USP – The User Services Platform

Issue: 1 Amendment 2 (release-1.2-20)

Issue Date: November 2022

- Executive Summary
- Purpose
- Scope
- Test Setup
  - Test Equipment
  - Test Setup and Execution
  - Universal Test Metrics
  - Notes about test case descriptions
- 1 Messages and Path Names
  - 1.1 Add message with allow partial false, single object, required parameters succeed
  - 1.2 Add message with allow partial true, single object, required parameters succeed
  - 1.3 Add message with allow partial false, single object, required parameters fail
  - 1.4 Add message with allow partial false, single invalid object
  - 1.5 Add message with allow partial false, multiple objects
  - 1.6 Add message with allow partial false, multiple objects with an invalid object
  - 1.7 Add message with allow partial false, multiple objects, required parameters fail in single object
  - 1.8 Add message with allow partial true, required parameters fail, invalid type, single object
  - 1.9 Add message with allow partial true, required parameters fail, multiple objects
  - 1.10 Add message with unique key addressing in path
  - 1.11 Set message with allow partial false, required parameters pass
  - 1.12 Set message with allow partial true, required parameters pass
  - 1.13 Set message with allow partial false, multiple objects
  - 1.14 Set message with allow partial false, required parameters fail
  - 1.15 Set message with allow partial false, multiple objects, required parameters fail in single object
  - 1.16 Set message with allow partial true, required parameter fails, multiple objects
  - 1.17 Set message with allow partial true, non-required parameter fails, multiple parameters
  - 1.18 Set message with unique key addressing in path
  - 1.19 Set message with wildcard search path, allow partial false, required parameters pass
  - 1.20 Set message with wildcard search path, allow partial false, required parameters fail

- 1.21 Set message with wildcard search path, allow partial true, required parameters fail
- 1.22 Set message with search expression search path
- 1.23 Set message with path that matches no objects
- 1.24 Delete message with allow partial false, valid object instance
- 1.25 Delete message with allow partial false, object instance doesn’t exist
- 1.26 Delete message with allow partial false, invalid object
- 1.27 Delete message with allow partial false, multiple objects
- 1.28 Delete message with allow partial false, multiple objects, invalid object
- 1.29 Delete message with allow partial true, object instance doesn’t exist
- 1.30 Delete message with allow partial true, invalid object
- 1.31 Delete message with allow partial true, multiple objects, invalid object
- 1.32 Delete message with allow partial true, multiple objects, object doesn’t exist
- 1.33 Delete message with unique key addressing
- 1.34 Delete message with wildcard search path, valid objects
- 1.35 Delete message with search expression search path
- 1.36 Get message with full parameter path
- 1.37 Get message with multiple full parameter paths, same object
- 1.38 Get message with multiple full parameter paths, different objects
- 1.39 Get message with object path
- 1.40 Get message with object instance path
- 1.41 Get message with invalid parameter
- 1.42 Get message with invalid parameter and valid parameter
- 1.43 Get message using unique key addressing
- 1.44 Get message using wildcard search path on full parameter
- 1.45 Get message using wildcard search path on object path
- 1.46 Get message using search expression search path (equivalence)
- 1.47 Get message using search expression search path (non-equivalence)
- 1.48 Get message using search expression search path (exclusive greater comparison)
- 1.49 Get message using search expression search path (exclusive lesser comparison)
- 1.50 Get message using search expression search path (inclusive greater comparison)
- 1.51 Get message using search expression search path (inclusive lesser comparison)
- 1.52 Notify - Subscription creation using Value Change
- 1.53 Notify - Subscription Deletion Using Value Change
- 1.54 Notification Retry using Value Change
- 1.55 Subscription Expiration using Value Change
- 1.56 Notification Retry Expiration using Value Change
- 1.57 ObjectCreation Notification
- 1.58 ObjectDeletion Notification
- 1.59 Event Notification using Periodic!
- 1.60 OnBoardRequest Notification
- 1.61 Operate message using Reboot() with send_resp true

- 1.62 Operate message using Reboot() with send_resp false
- 1.63 Operate message using input arguments (DEPRECATED by 1.79)
- 1.64 Asynchronous operation with send_resp true
- 1.65 Asynchronous operation with send_resp false
- 1.66 GetInstances using a single object, first_level_only true
- 1.67 GetInstances using a single object, first_level_only false
- 1.68 GetInstances with multiple objects
- *1.69 DELETED*
- 1.70 GetInstances with wildcard search path
- 1.71 GetInstances with search expression search path
- 1.72 GetSupportedDM using a single object, first_level_only false, all options
- 1.73 GetSupportedDM using a single object, first_level_only true, all options
- 1.74 GetSupportedDM using a single object, first_level_only true, no options
- 1.75 GetSupportedDM using multiple objects, first_level_only true, all options
- 1.76 GetSupportedDM on root object, all options
- 1.77 GetSupportedDM on unsupported object
- 1.78 Removal of subscriptions that have no associated controller
- 1.79 Operate message using input arguments
- 1.80 GetSupportedProtocol
- 1.81 Automatic unique key generation
- 1.82 Get message with unmatched search expression
- 1.83 GetInstances message with unmatched search expression
- 1.84 Notification - Subscription using search paths
- 1.85 (For future work) Get message with unresolved instances - addressing by instance number
- 1.86 Get message with unresolved instances - using a search path
- 1.87 Get message with unresolved instances - using an object path
- 1.88 Add message fails when unique key is invalid
- 1.89 Get message using max_depth
- 1.90 Delete message with search expression that matches no objects
- 1.91 Unknown arguments in an Operate message
- 1.92 Agent uses default values for Operate arguments

