ebXML Messaging (2.0) Basic Interoperability Profile Test Suite

Committee Specification Version 1.0

OASIS ebXML Implementation, Interoperability and Conformance Technical Committee

03 April, 2003

Document identifier:

ebxml-iic-basic-interop-test-suite-10

Location:

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

Authors/Editors:

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

Contributors:

Kazunori Iwasa <kiwasa@jp.fujitsu.com>
Rik Drummond, Drummong Group Inc. <rik@drummondgroup.com>
Mike Dillon, Drummond Group Inc. <mike@drummondgroup.com>
Masahiko Narita <masahiko.narita@jp.fujitsu.com>
Christopher Frank < C.Frank@seeburger.de>
Eric VanLydegraf, < ericv@kinzan.com>
Serm Kulvatunyou, NIST <serm@nist.gov>

See in Appendix C the complete list of IIC members having contributed.

Abstract:

This document specifies a basic test suite for ebXML Messaging interoperability, that can be used for the testing of global interoperability between all communities of business users.

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.

This version

V1.0

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 ebXML Glossary of Terms in the TestFramework specification [ebTestFramework]. 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),

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

1.6 Objectives and Methodology

1.6.1 Interoperability Profiles

It is in impractical to test all combinations of messaging features and configuration features for interoperability between two message handler implementations: there is generally a large number of combinations – and of possible failure scenarios. As two or more message handlers are involved, these combinations are even greater than for conformance testing, which typically focuses on a single message handler.

When testing interoperability, a small set of significant test cases must be selected. One way to do this selection is to observe the interoperability requirements of a user community, and to address them. Because of the “combinatorial” problem of features and scenarios, and also because it involves several business partners, interoperability testing usually must be restricted to reflect the particular needs of a business community. This is in contrast with conformance testing, which mostly focuses on verifying adherence to the standard.

Interoperability tests should then focus on the kind of usage that is most meaningful for a business community. These forms – or modes - of interoperability are called *profiles*. An interoperability profile should be verified by an appropriate test suite.

1.6.2 A Basic Interoperability Profile

This document specifies the Basic Interoperability Profile (BIP) for ebXML messaging. The primary objective of this profile is to define the baseline of business interoperability (it exercises basic ebXML MS core services, secure and reliable messaging). This profile may not be sufficient to address all the business requirements of a user community: Specific requirements – for example, using very large messages, or security features such as encryption - will be addressed by additional, more specific profiles that expand on basic functions or combinations of functions relevant to user communities.

The number of requirements test requirements for the Messaging BIP is relatively small (as compare to the number of test requirements for the conformance test suite.) This is intentional, to enable interoperability and lower the cost of entry of testing. The reasons for keeping an interoperability test suite small are:

- Interoperability testing requires more efforts in logistics than conformance testing, as coordination between parties is required.
- Interoperability may be affected by several factors such as operating environment, third-party software or utilities, testing should be done under normal operating conditions. This creates constraints and disturbance for business.

Users or industry groups are encouraged to design additional interoperability profiles, if these are not already specified in the test suites produced by the ebXML IIC Technical Committee. In order to be conforming to the IIC testing guidelines, any new messaging interoperability profile definition:

- MUST include the Basic Interoperability Profile (i.e. extend it)
- MUST be described using the test material (test case scripting, test architecture) specified in the ebXML IIC Test Framework.

1.6.3 Related Initiatives and Contributing Parties

In accordance with the notion that interoperability testing - more than conformance testing - should be aligned with business requirements –, the IIC TC has consulted some user communities in order to establish a minimal, yet universal set of messaging interoperability requirements.

- In the United States, UCC (Uniform Code Council) and DGI (Drummond Group, Inc.) have been conducting ebXML interoperability test rounds between several ebXML vendors. The requirements of UCC-DGI tests have been studied, and after investigation, a subset of test requirements defined by UCC-DGI have been used as an input for the Basic Interoperability profile test requirements.
- In Asia, ECOM (E-Commerce consortium of major Asian IT vendors and government agencies) has also organized ebXML interoperability testing rounds. The requirements of this community of users have also proved valuable and have been taken into account for the Basic Interoperability profile.
- In Europe, the e-Business Board for European Standardization workshop (eBES) is a forum for IT vendors and users, sponsored by the European Committee for Standardization (CEN) and Information Society Standardization System (ISSS). eBES focus is on business-to-business and interoperability testing. The group is also organizing ebXML testing, and has provided useful feedback to IIC, in particular about their implementation plan and test harness requirements.

The Basic Interoperability Profile (BIP) is the result of this consulting, and is addressing a common set of interoperability requirements. This common set may not cover every interoperability feature that each

community requires, but it addresses the most essential ones, and is reasonably complete. We noticed that the test plans in the above industry initiatives included both interoperability tests and (some) conformance tests. The IIC approach is to clearly separate test suites for conformance, and test suites for interoperability. One reason the BIP has a smaller number of test requirements is that only the requirements relevant to interoperability have been kept. Other requirements relevant to conformance have been moved to the MS conformance test requirements. By doing so, the cost of operating an interoperability test suite is reduced, as conformance should normally be verified prior to interoperability, by a testing procedure that does not require coordination with other parties.

1.7 Concept of Operation

1.7.1 Driving the Tests

The MS interoperability test harness described in this document is based on the ebXML Test Framework [ebTestFramework], described in another document. This test harness is assumed for testing the Basic Interoperability Profile, and has been designed to achieve the following objectives:

- The MS Interoperability Test Suite can be run entirely and validated from one component of the framework, called the Test Driver. This means that all test outputs will be generated - and test conditions verified - by the Test Driver, even if the test harness involves several – possibly remote – components of the framework. Significant events occurring in such components are communicated back to the Test Driver.

The verification of each Test Case can be done at run-time by the Test Driver itself, as soon as the test case is completed. The report of the verification can be generated immediately as the Test Suite has been completed.

1.7.2 Interoperability vs. Conformance

Interoperability in no way guarantees conformance (conformance being defined as the adherence of a software implementation to a specification). Two implementations can be made to interoperate well with each other without necessarily adhering to the specification. It is expected that some level of conformance testing be done prior to interoperability testing. For example, the interoperability test does not verify or diagnose the following:

- Invalid SOAP header and message
- Invalid ebXML information in SOAP header and message
- CPA Error and Resolution
- Unrecognized service
- Duplicate messages
- Simple error handling

All the tests above are defined in the ebXML Messaging conformance test suite, and are to be passed prior to undergoing interoperability tests. If only from a logistic perspective, it is preferable to do as many verifications as possible during conformance testing, which typically involves a single message service handler (MSH), and is much easier to set-up than interoperability testing.

Before testing two MSH implementations for the MS Basic Interoperability Profile (BIP), it is strongly RECOMMENDED that each candidate MSH has passed initially the MS Conformance test suite (released by the ebXML IIC in another specification, [ebMSConfTestSuite]), since otherwise, some problems might be observed when testing for BIP, which in fact are caused by a lack of conformance to the ebMS specification.

Any MSH behavior that can be verified in a test harness that includes a single MSH (plus a test driver simulating another MSH) is relevant to conformance. Testing such behaviors should not belong to an interoperability test suite, but instead to a conformance test suite. MSH behaviors, which necessitate exchanges between two MSH's for verification, should be tested in interoperability mode. Because organizing interoperability tests (administration and logistics) is usually costly, only those tests that are essential to interoperability are included here.

1.7.3 Interoperability and Testing

Having passed a round of interoperability testing only ensures interoperability with other software implementations that have participated in that specific round of testing. There are two major reasons for this:

- Specific implementation options defined by a testing body or the participants may affect interoperability. For example, because there are different ways to implement digital signatures, this can cause a MSH to reject a message as invalid. Where possible, this documents makes recommendations on these implementation options.
- Interoperability is not transferable (or transitive). In other words, if MSH A interoperates with MSH B, and MSH B interoperates with MSH C, this does not guarantee that MSH A interoperates with MSH C (although there is a high probability that it will).

1.7.4 Asymmetric Testing

The basic interoperability test suite defined here is intended to be driven from one party (or node) of the network called the "driver party" (this is the party that communicates with the Test Driver). As it involves two parties, it is called a "binary" test suite.

The test suite is asymmetric. This means, when run between two parties A and B, the same test suite may produce different results when driven from A (driver party = A) than when driven from B (driver party = B). For example, a test case that requires a party to sign a message, and the other party to validate the signature, may succeed from A to B, and fail from B to A. This is because the test cases in this suite do not verify exactly the same capability on each side.

In order to achieve a well-rounded interoperability testing, a binary, asymmetric interoperability test suite is supposed to be run twice. At each run, a different party acts as the driver party.

1.7.5 Interoperability as a Contract between Applications and Messaging

The test suites described here – in their current version - are interoperability testing at the application level only, not at "wire" level. This means that the combination:

{ MSH1 + communication medium(transport) + MSH2 }

is treated as a black box. The test cases only verify that the contract Application1 – Application2 is satisfied. The test cases actually verify another contract as well: the contract between the two parties at each end-point (the applications), and the communication middleware that includes the two MSH and the transport they use. A failure of the test cases will simply indicate that the communication layer did not fulfill the expected service. The test cases do not intend to verify the particular output of one MSH or the

other, as this is relevant to conformance. For example, no “sniffing” on the wire is needed in order to process these test cases, as everything related to the internal behavior of an MSH, or message conformance at transport level, is supposed to have been verified by conformance testing.

