = "ebTest:MessageRef"/>
                        </choice>
                        <element ref = "ebTest:Mutator" minOccurs = "0"/>
                        <element ref = "ebTest:DSign" minOccurs = "0"/>
                </sequence>
                <attribute name = "description" use = "required" type = "string"/>
                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
        </complexType>
   </element>
   <element name = "TestServiceConfigurator">
        <complexType>
                <sequence>
                        <element ref = "ebTest:ServiceMode"/>
                        <element ref = "ebTest:ResponseURL"/>
                        <element ref = "ebTest:NotificationURL"/>
                        <element ref = "ebTest:PayloadDigests" minOccurs = "0"/>
                </sequence>
        </complexType>
   </element>
   <element name = "MessageRef" type = "IDREF"/>
   <element name = "ConfigurationItem">
        <complexType>
                <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <element name = "Value" type = "ebTest:non-empty-string"/>
                </sequence>
        </complexType>
   </element>
   <element name = "ErrorURL" type = "anyURI"/>
   <element name = "NotificationURL" type = "anyURI"/>
   <element name = "SetXPathParameter">
        <complexType>
                <sequence>
                        <element name = "Name" type = "ebTest:non-empty-string"/>
                        <element name = "Expression" type = "ebTest:non-empty-
string"/>
                </sequence>
                <attribute name = "scope" use = "optional" type =
"ebTest:scope.type"/>
                <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
        </complexType>
   </element>
   <element name = "ResponseURL" type = "anyURI"/>
   <element name = "StoreAttachments" type = "boolean"/>
   <element name = "OperationMode" type = "string"/>
   <element name = "PayloadDigests">
        <complexType>
                <sequence>
                        <element name = "Payload" maxOccurs = "unbounded">
                                <complexType>
                                        <sequence>
                                                <element name = "Id" type = "anyURI"/>
                                                <element name = "Digest" type =
"base64Binary"/>
                                        </sequence>
                                </complexType>
                        </element>
                </sequence>
        </complexType>
   </element>
   <element name = "ServiceMode" type = "ebTest:serviceMode.type"/>
   <element name = "Transaction">
        <complexType>
                <sequence maxOccurs = "unbounded">
                        <choice maxOccurs = "unbounded">
                                <element ref = "ebTest:PutMessage"/>
                                <element ref = "ebTest:Initiator"/>
                        </choice>
```

```
                                <element ref = "ebTest:GetMessage" minOccurs = "0" maxOccurs =
"unbounded"/>
                        </sequence>
                        <attribute name = "timeToPerform" use = "optional" type = "duration"/>
                </complexType>
        </element>
        <element name = "VerifyTimeDifference">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:ParamName"/>
                                <element ref = "ebTest:ParamName"/>
                                <element ref = "ebTest:Operator"/>
                                <element ref = "ebTest:Difference"/>
                        </sequence>
                        <attribute name = "result" use = "required" type =
"ebTest:result.type"/>
                </complexType>
        </element>
        <element name = "Difference" type = "duration"/>
        <element name = "Operator" type = "ebTest:operator.type"/>
        <element name = "XPathExpression" type = "ebTest:non-empty-string"/>
        <element name = "Continue">
                <complexType/>
        </element>
        <element name = "ParamName" type = "ebTest:non-empty-string"/>
        <element name = "VerifyTimeToPerform">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:ThreadName" maxOccurs = "unbounded"/>
                        </sequence>
                        <attribute name = "maxTime" use = "required" type = "duration"/>
                </complexType>
        </element>
        <element name = "ThreadName" type = "IDREF"/>
        <element name = "Header">
                <complexType>
                        <sequence>
                                <element ref = "ebTest:Name"/>
                                <element ref = "ebTest:Value"/>
                        </sequence>
                </complexType>
        </element>
        <element name = "Name" type = "ebTest:non-empty-string"/>
        <element name = "Value" type = "ebTest:non-empty-string"/>
</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/tc/ebxml-iic/tests/messages"
    xmlns:tns = "http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
    xmlns:eb = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-header-
2_0.xsd"
    xmlns:ds = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
   <import namespace = "http://www.oasis-open.org/committees/ebxml-msg/schema/msg-
header-2_0.xsd" schemaLocation = "http://www.oasis-open.org/committees/ebxml-
msg/schema/msg-header-2_0.xsd"/>
   <import namespace = "http://www.oasis-open.org/tc/ebxml-iic/tests/xmldsig"
schemaLocation = "xmldsig.xsd"/>

