Reliable Secure Profile 1.0 Test Scenarios

Document Status: WS-I Working Group Draft

Version: 1.0

Date: November 24, 2008

Editors:

Ram Jeyaraman, Microsoft (ram.jeyaraman@microsoft.com)

Notices

Abstract

This document proposes a set of scenarios for testing interoperability for Reliable Secure Profile 1.0. This document is intended to provide the information necessary to implement the described scenarios.

WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

Feedback

The Web Services-Interoperability Organization (WS-I) would like to receive input, suggestions and other feedback ("Feedback") on this work from a wide variety of industry participants to improve its quality over time.

By sending email, or otherwise communicating with WS-I, you (on behalf of yourself if you are an individual, and your company if you are providing Feedback on behalf of the company) will be deemed to have granted to WS-I, the members of WS-I, and other parties that have access to your Feedback, a non-exclusive, non-transferable, worldwide, perpetual, irrevocable, royalty-free license to use, disclose, copy, license, modify, sublicense or otherwise distribute and exploit in any manner whatsoever the Feedback you provide regarding the work. You acknowledge that you have no expectation of confidentiality with respect to any Feedback you provide. You represent and warrant that you have rights to provide this Feedback, and if you are providing Feedback on behalf of a company, you represent and warrant that you have the rights to provide Feedback on behalf of your company. You also acknowledge that WS-I is not required to review, discuss, use, consider or in any way incorporate your Feedback into future versions of its work. If WS-I does incorporate some or all of your Feedback in a future version of the work, it may, but is not obligated to include your name (or, if you are identified as acting on behalf of your company, the name of your company) on a list of contributors to the work. If the foregoing is not acceptable to you and any company on whose behalf you are acting, please do not provide any Feedback.

Feedback on this document should be directed to wsi_rsp_comment@ws-i.org.

Table of Contents

These scenarios do not cover *all* requirements set forth in RSP 1.0, but should be adequate enough to cover most of the requirements.

1 Scenarios

1.1 Reliable one-way

Client establishes a Reliable Session, then sends a message and initiates a close of the Session. Expand over the following options, giving 8 scenarios:

1. Security
    a. No Security
    b. Secure
2. Basic Profile Version
    a. BP 1.2 (SOAP 1.1 + WS-Addressing 1.0)
    b. BP 2.0 (SOAP 1.2 + WS-Addressing 1.0)
3. Addressable / Non-Addressable locations
    a. Addressable Client, Addressable Service
    b. Non-Addressable Client, Addressable Service

The Secure Scenarios are not listed here, but besides setting up the Secure Session, the Message Exchange pattern is the same. See the Security section for details.

This first set of scenarios is also expanded over BP 1.2 and BP 2.0. Since the only difference is the SOAP version, and the Message Exchange Pattern is not affected, BP version is not called out explicitly. The message examples linked show SOAP 1.2 (BP 2.0).

For the request-reply scenarios, the term, “request sequence” is used to define the sequence initiated by the Client requests to the Service. The term, “response sequence” is used to define the sequence initiated by the Service.

1.1.1 Reliable_OneWay_BPx.x_AddressableClient_AddressableService

In this scenario, one-way exchange is used. The Client’s application-level message is a simple Ping message, and no application-level response is expected. Since one-way exchange is used, there is no requirement to use WS-MakeConnection messages.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. The Service responds on the HTTP response with a CreateSequenceResponse. The Client then sends a Ping message. The Service responds by acknowledging the message with a SequenceAcknowledgement; there are no application messages to be sent on the HTTP response. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds with a CloseSequenceResponse. Last, the Client sends a TerminateSequence to which the Service responds with a TerminateSequenceResponse.

1.1.2 Reliable_OneWay_BPx.x_NonAddressableClient_AddressableService

In this scenario, the Client is non-addressable. But since it is One-Way, the Server can use the HTTP response for responses. Use of ws-MakeConnection is not necessary.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. The Service responds with a CreateSequenceResponse on the HTTP response. The Client then sends a Ping message. The Service only responds by acknowledging the message with a SequenceAcknowledgement on the HTTP response. The "To" header of the SequenceAcknowledgement is the WS-Addressing anonymous URI, since no MakeConnection was used. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds with a CloseSequenceResponse. Last, the Client sends a TerminateSequence to which the Service responds with a TerminateSequenceResponse.