- 2 Authentication and Access Control Test Cases
  - 2.1 Agent does not accept messages from its own Endpoint ID
  - 2.2 Agent rejects messages that do not contain its to_id in the USP Record
  - 2.3 Agent does not process messages without ’s certificate information - DEPRECATED
  - 2.4 Agent rejects messages from Endpoint IDs that are not in subjectAltName - DEPRECATED
  - 2.5 Agent use of self-signed certificates - DEPRECATED
  - 2.6 Connecting without absolute time
  - 2.7 Agent ignores unsigned or invalid Record signatures
  - 2.8 Agent ignores invalid TLS certificate

  - 2.9 Use of the Untrusted role
  - 2.10 Adding a Role
  - 2.11 Permissions - Object Creation Allowed
  - 2.12 Permissions - Object Creation Not Allowed
  - 2.13 Permissions - Object Deletion Allowed
  - 2.14 Permissions - Object Deletion Not Allowed
  - 2.15 Permissions - Parameter Update Allowed
  - 2.16 Permissions - Parameter Update Not Allowed
  - 2.17 Permissions - Operation Allowed
  - 2.18 Permissions - Operation Not Allowed
  - 2.19 Permissions - Value Change Notification Allowed on Parameter
  - 2.20 Permissions - Value Change Notification Not Allowed on Parameter
  - 2.21 Permissions - Overlapping Permissions
  - 2.22 Using Get when no read permissions are available on some parameters
- 3 USP Record Test Cases
  - 3.1 Bad request outside a session context
  - 3.2 Agent Verifies Non-Payload Field Integrity
  - 3.3 Agent rejects invalid signature starting a session context
  - 3.4 Using TLS for USP Record Integrity
  - 3.5 Failure to Establish TLS
  - 3.6 Agent does not accept TLS renegotiation for E2E message exchange
  - 3.7 Use of X.509 Certificates
  - 3.8 Establishing a Session Context
  - 3.9 Receipt of a Record out of a Session Context
  - 3.10 Session Context Expiration
  - 3.11 Use of Sequence ID and Expected ID
  - 3.12 Preservation of USP Records
  - 3.13 Agent Rejects Records with Different Payload Security than the Established Context
  - 3.14 Use of retransmit_id
  - 3.15 Handling Duplicate Records
- 4 General MTP Test Cases
  - 4.1 Use of X.509 certificates at the MTP layer
- 5 CoAP Test Cases (DEPRECATED)
  - 5.1 Mapping a USP Record to a CoAP message (DEPRECATED)
  - 5.2 USP Records that exceed CoAP message size (DEPRECATED)
  - 5.3 Successful CoAP exchange (DEPRECATED)
  - 5.4 Failed CoAP exchange - timeout (DEPRECATED)
  - 5.5 Failed CoAP Exchange - Invalid Method (DEPRECATED)
  - 5.6 Failed CoAP Exchange - Invalid Content-Format (DEPRECATED)
  - 5.7 Failed CoAP Exchange - Invalid USP Record (DEPRECATED)

  - 5.8 Use of DTLS (DEPRECATED)
- 6 STOMP Test Cases
  - 6.1 Support of Required Profiles
  - 6.2 STOMP session establishment
  - 6.3 STOMP Connection Retry
  - 6.4 Successful USP message over STOMP with required headers
  - 6.5 STOMP destination - provided in subscribe-dest
  - 6.6 STOMP destination - configured in USP data model
  - 6.7 STOMP Destination - terminates unconfigured session
  - 6.8 Use of STOMP heartbeat mechanism
  - 6.9 Error Handling - Unprocessed Record
  - 6.10 Agent’s STOMP destination is changed
  - 6.11 STOMP - Use of TLS
  - 6.12 STOMP - Use of Connect Record
- 7 WebSocket Test Cases
  - 7.1 Session Establishment
  - 7.2 Use of only one session
  - 7.3 Agent session acceptance from Controller
  - 7.4 Closing a WebSocket Connection
  - 7.5 Rejection of Session Establishment
  - 7.6 Error Handling - Unprocessed Records
  - 7.7 Use of Ping and Pong frames
  - 7.8 WebSocket Session Retry
  - 7.9 Use of TLS
  - 7.10 WebSocket - Use of Connect Record
- 8 Discovery Test Cases
  - 8.1 DHCP Discovery - Agent Request Requirements
  - 8.2 DHCP Discovery - Agent handling of received options
  - 8.3 DHCP Discovery - FQDN Leads to DNS Query
  - 8.4 mDNS
  - 8.5 mDNS and Message Transfer Protocols
  - 8.6 DNS - DNS Record Requirements
  - 8.7 mDNS request response
- 9 Functionality Test Cases
  - 9.1 Use of the Timer! Event (DEPRECATED by 9.11)
  - 9.2 Use of Device.LocalAgent.AddCertificate()
  - 9.3 Upgraded the Agent’s Firmware - Autoactivate enabled
  - 9.4 Upgrading the Agent’s Firmware - Using TimeWindow, Immediate
  - 9.5 Upgrading the Agent’s Firmware - Using TimeWindow, AnyTime
  - 9.6 Upgrading the Agent’s Firmware - Validated Firmware

