ETSI CTI Plugtests Guide Draft V0.0.5 (2013-11)

IoT CoAP Plugtests;
Las Vegas, USA;
19 - 22 November 2013

Contents

1 Scope

This document forms the guidelines to lead the technical organization of the CoAP#3 and OMA LWM2M Plugtests event, in Las Vegas, from 19th to 22nd November 2013. This document is intended to be upgraded for future interoperability events.

This document describes:

- The testbed architecture showing which IoT CoAP systems and components are involved and how they are going to interwork
- The configurations used during test sessions, including the relevant parameter values of the different layers
- The interoperability test descriptions, describing the scenarios, which the participants will follow to perform the interoperability tests

2 References

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents, which are not found to be publicly available in the expected location, might be found at http://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication ETSI cannot guarantee their long term validity.

2.1 Normative references

The following referenced documents are necessary for the application of the present document.

[1] Constrained Application Protocol (CoAP); draft-ietf-core-coap-18

[2] Core Link Format; RFC 6690

[3] Observing Resources in CoAP; draft-ietf-core-observe-11

[4] Blockwise transfers in CoAP; draft-ietf-core-block-14

[5] ETSI TS 103 104: Test Interoperability Test Specification for CoAP Binding of ETSI M2M Primitives

3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ACK	Acknowledgement
CON	Confirmable
NON	Non-Confirmable
NA	Network Application
RST	Reset
TD	Test Description

4 Conventions

4.1 Interoperability test process

4.1.1 Introduction

The goal of interoperability test is to check that devices resulting from protocol implementations are able to work together and provide the functionalities provided by the protocols. As necessary, a message may be checked during an interoperability test, when a successful functional verification may result from an incorrect behaviour for instance. Detailed protocol checks are part of the conformance testing process and are thus avoided during the Interoperability tests.

The test session will be mainly executed between 2 devices from different vendors. For some test purposes, it may be necessary to have more than 2 devices involved. The information about the test configuration like the number of devices or the roles required are indicated in the test description tables below.

4.1.2 The test description proforma

The test descriptions are provided in proforma tables. The following different types of test operator actions are considered during the test execution:

- A **stimulus** corresponds to an event that enforces an EUT to proceed with a specific protocol action, like sending a message for instance
- A **verify** consists of verifying that the EUT behaves according to the expected behaviour (for instance the EUT behaviour shows that it receives the expected message)
- A **configure** corresponds to an action to modify the EUT configuration
- A **check** ensures the correctness of protocol messages on reference points, with valid content according to the specific interoperability test purpose to be verified.

For the execution of the interoperability test sessions, the following conventions apply:

- Every ‘Check’ step of a test description should be performed using a trace created by a monitor tool (see clause ‘Tooling’ below) and may be skipped due to time restrictions

4.2 Tooling

- Participant shall use their own tools (e.g. tcpdump, wireshark) for logging and analysing messages for the “check” purposes
- Participants will be given the opportunity to upload their log files to a central server for a format validity check. The checks defined in each test description will be automatically performed by the central server
- Except for the “check” events, the verification of the message correctness is not part of the Interoperability test process
- To realize the lossy context of tests TD_XXX (e.g. packet loss and packet delay) a gateway will be provided which will serve as an intermediate between the client and the server to simulate the lossy medium (technically this is implemented using NAT-style UDP port redirections)

4.3 Test Description naming convention

Table 1: TD naming convention

TD/<root>/<gr>/<nn>		
<root> = root	COAP	Constrained Application Protocol
<gr> = group	CORE	Core protocol
	LINK	Core Link Format
	BLOCK	Blockwise transfers
	OBS	Observing Resources
	DTLS	DTLS
<nn> = sequential number		01 to 99

4.4 Test Summary – Base CoAP Tests

Table 2: CoAP Tests

TD_COAP_CORE_01	Perform GET transaction (CON mode)
TD_COAP_CORE_02	Perform DELETE transaction (CON mode)
TD_COAP_CORE_03	Perform PUT transaction (CON mode)
TD_COAP_CORE_04	Perform POST transaction (CON mode)
TD_COAP_CORE_05	Perform GET transaction (NON mode)
TD_COAP_CORE_06	Perform DELETE transaction (NON mode)
TD_COAP_CORE_07	Perform PUT transaction (NON mode)
TD_COAP_CORE_08	Perform POST transaction (NON mode)
TD_COAP_CORE_09	Perform GET transaction with separate response (CON mode, no piggyback)
TD_COAP_CORE_10	Perform GET transaction containing non-empty Token option (CON mode)
TD_COAP_CORE_11	Perform GET transaction containing non-empty Token with a separate response (CON mode)
TD_COAP_CORE_12	Perform GET transaction using empty Token (CON mode)
TD_COAP_CORE_13	Perform GET transaction containing several URI-Path options (CON mode)
TD_COAP_CORE_14	Perform GET transaction containing several URI-Query options (CON mode)
TD_COAP_CORE_15	Perform GET transaction (CON mode, piggybacked response) in a lossy context
TD_COAP_CORE_16	Perform GET transaction (CON mode, delayed response) in a lossy context
TD_COAP_CORE_17	Perform GET transaction with a separate response (NON mode)
TD_COAP_CORE_18	Perform POST transaction with responses containing several Location-Path options (CON mode)
TD_COAP_CORE_19	Perform POST transaction with responses containing several Location-Query options (CON mode)
TD_COAP_CORE_20	Perform GET transaction containing the Accept option (CON mode)
TD_COAP_CORE_21	Perform GET transaction containing the ETag option (CON mode)
TD_COAP_CORE_22	Perform GET transaction with responses containing the ETag option and requests containing the If-Match option (CON mode)
TD_COAP_CORE_23	Perform PUT transaction containing the If-None-Match option (CON mode)
TD_COAP_CORE_31	Perform CoAP Ping (CON mode)

4.5 Test Summary – Link Tests

Table 3: Link Tests

TD_COAP_LINK_01	Access to well-known interface for resource discovery
TD_COAP_LINK_02	Use filtered requests for limiting discovery results
TD_COAP_LINK_03	Handle empty prefix value strings
TD_COAP_LINK_04	Filter discovery results in presence of multiple rt attributes
TD_COAP_LINK_05	Filter discovery results using if attribute and prefix value strings
TD_COAP_LINK_06	Filter discovery results using sz attribute and prefix value strings
TD_COAP_LINK_07	Filter discovery results using href attribute and complete value strings
TD_COAP_LINK_08	Filter discovery results using href attribute and prefix value strings
TD_COAP_LINK_09	Arrange link descriptions hierarchically

4.6 Test Summary – Block Tests

Table 4: Block Tests

TD_COAP_BLOCK_01	Handle GET blockwise transfer for large resource (early negotiation)
TD_COAP_BLOCK_02	Handle GET blockwise transfer for large resource (late negotiation)
TD_COAP_BLOCK_03	Handle PUT blockwise transfer for large resource
TD_COAP_BLOCK_04	Handle POST blockwise transfer for creating large resource
TD_COAP_BLOCK_05	Handle POST with two-way blockwise transfer
TD_COAP_BLOCK_06	Handle GET blockwise transfer for large resource (early negotiation, 16 byte block size)

4.7 Test Summary – Observ Tests

Table 5: OBS Tests

TD_COAP_OBS_01	Handle resource observation with CON messages
TD_COAP_OBS_02	Handle resource observation with NON messages
TD_COAP_OBS_04	Client detection of deregistration (Max-Age)
TD_COAP_OBS_05	Server detection of deregistration (client OFF)
TD_COAP_OBS_06	Server detection of deregistration (explicit RST)
TD_COAP_OBS_07	Server cleans the observers list on DELETE
TD_COAP_OBS_08	Server cleans the observers list when observed resource content-format changes
TD_COAP_OBS_09	Update of the observed resource
TD_COAP_OBS_10	GET does not cancel resource observation

4.8 Test Summary - DTLS Scenarios

Table 6: DTLS

TD_COAP_DTLS_01	Basic DTLS PSK (success case)
TD_COAP_DTLS_02	Basic DTLS PSK (failure case — wrong PSK)
TD_COAP_DTLS_03	Lossy DTLS PSK (success case)
TD_COAP_DTLS_04	Basic DTLS RPK (success case)
TD_COAP_DTLS_05	Basic DTLS RPK (client failure case)
TD_COAP_DTLS_06	Basic DTLS RPK (server failure case)
TD_COAP_DTLS_07	Lossy DTLS RPK (success case)

4.9 Test Summary – OMA LWM2M Scenarios

Table 8: LWM2M Tests

Registration	LightweightM2M-1.0-int-101 – Initial Registration
	LightweightM2M-1.0-int-102 – Registration Update
	LightweightM2M-1.0-int-103 – Deregistration
Device object-related use cases	Querying basic information from the client
	Querying the firmware version from the client
	Rebooting the device
	Querying power status of the terminal
Device firmware update	LightweightM2M-1.0-int-301 – Firmware update (via COAP)
	LightweightM2M-1.0-int-302 – Firmware update (via alternative mechanism)
Connectivity object monitoring	LightweightM2M-1.0-int-401 – Querying of connectivity parameters
Observe and Notify	LightweightM2M-1.0-int-501 – Observation and notification of parameter values inside MachineLink 3G