For example, when verifying that a digital signature is:

(a) well inserted by the sender, when the CPA requires so, and

(b) that the recipient is able to validate it should not require monitoring the wire or the internal behavior of an MSH, during interoperability tests.

Testing for (a) should occur during conformance tests, which involve monitoring the “wire” for conformance of message elements such as a well-formed signature. As for recipient validation (b), only the effect of the “Service” behavior (application contract) will be checked: i.e. the received signed message is passed to the application, and no error is generated.

2 Harness for MS Interoperability Testing

2.1 Architecture

This section describes how to configure the Test Framework elements for interoperability testing between two implementations of the ebXML Messaging Service specification (2.0), identified here as party A and party B.

As mentioned above, interoperability testing will be asymmetric: one party – called the driver party – will drive the test cases, the other party – called the responder – will respond to messages initiated by the driver party. Two options for the interoperability test harness are described in Appendix A. This section will focus on the "point-to-point" test harness. With this test harness, the Test Suite will be controlled from the "driver" party, and does not necessarily verify the same capabilities on both sides i.e. is asymmetric). In order to get a full interoperability test between Party A and Party B, the test suite should be repeated after both parties have swapped the (driver/responder) roles.

The components of the framework that are involved in interoperability testing are:

On the driver party:

- An instance of the Test Driver component, coupled with an instance of a Test Service. This coupling consists of: (1) the ability for the Test Driver to trigger an action of the Test Service (typically, the Initiator action), (2) the ability for the Test Driver to be notified of actions triggered in the Test Service by received messages. In this configuration, the Test Driver is in "service" mode (see [ebTestFramework]). The driver party will process and initiate all test cases from the Test Driver.
- An instance of the Test Service component, which will directly interact with the driver party's MSH Service Interface. Note that the Test Driver does not need to interact directly with the MSH. In this configuration, the Test Service will operate in "reporting" mode. When installed on the same host, as suggested here, the reporting will be local: notifying the test driver of received messages is done via the "Receive" interface.

On the responder party:

- An instance of the Test Service component (same as in the driver party), which will support test actions invoked by messages received by the responder MSH. This Test Service instance will operate in "loop" mode.

373 Figure 1 illustrates a point-to-point test harness for MS interoperability testing.

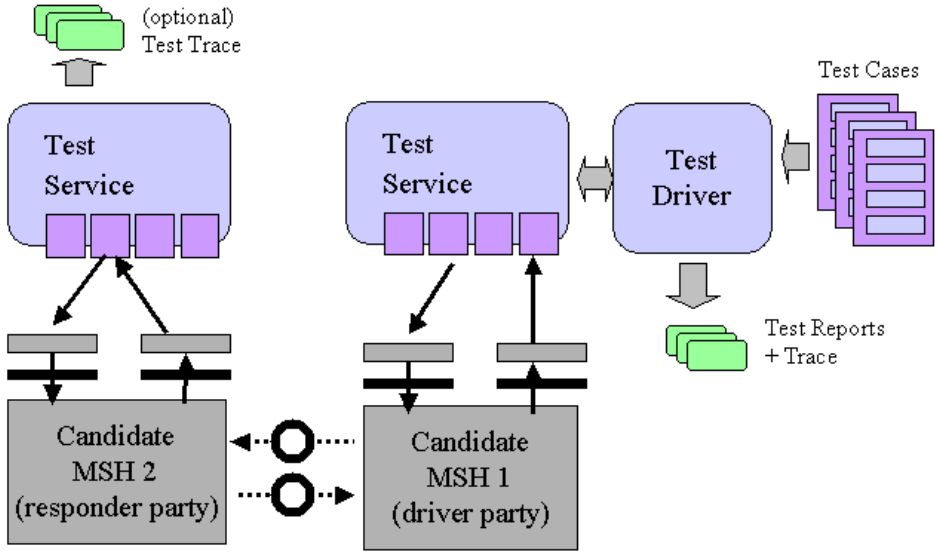


Fig 1. MS Interoperability Test Harness

20

374

375 The typical Interoperability test case procedure will consists of a sequence of test steps. The Test Driver
376 will control each of these steps. These steps will be:

377 • Sending messages – the content of which is specified in the test case – to some action of the
378 responder's Test Service.

379 • Receiving messages from the responder's Test Service.

380 • Analyzing the content of received messages, possibly in correlation with other message data,
381 received or sent during the same test case, in order to validate the requirement of the test case.

382 • Reporting on the test case outcome.

383 • Optionally (and prior to executing a test case), configure the MSH(s) for the message
384 conversation(s) that will be generated by the Test Case(s), with CPA data. Normally, the
385 installation of CPAs to be used for a test suite is supposed to be done prior to executing the test
386 suite. However, the Configurator action of a Test Service may be invoked – either locally by the
387 Test Driver on the driver party, or remotely by a message, with new CPA data. The expected
388 effect is the dynamic creation and installation of a new CPA, on the MSH associated with this
389 Test Service.

390 Appendix A illustrates how this test harness can be implemented.

391

2.2 The Test Service and its Actions

The Test Service name is: `urn:ebXML:iic:test`

A Test Case is described as a sequence of Test Steps. These Test Steps will consist of atomic operations executed by the components of the test Framework, e.g. sending a message, verifying a condition on a received message, etc. Most operations about messages are supported by the Test Service component, described in the Test Framework specification.

2.2.1 Test Service Actions

The standard test actions are completely described in the ebXML Test Framework specification [ebTestFramework]. They are:

- **Mute** action
- **Dummy** action
- **Reflector** action
- **Initiator** action
- **PayloadVerify** action
- **ErrorAppNotify** action
- **ErrorURLNotify** action
- **Configurator** action

3 The MS Basic Interoperability Profile Test Suite

3.1 Overview

In a nutshell, the MS-BIP is verifying:

- Various types of messages exchanged: no payloads, multiple payloads, different types of payloads.
- Asynchronous responses as well as Synchronous if the transport protocol allows for this, e.g. HTTP.
- All signals normally expected from an MSH (Acks and Errors). This ensures that other MSH will "understand" them properly. The "conformance" semantics of these signals has already been tested during conformance testing, e.g. they manifest as well-formed envelope elements, or they are generated when they should. When digital signatures are used, they must be properly understood and validated on each side, especially with various combinations and options that may affect interoperability (e.g., about key info, about signature of Ack signals.)

3.2 Options of the Basic Interoperability Profile

The ebXML MS basic interoperability profile (ebXML MS-BIP) provides users with options that must be specified, prior to testing. A primary set of options must be selected when testing such a profile. These options are:

- The transport protocol: The RECOMMENDED values are: HTTP/1.1 and SMTP.
- The canonization method (for digital signatures): The recommended value is: ([ebMS] section 4.1.3) "http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
- The signature algorithm: The recommended value is: "http://www.w3.org/2000/09/xmldsig#dsa-sha1"

When mentioning the MS Basic Interoperability Profile (e.g. when claiming the ability to interoperate according to the MS-BIP), the actual values chosen for these three options should always be mentioned. For example:

- Incorrect way to state an interoperability claim:
  - Partners A and B can interoperate according to the MS Basic Interoperability Profile.
- Correct way to state an interoperability claim:
  - Partners A and B can interoperate according to the MS Basic Interoperability Profile, with transport=" HTTP/1.1", canonization="http://www.w3.org/TR/2001/REC-xml-c14n-20010315", and sig-algorithm=" http://www.w3.org/2000/09/xmldsig#dsa-sha1".

These profile options define an interoperability space: if a set U1 of users can interoperate according to MS-BIP with a combination of option values, and another set U2 of users can interoperate according to MS-BIP with a different combination of option values, this does not tell anything about the ability of U1 and U2 to interoperate across them. In fact, it is very likely that different MSHs that are configured for different values of these options will not be able to interoperate.

3.3 Parameters of the Test Suite and of its Test Cases

The MS-BI test suite and its test cases have parameters that can be considered as parameters of the test harness.

Three of these parameters correspond to the MS-BIP options described in 3.2, and can be considered as *global parameters* of the MS-BIP test suite, i.e. they will characterize a particular instance of the MS-BIP test suite. The RECOMMENDED parameter notation and order, for precisely defining a particular instance of the MS-BIP test suite is: MS-BIP (<transport-protocol>, <canonicalization-method>, <signature-algorithm>)

Other parameters are used by the MS-BIP test suite, which are specific to each test case of the suite (i.e. they may change from one test case to the other.) All parameters are defined in the BIP testing parameter table, of which a sample is given below.

The recommended parameter values in the table below only reflect the most common - or expected - options, or those recommended by the Messaging specification [ebMS]. The table also reflects the recommended minimum set of parameters used for test execution. This representative set includes a subset of configuration options for an ebMS implementation and a subset of relevant attributes of a Collaboration Protocol Agreement (CPA) between the partners or endpoints. In addition, some parameters fall outside the scope of a CPA, but are nevertheless critical messaging features that must be set to correctly run a test or a test suite. The table contains a column with an XPath reference to the location within a CPA that a parameter refers (if it is defined in a CPA).

The set of instances of the parameter table, as needed by the set of test cases in the MS-BIP test suite, are reported in section 4.1.3.