- 9.7 Upgrading the Agent’s Firmware - Download to Active Bank
- 9.8 Upgrading the Agent’s Firmware - Cancelling a request using the Cancel() command
- 9.9 Adding a New Controller - OnBoardRequest
- 9.10 Use of the Boot! event and BootParameters
- 9.11 Use of the Timer! Event

- 10 Bulk Data Collection Test Cases
  - 10.1 Use BulkData collection using HTTP and JSON
  - 10.2 Use BulkData collection using HTTPS and JSON
  - 10.3 Use BulkData collection using HTTP and CSV
  - 10.4 Use BulkData collection using HTTPS and CSV
  - 10.5 Use BulkData collection using HTTP with URI Parameters
  - 10.6 Use BulkData collection using HTTPS with URI Parameters
  - 10.7 BulkData collection retry mechanism over HTTP
  - 10.8 Use BulkData collection using HTTP with wildcard parameter
  - 10.9 Use BulkData collection using HTTP with Object Path
  - 10.10 Use BulkData collection Push event
  - 10.11 Use BulkData collection Push event with Wildcard path
  - 10.12 Use BulkData collection Push event with Object path
  - 10.13 Use BulkData collection over MQTT
- 11 MQTT Test Cases
  - 11.1 Support of Required MQTT Profiles
  - 11.2 MQTT session establishment using a CONNECT packet
  - 11.3 MQTT Use of TLS
  - 11.4 MQTT 5.0 ClientID
  - 11.5 MQTT ClientID Persistence
  - 11.6 MQTT Message Retry
  - 11.7 MQTT Keep Alive
  - 11.8 MQTT SUBSCRIBE Packet
  - 11.9 MQTT New Subscription
  - 11.10 MQTT No Topic in CONNACK
  - 11.11 MQTT Failure to Subscribe
  - 11.12 MQTT PUBLISH Packet
  - 11.13 MQTT QoS
  - 11.14 MQTT Reply to Topic
  - 11.15 MQTT 5.0 Content Type
  - 11.16 MQTT Connection Retry
  - 11.17 MQTT - Use of Connect Record

List of Figures

1. Basic Test Setup

List of Tables

Notice

Intellectual Property

Terms of Use

Issue History

Issue Number	Changes
Release 1.0	• First release of this test plan, containing test cases for basic compliance with TR-369/USP
Release 1.0.1	• Deprecated test 7.3 • Added flag to the features list to indicate which features are “not-in-force” and not yet available for certification • Various procedure and metric fixes
Release 1.0.2	• Both mandatory and conditional mandatory tests can use alternate objects or parameters if available • Updated features and requirements • Altered test setup of 1.50 to include three objects • Fixed metrics of 1.25 to use new DeleteResp logic • Fixed tests 1.16 and 1.21 to use the correct error codes • Fixed test 1.20 to only check for at least one error • Fixed test 4.1 to include “OnBoardRequest()” as a conditional requirement • Fixed test 1.32 to use new DeleteResp logic • Fixed tests 1.73, 1.74, 1.75 to use new GetSupportedDM first_level_only logic • Fixed error code metric of test 1.8
Release 1.0.3	• The metrics of test 1.9 now do not imply order • Test 1.22 metric now requires “at least one” element rather than a “single” element • Renamed test 1.23 • Fixed the metric of test 1.38 to include the instance identifier of the path • Reworded the purpose of test 1.41 • Clarified the test setup of tests 1.66 and 1.67 • Fixed a typo in test 6.1 metrics (ServerRetryInitialMultiplier to ServerRetryIntervalMultiplier) • Fixes other typographical errors