5 Basic Configuration

5.1 Resources offered by servers under test

In order to ease test setup and execution, CoAP servers are requested to support the following resources:

Table 7: Resources offered by CoAP Servers

Resource name	Description	Used in
/test	Default test resource	TD_COAP_CORE_01 TD_COAP_CORE_02 TD_COAP_CORE_03 TD_COAP_CORE_04 TD_COAP_CORE_05 TD_COAP_CORE_06 TD_COAP_CORE_07 TD_COAP_CORE_08 TD_COAP_CORE_10 TD_COAP_CORE_11 TD_COAP_CORE_14 TD_COAP_CORE_18 TD_COAP_CORE_22 TD_COAP_LINK_08 TD_COAP_LINK_10

/validate	Resource which varies	TD_COAP_CORE_21 TD_COAP_CORE_27 TD_COAP_CORE_29
/create1	Resource which doesn't exist yet (to perform atomic PUT)	TD_COAP_CORE_23
/create2	Resource which doesn't exist yet	TD_COAP_CORE_24
/create3	Resource which doesn't exist yet	TD_COAP_CORE_28
/seg1/seg2/seg3	Long path resource	TD_COAP_CORE_12
/location1/location2/location3	Location path resource	TD_COAP_CORE_18 TD_COAP_CORE_24
/location-query	Resource accepting location query parameters	TD_COAP_CORE_19 TD_COAP_CORE_25
/query	Resource accepting query parameters	TD_COAP_CORE_13
/separate	Resource which cannot be served immediately and which cannot be acknowledged in a piggy-backed way	TD_COAP_CORE_09 TD_COAP_CORE_15 TD_COAP_CORE_16
/large	Large resource	TD_COAP_BLOCK_01 TD_COAP_BLOCK_02
/large-update	Large resource that can be updated using PUT method	TD_COAP_BLOCK_03
/large-create	Large resource that can be created using POST method	TD_COAP_BLOCK_04
/obs	Observable resource which changes every 5 seconds and for which the server is configured to send confirmable (CON) notifications	TD_COAP_OBS_01 TD_COAP_OBS_03 TD_COAP_OBS_04 TD_COAP_OBS_05 TD_COAP_OBS_06 TD_COAP_OBS_07 TD_COAP_OBS_08 TD_COAP_OBS_09
/obs-non	Observable resource which changes every 5 seconds and for which the server is configured to send non-confirmable (NON) notifications	TD_COAP_OBS_02
/.well-known/core	Core Link Format	TD_COAP_LINK_01 TD_COAP_LINK_02 TD_COAP_LINK_03 TD_COAP_LINK_04 TD_COAP_LINK_05 TD_COAP_LINK_06 TD_COAP_LINK_07 TD_COAP_LINK_08 TD_COAP_LINK_09 TD_COAP_LINK_10
/multi-format	Resource that exists in different content formats (text/plain utf8 and application/xml)	TD_COAP_CORE_20 TD_COAP_CORE_26
/link1	Link test resource	TD_COAP_LINK_07 TD_COAP_LINK_08

/link2	Link test resource	TD_COAP_LINK_07 TD_COAP_LINK_08
/link3	Link test resource	TD_COAP_LINK_07 TD_COAP_LINK_08
/path	Hierarchical link description entry	TD_COAP_LINK_09
/path/sub1	Hierarchical link description sub-resource	TD_COAP_LINK_09
/path/sub2	Hierarchical link description sub-resource	TD_COAP_LINK_09
/path/sub3	Hierarchical link description sub-resource	TD_COAP_LINK_09
/alternate	Alternate	TD_COAP_LINK_10

Note on resource sizes:

- Resources used in TD_COAP_CORE tests should not exceed 64 bytes
- Large resources used in TD_COAP_BLOCK tests shall not exceed 2048 bytes
- TD_COAP_LINK tests may require usage of Block options with some implementations

5.4 CoAP settings

Unless stated otherwise, the following settings shall be applied:

- Each equipment under test shall be configured with a unicast address
- Client cache shall be cleaned up after each test
- Use of ETag option shall be avoided, but implementation should be prepared to handle it
- Use of Token shall be avoided, but implementation should be prepared to handle it
- Use of Piggybacked responses shall be preferred

6 Test Configurations

This section defines the different test configurations.

6.1 Basic CoAP 1 (CoAP_CFG_01)

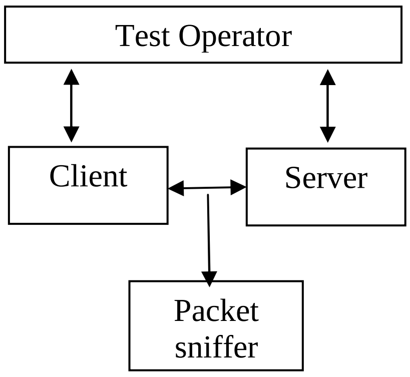

Figure 1: Basic One-2-One CoAP client/server Configuration

6.2 CoAP in lossy context (CoAP_CFG_02)

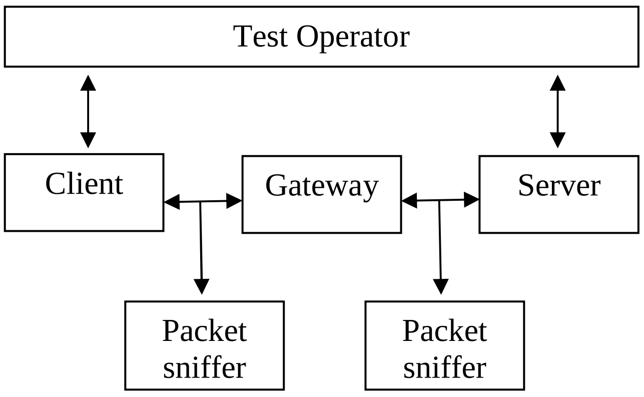

Figure 2: Basic One-2-One CoAP client/server Configuration in lossy context

The Gateway emulates a lossy medium between the client and the server. It does not implement the CoAP protocol itself (in other terms it is not a CoAP proxy), but works at the transport layer. It provides two features:

- It performs NAT-style UDP port redirections towards the server (thus the client contacts the gateway and is transparently redirected towards the server)
- It randomly drops packets that are forwarded between the client and the server

6.3 Test Configuration 3 (CoAP_CFG_03)

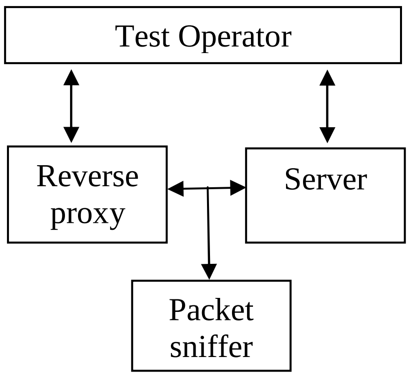

Figure 3: Basic One-2-One CoAP proxy/server Configuration

The reverse proxy shown in the Figure 3 is assumed as CoAP/CoAP proxy. Test operator includes an interface (it can be a CoAP client) that creates the stimulus to initiate the tests for reverse proxy.

More clearly, there exists two methods to create the stimulus for reverse proxy.

1. Reverse proxy can provide a direct interface to create and launch the stimulus
2. A CoAP client can be connected to reverse proxy to create and launch the stimulus for the tests

In the both cases, reverse proxy and client equally act as point of observation.

7 CoAP Scenarios

This section describes the different test scenarios. To ensure the good execution of these scenarios, it is assumed that the following settings are applied before each test execution:

- Each equipment under test shall be configured with a unicast address
- Client cache shall be cleaned up
- Use of ETag option shall be avoided except if explicitly stated in the test description, but implementation should be prepared to handle it
- Use of Token option shall be avoided except if explicitly stated in the test description, but implementation should be prepared to handle it
- Use of Piggybacked responses shall be preferred unless stated otherwise in the test description

7.1 CoAP protocol

Interoperability Test Description			
Identifier:	TD_COAP_CORE_01		
Objective:	Perform GET transaction (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 1.2, 2.1, 2.2, 3.1		
Pre-test conditions:	Server offers the resource /test with resource content is not empty that handles GET with an arbitrary payload		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a GET request with: • Type = 0 (CON) • Code = 1 (GET)
	2	Check	The request sent by the client contains: • Type=0 and Code=1 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the received information

Interoperability Test Description			
Identifier:	TD_COAP_CORE_02		
Objective:	Perform DELETE transaction (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.4, 1.2, 2.1, 2.2, 3.1		
Pre-test conditions:	Server offers a /test resource that handles DELETE		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a DELETE request with: • Type = 0 (CON)