1.2 Reliable request-response

These scenarios will follow the basic pattern of opening a sequence, sending 3 application messages, then closing the sequence. These scenarios are parameterized. The non-addressable scenarios using MakeConnection will cover R2000, R2001, R2004, R2100, R2101, R2110, R2111, R2112, and R2113. The Secure scenarios cover R3010, R3100, R3101, R3102, R3110, R3111, R3114, R3115, and R3300. The RequestReply scenarios will cover R0010, R0011, R0020, and R0200. All scenarios will be covering R0102, R0210, R0600, R0800, R0900, R0901, and R2005.

Reliable asynchronous two-way exchange: Expand over the following options, giving 16 scenarios:

1. Security
    a. No Security
    b. Secure
2. Basic Profile Version
    a. BP 1.2 (SOAP 1.1 + WS-Addressing 1.0)
    b. BP 2.0 (SOAP 1.2 + WS-Addressing 1.0)
3. Addressable / Non-Addressable locations
    a. Addressable Client, Addressable Service
    b. Non-Addressable Client, Addressable Service - Service responds to client requests with either a 202 or a SequenceAcknowledgement on the HTTP response, then Client sends MakeConnection request to get the response messages in the response sequence.
4. Offer: The "offer" element is used to establish the response.
    a. Use Offer
    b. Do not use Offer

1.2.1 Reliable_RequestReply_BPx.x_AddressableClient_AddressableService_Offer

This scenario introduces a Request-Reply pattern. For application-level messages, the Client will send an Echo message, to which the Server will respond with an EchoResponse message. In this case, both the

Client and Service are addressable, so the use of WS-MakeConnection is not necessary. The message exchanges happen reliably from the Client to the Service and from the Service to the Client.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. The body of this CreateSequence message contains the "offer" element so that the Service can establish a sequence with the endpoint in the "offer." This sequence will be referred to as the "return sequence." The "AcksTo" and "ReplyTo" headers in the CreateSequence uses an addressable URI same as the endpoint in the "offer". The Service responds with a CreateSequenceResponse. The Client then sends an Echo message. The Service responds with a SequenceAcknowledgement and an EchoResponse message on the return sequence. The Client responds to the EchoResponse with a SequenceAcknowledgement on the request sequence. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds with a CloseSequenceResponse. Then, the Client sends a TerminateSequence to which the Service responds with a TerminateSequenceResponse. Optionally, this sequence closing may also happen for the return sequence from the Service to the Client.

Note: The lifecycle messages may contain a WS-Addressing [reply endpoint] property set to a WS-Addressing anonymous URI.

1.2.2 Reliable_RequestReply_BPx.x_AddressableClient_AddressableService_NoOffer

The sequence begins with a CreateSequence message sent from the Client to the Service. The body of this CreateSequence message does not contain the "offer" element. The Service responds with a CreateSequenceResponse on the HTTP response. Since an "offer" is not included in the CreateSequence, the Service must issue a CreateSequence for the response sequence to the Client to which the Client responds with a CreateSequenceResponse. The Client then sends an Echo message. The Service responds with a SequenceAcknowledgement and an EchoResponse message on the return sequence. Note that piggybacking may or may not occur.

The Client responds to the EchoResponse with a SequenceAcknowledgement on the request sequence. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds with a CloseSequenceResponse. Then, the Client sends a TerminateSequence to which the Service responds with a TerminateSequenceResponse. Optionally, this sequence closing may also happen for the return from the Service to the Client.