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


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

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

```
                </sequence>
            </complexType>
    </element>
    <element name = "NotificationResponse">
            <complexType>
                    <sequence>
                            <element ref = "tns:Success"/>
                    </sequence>
            </complexType>
    </element>
    <complexType name = "GenericMessageType">
            <attribute name = "synchType" use = "required" type = "tns:synch.type"/>
            <attribute name = "id" use = "required" type = "string"/>
            <attribute name = "serviceInstanceId" use = "optional" type = "string"/>
            <attribute name = "serviceName" use = "optional" type = "string"/>
            <attribute name = "reportingAction" use = "optional" type = "string"/>
            <anyAttribute namespace = "##any" processContents = "strict"/>
    </complexType>
    <element name = "NotificationRequest">
            <complexType>
                    <complexContent>
                            <extension base = "tns:GenericMessageType">
                                    <sequence>
                                            <element ref = "tns:Part" maxOccurs =
"unbounded"/>
                                    </sequence>
                                    <attribute name = "notificationType" use = "required"
type = "tns:notification.type"/>
                            </extension>
                    </complexContent>
            </complexType>
    </element>
    <element name = "TestServiceConfiguratorRequest">
            <complexType>
                    <sequence>
                            <element name = "ServiceMode" type = "tns:serviceMode.type"/>
                            <element ref = "tns:ResponseURL"/>
                            <element ref = "tns:NotificationURL"/>
                            <element ref = "tns:PayloadDigests" minOccurs = "0"/>
                    </sequence>
            </complexType>
    </element>
    <element name = "ResponseURL" type = "anyURI"/>
    <element name = "TestServiceConfiguratorResponse">
            <complexType>
                    <sequence>
                            <element ref = "tns:Success"/>
                    </sequence>
            </complexType>
    </element>
    <element name = "Status" type = "boolean"/>
    <element name = "Mode" type = "tns:non-empty-string"/>
    <element name = "MessageId" type = "tns:non-empty-string"/>
    <simpleType name = "non-empty-string">
            <restriction base = "string">
                    <minLength value = "1"/>
            </restriction>
    </simpleType>
    <simpleType name = "configAction.type">
            <restriction base = "NMTOKEN">
                    <enumeration value = "query"/>
                    <enumeration value = "replace"/>
            </restriction>
    </simpleType>
    <simpleType name = "result.type">
            <restriction base = "NMTOKEN">
                    <enumeration value = "pass"/>
                    <enumeration value = "fail"/>
            </restriction>
    </simpleType>
    <simpleType name = "operationMode.type">
```

```
		<restriction base = "NMTOKEN">
			<enumeration value = "reporting"/>
			<enumeration value = "loop"/>
		</restriction>
	</simpleType>
	<simpleType name = "parameter.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "parameter"/>
			<enumeration value = "string"/>
		</restriction>
	</simpleType>
	<simpleType name = "serviceMode.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "remote-reporting"/>
			<enumeration value = "local-reporting"/>
			<enumeration value = "loop"/>
		</restriction>
	</simpleType>
	<simpleType name = "parameter.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "string"/>
			<enumeration value = "namespace"/>
		</restriction>
	</simpleType>
	<simpleType name = "notification.type">
		<restriction base = "NMTOKEN">
			<enumeration value = "errURL"/>
			<enumeration value = "errorApp"/>
			<enumeration value = "message"/>
		</restriction>
	</simpleType>
	<simpleType name = "synch.type">
		<restriction base = "string">
			<enumeration value = "synchronous"/>
			<enumeration value = "asynchronous"/>
		</restriction>
	</simpleType>
	<element name = "OperationMode" type = "tns:operationMode.type"/>
	<element name = "NotificationURL" type = "anyURI"/>
	<element name = "DSign">
		<complexType>
			<sequence>
				<element ref = "ds:Signature"/>
			</sequence>
		</complexType>
	</element>
	<element name = "PayloadDigests">
		<complexType>
			<sequence>
				<element ref = "tns:Payload" maxOccurs = "unbounded"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Id" type = "string"/>
	<element name = "Digest" type = "string"/>
	<element name = "SetPart">
		<complexType>
			<sequence>
				<element ref = "tns:Header" minOccurs = "0" maxOccurs =
"unbounded"/>
				<element ref = "tns:Declaration" minOccurs = "0"/>
				<element ref = "tns:DSign" minOccurs = "0"/>
			</sequence>
		</complexType>
	</element>
	<element name = "Header">
		<complexType>
			<sequence>
				<element ref = "tns:Name"/>
				<element ref = "tns:Value"/>
			</sequence>
			</complexType>
	</element>
	<element name = "Declaration">
			<complexType>
					<sequence>
							<any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
					</sequence>
			</complexType>
	</element>
	<element name = "Payload">
			<complexType>
					<sequence>
							<element ref = "tns:Id"/>
							<element ref = "tns:Success"/>
					</sequence>
			</complexType>
	</element>
	<element name = "Name" type = "string"/>
	<element name = "Value" type = "string"/>
	<element name = "Part">
			<complexType>
					<sequence>
							<element ref = "tns:Header" minOccurs = "0" maxOccurs =
"unbounded"/>
							<element ref = "tns:Content" minOccurs = "0"/>
					</sequence>
			</complexType>
	</element>
	<element name = "Content">
			<complexType>
					<sequence>
							<any namespace = "##other" processContents = "lax" minOccurs =
"0" maxOccurs = "unbounded"/>
					</sequence>
			</complexType>
	</element>
	<element name = "PayloadVerifyResponse">
			<complexType>
					<sequence>
							<element ref = "tns:Payload" minOccurs = "0" maxOccurs =
"unbounded"/>
					</sequence>
			</complexType>
	</element>
	<element name = "Success" type = "boolean"/>
</schema>
```