Release 1.1	• Adds test 1.78 Removal of subscriptions that have no associated controller • Adds test 1.80 GetSupportedProtocol • Adds language reinforcing required tests based on supported features • Adds test case 1.81 to test automatic unique key generation by the Agent • Adds negative test metrics to several Set and Delete tests to validate that operations did not occur upon error (1.7, 1.8, 1.14, 1.15, 1.20, 1.24, 1.27, 1.28, 1.31, 1.32, 1.33, 1.34, 1.35) • Adds section 10 for bulk data collection tests • Adds tests 1.82 and 1.83 to validate Get and GetInstances using expressions that match zero objects • Adds test 1.84 to exercise the use of search paths in Subscriptions • Defines deprecation for tests • Deprecates test 1.63 in favor of test 1.79 • Deprecates test 9.1 in favor of test 9.11 • Deprecates tests 2.3, 2.4, and 2.5 • Completely deletes test 1.69 • Updates functionality tag on test 1.71 to “supports at least one multi-instance object” • Clarifies the definition of an empty oper_success element in test 1.32 • Rewrites test 2.20 to accomplish its original intent • Changes test 3.6 to check that an Agent does not accept TLS renegotiation • Changes test 2.15 and 2.16 to use “Enable” rather than “Alias” as a test parameter • Simplifies test 4.1 • Various typographical fixes, some in test procedure path names • Normalizes all protobuf examples in test procedures • Fixes test 2.20 to use the correct permissions • Fixes test 1.47 to use the correct operator • Eases the requirements of test 9.7 for implementation flexibility • Removes the concept of “not-in-force” test cases and features • Updates tests 1.59, 1.84 to use allow_partial false” • Allows tests 1.3, 1.7, 1.8, 1.21 to use only an invalid parameter value (not an invalid parameter) and accept 7012 as an acceptable error code

Release 1.2	• Deprecates COAP tests • Adds MQTT test cases • Adds tests 1.86, 1.87 to test Get with unmatched search path • Adds new Connect Record test cases to STOMP and WebSocket sections • Adds MQTT bulk data collection test cases • Adds test 1.88 to check that unique keys are always treated as required • Adds test 1.89 to exercise max_depth feature in Get • Adds test 1.90 to check Delete with unmatched search path • Adds test 1.91 to very an Agent ignores unknown arguments in a command • Adds test 1.92 to test that operate uses default values when absent • Updates test 1.23 metrics to require the return of an empty oper_success • Updates GetSupportedDM test cases to check for new USP 1.2 fields • Updates tests 1.3, 1.4, 1.6, 1.21 to include additional test metrics • Updates tests 1.64 and 1.65 to check for correct command_key value • Updates tests to account for requirement R-MTP.5 in USP 1.2 • Fixes metric in test 1.20 and 1.21 to allow for single failure • Fixes test setup in 9.4, 9.5, 9.6 • Fixes test 8.5 to remove MTP specific language and require mDNS info • Reverts metric in test 6.7 • Fixes test 1.47 test metrics for correct equivalence • Set send_resp to true for several Operate tests where needed • Various typographical fixes

Work Area Directors

Name	Company	Email	Role
Jason Walls	QA Cafe, LLC	jason@qacafe.com	Broadband User Services Work Area Director
John Blackford	CommScope	john.blackford@commscope.com	Broadband User Services Work Area Director

Editors

Name	Company	Email	Role
Jason Walls	QA Cafe, LLC	jason@qacafe.com	Editor

Executive Summary

Testing is crucial to promoting the interoperability and adoption of standards. To meet this, the Broadband Forum regularly produces test suites that validate the conformance of implementations of their standards. This specification defines the test setup, test procedures, and test metrics to validate Agent and implementations of the User Services Platform (USP), published as BBF TR-369.

This test plan is used to validate USP Agent implementations for theBBF.369 USP Agent Certification Program. Companies looking to certify their products, or to acquire certified products, can find full details on the program, approved test tools and labs, and the list of certified products here.

Purpose

This purpose of this document is to provide a definitive guide for validating the compliance of USP Agents in accordance with the specification.

Scope

The tests defined below are intended to validate the specific requirements outlined in the USP specification, as well as those requirements defined in the Device:2 Data Model for USP Agents for objects, parameters, commands, and events necessary for the operation of USP.

Test Setup

Test Equipment

There are a number of components necessary to the implementation of this test suite.

Traffic Generator - One or more traffic generators are necessary in order to transmit the required traffic to execute the test procedures. Traffic generation can be done with script-able, real implementations of DHCP servers, mDNS endpoints, and USP endpoints (for example), or can be simulated through other means. For tests that exercise the presence of multiple Controllers or agents, the traffic generators can each represent a single endpoint, or multiple endpoints, depending on its capabilities, as long as the traffic can be differentiated by the Endpoint Under Test.

Analyzer - One or more traffic analyzers are necessary to confirm the receipt of messages and evaluate the test metrics outlined in the tests below. This analyzer may exist at the traffic generator source, in-line, or accessed through a replicated interface that will push traffic to the analyzer.

Test Network - The tests below require IP layer connectivity between the Traffic Generator and the Endpoint Under Test (EUT). Steps SHOULD be taken to unsure that the underlying network does not interfere with the test procedures or test metrics.

Test Setup and Execution

Basic Test Setup

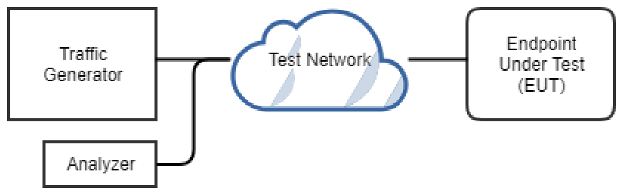

Figure 1: Basic Test Setup

Functionality Tags

USP contains both required and optional functionality. To ensure that all different classes of device can exercise this test suite, tests are marked as “Mandatory”, “Conditional Mandatory”, or “Deprecated”. This is indicated in each individual test case under the “Functionality Tag”.