This basic interoperability profile assumes symmetric configurations between partners, and therefore a symmetrically configured CPA.

BIP Testing Parameter Table

The parameters below identify the MSH configuration for a single Test Case, or a group of Test Cases. These parameters can be used to “profile” an MSH configuration under test, and provide a context for test reporting. In addition, such a set of parameters may (in a future versions of the ebXML Test Framework Specification) be used as metadata to “tag” conformance or interoperability tests, and permit filtering of test cases based upon these parameter values. Currently, these parameters serve only as a recommended MSH configuration context under which tests may be executed.

Name	Commonly Used Values	Equivalent CPA field(s) (using XPath notation)
Transport Protocol	HTTP 1.1 | SMTP	CollaborationProtocolAgreement/PartyInfo/Transport//TransportProtocol
Canonicalization Algorithm	“http://www.w3.org/TR/2001/REC-xml-c14n-20010315” (spec recommended)	N/A – explicitly defined in individual message declaration
Signature Algorithm	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec	CollaborationProtocolAgreement/PartyInfo/DocExchange//SenderNo

	recommended)	nRepudiation/SignatureAlgorithm
Signed Message	true|false	CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding//BusinessTransactionCharacteristics/isNonRepudiationRequired
Signed Acknowledgment	true|false	CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding//BusinessTransactionCharacteristics/isNonRepudiationReceiptRequired
Confidentiality (not required for BIP testing)	none | transient | persistent | transient-and-persistent	CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding//BusinessTransactionCharacteristics/isConfidential
Authentication (not required for BIP testing)	none | transient | persistent | transient-and-persistent	CollaborationProtocolAgreement/PartyInfo/CollaborationRole/ServiceBinding//BusinessTransactionCharacteristics/isAuthenticated
Retries	An integer value	CollaborationProtocolAgreement/PartyInfo/DocExchange//ReliableMessaging/Retries
RetryInterval	PT30S (a typical value)	CollaborationProtocolAgreement/PartyInfo/DocExchange//ReliableMessaging/RetryInterval
AckRequested	always | never | perMessage	CollaborationProtocolAgreement/PartyInfo/DeliveryChannel/AckRequested
PersistDuration	P10D (a typical value)	CollaborationProtocolAgreement/PartyInfo/DocExchange//ReliableMessaging/ReliableMessaging/PersistDuration
duplicateElimination	always | never | perMessage	CollaborationProtocolAgreement/PartyInfo/DeliveryChannel/MessagingCharacteristics/@duplicateElimination
MessageOrder Semantics	Guaranteed|NotGuaranteed	CollaborationProtocolAgreement/PartyInfo/DocExchange//ReliableMessaging/MessageOrderSemantics
HTTP Timeouts	PT5M (a typical value)	N/A – explicitly defined in Test Suite ConfigurationGroup XML
SyncReply (used to globally define all messages are sent witih a SyncReply element)	true|false	N/A – explicitly defined in Test Suite ConfigurationGroup XML
syncReplyMode	mshSignalsOnly | responseOnly | signalsAndResponse | signalsOnly | none	CollaborationProtocolAgreement/PartyInfo/DeliveryChannel/syncReplyMode
ErrorURL	URL of driver party MSH	CollaborationProtocolAgreement/PartyInfo/Transport/Endpoint/uri
NotifyURL	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)	N/A –explicitly defined in Test Suite ConfigurationGroup XML

3.4 MS-BIP Test Cases Specification

The following test cases are specified using test material described in the ebXML Test Framework specification. The test data used by these test cases (MSH settings, generated message headers, payloads, configuration) are described in section 4.

Some of the MSH settings can be set using a Collaboration Protocol Agreement (CPA). While this document does not provide specific CPA values, it does provide information on what these values should be. It is recommended that a full CPA be used to configure the MSH.

Each message in the test cases includes a Conversation ID; it is recommended that each test case have a unique Conversation ID (i.e. a new conversation be started for each test case execution). This will help test reporting, and also avoid possible run-time problems if messages of a test case get intertwined with messages of another test case, as message correlation within a test case is done based on the conversation ID.

3.4.1 Test Case 1.1: No payload basic exchange

Rationale:

The test case verifies that an incoming message is well received and triggers the correct action on Responder side. There is no check of the integrity of the received message, except its ability to trigger the **Dummy** action of the responder Test Service. A predefined response message (no payload) is generated by the Test Service of responder. There is no check on this message, except its ability to trigger the **Mute** action of the driver Test Service, which will record the reception.

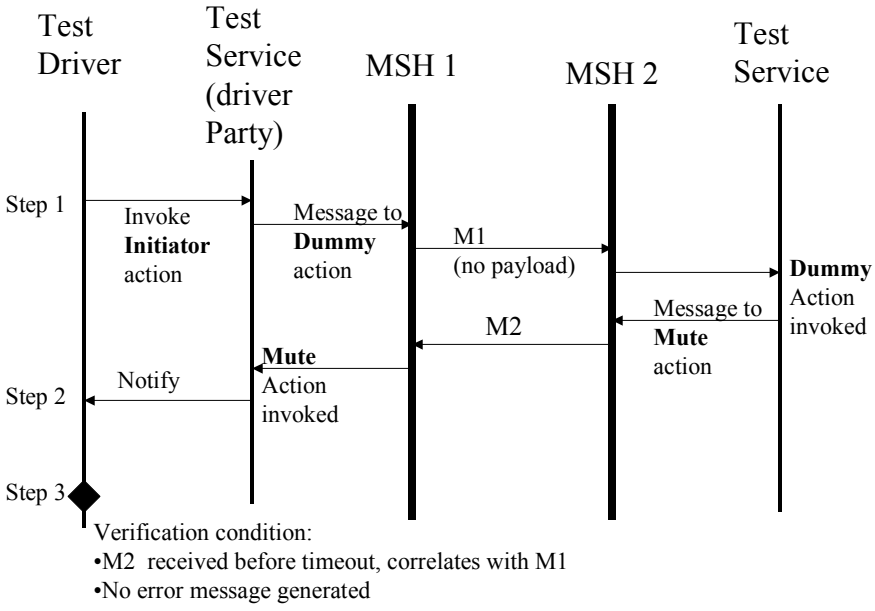
Test Data Material:

- MSH-configuration: mshc_1
- Message Payloads: none
- Message Header default: mhdr_0

Test Steps:

1. Test Driver (driver party) sends a sample message M1 to the **Dummy** action of the Test Service of the responder party. This is done by invoking the Initiator action of the driver party Test Service.
2. Test Driver (driver party) receives within time limit a response message M2 via the **Mute** action of its local Test Service M2 is generated by the Dummy action of Responder). Correlation: (M2.CPAId= M1.CPAId) and (M2.ConversationId = M1.ConversationId) and M2.Action = “Mute”.
3. Verification. Test Case succeeds if: (Step 2 successful within time limit)

Fig 2. Diagram for Test Case 1.1



3.4.2 Test Case 1.2: Basic exchange with one payload

Rationale:

The test case verifies that an incoming message is well received, triggers the right action on Responder side, and passes its payload to application (**Reflector** action of Test Service). A response message is generated by the Test Service of responder (Reflector action), sending back the same message - except for expected changes in header - with same payload. The received message triggers the **Mute** action of the driver Test Service, which will record the reception. The received payload is compared with the payload initially sent.

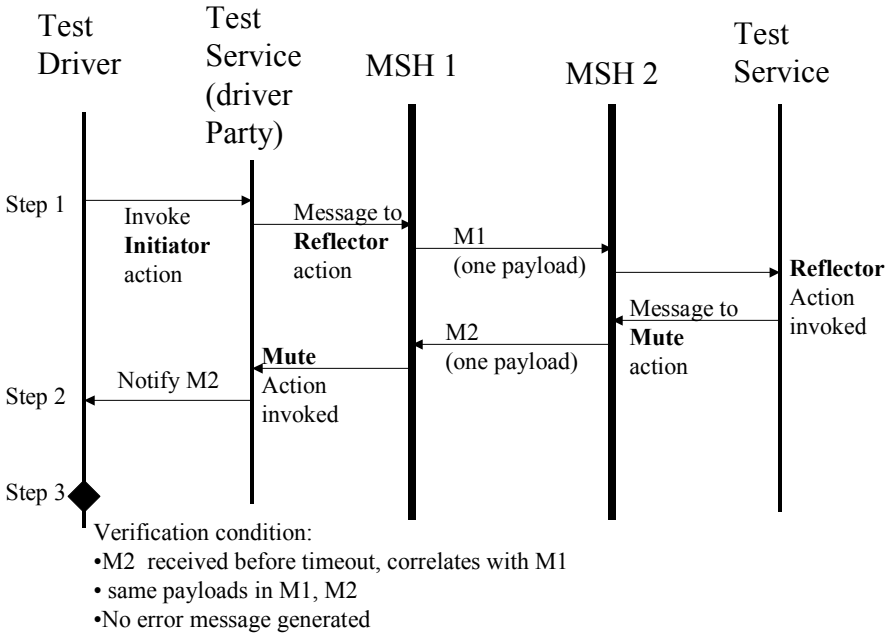
Test Data Material:

- MSH-configuration: mshc_1
- Message Payloads: payload_1
- Message Header default: mhdr_1

Test Steps:

1. Test Driver (driver party) sends a sample message M1 to the **Reflector** action of the Test Service of the responder party.
2. Test Driver (driver party) receives within time limit a response message M2 via the **Mute** action of its local Test Service (M2 is generated from the Reflector action of Responder). Correlation: (M2.CPAId= M1.CPAId) and (M2.ConversationId = M1.ConversationId) and M2.Action = “Mute”.
3. Verification. Test Case succeeds if: (Step 2 successful) AND (M2.payload = M1.payload)

Fig 3. Diagram for Test Case 1.2



3.4.3 Test Case 1.3: Basic exchange with three payloads

Rationale:

The test case verifies that an incoming message with multiple payloads of different types (two XML, one binary) is well received, triggers the correct action on Responder side, and passes its payload to the application (**Reflector** action of Test Service). A response message is generated by the Reflector action of the responder Test Service, sending back the same message - except for expected changes in the header - with same payloads. The received message triggers the **Mute** action of the driver Test Service, which will record the reception. The received payloads are compared with the initially sent payloads

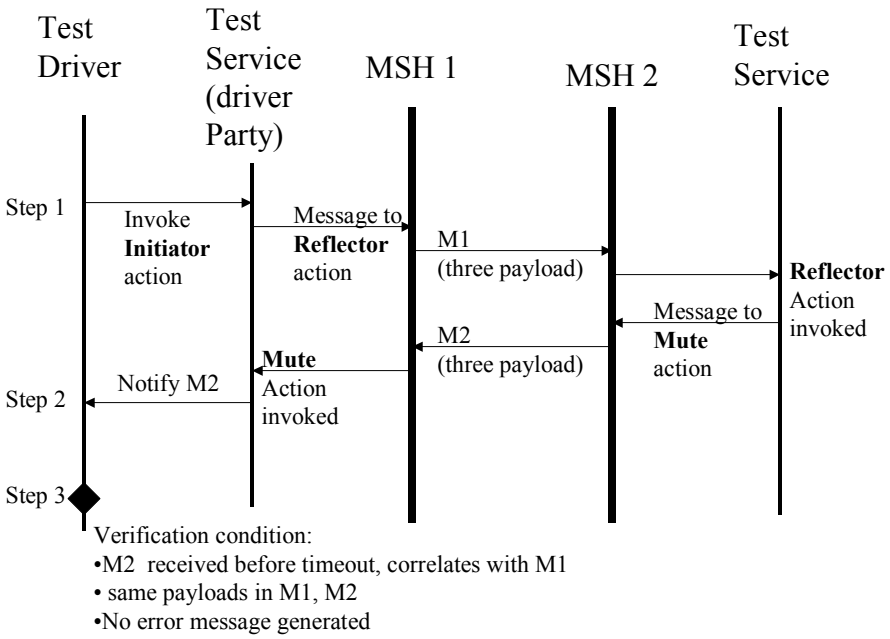
Test Data Material:

- MSH-configuration: mshc_1
- Message Header default: mhdr_3
- Message Payloads: payload_1, payload_2, payload_3

Test Steps:

1. Test Driver (driver party) sends a sample message M1 to the **Reflector** action of the Test Service of the responder party.
2. Test Driver (driver party) receives within time limit a response message M2 via the **Mute** action of its local Test Service (generated from the Reflector action of the Responder). Correlation: (M2.CPAId= M1.CPAId) and (M2.ConversationId = M1.ConversationId) and M2.Action = “Mute”.
3. Verification. Test Case succeeds if: (Step 2 successful) AND (M2.payload1 = M1.payload1) AND (M2.payload2 = M1.payload2) AND (M2.payload3 = M1.payload3)

Fig 4. Diagram for Test Case 1.3



3.4.4 Test Case 1.4: Basic exchange with Error message

Rationale:

The test case verifies that error messages are well received by the driver party. The driver party should provide its URL as ErrorURL, as mandated by the CPA “mshc_1”. The test does not cover that errors are generated with the right code: that is done by conformance tests. An erroneous message is sent to a non-existent action of the responder Test Service. The responder MSH will send back an Error, which should be notified to the sender (driver party) via its **ErrorURLNotify** action, which will record the reception.

Test Data Material:

- MSH-configuration: mshc_1
- Message Header default: mhdr_1
- Message Payloads: payload_1

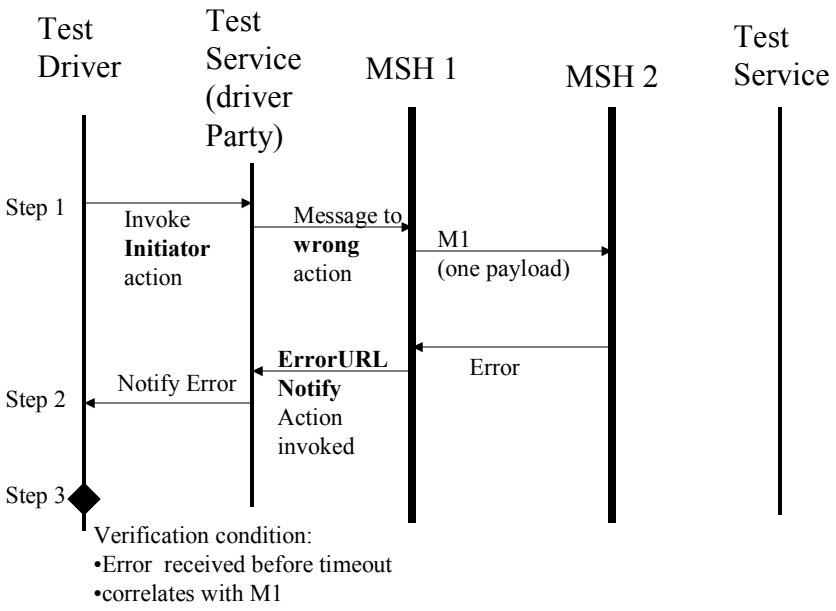
Test Steps:

1. Test Driver (driver party) sends a sample message M1 to the unresolvable action of the Test Service of the responder party. In the message header, the Service/Action fields are set to non-existantService/Action values.
    - Header modified: mhdr_1’ <here, introduce the error by modifying header Service/Action in default mhdr_1>. It is recommended to use the erroneous Action value: “non-existent-action”, with the correct Service value for the Test Service.
2. Test Driver (driver party) receives within time limit an error message M2 via the **ErrorURLNotify** action of its local Test Service. Correlation: (M2.CPAId= M1.CPAId) and M2.Action = “ErrorURLNotify”.

NOTE: the only reliable way to correlate an error message to its cause, is based on RefToMessageId, which is communicated to the recipient (here the Test Driver). The correlation: (M2.RefToMessageId = M1.MessageId) should then be used. However, this correlation assumes that the Test Driver, as sender of the error-causing message (M1) knows the MessageId generated by the MSH. This is not always easy to obtain and depends on the implementation: the MessageId may not be returned to the Test Driver, and instead be reported in a log that needs to be accessed separately, e.g. browsed by the user, for doing this correlation. Because of this, automating this test cannot always rely on RefToMessageId. In case the participant MSHs can return MessageIds, then (M2.RefToMessageId = M1.MessageId) should be used.

3. Verification. Test Case succeeds if: (Step 2 successful)

Fig 5. Diagram for Test Case 1.4



3.4.5 Test Case 1.5: Simple Signed Exchange Using Certificate

Rationale:

The test case verifies message exchange with digital signature (without key info). The key info is NOT embedded in the message. It is available on recipient side from a certificate. This test case exercises the ability to resolve the key info based on the right certificate. It is not essential for the response to be signed, although the CPA setting will require so for the convenience of having similar configurations on each party (the ability to sign messages from the other party, will be tested when running the same test case from the other party, as the test suite is asymmetric, see Section 1).

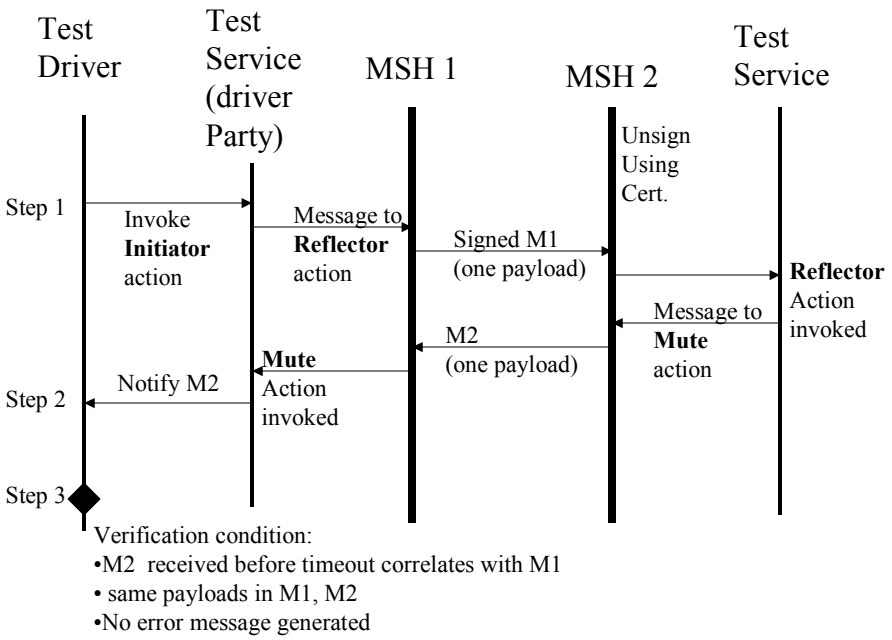
Test Data Material:

- MSH-configuration: mshc_4
- Message Payloads: payload_1
- Message Header default: mhdr_1

Test Steps:

1. "Initiator" on driver side sends signed message to Reflector action of recipient. The entire message is signed.
2. "Mute" action on driver side receives (unsigned) notification message from Reflector, with the same payload.
3. Verification: (payloads are same) and (no error message received)

Fig 6. Diagram for Test Case 1.5



3.4.6 Test Case 1.6: Synchronous Basic Exchange with one payload

Rationale:

This is the synchronized version of Test Case 1.2 (SyncReply element is present in sent message). The CPA used will have SyncReplyMode set to "signalsAndResponse".This test case is for synchronous transport only (test suite parameter: < transport-protocol >).