			• Code = 4 (DELETE)
	2	Check	The request sent by the client contains: • Type=0 and Code=4 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Check	Server sends response containing: • Code = 2.02 (Deleted) • Message ID = CMID, Token = CTOK • Content-format option if payload non-empty • Empty or non-empty Payload
	4	Verify	Client displays the received information

Interoperability Test Description			
Identifier:	TD_COAP_CORE_03		
Objective:	Perform PUT transaction (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.3, 1.2, 2.1, 2.2, 3.1		
Pre-test conditions:	Server offers already available resource /test or accepts creation of new resource on /test that handles PUT		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a PUT request with: • Type = 0 (CON) • Code = 3 (PUT) • Content-format option • Empty or non-empty Payload
	2	Check	The request sent by the client contains: • Type=0 and Code=3 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Verify	Server displays received information
	4	Check	Server sends response containing: • Code = 2.04 (Changed) or 2.01 (Created) • Message ID = CMID, Token = CTOK • Content-format option if payload non-empty

			• Empty or non-empty Payload
	5	Verify	Client displays the received response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_04		
Objective:	Perform POST transaction (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.2, 1.2, 2.1, 2.2, 3.1		
Pre-test conditions:	Server accepts POST request on /test		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a POST request with: • Type = 0 (CON) • Code = 2(POST) • Content-format option • Empty or non-empty Payload
	2	Check	The request sent by the client contains: • Type=0 and Code=2 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Verify	Server displays received information
	4	Check	Server sends response containing: • Code = 2.01 (Created) or 2.04 (Changed) • Message ID = CMID, Token = CTOK • Content-format option if payload non-empty • Zero or more Location-path options • Empty or non-empty Payload
	5	Verify	Client displays the received response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_05		
Objective:	Perform GET transaction (NON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.2.3		
Pre-test conditions:	Server offers a /test resource with resource content is not empty that handles GET		
Test Sequence:	Step	Type	Description

	1	Stimulus	Client is requested to send a GET request with: • Type = 1 (NON) • Code = 1 (GET)
	2	Check	The request sent by the client contains: • Type=1 and Code=1 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Check	Server sends response containing: • Type = 1 (NON) • Code = 2.05 (Content) • Server-generated Message ID (➔ SMID) • Token = CTOK • Content-format option
	4	Verify	Client displays the received information

Interoperability Test Description			
Identifier:	TD_COAP_CORE_06		
Objective:	Perform DELETE transaction (NON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.4, 5.2.3		
Pre-test conditions:	Server offers a /test resource that handles DELETE		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a DELETE request with: • Type = 1 (NON) • Code = 4 (DELETE)
	2	Check	The request sent by the client contains: • Type=1 and Code=4 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Check	Server sends response containing: • Type = 1 (NON) • Code = 2.02 (Deleted) • Server-generated Message ID (➔ SMID)

			• Token = CTOK • Content-format option if payload non-empty • Empty or non-empty Payload
	4	Verify	Client displays the received information

Interoperability Test Description			
Identifier:	TD_COAP_CORE_07		
Objective:	Perform PUT transaction (NON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.3, 5.2.3		
Pre-test conditions:	Server offers a /test resource that handles PUT		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a PUT request with: • Type = 1 (NON) • Code = 3 (PUT) • An arbitrary payload • Content-format option
	2	Check	The request sent by the client contains: • Type=1 and Code=3 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Verify	Server displays the received information
	4	Check	Server sends response containing: • Type = 1 (NON) • Code = 2.04 (Changed) or 2.01 (Created) • Server-generated Message ID (➔ SMID) • Token = CTOK • Content-format option if payload non-empty • Empty or non-empty Payload
	5	Verify	Client displays the received response

Interoperability Test Description	
Identifier:	TD_COAP_CORE_08
Objective:	Perform POST transaction (NON mode)
Configuration:	CoAP_CFG_BASIC
References:	[COAP] 5.8.2, 5.2.3
Pre-test	Server accepts POST request on /test

conditions:			
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a POST request with: • Type = 1 (NON) • Code = 2(POST) • An arbitrary payload • Content-format option
	2	Check	The request sent by the client contains: • Type=1 and Code=2 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Verify	Server displays the received information
	4	Check	Server sends response containing: • Type = 1 (NON) • Code = 2.01 (Created) or 2.04 (Changed) • Server-generated Message ID (➔ SMID) • Token = CTOK • Zero or more Location-path options • Content-format option if payload non-empty • Empty or non-empty Payload
	5	Verify	Client displays the received response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_09		
Objective:	Perform GET transaction with separate response (CON mode, no piggyback)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.2.2		
Pre-test conditions:	Server offers a resource /separate which is not served immediately and which therefore is not acknowledged in a piggybacked way.		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request to server’s resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID)

	3	Check	Server sends response containing: • Type = 2 (ACK) • Code = 0 • Message ID = CMID • Empty Payload
Some time (a couple of seconds) elapses.			
	4	Check	Server sends response containing: • Type = 0 (CON) • Code = 2.05 (Content) • Server-generated Message ID (➔ SMID) • Token = CTOK • Content-format option • Non-empty Payload
	5	Check	Client sends response containing: • Type = 2 (ACK) • Code = 0 • Message ID = SMID • Empty Payload
	6	Verify	Client displays the response
Notes:	Steps 3 and 4 may occur out-of-order		

Interoperability Test Description			
Identifier:	TD_COAP_CORE_10		
Objective:	Perform GET transaction containing non-empty Token (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 2.2, 5.8.1, 5.10.1		
Pre-test conditions:	Server offers a /test resource with resource content is not empty that handles GET		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a GET request to server's resource with non-empty Token option
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • Length of the token should be between 1 to 8 Bytes

	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_11		
Objective:	Perform GET transaction containing non-empty Token with a separate response (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 2.2, 5.2.2, 5.8.1		
Pre-test conditions:	Server offers a resource /separate which is not served immediately and which therefore is not acknowledged in a piggybacked way.		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a GET request to server's resource including Token option
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • Length of the token should be between 1 to 8 Bytes
	3	Check	Server sends response containing: • Type = 2 (ACK) • Code = 0 • Message ID = CMID • Empty Payload
Some time (a couple of seconds) elapses.			
	4	Check	Server sends response containing: • Type = 0 (CON) • Code = 2.05 (Content) • Server-generated Message ID (➔ SMID) • Token = CTOK • Non-empty Payload
	5	Check	Client sends response containing:

			• Type = 2 (ACK) • Code = 0 • Message ID = SMID • Empty Payload
	6	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_12		
Objective:	Perform GET transaction using empty Token (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 2.2, 5.8.1, 5.10.1		
Pre-test conditions:	Server offers the resource /test with resource content is not empty that handles GET with an arbitrary payload		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request using zero-length Token to server's resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Zero-Length Token ➔ CTOK
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the response
Notes:	Not all clients may be able to send a zero-length Token		

Interoperability Test Description			
Identifier:	TD_COAP_CORE_13		
Objective:	Perform GET transaction containing several URI-Path options (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.4.5, 5.10.2, 6.5		
Pre-test conditions:	Server offers a /seg1/seg2/seg3 resource with resource content is not empty		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request to

			server’s resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) and three options of type Uri-Path, with the values: • seg1 • seg2 • seg3
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_14		
Objective:	Perform GET transaction containing several URI-Query options (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.4.5, 5.10.2, 6.5		
Pre-test conditions:	Server offers a /query resource with resource content is not empty		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request with three Query parameters (e.g. ?first=1&second=2&third=3) to the server’s resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) and two options of Uri-Query, with values such as: • first=1 • second=2

	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_15		
Objective:	Perform GET transaction (CON mode, piggybacked response) in a lossy context		
Configuration:	CoAP_CFG_LOSSY		
References:	[COAP] 4.4.1, 5.2.1, 5.8.1		
Pre-test conditions:	• Gateway is introduced and configured to produce packet losses • Server offers a /test resource with resource content is not empty that can handle GET		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request to server's resource
	2	Check	Sent request must contain: • Type = 0 • Code = 1 • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Content-format option • Non-empty Payload
	4	Verify	Client displays the response
	5	Check	Repeat steps 1-4 until at least one of the following actions has been observed: • One dropped request • One dropped response
	6	Verify	• For each case mentioned in step 5: • Observe that retransmission is launched

Interoperability Test Description			
Identifier:	TD_COAP_CORE_16		
Objective:	Perform GET transaction (CON mode, delayed response) in a lossy context		
Configuration:	CoAP_CFG_LOSSY		
References:	[COAP] 4.4.1, 5.2.2, 5.8.1		
Pre-test conditions:	• Gateway is introduced and configured to produce packet losses • Server offers a resource /separate which is not served immediately and which therefore is not acknowledged in a piggybacked way.		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable GET request to server's resource
	2	Check	The requested sent by the client contains: • Type = 0 • Code = 1 • Client-generated Message ID (➔ CMID)
	3	Check	Server sends response containing: • Type = 2 (ACK) • Code = 0 • Message ID = CMID • Empty Payload
	4	Check	Server sends response containing: • Type = 0 (CON) • Code = 2.05 (Content) • Server-generated Message ID (➔ SMID) • Non-empty Payload • Content-format option
	5	Check	Client sends response containing: • Type = 2 (ACK) • Code = 0 • Message ID = SMID • Empty Payload
	6	Verify	Client displays the response
	7	Check	Repeat steps 1-6 until at least one of the following actions has been observed: • One dropped request