Tests that are “Deprecated” represent tests that were removed or replaced with newer tests.

Alternate parameters, objects, commands, and events in test procedures

For tests that make use of particular parameter, object, command, event to validate the test metrics, a different subject can be substituted that meets the needs of the test. For example, if an EUT does not support the Reboot:1 profile, another synchronous operation can be substituted for tests 1.61 and 1.62.

Endpoint Requirements and Metadata Collection

Required Profiles

The Device:2 Data Model for USP Agents outlines several profiles that contain data model objects, parameters, commands, and events necessary to the operation of USP. In order to be able to perform the tests below, a USP Agent MUST implement, at minimum, the following profiles:

- LocalAgent:1
- Subscriptions:1

Conditional mandatory tests may require the implementation of additional profiles.

Test Cases Required by Profile and Option Support

Those seeking to utilize this test plan can use the following feature IDs to specify their support for conditional mandatory test cases. Since the types of endpoints under test may vary widely in use cases and complexity, this list is meant to act as a guide to ensure that many kinds of products can achieve compliance. However, when determining which tests must be passed to achieve compliance, testers must know that:

An Endpoint Under Test (EUT) MUST complete all Mandatory test cases, and MUST complete all Conditional Mandatory test cases for ALL features supported by the EUT (see table below).

Feature ID	Feature name	Test Cases	Notes
1	At least one command	1.61, 1.62	
2	At least one command with input arguments	1.63	
3	At least one asynchronous command	1.64, 1.65	
4	Subscription.{i}.NotifExpiration parameter	1.56	An extension to the Subscription:1 profile
5	Controller:1 profile	1.59	

6	Device.LocalAgent.Subscription.{i}.TimeToLive	1.55	
7	Controller:1 profile (writeable)	1.78, 9.9	EUT allows the creation of Device.LocalAgent.Controller.{i}. objects
8	Device.LocalAgent.Controller.{i}.SendOnBoardRequest()	1.60, 4.1, 9.9	
9	Device.ScheduleTimer()	1.63, 1.79, 9.1	
10	Reboot:1 profile	1.61, 1.62, 9.10	
11	TraceRoute:1 profile	1.64, 1.65	
12	ControllerTrust:1 profile	2.9, 2.10	
13	ControllerTrust:1 profile (writeable)	2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17, 2.18, 2.19, 2.20, 2.21, 2.22	Additionally supports at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.
14	Self-signed controller certificates	TBD	
15	TLS at the MTP Layer	4.1	
16	CoAP MTP (DEPRECATED)	5.*	
17	STOMP MTP	6.*	Excludes 6.8 unless option 18 is supported

18	STOMPHeartbeat:1 profile	6.8	
19	WebSocket MTP	7.*	Excludes 7.3 unless option 20 is supported
20	TR-369 requirement R-WS.6	7.3	
21	Discovery via DHCP Options	8.1, 8.2, 8.3	
22	Discovery via mDNS	8.4, 8.5, 8.6, 8.7	
23	Secure Message Exchange (TLS for USP Record Integrity)	3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.13	
24	USP session context	2.6, 2.7, 2.8, 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15	
25	Device.LocalAgent.AddCertificate()	9.2	
26	Firmware:1 profile	9.3, 9.5, 9.6, 9.7	

27	Firmware:1 profile (Activate)	9.4, 9.5	Supports Firmware:1 profile and additionally supports the Activate() operation
28	Device.LocalAgent.Request.{i}.Cancel()	9.8	Applies only if option 26 is supported
29	UntrustedRole disabled	TBD	The use of UntrustedRole must be either unsupported, or capable of being disabled, to run this test
30	Device.DeviceInfo.BootFirmwareImage	9.10	
31	The product supports at least one nested multi-instance object	1.10, 1.71, 1.83	
32	HTTP bulk data collection with JSON encoding	10.1, 10.2, 10.5, 10.6, 10.7, 10.8, 10.9	
33	HTTP bulk data collection with CSV encoding	10.3, 10.4, 10.5, 10.6, 10.7, 10.8, 10.9	
34	Bulk data collection via the Push! Event	10.10, 10.11, 10.12	
35	MQTT MTP, version 3.1.1	11.*	Excludes 11.4, 11.7, 11.8, 11.10, & 11.15, MQTT 5.0 only tests
36	MQTT MTP, version 5.0	11.*	

Elements Specified in the Test Procedure

Many of the mandatory and conditional mandatory tests specify the objects, parameters, or operations to be used for the test. If the specific elements are not supported by the EUT, other elements that will satisfy the test criteria MAY be used instead. If so, the test report MUST include the alternate elements used.

Required EUT Information and Resources

In order to be able to perform the tests and create a report of the results, the following must be provided concerning the Endpoint Under Test:

1. The software and/or firmware version of the EUT.
2. The number of firmware banks available if the Firmware:1 profile is supported.
3. A list of the feature IDs supported.
4. If the service elements specified in the tests are not supported, provide a list of alternate elements used in the testing.