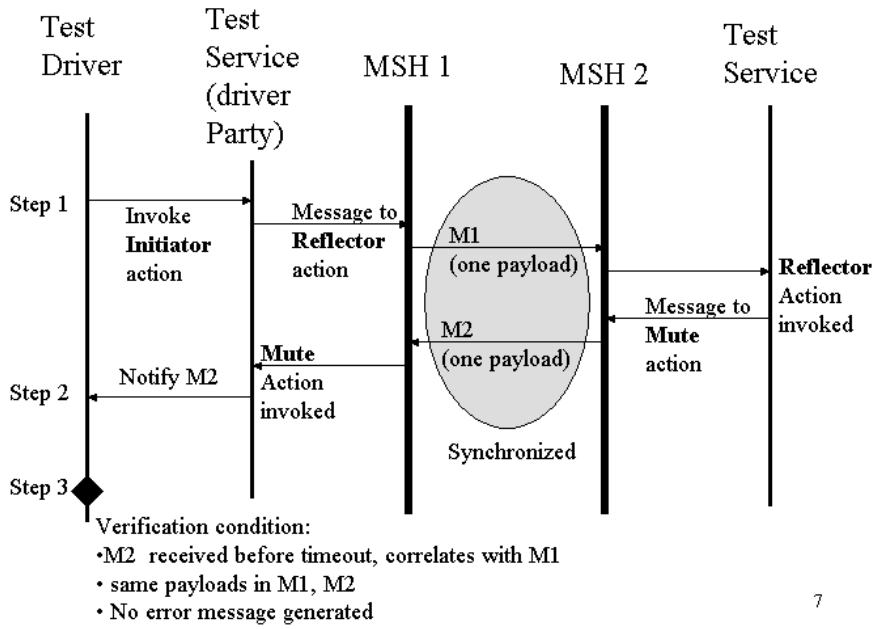
Test Data Material:

- MSH-configuration: mshc_5
- Message Payloads: payload_1
- Message Header default: mhdr_1

Test Steps:

1. Test Driver (driver party) sends a sample message M1 to the **Reflector** action of the Test Service of the responder party.
2. Test Driver (driver party) receives within time limit a response message M2 via the **Mute** action of its local Test Service (from the Reflector action of Responder). Correlation: (M2.CPAId= M1.CPAId) and (M2.ConversationId = M1.ConversationId) and M2.Action = "Mute".
3. Verification. Test Case succeeds if: (Step 2 successful) AND (M2.payload = M1.payload)

Fig 7. Diagram for test Case 1.6



644

645 3.4.7 Test Case 1.7: Acknowledgment exchange: Unsigned Data, Unsigned 646 Ack

647 Rationale:

648 Test the ability of two MSHs to exchange and understand each other's ack signals.

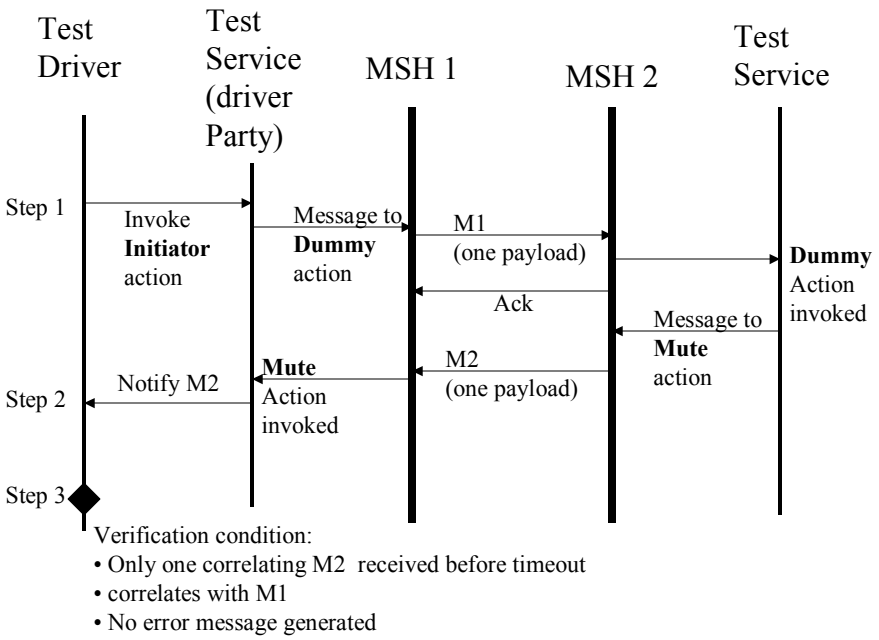
649 Test Data Material:

650 • MSH-configuration: mshc_1
651 • Message Payloads: payload_1
652 • Message Header default: mhdr_1 (add Acknowledge element)
653

654 Test Steps:

655 1. "Initiator" on driver side sends unsigned message to Dummy action of recipient, with AckRequested
656 element.
657 2. "Mute" action on driver side receives a single (unsigned) response message from Dummy. NOTE: in case
658 Ack is not received or understood, driver MSH will resend message of step 1, and several responses from
659 Dummy will be observed.
660 3. Verification: within a time period equal or greater than (Retries + 1) * RetryInterval from (step 1): (exactly
661 ONE response message from Dummy is received in Step 2)and (no error message received)
662

Fig 8. Diagram for Test Case 1.7 (pass)



Because Acknowledgements are MSH-level signals, it is not possible to observe them from the application side. However, the objective of this test is not to verify the proper generation of well-formed Ack signals: this must have previously been verified using conformance tests.

The objective of this test only consists of verifying that Acks generated by an MSH are well interpreted by the other MSH implementation. Two failure cases may be observed by the test driver

- Two or more response messages (M2), (with different message Ids), are received by the test driver, within a time period equal or greater than (Retries + 1) * RetryInterval. This means that the receiver party (Test Service, "Dummy" action) has responded several times to as many incoming messages (M1). The reason why M1 was resent several times, is that the Ack from the receiver party has either not been received, or not been understood by the driver party. This situation is illustrated in figure 9 below.
- No response (M2) is received, within the time period equal or greater than (Retries + 1) * RetryInterval. This however does not imply anything on the interoperability of Ack messages. Rather, it reveals another type of failure, e.g., the initial message (M1) has not been received by the receiver party, or (2) the response message (M2) has not been received by the driver party.

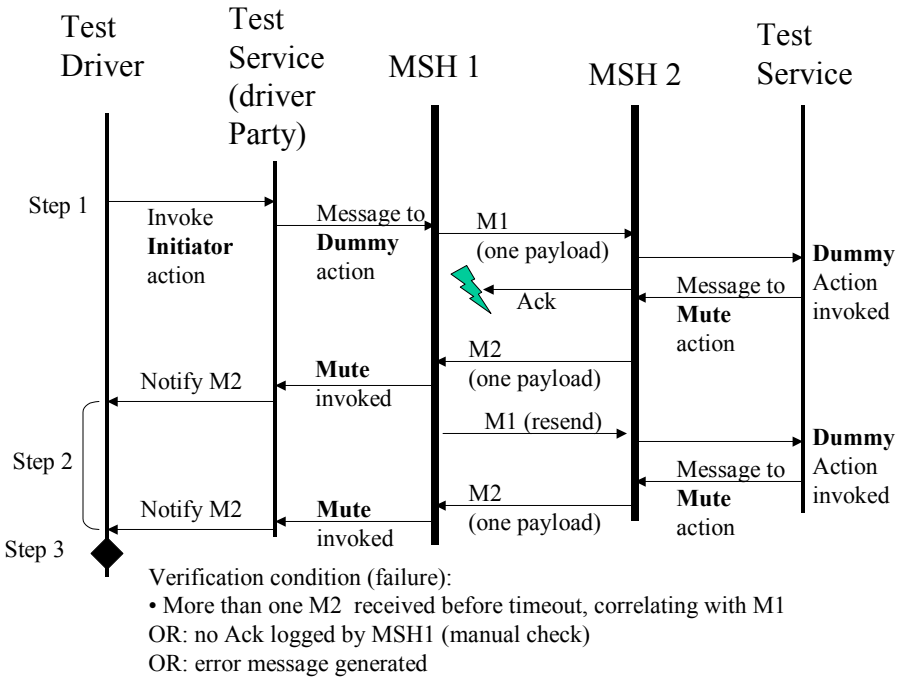
However, even if one and only one response message M2 is received by the sender, it is not possible to infer that the test case successfully demonstrated Ack interoperability, only by observing the events occurring in the test driver. The following failures will still result in a single response message to the test driver:

- The sender retry mechanism is not working properly, so no multiple invocations of the Dummy action on receiver side will occur – only the initial invocation (message M1). In that case, a single response will be observed on sender side, which is also the observed effect in case of successful verification. Therefore, the only way to detect such a failure is to "manually" access the log of the MSH to ensure the Ack was well received by the driver party. It must be noted that this case should be considered as exceptional, since the ability to resend is supposed to have been checked by conformance testing.