			• One dropped request ACK • One dropped response • One dropped response ACK and its retransmission
	8	Verify	• For each case mentioned in step 7: • Observe that retransmission is launched

Interoperability Test Description			
Identifier:	TD_COAP_CORE_17		
Objective:	Perform GET transaction with a separate response (NON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 2.2, 5.2.2, 5.8.1		
Pre-test conditions:	Server offers a resource /separate which is not served immediately and which therefore is not acknowledged in a piggybacked way.		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a non-confirmable GET request to server's resource
	2	Check	The request sent by the client contains: • Type = 1 (NON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID)
	3	Check	Server DOES NOT send response containing: • Type = 2 (ACK) • Same message ID as in the request in step 2 • Empty Payload
Some time (a couple of seconds) elapses.			
	4	Check	Server sends response containing: • Type = 1 (NON) • Code = 2.05 (Content) • Server-generated Message ID (➔ SMID) • Content-format option • Non-empty Payload
	5	Verify	Client displays the response

Interoperability Test Description	
Identifier:	TD_COAP_CORE_18
Objective:	Perform POST transaction with responses containing several Location-Path options (CON mode)

Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.8, 5.9.1.1		
Pre-test conditions:	Server accepts creation of new resource on /test and the created resource is located at /location1/location2/location3 (resource does not exist yet)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable POST request to server's resource
	2	Check	The request sent by the client contains: • Type = 0 (CON • Code = 2 (POST) • An arbitrary payload • Content-format option
	3	Check	Server sends response containing: • Code = 2.01 (Created) and three options of type Location-Path, with the values (none of which contains a "/"): • location1 • location2 • location3
	4	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_19		
Objective:	Perform POST transaction with responses containing several Location-Query options (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.8, 5.9.1.1		
Pre-test conditions:	Server accepts creation of new resource on uri /location-query, the location of the created resource contains two query parameters ?first=1&second=2		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a confirmable POST request to server's resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 2 (POST) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK)

			• Content-format option • Empty or non-empty Payload
	3	Check	Server sends response containing: • Code = 2.01 (Created) • Message ID = CMID, Token = CTOK • Content-format option if payload non-empty • Zero or more Location-path options • Empty or non-empty Payload and two options of type Location-Query, with the values (none of which contains a "?" or "&"): • first=1 • second=2
	4	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_20		
Objective:	Perform GET transaction containing the Accept option (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.5, 5.10.4		
Pre-test conditions:	Server should provide a resource /multi-format which exists in two formats: • text/plain;charset=utf-8 • application/xml		
Test Sequence:	Step	Type	Description
client requests a resource in text format			
	1	Stimulus	Client is requested to send a confirmable GET request to server's resource
	2	Check	The request sent request by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • Option type = Accept, value = 0 (text/plain;charset=utf-8)
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK

			• Option type = Content-Format, value = 0 (text/plain;charset=utf-8) • Payload = Content of the requested resource in text/plain;charset=utf-8 format
	4	Verify	Client displays the response
client requests a resource in xml format			
	5	Stimulus	Client is requested to send a confirmable GET request to server's resource
	6	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Another client-generated Message ID ≠ CMID (➔ CMID2) • Client-generated Token which may or may not be ≠ CTOK (➔ CTOK2) • Option type = Accept, value = 41 (application/xml)
	7	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID2, Token = CTOK2 • Option type = Content-Format, value = 41 (application/xml) Payload = Content of the requested resource in application/xml format
	8	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_21		
Objective:	Perform GET transaction containing the ETag option (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.7, 5.10.10, 12.1.12		
Pre-test conditions:	• Server should offer a /validate resource which may be made to vary over time • Client & server supports ETag option • The Client's cache must be purged		
Test Sequence:	Step	Type	Description
Verifying that client cache is empty			
	1	Stimulus	Client is requested to send a confirmable GET request to server's resource

	2	Check	The request sent request by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • No ETag option
	3	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Option type = ETag, value = a value chosen by the server (➔ ETAG1) • Non-empty Payload
	4	Verify	Client displays the response
Verifying client cache entry is still valid			
	5	Stimulus	Client is requested to send a confirmable GET request to server's resource so as to check if the resource was updated
	6	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Another client-generated Message ID ≠ CMID (➔ CMID2) • Client-generated Token which may or may not be ≠ CTOK (➔ CTOK2) • Option Type = ETag, value = ETAG1 (the ETag value received in step 3)
	7	Check	Server sends response containing: • Code = 2.03 (Valid) • Message ID = CMID2, Token = CTOK2 • Option type = ETag, value = ETAG1 • Empty Payload
	8	Verify	Client displays the response
Verifying that client cache entry is no longer valid			
	9	Stimulus	Update the content of the server's resource from a CoAP client
	10	Stimulus	Client is requested to send a confirmable GET request to server's resource so as to check if the resource was updated

	11	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Another client-generated Message ID ≠ CMID and ≠ CMID2 (➔ CMID3) • Client-generated Token which may or may not be ≠ CTOK or CTOK2 (➔ CTOK3) • Option Type = ETag, value = ETAG1 (the ETag value received in step 3)
	12	Check	Server sends response containing: • Code = 2.05 (Content) • Message ID = CMID3, Token = CTOK3 • Option type = ETag, value = another ETag value ≠ ETAG1 • The payload of the requested resource, which should be different from the payload in step 3
	13	Verify	Client displays the response

Interoperability Test Description			
Identifier:	TD_COAP_CORE_22		
Objective:	Perform GET transaction with responses containing the ETag option and requests containing the If-Match option (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.7, 5.10.9, 12.1.12		
Pre-test conditions:	• Server offers a /validate resource • Client & server supports ETag and If-Match option • The Client ‘s cache must be purged		
Test Sequence:	Step	Type	Description
client gets the resource			
	1	Stimulus	Client is requested to send a confirmable GET request to server’s resource
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • No ETag option
	3	Check	Server sends response containing:

			• Code = 2.05 (Content) • Message ID = CMID, Token = CTOK • Option type = ETag, value = a value chosen by the server (➔ ETAG1) • Non-empty Payload
single update			
	4	Stimulus	Client is requested to send a confirmable PUT request to server's resource so as to perform an atomic update
	5	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 3 (PUT) • Another client-generated Message ID ≠ CMID (➔ CMID2) • Client-generated Token which may or may not be ≠ CTOK (➔ CTOK2) • Option type = If-Match, value = ETAG1 (ETag value received in step 3) • An arbitrary payload (which differs from the payload received in step 3)
	6	Check	Server sends response containing: • Code = 2.04 (Changed) • Message ID = CMID2, Token = CTOK2 • Content-format option if payload non-empty • Empty or non-empty Payload
	7	Verify	Client displays the response and the server changed its resource
concurrent updates			
	8	Stimulus	Client is requested to send a confirmable GET request to server's resource
	9	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 1 (GET) • Another client-generated Message ID ≠ CMID and ≠ CMID2 (➔ CMID3) • Client-generated Token which may or may not be ≠ CTOK or CTOK2 (➔ CTOK3)
	10	Check	Server sends response containing: • Code = 2.05 (Content)

			• Message ID = CMID3, Token = CTOK3 • Option type = ETag, value = a value ≠ ETAG1 chosen by the server (➔ ETAG2) • The Payload sent in step 5
	11	Verify	Client displays the response
	12	Stimulus	Update the content of the server's resource from a CoAP client
	13	Stimulus	Client is requested to send a confirmable PUT request to server's resource so as to perform an atomic update
	14	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 3 (PUT) • Another client-generated Message ID ≠ CMID, CMID2, CMID3 (➔ CMID4) • Client-generated Token which may or may not be ≠ CTOK, CTOK2, CTOK3 (➔ CTOK4) • Option type = If-Match, value = ETAG2 (ETag value received in step 10) • An arbitrary payload (which differs from the previous payloads)
	15	Check	Server sends response containing: • Code = 4.12 (Precondition Failed) • Message ID = CMID4, Token = CTOK4 • Optional Content-format option • Empty or non-empty Payload
	16	Verify	Client displays the response and the server did not update the content of the resource