1.2.3 Reliable_RequestReply_BPx.x_NonAddressableClient_AddressableService_Offer

This scenario uses MakeConnection messages issued by the Client to get messages from the Service. Because the Client is not addressable, all Reply-To and AcksTo addresses from the Client point to the MakeConnection Anonymous URI.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. Since the Client is Non-Addressable the Service responds on the HTTP response with a CreateSequenceResponse. An "offer" element (containing an instance of MakeConnection Anonymous URI) is used in the body of the CreateSequence for the Service to establish a return session with the Client. The Client then sends an Echo message. The Client will also send a MakeConnection message to the Service on the response sequence to get the EchoResponse message.

The Service replies to the Echo message with a SequenceAcknowledgement on either the HTTP response of the request sequence or directly on the response sequence in response to the MakeConnection. The Service sends Sequence Acknowledgements on the HTTP response if the wsa:ReplyTo of the request matches the AcksTo EPR (including reference parameters). Otherwise, the Service sends the Sequence Acknowledgement only as a result of a Make Connection.

When the Service has the EchoResponse available, it sends the EchoResponse on the response sequence in response to a MakeConnection issued by the Client. When the Client gets the EchoResponse, it responds with a SequenceAcknowledgement on the request sequence.

Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds on the HTTP response with a CloseSequenceResponse. Then, the Client sends a TerminateSequence, to which the Service responds with a TerminateSequenceResponse. A similar exchange may happen for the response sequence, but the Client must issue MakeConnection messages to get the CloseSequence and TerminateSequence messages from the Service.

Note, in this message exchange pattern, piggybacking may or may not occur.

1.2.4 Reliable_RequestReply_BPx.x_NonAddressableClient_AddressableService_NoOffer

This scenario uses MakeConnection messages issued by the Client to get messages from the Service.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. Since the Client is Non-Addressable, and exchange is Request-Reply, the Service responds on the HTTP response with a CreateSequenceResponse. Additionally, since the CreateSequence does not contain an "offer," the Client must issue a MakeConnection on what is to be the Server's response sequence with the Client. The Server responds to this with a CreateSequence. The Client responds to that CreateSequence with a CreateSequenceResponse. The Client then sends an Echo message. The Client will also send a MakeConnection message to the Service to get the EchoResponse message. The Service replies to the Echo message with a SequenceAcknowledgement either on the HTTP response of the request sequence or directly on the response sequence in response to the MakeConnection. When the Service has the EchoResponse available, it sends the EchoResponse on the response sequence in response to a MakeConnection issued by the Client. When the Client gets the EchoResponse, it responds with a SequenceAcknowledgement on the request sequence. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds on the HTTP response with a

CloseSequenceResponse. Then, the Client sends a TerminateSequence, to which the Service responds with a TerminateSequenceResponse. A similar exchange happens on the response sequence, but the Client must issue MakeConnection messages to get the CloseSequence and TerminateSequence messages from the Service.

1.3 Retransmission Scenarios

A mechanism for dropping messages is required for these scenarios. This will test retrying messages. These scenarios cover R0101, R0102, R0110, and R0120. The secure retransmission scenarios cover R3301.

The following describes the retransmission scenarios. This section only describes the message exchange patterns. These scenarios are to be expanded by running both in the absence of Security and with Security enabled.

A mechanism for dropping specific sequence messages is required for these scenarios.

1.3.1 Reliable_OneWay_BP20_AddressableClient_AddressableService_DropMessage

This scenario tests retransmission of dropped messages with a One-Way scenario and addressable RM nodes. Retransmit unacknowledged sequence messages (One-Way pattern): Drop a sequence message from the Client at least once before allowing it to be sent to the Service. Both Client and Service are addressable.

Message Exchange Pattern

The sequence begins with a CreateSequence message sent from the Client to the Service. The Service responds with a CreateSequenceResponse. The Client then sends 1 Ping message. The message is dropped before the Server can receive it. Since the Client doesn't receive a SequenceAcknowledgement from the Server, the Client will resend the message. Provided the message is not dropped this time, the Service will respond by acknowledging the message with a SequenceAcknowledgement on the HTTP response. Finally, the sequence is closed by the Client sending a CloseSequence message. The Service responds with a CloseSequenceResponse. The Client sends a TerminateSequence to which the Service responds with a TerminateSequenceResponse.