Clean-Up Procedures

A number of tests that make changes to the EUT have procedures that are not part of the validation portion of the test case. These procedures are intended to "clean up" any changes that were made during the test to ensure that the EUT is in a relatively known state from one test case to the next. The most obvious example is using the Delete message to remove any objects that were added as part of the procedure, but the clean-up procedure may include any number of steps.

Universal Test Metrics

Due to the nature of performative testing of protocol messages, certain requirements in the specification are effectively tested every time. Writing additional test cases for these metrics is unnecessary, but the requirements must still be met by endpoint implementations.

1. The Endpoint ID of the Endpoint Under Test is valid (ARC.3, ARC.4, ARC.5, and the requirements outlined in the authority-scheme table).

2. The USP records and USP messages of the Endpoint Under Test are valid according to the usp-record.proto and usp-msg.proto schemas (ENC.0, ENC.1).

3. The Path Names and Search Paths used in messages sent by the Endpoint Under Test are valid according to Data Model Path Grammar and TR-106 (ARC.7).

4. Path Names in messages originating from the EUT use instance number addressing (R-MSG.3).

Notes about test case descriptions

Each of the test cases below have the following sections:

Purpose - The purpose describes the reasoning for the test case, based on the normative requirements defined in USP.

Functionality Tag - The functionality tag indicates whether the test is mandatory or conditional mandatory. If it is the latter, this section will list any additional Device:2 profiles necessary to the performance of the test case.

Test Setup - The test setup section indicates any special prior conditions that must be configured before performing the test.

Test Procedure - The procedure indicates the steps, in order, taken to perform the test.

Test Metrics - The metrics indicate the required behavior that must be observed to consider the test passed.

Use of examples

The test setup, procedure, and metrics in each test case may contain examples of the data to be sent to or received from the EUT. In these examples, elements that are to be filled with a known value dependent on the protocol’s behavior are indicated with greater-than/less-than brackets (<for example>), to indicate a variable. These examples should not be taken literally.

1 Messages and Path Names

1.1 Add message with allow partial false, single object, required parameters succeed

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, and all required parameters to be set upon Object Creation succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add1'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
           required: true
         }
       }
     }
   }
 }
```

2. Allow the EUT to send an AddResp.
3. Record the instance identifier of the created object as reported by the EUT.
4. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

5. Allow the EUT to send a GetResp.

6. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

7. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT sends an AddResp.

2. The AddResp contains a single CreatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains no parameter errors and 3 elements in the unique key map: Alias, Recipient, and ID. Alternatively, the OperationSuccess contains 2 elements in the unique key map if the Alias parameter is not supported: Recipient, and ID.

3. The EUT creates the Subscription object.

4. The Subscription object’s values match the values set in the`param_settings` element.

1.2 Add message with allow partial true, single object, required parameters succeed

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to true, and all required parameters to be set upon Object Creation succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: true
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add2'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
           required: true
         }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp.
3. Record the instance identifier of the created object as reported by the EUT.
4. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

5. Allow the EUT to send a GetResp.

6. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

7. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT AddResp is valid.

2. The AddResp contains a single CreatedObjectResult that has an OperationStatus of OperationSuccess. The OperationSuccess contains no parameter errors and 3 elements in the unique key map: Alias, Recipient, and ID. Alternatively, the OperationSuccess contains 2 elements in the unique key map if the Alias parameter is not supported: Recipient, and ID.

3. The EUT creates the Subscription object.

4. The Subscription object’s values match the values set in the`param_settings` element.

1.3 Add message with allow partial false, single object, required parameters fail

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, and at least one required parameter fails, and only a single object is set.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'InvalidValue'
           required: true
         }
        param_settings {
           param: 'ID'
           value: 'add3'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
         }
        }
      }
    }
  }
```

2. Allow the EUT to send an Error message.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends an Error message.
2. The Error message contains an `err_code` of 7004, ‘Invalid Arguments’, with the`param_errs` element containing a single error with a `param_path` that indicates the Enable parameter, and an `err_code` of ‘7011’ (Invalid Type) or ‘7012’ (Invalid Value).
3. The EUT did not create a new Subscription object.

1.4 Add message with allow partial false, single invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, and a single invalid object is set.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.InvalidObject.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add4'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
         }
        }
      }
    }
  }
```

2. Allow the EUT to send an Error message.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends an Error message.

2. The Error message contains an `err_code` of 7004, ‘Invalid Arguments’, with the`param_errs` element containing a single error with a `param_path` of ‘Device.LocalAgent.InvalidObject.’, and an `err_code` of 7026, ‘Invalid Path’.

3. The EUT did not create a new Subscription object.

1.5 Add message with allow partial false, multiple objects

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, multiple objects are attempted, and all required parameters to be set upon Object Creation succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add51'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
```

```
                required: true
              }
            }
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'add52'
              }
            param_settings {
                param: 'NotifType'
                value: 'ValueChange'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.EndpointID'
                required: true
              }
            }
          }
        }
      }
```

2. Allow the EUT to send an AddResp.

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier 1>.'
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier 2>.'
    }
  }
}
```

5. Allow the EUT to send a GetResp.

6. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
msg_id: '<msg_id>'
msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier 1>.'
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier 2>.'
    }
  }
}
```

7. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT AddResp is valid.