Interoperability Test Description			
Identifier:	TD_COAP_CORE_23		
Objective:	Perform PUT transaction containing the If-None-Match option (CON mode)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP] 5.8.1, 5.10.7, 5.10.10, 12.1.12		
Pre-test conditions:	• Server offers a /create1 resource, which does not exist and can be created by the client • Client & server support If-Non-Match		
Test Sequence:	Step	Type	Description
single creation			
	1	Stimulus	Client is requested to send a confirmable PUT request to

			server's resource so as to atomically create the resource.
	2	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 3 (PUT) • Client-generated Message ID (➔ CMID) • Client-generated Token (➔ CTOK) • Option Type=If-None-Match • An arbitrary payload
	3	Check	Server sends response containing: • Code = 2.01 (Created) • Message ID = CMID, Token = CTOK • Content-format option if payload non-empty • Empty or non-empty Payload
	4	Verify	Client displays the response and the server created a new resource
concurrent creations			
	5	Stimulus	Client is requested to send a confirmable PUT request to server's resource so as to atomically create the resource.
	6	Check	The request sent by the client contains: • Type = 0 (CON) • Code = 3 (PUT) • Another client-generated Message ID ≠ CMID (➔ CMID2) • Client-generated Token which may or may not be ≠ CTOK (➔ CTOK2) • Option Type=If-None-Match • An arbitrary payload
	7	Check	Server sends response containing: • Code = 4.12 (Precondition Failed) • Message ID = CMID2, Token = CTOK2 • Optional Content-format option • Empty or non-empty Payload
	8	Verify	Client displays the response

Interoperability Test Description	
Identifier:	TD_COAP_CORE_31
Objective:	Perform CoAP Ping (CON mode)
Configuration:	CoAP_CFG_BASIC
References:	[COAP] 4.3

Pre-test conditions:	(Should work with any CoAP server)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a "Ping" request with: • Type = 0 (CON) • Code = 0 (empty)
	2	Check	The request sent by the client is four bytes and contains: • Type=0 and Code=0 • Client-generated Message ID (➔ CMID) • Zero-length Token • No payload
	3	Check	Server sends four-byte RST response containing: • Type=3 and Code=0 • Message ID = CMID • Zero-length Token • No payload
	4	Verify	Client displays that the "Ping" was successful

7.2 CoRE Link Format

Interoperability Test Description			
Identifier:	TD_COAP_LINK_01		
Objective:	Access to well-known interface for resource discovery		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK]		
Pre-test conditions:	• Client and server supports CoRE Link Format • Server supports /.well-known/core resource and the CoRE Link Format		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s list of resource
	2	Check	Client sends a GET request to Server for /.well-known/core resource
	3	Check	• Server sends response containing: • Content-format option indicating 40

			(application/link-format) • Code indicating 2.05 (Content) • Payload indicating all the links available on Server
	4	Verify	Client displays the list of resources available on Server

Interoperability Test Description			
Identifier:	TD_COAP_LINK_02		
Objective:	Use filtered requests for limiting discovery results		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 4.1		
Pre-test conditions:	• Client supports CoRE Link Format • Server supports CoRE Link Format • Server offers different types of resources (Type1, Type2, ...; see Note)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's list of resource of a specific type Type1
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating "rt=Type1"
	3	Check	• Server sends response containing: • Content- format option indicating 40 (application/link-format) Payload indicating only the links of type Type1 available on Server
	4	Verify	Client displays the list of resources of type Type1 available on Server
Notes:	Type1, Type2, ... refer to real resource types available on Server and shall be extracted from Server's /.well-known/core resource		

Interoperability Test Description			
Identifier:	TD_COAP_LINK_03		
Objective:	Handle empty prefix value strings		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 4.1 §2		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers different types of resources (Type1, Type2, ...; see Note) • Server offers resources with no type (i.e. no rt attribute)		
Test	Step	Type	Description

Sequence:			
	1	Stimulus	Client is requested to retrieve Server’s list of resources matching an rt empty value
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating rt=“*”
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) • Payload indicating only the links having an rt attribute
	4	Verify	Client displays the list of resources with rt attribute available on Server
Notes:	Type1, Type2, ... refer to real resource types available on Server and shall be extracted from Server’s /.well-known/core resource		

Interoperability Test Description			
Identifier:	TD_COAP_LINK_04		
Objective:	Filter discovery results in presence of multiple rt attributes		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 3.1, 4.1 §2		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers 4 groups of resources: o Resources with rt=“Type1 Type2” o Resources with rt=“Type2 Type3” o Resources with rt=“Type1 Type3” o Resources with rt=“”		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s list of resources of a specific type Type2
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating rt=“Type2”
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format)

			• Payload indicating only the links of groups 1 and 2
	4	Verify	Client displays the list of resources of type Type2 available on Server

Interoperability Test Description			
Identifier:	TD_COAP_LINK_05		
Objective:	Filter discovery results using if attribute and prefix value strings		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 3.2, 4.1 §5		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers 4 groups of resources: o Resources with if=“If1” o Resources with if=“If2” o Resources with if=“foo” o Resources with no if attribute		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s list of resources matching the interface description pattern “If*”
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating if=“If*”
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) • Payload indicating only the links of groups 1 and 2
	4	Verify	Client displays the retrieved list of resources

Interoperability Test Description	
Identifier:	TD_COAP_LINK_06
Objective:	Filter discovery results using sz attribute and prefix value strings
Configuration:	CoAP_CFG_BASIC
References:	[LINK] 3.3, 4.1 §5

Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers resource with sz attribute • Server offers resources with no sz attribute		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's list of resources having a sz attribute
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating sz="*"
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) • Payload indicating only the links having a sz attribute
	4	Verify	Client displays the retrieved list of resources

Interoperability Test Description			
Identifier:	TD_COAP_LINK_07		
Objective:	Filter discovery results using href attribute and complete value strings		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 4.1		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers resources /link1 /link2 and /link3		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve the link-value anchored at /link1
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating href="/link1"
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) • Payload indicating only the link for /link1
	4	Verify	Client displays the retrieved list of resources

Interoperability Test Description

Identifier:	TD_COAP_LINK_08		
Objective:	Filter discovery results using href attribute and prefix value strings		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 4.1		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers resources /link1 /link2 and /link3 • Server offers resource /test		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve the link-value anchored at /link*
	2	Check	Client sends a GET request to Server for /.well-known/core resource containing URI-Query indicating href="/link*"
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) • Payload indicating only the link matching /link*
	4	Verify	Client displays the retrieved list of resources

Interoperability Test Description			
Identifier:	TD_COAP_LINK_09		
Objective:	Arrange link descriptions hierarchically		
Configuration:	CoAP_CFG_BASIC		
References:	[LINK] 5 §4		
Pre-test conditions:	• Client supports Core Link Format • Server supports Core Link Format • Server offers an entry located at /path with ct=40 • Server offers sub-resources /path/sub1, /path/sub2, … (see Note)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve one of the sub-resources
	2	Check	Client sends a GET request to Server for /.well-known/core resource
	3	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) Payload indicating the link description for /path
	4	Check	Client sends a GET request for /path to Server

	5	Check	• Server sends response containing: • Content-format option indicating 40 (application/link-format) Payload indicating the link description for /path/sub1, /path/sub2, …
	6	Check	Client sends a GET request for /path/sub1
	7	Check	• Server sends 2.05 (Content) response. • Payload contains /path/sub1
	8	Verify	Client displays the retrieved sub-resource.
Notes:	/path/sub1, /path/sub2, … refer to real resources available on Server and shall be extracted from Server's /.well-known/core resource		

7.3 Blockwise transfers

Interoperability Test Description			
Identifier:	TD_COAP_BLOCK_01		
Objective:	Handle GET blockwise transfer for large resource (early negotiation)		
Configuration:	CoAP_CFG_BASIC		
References:	[BLOCK] 2.2–2.4		
Pre-test conditions:	• Client supports Block2 transfers • Server supports Block2 transfers • Server offers a large resource /large • Client knows /large requires block transfer		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve resource /large
	2	Check	Client sends a GET request. The request contains a Block2 option indicating: • NUM = 0; • M = 0; • SZX (➔DES_SZX) is the desired block size.
	3	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = 0; • M = 1; • SZX (➔ACT_SZX) is less than or equal to

			DES_SZX. Payload size is 2**(SZX+4) bytes.
Start of loop			
	4	Check	Client send GET requests for further blocks indicating: • NUM = i where “i” is the block number of the current block; • M = 0; • SZX is ACT_SZX.
	5	Check	Server sends 2.05 (Content) response containing Block2 option indicating: • NUM = i where “i” is the block number used at step 4; • M = 1; • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
end of loop; final slice:			
	6	Check	Client send GET request for the last block indicating: • NUM = n where “n” is the last block number; • M = 0; • SZX is ACT_SZX.
	7	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = n where “n” is the block number used at step 6; • M = 0; • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	8	Verify	Client displays the received information (no gaps, right order)
Notes:	Steps 4 and 5 are in a loop.		