1.3.2 Reliable_RequestReply_BP20_NonAddressableClient_AddressableService_DropFirstMessageOnce

This scenario will use MakeConnection to get a dropped EchoResponse message from the Service. Since the scenario uses RequestReply with a Non-Addressable Client, the Client already issues MakeConnection calls to get every message from the Service.

Use MakeConnection to get unacknowledged messages.

- Request-Reply with Non-Addressable Client, Addressable Service.

- Client establishes a Sequence and sends an application request message (Echo).
- Server sends the application response message (EchoResponse), but it is dropped.
- The Server will resend the message because it never received a SequenceAcknowledgement.

Message Exchange Pattern

The sequence begins the same way as Reliable_RequestReply_BPx.x_NonAddressableClient_AddressableService_Offer. The Client then sends an Echo message. The Server replies on the HTTP response with a SequenceAcknowledgement. The Client sends a MakeConnection message to the Service on the response sequence to get the EchoResponse message. The Server replies to the MakeConnection with the EchoResponse message. This message is dropped before reaching the Client. The Client should continue to send MakeConnection, since it didn't receive a response on the first MakeConnection request. The Server receives the MakeConnection and resends the EchoResponse. The message is not dropped this time. Upon receipt of the EchoResponse message, the Client sends a SequenceAcknowledgement back to the Service via a new HTTP request. The message exchanges are closed in the same manner as with Reliable_RequestReply_BPx.x_NonAddressableClient_AddressableService_Offer.

1.3.3 SecureReliable One-Way Addressable Drop Message

Same as 1.3.1, but run under a secure environment.

1.3.4 SecureReliable Request-Reply Non-Addressable Drop Message

Same as 1.3.2, but run under a secure environment.

1.4 Negative Scenarios

These two scenarios cover the transmission of a fault generated at the RMD while processing a CreateSequence message. The non-secure scenario covers R0400. The secure scenario covers R0400, R3120, R3121, and R3122.

1.4.1 Reliable Request-Reply Addressable Fault-CreateSequence

This scenario tests the transmission of a fault from the Service to the Client.

Message Exchange Pattern

The Client sends a CreateSequence message to the Service. While processing the message, the Service faults. This fault must be retransmitted back to the Client. The Service does not create a sequence or send back a CreateSequenceResponse.

1.4.2 Secure Reliable Request-Reply Addressable Fault-CreateSequence

This scenario is identical to scenario 1.4.1, but run under a secure environment. This is to ensure coverage of requirement R3120.

1.5 Request/Response - non-addressable client

This scenario involves a client sending a request message to a service and using MC to pull back the response. In this case we will mimic a situation where the service is a long-running service and thus cannot (or does not want) to keep the current transport back-channel open the entire time.

Message Exchange Pattern

Client sends a request message to the service. The request message includes a wsa:ReplyTo and wsa:FaultTo set to an instance of the MC Anonymous URI. The service returns an HTTP 202 Accepted. Periodically, the client will send a MakeConnection message to the service to allow for any possible response to flow. If the client used a different MC Anonymous URI for wsa:ReplyTo and wsa:FaultTo then it is expected that the client will send two different MakeConnection messages - however, the use of one or two URIs is an implementation choice. Upon receipt of a MC message, the service will either allow a pending message targeted for the included MC Anonymous URI to flow, or after waiting an certain amount of time for a message to be ready, it will return an HTTP 202 Accepted. The client is expected to continue to send MakeConnection messages until a response is received.

1.6 Non-addressable Service

1.6.1 Reliable One-way – Addressable Client / Non-addressable Service

In this scenario the client is addressable and the service is non-addressable. To initiate the message changes from the client, the client must first be given an EPR (using the MC Anonymous URI).