2. The AddResp contains two CreatedObjectResults that each have an OperationStatus of OperationSuccess. The OperationSuccess elements contains no parameter errors and 3 elements in the unique key map: Alias, Recipient, and ID. Alternatively, the OperationSuccess contains 2 elements in the unique key map if the Alias parameter is not supported: Recipient, and ID.

3. The EUT creates the Subscription objects.
4. The first Subscription object's values match the values set in the`param_settings` element.
5. The second Subscription object's values match the values set in the`param_settings` element.

1.6 Add message with allow partial false, multiple objects with an invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, multiple objects are attempted, and one of the objects is invalid.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add61'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
           required: true
         }
        }
      create_objs {
        obj_path: 'Device.LocalAgent.InvalidObject.'
        param_settings {
           param: 'Enable'
           value: 'true'
         }
        param_settings {
           param: 'ID'
           value: 'add62'
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.EndpointID'
           required: true
         }
          }
        }
      }
    }
```

2. Allow the EUT to send an Error.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends an Error message.

2. The Error message contains an `err_code` of 7004, ‘Invalid Arguments’, with the`param_errs` element containing a single error with a `param_path` of ‘Device.LocalAgent.InvalidObject.’, and an `err_code` of 7026, ‘Invalid Path’.

3. The EUT did not create a new Subscription object.

1.7 Add message with allow partial false, multiple objects, required parameters fail in single object

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to false, and at least one required parameter fails in one of multiple objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'InvalidValue'
           required: true
         }
        param_settings {
           param: 'ID'
           value: 'add71'
```

```
             }
            param_settings {
               param: 'NotifType'
               value: 'ValueChange'
             }
            param_settings {
               param: 'ReferenceList'
               value: 'Device.LocalAgent.SoftwareVersion'
               required: true
             }
            }
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
               param: 'Enable'
               value: 'true'
             }
            param_settings {
               param: 'ID'
               value: 'add72'
             }
            param_settings {
               param: 'NotifType'
               value: 'ValueChange'
             }
            }
          }
        }
      }
```

2. Allow the EUT to send an Error.

3. Send a Get message to the EUT with the request path of `Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends an Error message.

2. The Error message contains an `err_code` of 7004, ‘Invalid arguments’, with the `param_errs` element containing a single error with a `param_path` that indicates the Enable parameter, and an `err_code` of ‘7011’ (Invalid Type) or ‘7012’ (Invalid Value).

3. The GetResp from the EUT does not contain a Subscription instance with ID ‘add71’ or ‘add72’.

1.8 Add message with allow partial true, required parameters fail, invalid type, single object

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to true, and at least one required parameter fails (with an invalid value) in a single object.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: ADD
    }

    body {
      request {
        add {
          allow_partial: true
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
               param: 'Enable'
               value: 'InvalidValue'
               required: true
             }
            param_settings {
               param: 'ID'
               value: 'add8'
             }
            param_settings {
               param: 'NotifType'
               value: 'ValueChange'
             }
            param_settings {
               param: 'ReferenceList'
               value: 'Device.LocalAgent.SoftwareVersion'
               required: true
             }
            }
          }
        }
      }
```

2. Allow the EUT to send an AddResp.
3. Send a Get message to the EUT with the request path of ‘Device.LocalAgent.Subscription.’.

Test Metrics

1. The EUT sends an AddResp message.

2. The AddResp contains a single CreatedObjectResult that has an OperationStatus that is an element of type OperationFailure. The OperationFailure element contains an `err_code` of ‘7011’ (Invalid Type) or ‘7012’ (Invalid Value).

3. The GetResp from the EUT does not contain a Subscription instance with ID ‘add8’.

1.9 Add message with allow partial true, required parameters fail, multiple objects

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the `allow_partial` element is set to true, and at least one required parameter fails in one of multiple objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
```

```
    body {
      request {
        add {
          allow_partial: true
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'add91'
              }
            param_settings {
                param: 'NotifType'
                value: 'ValueChange'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.SoftwareVersion'
              }
            }
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'add91'
              }
            param_settings {
                param: 'NotifType'
                value: 'ValueChange'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.SoftwareVersion'
              }
            param_settings {
                param: 'InvalidParameter'
                value: 'IrrelevantValue'
                required: true
              }
            }
          }
        }
      }
```

2. Allow the EUT to send an AddResp.

3. Record the instance identifier of the created object as reported by the EUT.

4. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

5. Allow the EUT to send a GetResp.

6. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

7. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT sends an AddResp message.
2. The AddResp contains two CreatedObjectResults.
    a. One CreateObjectResult is an element of type OperationSuccess. The OperationSuccess element contains no parameter errors and 3 elements in the unique key map: Alias, Recipient, and ID. Alternatively, the OperationSuccess contains 2 elements in the unique key map if the Alias parameter is not supported: Recipient, and ID.
    b. The other CreateObjectResult is an element of type OperationFailure. The OperationFailure element contains an `err_code` of ‘7010’ (‘Unsupported Parameter’), ‘7017’ (‘Object could not be created’), or ‘7026’ (‘Invalid Path’).
3. The EUT creates the first Subscription object, and does not create the second Subscription object.
4. The Subscription object’s values match the values set in the`param_settings` element.