Interoperability Test Description	
Identifier:	TD_COAP_BLOCK_02
Objective:	Handle GET blockwise transfer for large resource (late negotiation)
Configuration:	CoAP_CFG_BASIC
References:	[BLOCK] 2.2–2.4
Pre-test conditions:	• Client supports Block2 transfers • Server supports Block2 transfers

	• Server offers a large resource /large • Client does not know /large requires block transfer		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve resource /large
	2	Check	Client sends a GET request not containing a Block2 option
	3	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = 0; • M = 1; • SZX (➔DES_SZX) is the desired block size. Payload size is 2**(SZX+4) bytes.
	4	Check	Client switches to blockwise transfer mode and sends a GET request with a Block2 option indicating: • NUM is the next block number k = (2**(DES_SZX – ACT_SZX)); • M = 0; • SZX (➔ACT_SZX) is less than or equal to DES_SZX.
	5	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = k where “k” is the block number used at step 4; • M = 1; • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
Start of loop			
	6	Check	Client sends GET request for further blocks indicating: • NUM = i where “i” is the block number of the current block; • M = 0; • SZX is ACT_SZX.
	7	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = i where “i” is the block number used at step 6; • M = 1;

			• SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
end of loop; final slice:			
	8	Check	Client send GET request for the last block indicating: • NUM = n where “n” is the last block number; • M = 0; • SZX is ACT_SZX.
	9	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = n where “n” is the block number used at step 8; • M = 0; • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	10	Verify	Client displays the received information
Notes:	Steps 6 and 7 are in a loop.		

Interoperability Test Description			
Identifier:	TD_COAP_BLOCK_03		
Objective:	Handle PUT blockwise transfer for large resource		
Configuration:	CoAP_CFG_BASIC		
References:	[BLOCK] 2.2, 2.3, 2.5		
Pre-test conditions:	• Client supports Block1 transfers • Server supports Block1 transfers • Server offers a large updatable resource /large-update		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to update resource /large-update on Server
	2	Check	Client sends a PUT request containing Block1 option indicating: • NUM = 0; • M = 1; • SZX (➔DES_SZX) is the desired block size. Payload size is 2**(SZX+4) bytes.
	3	Check	Server sends 2.04 (Changed) response with a Block1 option indicating:

			• NUM = 0; • M = 0 (stateless) or 1 (atomic); • SZX (➔ACT_SZX) is less than or equal to DES_SZX.
Start of loop			
	4	Check	Client sends further requests containing Block1 option indicating: • NUM = i where “i” is the block number of the current block. If the server decreased the SZX parameter in step 3, then the client needs to adapt the block size accordingly and resume the transfer from block number 2**(ACT_SZX – DES_SZX) instead of block 1. • M = 1; • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
	5	Check	Server sends 2.04 (Changed) response containing Block1 option indicating: • NUM = i where “i” is the block number used at step 4; • M = 0 (stateless) or 1 (atomic); • SZX is ACT_SZX.
end of loop; final slice:			
	6	Check	Client send PUT request containing the last block and indicating: • NUM = n where “n” is the last block number; • M = 0; • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	7	Check	Server sends 2.04 (Changed) response with a Block1 option indicating: • NUM = n where “n” is the block number used at step 6; • M = 0; • SZX is ACT_SZX.
	8	Verify	Server indicates presence of the complete updated resource /large-update
Notes:	Steps 4 and 5 are in a loop.		

Interoperability Test Description			
Identifier:	TD_COAP_BLOCK_04		
Objective:	Handle POST blockwise transfer for creating large resource		
Configuration:	CoAP_CFG_BASIC		
References:	[BLOCK] 2.2, 2.3, 2.5		
Pre-test conditions:	• Client supports Block1 transfers • Server supports Block1 transfers • Server accepts creation of new resources on /large-create		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to create a new resource /large-create on Server
	2	Check	Client sends a POST request containing Block1 option indicating: • NUM = 0; • M = 1 (more); • SZX (➔DES_SZX) is the desired block size. Payload size is 2**(SZX+4) bytes.
	3	Check	Server sends 2.31 (Continue) response containing Block1 option indicating: • NUM = 0; • M = 1 (atomic); • SZX (➔ACT_SZX) is less or equal to DES_SZX.
Start of loop			
	4	Check	Client sends further POST requests containing Block1 option indicating: • NUM = i where "i" is the block number of the current block. If the server decreased the SZX parameter in step 3, then the client needs to adapt the block size accordingly and resumes the transfer from block number 2**(DES_SZX - ACT_SZX) instead of block 1. • M = 1 (more); • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
	5	Check	Server sends 2.31 (Continue) response containing Block1 option indicating: • NUM = i where "i" is the block number used at step 4;

			• M = 1 (atomic); • SZX is ACT_SZX
end of loop; final slice:			
	6	Check	Client sends POST request containing the last block and indicating: • NUM = n where "n" is the last block number; • M = 0 (final); • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	7	Check	Server sends 2.01 (Created) response containing Block1 option indicating: • NUM = n where "n" is the block number used at step 6; • M = 0 (final); • SZX is ACT_SZX. and two Location-Path options • First option value must contain "large-create" • Second option value is a (single) path segment chosen by the server (PS) • none of the Location-Path options contain a '/'
	8	Verify	Client displays the response
	9	Verify	Server indicates presence of the complete new resource /large-create/PS
verify resource creation (optional):			
	10	Check	Client sends GET request to /large-create/PS (i.e., using Uri-Path options simpley copied from the Location-Path of step 7)
	11	Check	Server sends 2.05 (Content) response with representation of created resource, potentially making use of the Block2 protocol
	12	Verify	Client indicates the value of the newly created resource
Notes:	Steps 4 and 5 are in a loop.		

Interoperability Test Description	
Identifier:	TD_COAP_BLOCK_05
Objective:	Handle POST with two-way blockwise transfer
Configuration:	CoAP_CFG_BASIC
References:	[BLOCK] 2.2, 2.3, 2.5
Pre-test	• Client supports Block1 and Block2 transfers

conditions:	• Server supports Block1 and Block2 transfers • Server accepts large post requests on /large-post		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send a large represenation to /large-post on Server
	2	Check	Client sends a POST request containing Block1 option indicating: • NUM = 0; • M = 1 (more); • SZX (➔DES_SZX) is the desired block size. Payload size is 2**(SZX+4) bytes.
	3	Check	Server sends 2.31 (Continue) response containing Block1 option indicating: • NUM = 0; • M = 1 (atomic); • SZX (➔ACT_SZX) is less or equal to DES_SZX.
Start of loop			
	4	Check	Client sends further POST requests containing Block1 option indicating: • NUM = i where "i" is the block number of the current block. If the server decreased the SZX parameter in step 3, then the client needs to adapt the block size accordingly and resumes the transfer from block number 2**(DES_SZX - ACT_SZX) instead of block 1. • M = 1 (more); • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
	5	Check	Server sends 2.31 (Continue) response containing Block1 option indicating: • NUM = i where "i" is the block number used at step 4; • M = 1 (atomic); • SZX is ACT_SZX
end of loop; final request slice and first response slice:			
	6	Check	Client sends POST request containing the last block and indicating:

			• NUM = n where “n” is the last block number; • M = 0 (final); • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	7	Check	Server sends 2.04 (Changed) response containing Block1 option indicating: • NUM = n where “n” is the block number used at step 6; • M = 0 (final); • SZX is ACT_SZX. and a Block2 option indicating: • NUM = 0 • M = 1 (more); • SZX (➔rDES_SZX) is the desired block size. Payload size is 2**(SZX+4) bytes.
	8	Check	Client switches to blockwise retrieval of response and sends a POST request, with the same options except for Block1, without payload, with a Block2 option indicating: • NUM is the next block number k = (2**(rDES_SZX – rACT_SZX)); • M = 0; • SZX (➔rACT_SZX) is less than or equal to rDES_SZX.
	9	Check	Server sends 2.04 (Changed) response with a Block2 option indicating: • NUM = k where “k” is the block number used at step 8; • M = 1; • SZX is rACT_SZX. Payload size is 2**(SZX+4) bytes.
Start of retrieval loop			
	10	Check	Client sends a similar POST request for retrieving a further block indicating: • NUM = i where “i” is the block number of the current block; • M = 0; • SZX is rACT_SZX.

	11	Check	Server sends 2.04 (Changed) response with a Block2 option indicating: • NUM = i where “i” is the block number used at step 10; • M = 1; • SZX is rACT_SZX. Payload size is 2**(SZX+4) bytes.
end of retrieval loop; final slice:			
	12	Check	Client sends another POST request (which will retrieve the last block) indicating: • NUM = n where “n” is the last block number; • M = 0; • SZX is rACT_SZX.
	13	Check	Server sends 2.04 (Changed) response with a Block2 option indicating: • NUM = n where “n” is the block number used at step 12; • M = 0; • SZX is rACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.
	14	Verify	Client displays the response
Notes:	• Steps 4 and 5 are in a loop. • Steps 10 and 11 are in a loop. • There is no initiative change in block-13.		