Message Exchange Pattern

The scenario begins with the service sending its EPR to the client (once we get the WSDL for the app msgs we can add this one) - the EPR uses an instance of the MC Anonymous URI since the service is non-addressable. The client then initiates a new RM Sequence with the service by trying to send a CreateSequence to the EPR it was provided, however the transmission of the CreateSequence is done by the service sending MC messages to the client. The service sends a MC to the client and pulls back the CreateSequence. The service response with a CSR to the CS's ReplyTo EPR. Normal one-way message scenario processing occurs (one-way app msgs, terminateSequence msgs are sent from the client to the service, acks are sent from the service to the client's CS/AcksTo EPR). Each of the messages from the client to the service is transmitted on the back-channel of a MC request message.

1.6.2 Request/Response - non-addressable service

This scenario involves a client initiating a sequence of messages exchanges to a service but the service is not addressable. As mentioned in the MC specification, a classic example of this is an event producer sending notifications to an event consumer, where the event consumer is behind a firewall. In this scenario the notion of a client and service are turned around and as such a new term, "Initiator", is introduced to represent the endpoint that starts the processing of the scenario. Note, the initiator may

in reality be the "service" to keep the number of endpoints needed to run this scenario down to two.

Message Exchange Pattern

An initiator provides the client with the EPR to the service (a new operation may be needed here). This EPR contain an instance of the MC Anonymous URI to uniquely identify the service. The client will initiate a series of one-way and request-response messages with the service. Note, in the request-response case the wsa:ReplyTo will be an addressable EPR. Since the [destination] EPR for these messages is an instance of the MC Anonymous URI they will only be transmitted as result of the service sending a MC to the client. Thus, the service must periodically send a MC to the client. Upon receive of a message from the client, the service will process each message. If there is a response then it is sent to the appropriate EPR (replyTo vs faultTo) over a new HTTP connection.

1.6.3 Request/Response - non-addressable secure service

1.7 This scenario is the same as Non-addressable Service

1.7.1 Reliable One-way – Addressable Client / Non-addressable Service

In this scenario the client is addressable and the service is non-addressable. To initiate the message changes from the client, the client must first be given an EPR (using the MC Anonymous URI).

Message Exchange Pattern

The scenario begins with the service sending its EPR to the client (once we get the WSDL for the app msgs we can add this one) - the EPR uses an instance of the MC Anonymous URI since the service is non-addressable. The client then initiates a new RM Sequence with the service by trying to send a CreateSequence to the EPR it was provided, however the transmission of the CreateSequence is done by the service sending MC messages to the client. The service sends a MC to the client and pulls back the CreateSequence. The service response with a CSR to the CS's ReplyTo EPR. Normal one-way message scenario processing occurs (one-way app msgs, terminateSequence msgs are sent from the client to the service, acks are sent from the service to the client's CS/AcksTo EPR). Each of the messages from the client to the service is transmitted on the back-channel of a MC request message.

1.8 Request/Response - non-addressable service except the first message sent by the client to the service is a RequestSecurityToken message. This will establish a secure conversation between the two endpoints. The remainder of the messages defined in Non-addressable Service

1.8.1 Reliable One-way – Addressable Client / Non-addressable Service

In this scenario the client is addressable and the service is non-addressable. To initiate the message changes from the client, the client must first be given an EPR (using the MC Anonymous URI).

Message Exchange Pattern

The scenario begins with the service sending its EPR to the client (once we get the WSDL for the app

msgs we can add this one) - the EPR uses an instance of the MC Anonymous URI since the service is non-addressable. The client then initiates a new RM Sequence with the service by trying to send a CreateSequence to the EPR it was provided, however the transmission of the CreateSequence is done by the service sending MC messages to the client. The service sends a MC to the client and pulls back the CreateSequence. The service response with a CSR to the CS's ReplyTo EPR. Normal one-way message scenario processing occurs (one-way app msgs, terminateSequence msgs are sent from the client to the service, acks are sent from the service to the client's CS/AcksTo EPR). Each of the messages from the client to the service is transmitted on the back-channel of a MC request message.

Request/Response - non-addressable service will be the same, except they are protected using this newly established security context. Upon completion of the application messages, the client will then send Cancel to shut down the secure conversation.