- The Ack was not well received by the driver party, but in addition, the retry mechanism did not work well, so no resending occurred. Consequently, a single response M2 was received by the test driver.

In order to confirm a successful outcome of this test case, a "manual" check of the message log in the driver party MSH is required in order to reveal the presence of a received Ack.

Fig 9. Diagram for Test Case 1.7 (failure)



3.4.8 Test Case 1.8: Acknowledgment exchange: Signed Data, Signed Ack

Rationale:

Test the ability of two MSHs to exchange and understand each other's signed ack signals (for non-repudiation), while the business messages are signed.

Test Data Material:

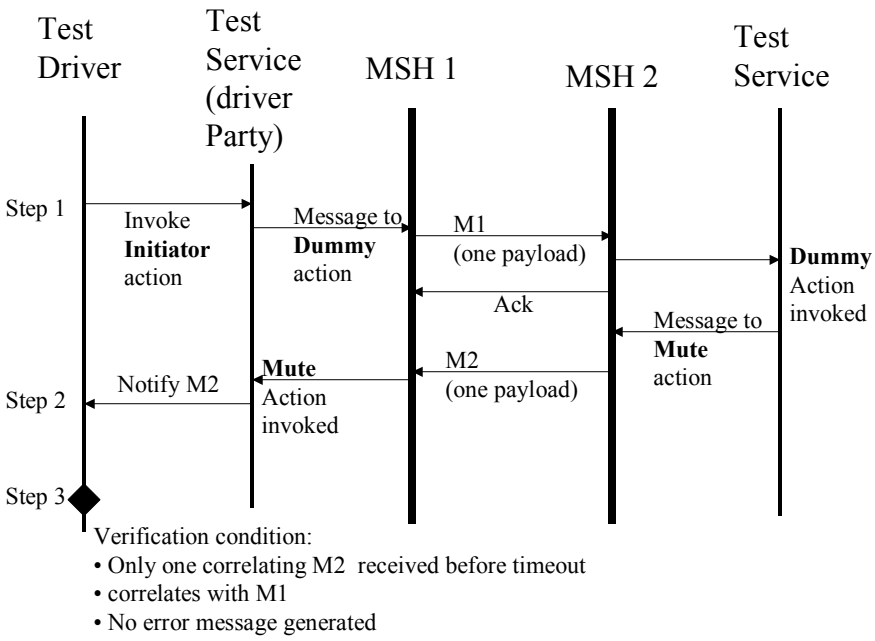
- MSH-configuration: mshc_2
- Message Payloads: payload_1
- Message Header default: mhdr_1 (add Acknowledge element)

Test Steps:

1. "Initiator" on driver side sends a signed message to Dummy action of recipient, with AckRequested element.
2. "Mute" action on driver side receives a single (unsigned) response message from Dummy. NOTE: in case Ack is not received or understood, driver MSH will resend message of step 1, and several responses from Dummy will be observed.
3. Verification: within a time period equal or greater than (Retries + 1) * RetryInterval, from (step 1): (exactly ONE response message from Dummy is received in Step 2) and (no error message received)

717

Fig 10. Diagram for Test Case 1.8



718

719 ## 3.4.9 Test Case 1.9: Synchronous Unsigned Acknowledgment exchange

720 Rationale:

721 Test the ability of two MSHs to exchange and understand each other's ack signals, in a synchronous set-
722 up. The CPA will have SyncReplyMode set to "mshSignalsOnly", so there is not overlap with Test Case
723 1.7. This is a fairly common case where the HTTP connection is not kept open for business messages
724 (for which response time may be long), but is kept open for MSH signals, for efficiency purpose. So the
725 Ack is immediately sent back on the same connection as the message.

726 Notes:

727 • The actual ability of each party to send Acks (e.g. on a same HTTP connection), based on CPA
728 requirement, is assumed to be to be previously tested by conformance tests. Only the
729 interoperability aspect of it is tested here.

730 • This test case is only to be used with a synchronous transport protocol (test suite parameter: <
731 transport-protocol >).

732

733 Test Data Material:

734 • MSH-configuration: mshc_3

735 • Message Payloads: payload_1

736 • Message Header default: mhdr_1 (add Acknowledge element)

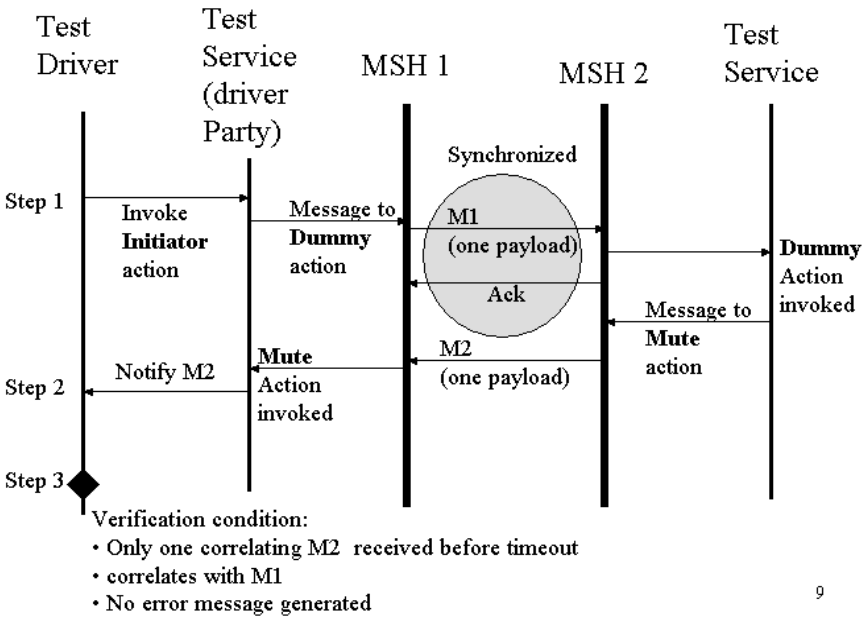
737

738 Test Steps:

739 1. "Initiator" on driver side sends unsigned message to Dummy action of recipient, with AckRequested
740 element.

2. “Mute” action on driver side receives a single (unsigned) response message from Dummy. NOTE: in case Ack is not received or understood, driver MSH will resend message of step 1, and several responses from Dummy will be observed.
3. Verification: (exactly ONE response message from Dummy is received in Step 2) and (no error message received)

Fig 11. Diagram for test Case 1.9



3.5 Two Instances of the Basic Interoperability Profiles and related Test Suites

3.5.1 The HTTP/1.1 Basic Interoperability Profile

The test suite, MS-BIP(“HTTP/1.1”), verifies the Basic Interoperability Profile for messaging over HTTP/1.1. It includes synchronous and asynchronous test cases (a total of 9) which exercise the capabilities of HTTP/1.1. The Test Cases are:

- Test Case 1.1: No payload basic exchange over HTTP/1.1.
- Test Case 1.2: Basic exchange with one payload over HTTP/1.1.
- Test Case 1.3: Basic exchange with three payloads over HTTP/1.1.
- Test Case 1.4: Basic exchange with Error message over HTTP/1.1.
- Test Case 1.5: Signed Message Without Embedded Key Info over HTTP/1.1.
- Test Case 1.6: Synchronous Basic Exchange with one payload over HTTP/1.1.
- Test Case 1.7: Acknowledgment exchange: Unsigned Data, Unsigned Ack over HTTP/1.1.
- Test Case 1.8: Acknowledgment exchange: Signed Data, Signed Ack over HTTP/1.1.
- Test Case 1.9: Synchronous Unsigned Acknowledgment exchange over HTTP/1.1.

764

765 3.5.2 The SMTP Basic Interoperability Profile

766 The test suite, MS-BIP (“SMTP”), verifies the Basic Interoperability Profile for messaging over SMTP. It
767 includes only asynchronous test cases (a total of 7), which exercise the capabilities of SMTP. The Test
768 Cases are:

769 • Test Case 1.1: No payload basic exchange over SMTP.

770 • Test Case 1.2: Basic exchange with one payload over SMTP.

771 • Test Case 1.3: Basic exchange with three payloads over SMTP.

772 • Test Case 1.4: Basic exchange with Error message over SMTP.

773 • Test Case 1.5: Signed Message Without Embedded Key Info over SMTP.

774 • Test Case 1.7: Acknowledgment exchange: Unsigned Data, Unsigned Ack over SMTP.

775 • Test Case 1.8: Acknowledgment exchange: Signed Data, Signed Ack over SMTP.