Interoperability Test Description			
Identifier:	TD_COAP_BLOCK_06		
Objective:	Handle GET blockwise transfer for large resource (early negotiation, 16 byte block size)		
Configuration:	CoAP_CFG_BASIC		
References:	[BLOCK] 2.2–2.4		
Pre-test conditions:	• Client supports Block2 transfers • Server supports Block2 transfers • Server offers a large resource /large • Client knows /large requires block transfer		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve resource /large
	2	Check	Client sends a GET request. The request contains a Block2

			option indicating: • NUM = 0; • M = 0; • SZX (➔DES_SZX) is the desired block size.
	3	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = 0; • M = 1; • SZX (➔ACT_SZX) is less than or equal to DES_SZX. Payload size is 2**(SZX+4) bytes.
Start of loop			
	4	Check	Client send GET requests for further blocks indicating: • NUM = i where "i" is the block number of the current block; • M = 0; • SZX is ACT_SZX.
	5	Check	Server sends 2.05 (Content) response containing Block2 option indicating: • NUM = i where "i" is the block number used at step 4; • M = 1; • SZX is ACT_SZX. Payload size is 2**(SZX+4) bytes.
end of loop; final slice:			
	6	Check	Client send GET request for the last block indicating: • NUM = n where "n" is the last block number; • M = 0; • SZX is ACT_SZX.
	7	Check	Server sends 2.05 (Content) response with a Block2 option indicating: • NUM = n where "n" is the block number used at step 6; • M = 0; • SZX is ACT_SZX. Payload size is less than or equal to 2**(SZX+4) bytes.

	8	Verify	Client displays the received information (no gaps, right order)
Notes:	Steps 4 and 5 are in a loop.		

7.4 Observing Resources

Interoperability Test Description			
Identifier:	TD_COAP_OBS_01		
Objective:	Handle resource observation with CON messages		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 1.2, 3, 4		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
After some time elapses, repeatedly:			
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content)

			• Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Check	Client sends an ACK
	6	Verify	Client displays the received information
Notes:	Steps 4-6 are in a loop.		

Interoperability Test Description			
Identifier:	TD_COAP_OBS_02		
Objective:	Handle resource observation with NON messages		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 1.2, 3, 4		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs-non which changes periodically (e.g. every 5s) which produces non-confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a non-confirmable GET request with observe option for resource /obs-non
	2	Check	The request sent by client contains: • Type = 1 (NON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
After some time elapses, repeatedly:			
	3	Check	Server sends a notification containing: • Type = 1 (NON) • Code = 2.05 (Content) • Content-format = the same for all notifications • Token value = t, same as one found in the step 2 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	4	Verify	Client displays the received information
Notes:	• Steps 3-4 are in a loop. • We don't run the test long enough to invoke the 24-hour rule in		

	[OBSERVE] 4.5, but in step 4 the server could still occasionally send a confirmable message, which then needs to be acknowledged by the client • (The request in step 2 could as well be a confirmable request.)

Interoperability Test Description			
Identifier:	TD_COAP_OBS_04		
Objective:	Client detection of deregistration (Max-Age)		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 3.3.1 §4		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 2 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Verify	Client displays the received information
	6	Check	Client sends an ACK

forcibly remove the observation relationship from the server			
	7	Stimulus	Server is rebooted or in another way caused to lose its observation state
	8	Check	Server does not send notifications
	9	Verify	Client does not display updated information
Client re-registers			
	10	Verify	After a while (see note) the client internally decides to send another GET request to the server with observe option for resource /obs, using Token t again to confirm the registration
	11	Verify	Client sends a GET request to the server for resource /obs: • Type = 0 (CON) • Code = 1 (GET) • Token value = t, same as one found in the step 2 • Observe option = empty
	12	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number that is not necessarily increasing
	13	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = same as one found in the step 12 • Token value = t, same as one found in the step 2 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	14	Verify	Client displays the received information
	15	Check	Client sends an ACK
Notes:	• Steps 4-6 are in a loop. • Step 7-9 are asynchronous to the loop 4-6. • Steps 13-15 are in a loop. • A new registration should be attempted after Max-Age + MAX_LATENCY as recommended by [OBSERVE]. MAX_LATENCY is defined by [COAP] and set to 100 seconds.		

Interoperability Test Description

Identifier:	TD_COAP_OBS_05		
Objective:	Server detection of deregistration (client OFF)		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 4.5 item 2 (see also ticket #350)		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Check	Client displays the received information
	6	Check	Client sends an ACK
Cause a timeout			
	7	Stimulus	Client is switched off
	8	Check	• Server's confirmable notifications are not acknowledged • Server's retransmissions have an updated Observe

			option value
	9	Verify	Server can keep retransmitting the responses for a while, but stops transmitting notifications after a final timeout
Notes:	• Steps 4-6 are in a loop. • Step 7-9 are asynchronous to the loop.		

Interoperability Test Description			
Identifier:	TD_COAP_OBS_06		
Objective:	Server detection of deregistration (explicit RST)		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 4.2 item 2		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24),

			unless more than 256 seconds elapsed
	5	Check	Client displays the received information
	6	Check	Client sends an ACK
Cause an RST			
	7	Stimulus	Client is rebooted
	8	Check	Server is still sending notifications for the request in step 2 as in step 4
	9	Verify	Client discards response and does not display information
	10	Check	Client sends RST to Server
	11	Verify	Server does not send further response
	12	Verify	Client does not display further received information
Notes:	• Steps 4-6 are in a loop. • Step 7-12 are asynchronous to the loop.		

Interoperability Test Description			
Identifier:	TD_COAP_OBS_07		
Objective:	Server cleans the observers list on DELETE		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 3.2 §2		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number

	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Check	Client displays the received information
	6	Check	Client sends an ACK
	7	Stimulus	Delete the /obs resource of the server (either locally or by having another CoAP client perform a DELETE request)
	8	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 4.04 (Not Found) • Token value = t, same as one found in the step 2 • No Observe option any more
	9	Verify	Server does not send further notification
	10	Verify	Client does not display further received information
Notes:	• Steps 4-6 are in a loop. • Step 7-10 are asynchronous to the loop.		

Interoperability Test Description			
Identifier:	TD_COAP_OBS_08		
Objective:	Server cleans the observers list when observed resource content-format changes		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 4.2 §3		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET)

			• Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Check	Client displays the received information
	6	Check	Client sends an ACK
	7	Stimulus	Update the /obs resource of the server's resource with a new payload having a different Content-Format (either locally or by having another CoAP client perform a DELETE request)
	8	Check	Server sends notification containing: • Type = 0 (CON) • Code = 5.00 (Internal Server Error) • Token value = t, same as one found in the step 2 • No Observe option any more
	9	Verify	Server does not send further notifications
	10	Verify	Client does not display further received information
Notes:	• Steps 4-6 are in a loop. • Step 7-10 are asynchronous to the loop.		

Interoperability Test Description	
Identifier:	TD_COAP_OBS_09
Objective:	Update of the observed resource
Configuration:	CoAP_CFG_BASIC
References:	[OBSERVE] 4.2 §3

Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 3 • Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed
	5	Check	Client displays the received information
	6	Check	Client sends an ACK
	7	Stimulus	Update the /obs resource of the server's resource with a new payload having the same Content-Format (either locally or by having another CoAP client perform a DELETE request)
	8	Check	Server notifications contains: • Type = 0 (CON) • Code = 2.05 (Content) • Content-format = f, same as one found in the step 3 • Token value = t, same as one found in the step 2

			• Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed • Payload = the new value sent at step 7
	9	Verify	Client displays the new value of /obs sent in step 8
	10	Check	Client sends an ACK
Notes:	• Steps 4-6 are in a loop. • Step 7-9 are asynchronous to the loop 4-6. • Steps 8-10 are in a loop (the same loop at steps 4-6 but /obs is updated).		

Interoperability Test Description			
Identifier:	TD_COAP_OBS_10		
Objective:	GET does not cancel resource observation		
Configuration:	CoAP_CFG_BASIC		
References:	[OBSERVE] 1.2, 3, 4		
Pre-test conditions:	• Client supports Observe option • Server supports Observe option • Server offers an observable resource /obs which changes periodically (e.g. every 5s) which produces confirmable notifications		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to send to the server a confirmable GET request with observe option for resource /obs
	2	Check	The request sent by client contains: • Type = 0 (CON) • Code = 1 (GET) • Token value ➔ t, a value generated by the client • Observe option = empty
	3	Check	Server sends the response containing: • Type = 2 (ACK) • Code = 2.05 (Content) • Content-format of the resource /obs, ➔ f • Token value = t, same as one found in the step 2 • Observe option with a sequence number
After some time elapses, repeatedly:			
	4	Check	Server sends a notification containing: • Type = 0 (CON) • Code = 2.05 (Content)

			<ul><li>Content-format = f, same as one found in the step 3</li><li>Token value = t, same as one found in the step 3</li><li>Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed</li></ul>
	5	Check	Client sends an ACK
	6	Verify	Client displays the received information
Perform an unrelated GET			
	7	Stimulus	Client is requested to send to the server a confirmable GET request *without* observe option for resource /obs
	8	Check	The request sent by client contains:<ul><li>Type = 0 (CON)</li><li>Code = 1 (GET)</li><li>Token value ➔ t2, a value generated by the client ≠ t</li><li>*No* Observe option</li></ul>
	9	Check	Server sends the response containing:<ul><li>Type = 2 (ACK)</li><li>Code = 2.05 (Content)</li><li>Content-format of the resource /obs, = f</li><li>Token value = t2, same as one found in the step 8</li><li>*No* Observe option</li></ul>
After some time elapses, the notifications still arrive:			
	10	Check	Server sends a notification containing:<ul><li>Type = 0 (CON)</li><li>Code = 2.05 (Content)</li><li>Content-format = f, same as one found in the step 3</li><li>Token value = t, same as one found in the step 3</li><li>Observe option indicating increasing values (sequence number arithmetic modulo 2**24), unless more than 256 seconds elapsed</li></ul>
	11	Check	Client sends an ACK
	12	Verify	Client displays the received information
Notes:	Steps 4-6 and 10-12 are in a loop.		