Message Exchange Pattern

An initiator provides the client with the EPR to the service (a new operation may be needed here). This EPR contain an instance of the MC Anonymous URI to uniquely identify the service. The client send an RST to the service to initiate a new secure conversation. The service will response to the wsa:ReplyTo EPR over a new HTTP connection. The client will initiate a series of one-way and request-response messages with the service - each will be protected with the newly created security context. Note, in the request-response case the wsa:ReplyTo will be an addressable EPR. Since the [destination] EPR for these messages is an instance of the MC Anonymous URI they will only be transmitted as result of the service sending a MC to the client. Thus, the service must periodically send a MC to the client. Upon receive of a message from the client, the service, after verifying the messages are properly secured, will process each message. If there is a response then it is sent to the appropriate EPR (replyTo vs faultTo) over a new HTTP connection - again protected with the established security context. When the client is done it will send a Cancel to the service, also thru the use of a MC message exchange.

1.9 Secure Conversation

In this scenario we test the basics of the SecureConversation w.r.t. how RSP profiles its use.

Message Exchange Pattern

This scenario will involve the creation of a new security context, modifying it and then canceling it. The message overall flow will consist of:

1 - Client sends an RST to the service

2 - Service responds with an RSTRC

3 - Client sends an Echo, Service responds with an EchoResponse - both secured by the context established in 1/2.

4 - Client Renews the context (RST <-> RSTR)

5 - Client sends an Echo, Service responds with an EchoResponse - both secured by the context established in 1/2.

6 - Client Cancels the context (RST <-> RSTR)
7 - Client sends an Echo, Service rejects the request due to the context no longer being valid.

1.10 Reliable request-response with App fault

This scenario is the same as a reliable request/response except an application fault is generated for one of the messages.

Message Exchange Pattern
Follow the same MEP as for Reliable Request/Response+Offer: RMS send CS+Offer. RMD responds with CSR+Accept. RMS sends a series of request messages. RMD responds with either non-faulting response, or with a fault (based on the data within the request - e.g. perhaps an Echo w/o 'text' generates a fault). Each response, non-fault or fault, will be sent using the Offered sequence from the original CS/CSR exchange. Finally, RMS sends a TS and RMD responds with a TSR.

1.11 Sequence carry-over and independence

In this scenario we verify that the two sequences created through a CS+Offer are truly independent.

Message Exchange Pattern

1. Client side RMS sends CS+Offer to service-side RMD. Service side responds with a CSR and accepts the offered sequence.
2. Client sends requests req1 and req2 using original sequence. For req1 and req2, service sends acknowledgements on HTTP backchannel containing 202 Accepted.
3. Client-side RMS sends TerminateSequence to service-side RMD. This terminates the original sequence.
4. Service side RMS sends responses resp1 and resp2 (corresponding to req1 and req2) using the offered sequence. For resp1 and resp2, client sends acknowledgements on the HTTP backchannel containing 202 Accepted.
5. RMS sends CS to RMD with no Offer.
6. Client side RMS sends requests req3 and req4 using new sequence. For req3 and req4, service sends acknowledgements on HTTP backchannel containing 202 Accepted.
7. Service side RMS sends TerminateSequence to client-side RMD. This cancels the offered sequence from client-side RMS.
8. Service-side RMS sends a CS to client side RMD. Client side RMS responds with CSR.

9. Service side RMS sends responses resp3 and resp4 (corresponding to req3 and req4) using the newly created sequence (step 8). For resp3 and resp4, client sends acknowledgements on the HTTP backchannel containing 202 Accepted.

NOTE: It is possible for the RMD to reuse the original offered sequence for responses #3 and #4, in which case steps 7 and 8 would not happen and the sequence mentioned in step 9 would be the same sequence as mentioned in step 4.

1.12 Dropping of lifecycle messages

In this scenario RM Lifecycle Messages are dropped and the sender is expected to resend them. Aside from the choice of which message to drop, there is the variant of the request message vs the response message being dropped and what the receiver will do upon receipt of a resent request message.

Message Exchange Pattern

Variant 1 - RMS sends a CreateSequence that is lost before the RMD receives it. The RMS resends it at some point later. The RMD responds with a CreateSequenceResponse.