776

4 Details of Test Material

4.1 Configuration of the Test Harness and MSH Implementation

4.1.1 Test Harness and MSH Settings

As described in **[ebTestFramework]**, Test Harness and MSH settings are defined through either:

- Explicit declaration of MSH parameters in a Test Suite ConfigurationGroup declaration

MSH configuration through CPA (or CPA-like) methods

Explicit declaration of message content value in message declarations

4.1.2 Test-specific MSH Configuration Parameters

The table below contains the recommended and required MSH configuration parameters defined for the BIP Test Suite. The configuration groups are identified using the corresponding CPAId specified in individual Test Cases in the Test Suite.

Required (bold/highlighted) and Recommended Parameter Values for all test MSH configurations

Parameter Name	**mshc_1**	**mshc_2**	**mshc_3**	**mshc_4**	**mshc_5**
Transport Protocol	HTTP 1.1 or SMTP	HTTP 1.1 or SMTP	HTTP 1.1 or SMTP	HTTP 1.1 or SMTP	HTTP 1.1 or SMTP
Canonicalization Algorithm	"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" (spec recommended)	"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" (spec recommended)	"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" (spec recommended)	"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" (spec recommended)	"http://www.w3.org/TR/2001/REC-xml-c14n-20010315" (spec recommended)
Signature Algorithm	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec recommended)	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec recommended)	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec recommended)	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec recommended)	http://www.w3.org/2000/09/xmldsig#dsa-sha1 (spec recommended)
Signed Message	false	**true**	false	**true**	false
Signed Acknowledgment	false	**true**	false	false	false
Confidentiality (not required for BIP testing)	none	none	none	none	none
Authentication (not required for BIP testing)	none	none	none	none	none
Retries	3	3	3	3	3
RetryInterval	PT30S	PT30S	PT30S	PT30S	PT30S

AckRequested	perMessage	perMessage	perMessage	perMessage	perMessage
PersistDuration	P10D	P10D	P10D	P10D	P10D
duplicateElimination	perMessage	perMessage	perMessage	perMessage	perMessage
MessageOrder Semantics	NotGuaranteed	NotGuaranteed	NotGuaranteed	NotGuaranteed	NotGuaranteed
HTTP Timeouts	PT5M (if HTTP)	PT5M (if HTTP)	PT5M (if HTTP)	PT5M (if HTTP)	PT5M (if HTTP)
SyncReply (used to globally define all messages are sent witih a SyncReply element)	false (if HTTP)	false (if HTTP)	true (if HTTP)	false (if HTTP)	false (if HTTP)
syncReplyMode	none	none	**mshSignalsOnly**	none	**signalsAndResponse**
ErrorURL	URL of driver party MSH	URL of driver party MSH	URL of driver party MSH	URL of driver party MSH	URL of driver party MSH
NotifyURL	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)	URL of the Test Driver (in a hub configuration), or to the driver party MSH (in point-to-point config)

4.1.3 Generated Message Headers

The ebXML Message Headers below are dynamically generated by the Test Harness, using the declarative message syntax described in **[ebTestFramework]**. Key message content value is supplied by the Test Harness, either through configuration parameters or through interpretation of the values provided in the message declaration itself.

4.1.4 Key Message Parameters

The default values for these run-time parameters should be set in the test suite ConfigurationGroup element when the test suite XML file is deployed:

$SenderParty (set to the Test Driver MSH host)

$ReceiverParty (set to the remote MSH host)

The values of the parameters below must be set (either by the Test Harness or through explicit declaration in a message) for each test case:

$CPA

$ConversationId

The value of this parameter may vary (in the MessageDeclaration element) for each test step:

$Action

The value of these parameters is not under control of the Test Driver, and will be set by the MSH implementation at run-time:

$MessageId
$TimeStamp

4.1.5 Sample Headers

4.1.5.1 mhdr_0

This sample header is constructed for messages with no payload. The parameters will be instantiated by the Test Driver or the MSH implementation.

```
<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> $SenderParty</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>$ReceiverParty </eb:PartyId>
    </eb:To>
    <eb:CPAId>$CPA </eb:CPAId>
    <eb:ConversationId> $ConversationId</eb:ConversationId>
    <eb:Service> urn:ebXML:iic:test</eb:Service>
    <eb:Action>$Action </eb:Action>
    <eb:MessageData>
      <eb:MessageId>$MessageId </eb:MessageId>
      <eb:Timestamp>$Timestamp </eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body>
</SOAP:Body>
</SOAP:Envelope>
```

4.1.5.2 mhdr_1

This sample header is constructed for messages with one payload, before instantiation of parameters.

```
<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>$SenderParty_</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>$ReceiverParty_</eb:PartyId>
    </eb:To>
    <eb:CPAId>$CPA_</eb:CPAId>
    <eb:ConversationId>$ConversationId_</eb:ConversationId>
    <eb:Service> urn:ebXML:iic:test</eb:Service>
    <eb:Action>$Action_</eb:Action>
    <eb:MessageData>
      <eb:MessageId>$MessageId_</eb:MessageId>
      <eb:Timestamp>$Timestamp_</eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body>
  <eb:Manifest eb:version="2.0">
    <eb:Reference xlink:href="cid:_payload_1"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP:Body>
</SOAP:Envelope>
```

4.1.5.3 mhdr_2

This sample header is constructed for messages with two payloads, before instantiation of parameters.

```
<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>$SenderParty_</eb:PartyId>
    </eb:From>
    <eb:To>
      <eb:PartyId>$ReceiverParty_</eb:PartyId>
    </eb:To>
    <eb:CPAId>$CPA_</eb:CPAId>
    <eb:ConversationId>$ConversationId_</eb:ConversationId>
    <eb:Service> urn:ebXML:iic:test</eb:Service>
    <eb:Action>$Action_</eb:Action>
    <eb:MessageData>
      <eb:MessageId>$MessageId_</eb:MessageId>
      <eb:Timestamp>$Timestamp_</eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body>
  <eb:Manifest eb:version="2.0">
    <eb:Reference xlink:href="cid:payload_1_"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
    </eb:Reference>
    <eb:Reference xlink:href="cid:payload_2_"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">CPPA</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP:Body>
</SOAP:Envelope>
```

4.1.5.4 mhdr_3

This sample header is constructed for messages with three payloads, before instantiation of parameters.

```
<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>$SenderParty_</eb:PartyId>
    </eb:From>
    <eb:To>
```

```
      <eb:PartyId>$ReceiverParty_</eb:PartyId>
    </eb:To>
    <eb:CPAId>$CPA_</eb:CPAId>
    <eb:ConversationId>$ConversationId_</eb:ConversationId>
    <eb:Service> urn:ebXML:iic:test</eb:Service>
    <eb:Action>$Action_</eb:Action>
    <eb:MessageData>
      <eb:MessageId>$MessageId_</eb:MessageId>
      <eb:Timestamp>$Timestamp_</eb:Timestamp>
    </eb:MessageData>
  </eb:MessageHeader>
</SOAP:Header>
<SOAP:Body>
  <eb:Manifest eb:version="2.0">
    <eb:Reference xlink:href="cid:payload_1_"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">Purchase Order 1</eb:Description>
    </eb:Reference>
    <eb:Reference xlink:href="cid:payload_2_2"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">CPPA</eb:Description>
    </eb:Reference>
  <eb:Reference xlink:href="cid:payload_3_"
         xlink:role="XLinkRole"   xlink:type="simple">
        <eb:Description xml:lang="en-US">Binary Document</eb:Description>
    </eb:Reference>
  </eb:Manifest>
</SOAP:Body>
</SOAP:Envelope>
```

4.1.6 Message Payloads

Message payloads for the BIP Test Suite are supplied in the normative BIP Test Suite described in section 4.3. There are three payloads used for testing in this test suite. They include:

4.1.6.1 Payload_1

Payload_1 is representative of a “small XML payload”. This payload is embedded in the Test Suite and is included in the message using an ID reference. The code for this payload is:

```
  <purchase_order>
  <po_number>1</po_number>
  <part_number>123</part_number>
  <price_currency=”USD”>500.00</price>
</purchase_order>
```

4.1.6.2 Payload_2

This payload represents an “average size” (22KB) XML business document. This payload is included in the test message through a file reference. The XML code used for this payload is the OASIS ebXML CPP/A example 2.0b on the OASIS CPPA Technical Committee web page.

4.1.6.3 Payload_3

This payload represents a “large” (1.236MB) binary document payload. This Test Suite uses the OASIS/ebXML Messaging Services Specification V2.0 document, available on the OASIS ebXML MS Technical Committee web page to represent a large binary ebXML message payload

4.2 Non-normative Basic Interoperability Profile Test Requirements

The table below defines the testing requirement for the ebXML MS V2.0 Basic Interoperability Profile. These data values map to the test requirements schema defined in [ebTestFramework] and its semantic test requirement model. The XML version of the test requirements, conforming to the schema defined in the ebXML Test Framework Specification, can be found in [ebMSInteropReqs].