8 DTLS Scenarios

Interoperability Test Description			
Identifier:	TD_COAP_DTLS_01		
Objective:	Basic DTLS PSK (success case)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP]		
Pre-test conditions:	• Client and server support DTLS PSK with TLS_PSK_WITH_AES_128_CCM_8 • Server listens for DTLS connections on port 5684 • Server has been set up to accept PSK "sesame" on PSK identity "password" (ASCII strings without quotes as byte strings) • Client has been set up to use PSK "sesame" on PSK identity "password" • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_PSK_WITH_AES_128_CCM_8 • server selects TLS_PSK_WITH_AES_128_CCM_8 in ServerHello • DTLS setup is successful and leads to the exchange of Finished messages
	3	Check	Client sends a GET request to Server for /test resource
	4	Check	• Server sends response containing: • Code indicating 2.05 (Content) • Payload as set up on the Server
	5	Verify	Client displays the received information

Interoperability Test Description	
Identifier:	TD_COAP_DTLS_02
Objective:	Basic DTLS PSK (failure case — wrong PSK)
Configuration:	CoAP_CFG_BASIC
References:	[COAP]

Pre-test conditions:	• Client and server support DTLS PSK with TLS_PSK_WITH_AES_128_CCM_8 • Server listens for DTLS connections on port 5684 • Server has been set up to accept PSK "sesame" on PSK identity "password" (ASCII strings without quotes as byte strings) • Client has been set up to use PSK "wrong" on PSK identity "password" • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_PSK_WITH_AES_128_CCM_8 • server selects TLS_PSK_WITH_AES_128_CCM_8 in ServerHello • DTLS setup fails and leads to an Alert message (decrypt_error)
	3	Verify	Client displays error indication

Interoperability Test Description			
Identifier:	TD_COAP_DTLS_03		
Objective:	Lossy DTLS PSK (success case)		
Configuration:	CoAP_CFG_LOSSY		
References:	[COAP]		
Pre-test conditions:	• Client and server support DTLS PSK with TLS_PSK_WITH_AES_128_CCM_8 • Server listens for DTLS connections on port 5684 • Server has been set up to accept PSK "sesame" on PSK identity "password" (ASCII strings without quotes as byte strings) • Client has been set up to use PSK "sesame" on PSK identity "password" • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01) • Gateway is introduced and configured to produce packet losses		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_PSK_WITH_AES_128_CCM_8

			• server selects TLS_PSK_WITH_AES_128_CCM_8 in ServerHello • DTLS setup is successful and leads to the exchange of Finished messages
	3	Check	Client sends a GET request to Server for /test resource
	4	Check	• Server sends response containing: • Code indicating 2.05 (Content) • Payload as set up on the Server
	5	Verify	Client displays the received information
	6	Stimulus	Repeat steps 1-5 until at least one of each of the DTLS handshake packets in a normal interchange has been lost
	7	Verify	• For each packet loss case mentioned in step 6: • Observe that retransmission is launched

Interoperability Test Description			
Identifier:	TD_COAP_DTLS_04		
Objective:	Basic DTLS RPK (success case)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP]		
Pre-test conditions:	• Client and server support DTLS RPK (using 122 for the client_certificate_type and 123 for the server_certificate_type) with TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (using 0xC0 0xAC as the cipher suite identifier) • Server listens for DTLS connections on port 5684 • Server has been set up to accept a raw public key RPK_C of key type ECDSA defined by the client • Client has been set up to use RPK_C as its client_certificate • Client has been set up to accept a raw public key RPK_S of key type ECDSA defined by the server • Server has been set up to use RPK_S as its server_certificate • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 • server selects TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 in ServerHello

			• DTLS setup is successful and leads to the exchange of Finished messages
	3	Check	Client sends a GET request to Server for /test resource
	4	Check	• Server sends response containing: • Code indicating 2.05 (Content) • Payload as set up on the Server
	5	Verify	Client displays the received information

Interoperability Test Description			
Identifier:	TD_COAP_DTLS_05		
Objective:	Basic DTLS RPK (client failure case)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP]		
Pre-test conditions:	• Client and server support DTLS RPK (using 122 for the client_certificate_type and 123 for the server_certificate_type) with TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (using 0xC0 0xAC as the cipher suite identifier) • Server listens for DTLS connections on port 5684 • Server has been set up to accept a raw public key RPK_C of key type ECDSA defined by the client • Client has been set up to use RPK_C as its client_certificate • Client has *NOT* been set up to accept a raw public key RPK_S of key type ECDSA defined by the server but does require server authentication • Server has been set up to use RPK_S as its server_certificate • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 • server selects TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 in ServerHello • DTLS setup fails and leads to an Alert message (certificate_unknown)
	3	Verify	Client displays error indication

Interoperability Test Description			
Identifier:	TD_COAP_DTLS_06		
Objective:	Basic DTLS RPK (server failure case)		
Configuration:	CoAP_CFG_BASIC		
References:	[COAP]		
Pre-test conditions:	• Client and server support DTLS RPK (using 122 for the client_certificate_type and 123 for the server_certificate_type) with TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (using 0xC0 0xAC as the cipher suite identifier) • Server listens for DTLS connections on port 5684 • Server has *NOT* been set up to accept a raw public key RPK_C of key type ECDSA defined by the client but does require client authentication • Client has been set up to use RPK_C as its client_certificate • Client has been set up to accept a raw public key RPK_S of key type ECDSA defined by the server • Server has been set up to use RPK_S as its server_certificate • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01)		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server’s resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 • server selects TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 in ServerHello • DTLS setup fails and leads to an Alert message (certificate_unknown)
	3	Verify	Client displays error indication

Interoperability Test Description	
Identifier:	TD_COAP_DTLS_07
Objective:	Lossy DTLS RPK (success case)
Configuration:	CoAP_CFG_LOSSY
References:	[COAP]
Pre-test conditions:	• Client and server support DTLS RPK (using 122 for the client_certificate_type and 123 for the server_certificate_type) with TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (using 0xC0 0xAC as the cipher suite identifier) • Server listens for DTLS connections on port 5684 • Server has been set up to accept a raw public key RPK_C of key type ECDSA defined by the client

	• Client has been set up to use RPK_C as its client_certificate • Client has been set up to accept a raw public key RPK_S of key type ECDSA defined by the server • Server has been set up to use RPK_S as its server_certificate • Server offers the resource coaps://.../secure with a non-empty representation available upon GET, but only in DTLS-secured connections (coap://.../secure, if available, might lead to 4.01) • Gateway is introduced and configured to produce packet losses		
Test Sequence:	Step	Type	Description
	1	Stimulus	Client is requested to retrieve Server's resource /secure
	2	Check	• Client opens a DTLS connection to server • cipher_suites in ClientHello contains TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 • server selects TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 in ServerHello • DTLS setup is successful and leads to the exchange of Finished messages
	3	Check	Client sends a GET request to Server for /test resource
	4	Check	• Server sends response containing: • Code indicating 2.05 (Content) • Payload as set up on the Server
	5	Verify	Client displays the received information
	6	Stimulus	Repeat steps 1-5 until at least one of each of the DTLS handshake packets in a normal interchange has been lost
	7	Verify	• For each packet loss case mentioned in step 6: • Observe that retransmission is launched

9 OMA Lightweight M2M Scenarios

Table 9: LWM2M Tests

1	Registration	LightweightM2M-1.0-int-101 – Initial Registration
2		LightweightM2M-1.0-int-102 – Registration Update
3		LightweightM2M-1.0-int-103 – Deregistration
4	Device object-related use cases	Querying basic information from the client
5		Querying the firmware version from the client
6		Rebooting the device
7		Querying power status of the terminal
8	Device firmware update	LightweightM2M-1.0-int-301 – Firmware update (via COAP)
9		LightweightM2M-1.0-int-302 – Firmware update (via alternative mechanism)
10	Connectivity object monitoring	LightweightM2M-1.0-int-401 – Querying of connectivity parameters
11	Observe and Notify	LightweightM2M-1.0-int-501 – Observation and notification of parameter values inside MachineLink 3G

The Test descriptions of the above tests are defined in the document OMA-ETS-LightweightM2M-V1_0-20131017-D

Change History

Document history		
0.0.1	15.11.2013	First Draft
0.0.2	18.11.2013	Updated with DTLS test cases
0.0.3	18.11.2013	Added OMA LWM2M test cases list
0.0.5	18.11.2013	Version used at the Plugtests