Variant 2 - RMS sends a CreateSequence, the CreateSequenceResponse is lost. The RMS resends the CreateSequence. If the RMS uses the same SeqID then the RMD returns a "Create Sequence Refused" fault. The RMS sends a CreateSequence with a new SeqID and the RMD responds with a CreateSequenceResponse.

Variant 3 - RMS sends a CreateSequence, RMD responds with a CreateSequenceResponse. Some number of application messages are sent. The RMS sends a CloseSequence that is lost before the RMD receives it. The RMS resends the CloseSequence and the RMD responds with a CloseSequenceResponse.

Variant 4 - RMS sends a CreateSequence, RMD responds with a CreateSequenceResponse. Some number of application messages are sent. The RMS sends a CloseSequence and the CloseSequenceResponse is lost. The RMS resends the CloseSequence and the RMD responds with a CloseSequenceResponse.

Variant 5 - RMS sends a CreateSequence, RMD responds with a CreateSequenceResponse. Some number of application messages are sent. The RMS sends a TerminateSequence that is lost before the RMD receives it. The RMS resends the TerminateSequence and the RMD responds with a TerminateSequenceResponse.

Variant 6 - RMS sends a CreateSequence, RMD responds with a CreateSequenceResponse. Some number of application messages are sent. The RMS sends a TerminateSequence and the TerinateSequenceResponse is lost. The RMS resends the TerminateSequence and the RMD responds with a "Unknown Sequence Fault".

1.13 Reliable_One_Way_Requests_Anon_AcksTo_Multiple_Sequences

The client creates two sequences to the service in the identical way; the CreateSequence message contains an anonymous AcksTo and there is no Offer. The service supports a one-way message. The client reliably sends a number of messages to the service, alternating the sequence for each message.

1.14 Reliable_Request_Response_AcksTo_RefParm

The client creates a sequence to the service with a CreateSequence message that contains a non-anonymous AcksTo with ReferenceParameters. The CreateSequence may or may not contain an Offer. The service supports reliable requests and either reliable or non-reliable responses. The client reliably sends a number of messages to the service. The /wsa:ReplyTo/wsa:Address for all these request should be identical to the /wsrm:CreateSequence/wsrm:AcksTo/wsa:Address but the /wsa:ReplyTo/wsa:ReferenceParameters should alternate between a set that is identical to /wsrm:CreateSequence/wsrm:AcksTo/wsa:ReferenceParameters and a set that is different.

Note: An implementation must implement the service side behavior described in this scenario, but the client side behavior described in this scenario is optional.

1.15 Reliable_Request_Empty_Response

Create a WSDL with a `portType` that defines a single request-response operation. The `wsdl:output` of this operation should have an empty message and a non-empty `wsam:Action` attribute. A client implementation of this WSDL reliably sends the request messages and either reliably or non-reliably receives the response messages.

2 Appendix – WSDL operations

Notify

A one-way operation with the following outline:
Request:
[Action] http://example.com/rsp/Notify
[Body]
<rsp:Notify>
<rsp:ID> myID </rsp:ID>
<rsp:text> Hello World <rsp:text>
</rsp:Notify>

- Has two pieces of data:

- ID: a string representing a unique identifier used to associate a series of Notify and Echo messages.
- text: a string of the client's choosing.

- All calls to Notify and Echo will concatenate the 'text' values for matching IDs.
- An empty string or "fault" for the 'text' parameter must cause the service to generate a fault.

Echo

A two-way operation with the following outline:
Request:
[Action] http://example.com/rsp/Echo
[Body]
<rsp:Echo>
<rsp:ID> myID <rsp:ID>
<rsp:text> Hello World <rsp:text>
</rsp:Echo>

Response:
[Action] http://example.com/rsp/EchoResponse
[Body]
<rsp:EchoResponse>
<rsp:text> Hello World <rsp:text>
</rsp:EchoResponse>

- Has two pieces of data:
  - ID: a string representing a unique identifier used to associate a series of Notify and Echo messages.
  - text: a string of the client's choosing.
- All calls to Notify and Echo will concatenate the 'text' values for matching ID's.
- An empty string or "fault" for the 'text' parameter must cause the service to generate a fault.