Appendix G WSDL Definitions for Test Service

WSDL Definition of the Test Service initiator SOAP method

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mike Kass (Personal) -
->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd1="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
name="RegistryService">
   <wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
location="schemas/TestServiceMessages.xsd"/>
   <wsdl:message name="InitiatorRequest">
           <wsdl:part name="InitiatorRequest" element="xsd1:InitiatorRequest"/>
   </wsdl:message>
   <wsdl:message name="InitiatorResponse">
           <wsdl:part name="InitiatorResponse" element="xsd1:InitiatorResponse"/>
   </wsdl:message>
   <wsdl:portType name="SendPortType">
           <documentation>Maps to the Initiator interface of Test Framework
spec.</documentation>
           <wsdl:operation name="initiator">
                   <wsdl:input message="tns:InitiatorRequest"/>
                   <wsdl:output message="tns:InitiatorResponse"/>
           </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="InitiatorSOAPBinding" type="tns:SendPortType">
           <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
           <wsdl:operation name="initiator">
                   <soap:operation
soapAction="uri:oasis:ebxml:iic:testservice:Send:initiator"/>
                   <wsdl:input>
                           <mime:multipartRelated>
                                   <mime:part>
                                           <soap:body use="literal"/>
                                   </mime:part>
                           </mime:multipartRelated>
                   </wsdl:input>
                   <wsdl:output>
                           <mime:multipartRelated>
                                   <mime:part>
                                           <soap:body use="literal"/>
                                   </mime:part>
                           </mime:multipartRelated>
                   </wsdl:output>
           </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="TestService">
           <documentation>The QueryManager service of OASIS ebXML Test Framework version
1.1</documentation>
           <wsdl:port name="InitiatorSOAPBinding" binding="tns:InitiatorSOAPBinding">
                   <soap:address
location="http://your_URL_to_your_ConfigurationService"/>
```

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

WSDL Definitnion of the Test Service configure method

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mike Kass (Personal) -
->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd1="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
name="RegistryService">
	<wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
location="schemas/TestServiceMessages.xsd"/>
	<wsdl:message name="TestServiceConfiguratorRequest">
		<wsdl:part name="TestServiceConfiguratorRequest"
element="xsd1:TestServiceConfiguratorRequest"/>
	</wsdl:message>
	<wsdl:message name="TestServiceConfiguratorResponse">
		<wsdl:part name="TestServiceConfiguratorResponse"
element="xsd1:TestServiceConfiguratorResponse"/>
	</wsdl:message>
	<wsdl:portType name="ConfigurationPortType">
		<documentation>Maps to the Configurator interface of Test Framework
spec.</documentation>
		<wsdl:operation name="configurator">
			<wsdl:input message="tns:TestServiceConfiguratorRequest"/>
			<wsdl:output message="tns:TestServiceConfiguratorResponse"/>
		</wsdl:operation>
	</wsdl:portType>
	<wsdl:binding name="ConfiguratorSOAPBinding" type="tns:ConfigurationPortType">
		<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
		<wsdl:operation name="configurator">
			<soap:operation
soapAction="uri:oasis:ebxml:iic:testservice:Configuration:configurator"/>
			<wsdl:input>
				<mime:multipartRelated>
					<mime:part>
						<soap:body/>
					</mime:part>
				</mime:multipartRelated>
			</wsdl:input>
			<wsdl:output>
				<mime:multipartRelated>
					<mime:part>
						<soap:body/>
					</mime:part>
				</mime:multipartRelated>
			</wsdl:output>
		</wsdl:operation>
	</wsdl:binding>
	<wsdl:service name="TestService">
```

```
          <documentation>The QueryManager service of OASIS ebXML Test Framework version
1.1</documentation>
          <wsdl:port name="ConfiguratorSOAPBinding"
binding="tns:ConfiguratorSOAPBinding">
                 <soap:address
location="http://your_URL_to_your_ConfigurationService"/>
          </wsdl:port>
   </wsdl:service>
   <documentation>This is the the normative abstract WSDL service definition for the
OASIS ebXML Test Service</documentation>
</wsdl:definitions>
```