ID	Name	Specification Ref	Precondition	Requirement Level	Assertion
req_id_1	BasicInteroperabilityProfileTests	ebMSBIP# 3.3			
funreq_id_1	CorrectMessageHeaderNoPayload	ebMSBIP# 3.3.1	(After receing a message with no payload addressed to the test service Dummy action,)	REQUIRED	The candidate Test Service returns a response message that correlates with the sent message based on CPAId, ConversationId and contains a “Mute” Action name. .
funreq_id_2	ValidOnePayloadMessage	ebMSBIP# 3.3.2	(After receing a message with one payload addressed to the test service Reflector action.)	REQUIRED	The response message correlates with the sent message based on CPAId, ConversationId and a “Mute” Action name, and the received payload is identical to the sent payload.
funreq_id_3	ValidateThreePayloadMessage	ebMSBIP# 3.3.3	**(**After receing a message with one payload addressed to the test service Reflector action**)**	REQUIRED	The response message correlates with the sent message based on CPAId, ConversationId and a “Mute” Action name, and the received payloads are identical to the sent payloads.
funreq_id_4	ReportBasicError	ebMSBIP# 3.3.4	(For a received response message, after sending a message with an unresolvable Service/Action element value)	REQUIRED	The response message contains an error message, directed to the the ErrorURLNotify action, and reports the CPAId, ConversationId and Action name of the erroneous message in the message

					payload.
funreq_id_5	VerifyMessageSignature	ebMSBIP# 3.3.5	(For a received response message, after sending a signed message with one payload to the Reflector action)	REQUIRED	The response message correlates with the sent message based on CPAId, ConversationId and “Mute” Action name, and the received payload is identical to the sent payload.(this means the certificate for that message has been resolved and the signature verified.)
funreq_id_6	SyncMessageOnePayload	ebMSBIP# 3.3.6	(For a received synchronous response message, after sending a synchronous unsigned message with one payload to the Reflector action and CPA syncReplyMode is set to “signalsAndResponse”)	REQUIRED	The response message correlates with the sent message based on CPAId, ConversationId and “Mute” Action name, and the received payload is identical to the sent payload
funreq_id_7	UnsignedMessageUnsignedAck	ebMSBIP# 3.3.7	(For all received response messages, after sending an unsigned , asynchronous request message with one payload to the Dummy action, with an AckRequested element AND the AckRequested "signed" attribute is set to "false" AND CPA "isNonRepudiationReceipt Required" is set to "false" AND CPA "isNonRepudiationRequire d" is set to "false")	REQUIRED	There is only one response message that correlates with the sent message based on CPAId, ConversationId and Action name, and this response has triggered the Mute action, and the received payload is identical to the sent payload
funreq_id_8	SignedMessageSignedAck	ebMSBIP# 3.3.8	(For all received response messages, after sending a signed , asynchronous message with one payload to the Dummy action, with an AckRequested element AND the AckRequested "signed" attribute is set to "true" AND CPA "isNonRepudiationReceipt Required" is set to "true" AND CPA "isNonRepudiationRequire d" is set to "true"))	REQUIRED	There is only one response message that correlates with the sent message based on CPAId, ConversationId and Action name, and this response has triggered the Mute action, and the received payload is identical to the sent payload
funreq_id_9	SyncUnsignedAck	ebMSBIP# 3.3.9	(For all received response messages, after sending a synchronous request message to the Dummy action with an unsigned AckRequested element AND CPA syncReplyMode is set to "mshSignalsOnly")	REQUIRED	There is only one response message that correlates with the sent message based on CPAId, ConversationId and Action name, and this response has triggered the Mute action, and the received payload is identical to the sent payload

1017

4.3 Normative ebXML MS Basic Interoperability Profile Executable Test Suite

[ebMSInteropTests] is an XML document containing the executable ebXML MS V2.0 Interoperability Test Suite. The XML document consists of a “bootstrap” ConfigurationGroup data, Test Case, Test Step and Test Operation XML content that provides the necessary information for the execution of the Test Suite by the Test Driver. The syntax and semantics of this Test Suite are described in detail in the [ebTestFramework].

Appendix A Implementations of the Test Harness

Two variants of the test harness described in Section 2 are described below.

A.1 The "Point-to-point" Test Harness Implementation

This configuration (Figure 12) is appropriate when two parties engage in interoperability testing without any third-party assistance, Each party will in turn play the driver party, and operate the Test Driver (install test cases, drive the executions, generate the reports.)

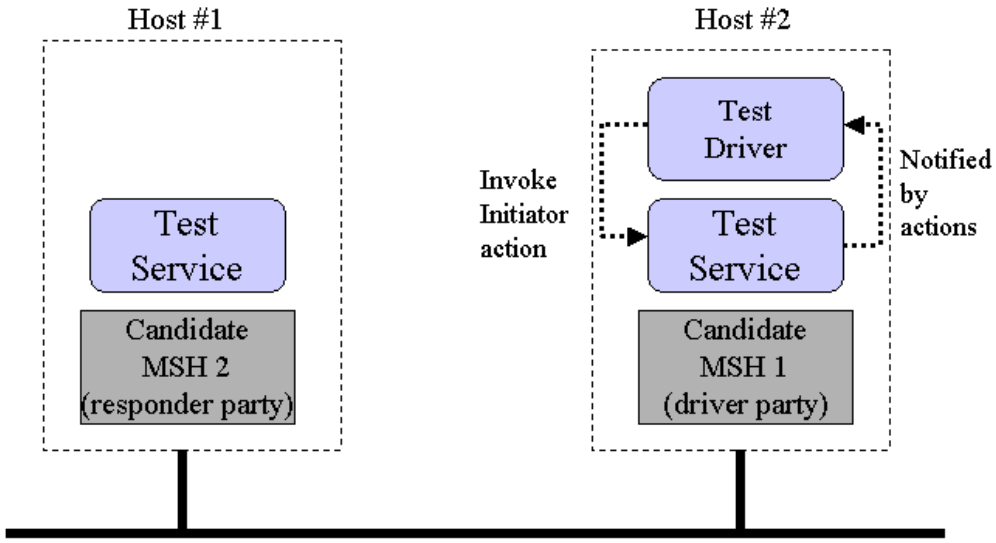


Fig 12. Point-to-point implementation

23

In this configuration, the Test Driver invokes directly the Initiator action of the associated Test Service in order to trigger an exchange. The Test Driver is in service mode, and the associated Test Service is in local reporting mode, as it directly notifies the Test Driver. There is no need to generate messages on the wire for doing this, as both components reside on the same host.

A.2 The "Hub Driver" Test Harness Implementation

This configuration (Figure 13) is appropriate when two parties engage in interoperability testing with the help of a third-party, which facilitates the testing. Each party will still in turn play the driver party (due to the asymmetric character of the BIP test suite), but the third party will operate the Test Driver (install test cases, drive the executions, generate the reports.) The two candidate parties would only make sure their

MSH and Test Service are up and running, and that the CPAs associated with the test suite are accessible.

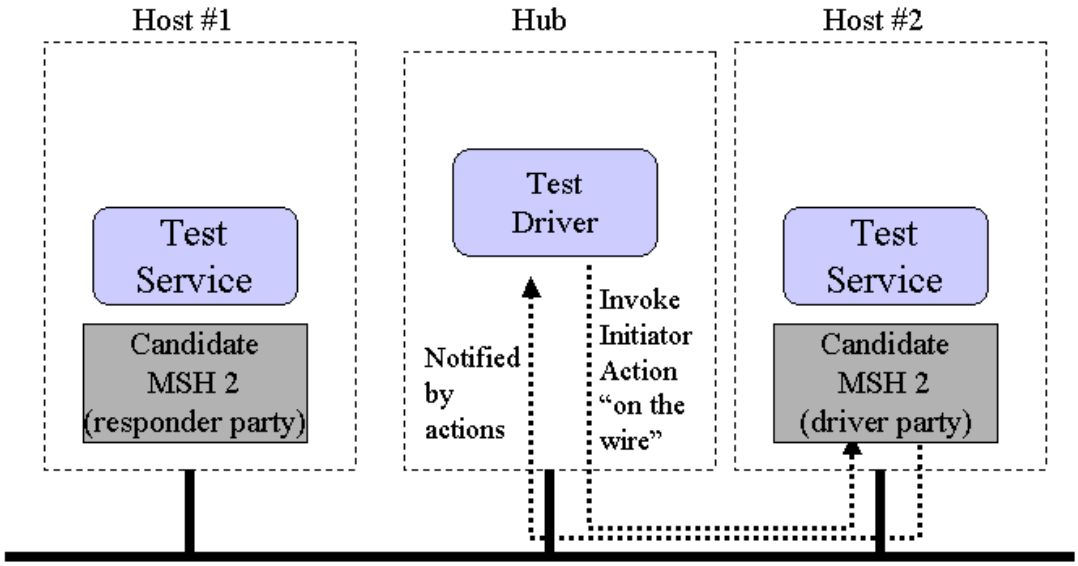


Fig 13. Hub-driver implementation

24

In this configuration, the Test Driver invokes remotely the Initiator action of the Test Service of the driver party, in order to trigger an exchange. The Test Driver, in connection mode, interfaces directly at transport level, generating message material as done in conformance testing. The notification from the actions of the Test Service (driver party side) will be done by messages sent to the Test Driver (Hub URL), which is proper to a Test Service in remote reporting mode. Once an exchange is triggered, both end-points can send messages to each other, directly or through the Hub node, used as a simple route.

Appendix B References

B.1 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/documents.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.

Appendix C 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.

C.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>

The OASIS ebXML IIC TC would especially like to thank the Drummond Group for their contribution to the test cases.

Appendix D Notices

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

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

Appendix E Revision History

Rev	Date	By Whom	What
cs-10	2003-04-03	Michael Kass	Initial version