  Note: leading and trailing spaces in the 'text' parameter will be ignored.

3 Appendix – Test WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://example.com/rsp"
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
                xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
            xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
                xmlns:tns="http://example.com/rsp">

  <wsdl:types>
    <xs:schema targetNamespace="http://example.com/rsp">
      <xs:include schemaLocation="http://...ws-i.../rsp.xsd"/>
      <xs:element name="Notify" type="tns:NotifyType"/>
      <xs:element name="Echo" type="tns:EchoType"/>
      <xs:element name="EchoResponse" type="tns:EchoResponseType"/>

    </xs:schema>
  </wsdl:types>

  <wsdl:message name="Notify">
    <wsdl:part name="Notify" element="tns:Notify"/>
  </wsdl:message>

  <wsdl:message name="Echo">
    <wsdl:part name="Echo" element="tns:Echo"/>

   </wsdl:message>

  <wsdl:message name="EchoResponse">
    <wsdl:part name="EchoResponse" element="tns:EchoResponse"/>
  </wsdl:message>

  <wsdl:portType name="RspPort">
    <wsdl:operation name="Notify">
      <wsdl:input message="tns:Notify"
      wsam:Action="http://example.com/rsp/Notify" />
    </wsdl:operation>
    <wsdl:operation name="Echo">
      <wsdl:input message="tns:Echo"
      wsam:Action="http://example.com/rsp/Echo" />
      <wsdl:output message="tns:EchoResponse"
      wsam:Action="http://example.com/rsp/EchoResponse" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="RspSOAP11Binding" type="tns:RspPort">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Notify">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal" parts="Notify"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="Echo">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal" parts="Echo"/>
      </wsdl:input>
      <wsdl:output>
```

```
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:binding name="RspSOAP12Binding" type="tns:RspPort">
    <soap12:binding style="document"
                    transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="Notify">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal" parts="Notify"/>
      </wsdl:input>
    </wsdl:operation>
    <wsdl:operation name="Echo">
      <soap12:operation/>
      <wsdl:input>
        <soap12:body use="literal" parts="Echo"/>
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="RspService">
    <wsdl:port  name="Soap11port" binding="tns:RspSOAP11Binding">
      <soap:address location="http://example.com/rsp/rspSOAP11"/>
    </wsdl:port>
    </wsdl:service>
    <wsdl:service name="RspService12">
    <wsdl:port name="Soap12port" binding="tns:RspSOAP12Binding">
      <soap12:address location="http://example.com/rsp/rspSOAP12"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```

4 Appendix – XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://example.com/rsp"
  xmlns:tns="http://example.com/rsp"
  elementFormDefault="qualified">

  <xs:complexType name="NotifyType">
    <xs:sequence>
      <xs:element name="ID" type="xs:string" />
      <xs:element name="text" type="xs:string" />
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
```

```
  <xs:complexType name="EchoType">
    <xs:sequence>
      <xs:element name="ID" type="xs:string" />
      <xs:element name="text" type="xs:string" />
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="EchoResponseType">
    <xs:sequence>
      <xs:element name="text" type="xs:string" />
      <xs:any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

5 Acknowledgements

The following experts have contributed to the development and submission of the test scenarios:

James Osborne, Microsoft

6 Revision History

Date	Revisions	Comments
28 August 2008	Contributed documents provided in WS-I RSP WG F2F meeting in Sunnyvale, CA.	
19 September 2008	Document revised into WS-I format and included WG F2F comments.	Revisions integrated.
26 September 2008	Added new scenarios.	
05 October 2008	Updated scenarios descriptions.	
06 October 2008	Updated WSDL and XSD files.	
10 October 2008	Added additional scenarios relating to requirements R0501, R0510, R0530.	
27 October 2008	Updated description of scenario 1.12 (reliable request response AckTo RefParam) to indicate that	

	the client-side behavior is optional for implementations.	
24 November 2008	Updated document status to “Working Group Draft”. Updated WSDL.	