WSDL Definition of the Test Driver notify method

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 3 U (http://www.xmlspy.com) by Mike Kass (Personal) -
->
<wsdl:definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="urn:oasis:names:tc:ebxml-
iic:testservice:wsdl:2.0" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd1="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
targetNamespace="urn:oasis:names:tc:ebxml-iic:testservice:wsdl:2.0"
name="RegistryService">
   <wsdl:import namespace="http://www.oasis-open.org/tc/ebxml-iic/tests/messages"
location="schemas/TestServiceMessages.xsd"/>
   <wsdl:message name="NotificationRequest">
          <wsdl:part name="NotificationRequest" element="xsd1:NotificationRequest"/>
   </wsdl:message>
   <wsdl:message name="NotificationResponse">
          <wsdl:part name="NotificationResponse" element="xsd1:NotificationResponse"/>
   </wsdl:message>
   <wsdl:portType name="NotificationPortType">
          <documentation>Maps to the Notification interface of Test Framework
spec.</documentation>
          <wsdl:operation name="Notify">
                 <wsdl:input message="tns:NotificationRequest"/>
                 <wsdl:output message="tns:NotificationResponse"/>
          </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="NotificationSOAPBinding" type="tns:NotificationPortType">
          <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
          <wsdl:operation name="Notify">
                 <soap:operation
soapAction="uri:oasis:ebxml:iic:testservice:Receive:Notification"/>
                 <wsdl:input>
                        <mime:multipartRelated>
                               <mime:part>
                                      <soap:body use="literal"/>
                               </mime:part>
                        </mime:multipartRelated>
                 </wsdl:input>
                 <wsdl:output>
                        <mime:multipartRelated>
                               <mime:part>
                                      <soap:body use="literal"/>
                               </mime:part>
                        </mime:multipartRelated>
                 </wsdl:output>
          </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="TestDriverReceiveService">
          <documentation>The Receive service of OASIS ebXML Test Framework version
1.1</documentation>
          <wsdl:port name="NotifySOAPBinding" binding="tns:NotificationSOAPBinding">
                 <soap:address location="http://your_URL_to_your_ReceiveService"/>
          </wsdl:port>
   </wsdl:service>
   <documentation>This is the the normative abstract WSDL service definition for the
OASIS ebXML Test Service</documentation>
</wsdl:definitions>
```

Appendix H 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 Threads, aimed at verifying a test requirement.
Test Requirements coverage	Specifies the degree that the test requirements are satisfied by the set of test cases listed in the test suite document. Coverage can be full, contingent, partial or none.

Appendix I References

I.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, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[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/

I.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 J 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.

J.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>

6810 Appendix K Revision History

6811

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

Appendix L Notices