1.10 Add message with unique key addressing in path

Purpose

The purpose of this test is to validate that the EUT properly handles an Add message when the Controller uses unique key addressing.

Functionality Tag

Conditional Mandatory (Product supports at least one nested multi-instance object)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Controller.[EndpointID=="<
EndpointID>"&&Alias=="<Alias if supported>"].BootParameter.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ParameterName'
            value: 'Device.LocalAgent.SoftwareVersion'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp.

3. Record the instance identifier of the created object as reported by the EUT.

4. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Controller.<instance identifier of
Controller>.BootParameter.<instance identifier>.'
    }
  }
}
```

5. Allow the EUT to send a GetResp.

6. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Controller.<instance identifier of
Controller>.BootParameter.<instance identifier>.'
    }
  }
}
```

7. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT sends an AddResp.

2. The AddResp contains a single CreatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains no parameter errors and 2 elements in the unique key map: Alias and ParameterName. Alternatively, the OperationSuccess contains one element in the unique key map if the Alias parameter is not supported: ParameterName.

3. The EUT creates the BootParameter object.
4. The BootParameter object's values match the values set in the`param_settings` element.

1.11 Set message with allow partial false, required parameters pass

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to false, and all required parameters to be updated succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<instance identifier from
test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier from
test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains a single UpdatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdateInstanceResult, with the `affected_path` equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
3. The retrieved value matches the value set in the`param_settings` element.

1.12 Set message with allow partial true, required parameters pass

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to true, and all required parameters to be updated succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: true
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<instance identifier from
test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
  body {
    request {
      get {
        param_paths: 'Device.LocalAgent.Subscription.<instance identifier
from test setup>.NotifRetry'
      }
    }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains a single UpdatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdateInstanceResult, with the `affected_path` equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
3. The retrieved value matches the value set in the`param_settings` element.

1.13 Set message with allow partial false, multiple objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to false, and all required parameters to be updated succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT, and the instance identifiers are known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<second instance identifier
from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
         }
        }
      }
    }
  }
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.NotifRetry'
      param_paths: 'Device.LocalAgent.Subscription.<second instance
identifier from test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains two UpdatedObjectResults that each have an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdateInstanceResult, with the affected_path equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
3. The retrieved value matches the value set in the`param_settings` element for each object.

1.14 Set message with allow partial false, required parameters fail

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to false, and a required parameter fails.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<instance identifier from
test setup>.'
        param_settings {
         param: 'InvalidParameter'
         value: 'IrrelevantValue'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send an Error.
3. Send a Get message to the EUT with a request path of 'Device.LocalAgent.Subscription.<instance identifier from test setup>.'.

Test Metrics

1. The EUT sends an Error.
2. The Error contains `err_code` '7004', 'Invalid Arguments', and a single ParamError element. The ParamError element contains a `param_path` of 'Device.LocalAgent.Subscription.<instance identifier>.InvalidParameter' and an `err_code` of '7010', 'Unsupported Parameter'.
3. The GetResp contains a single Subscription instance that does not include a 'InvalidParameter' parameter.

1.15 Set message with allow partial false, multiple objects, required parameters fail in single object

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to false, and required parameters in one of multiple objects fail.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT, and the instance identifiers are known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
         }
      }
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<second instance identifier
from test setup>.'
        param_settings {
         param: 'InvalidParameter'
         value: 'IrrelevantValue'
         required: true
         }
        }
      }
    }
  }
```

2. Allow the EUT to send an Error.

3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends an Error.

2. The Error contains `err_code` ‘7004’, ‘Invalid Arguments’, and a single ParamError element. The ParamError element contains a `param_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.InvalidParameter’ and an `err_code` of ‘7010’, ‘Unsupported Parameter’.

3. The GetResp contains at least two Subscription instances, neither of which contain a `InvalidParameter` parameter and the first instance from the test setup does not have an updated `NotifRetry` value.

1.16 Set message with allow partial true, required parameter fails, multiple objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to true, and a required parameter on one of multiple objects fails.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT, and the instance identifiers are known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: true
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<second instance identifier
from test setup>.'
        param_settings {
         param: 'InvalidParameter'
         value: 'IrrelevantValue'
         required: true
         }
        }
      }
    }
  }
```

2. Allow the EUT to send a SetResp.
3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains two UpdatedObjectResults.
    a. The first UpdatedObjectResult has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdatedInstanceResult, with the `affected_path` equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
    b. The second UpdatedObjectResult has an OperationStatus that is an element of type OperationFailure. The OperationFailure contains an `err_code` of '7021', 'Required parameter failed', and a single UpdatedInstanceFailure element. The UpdatedInstanceFailure has an `affected_path` with a value of 'Device.LocalAgent.Subscription.<instance identifier>.', and a single ParameterError element.
    c. The ParameterError has a `param` element with a value of 'InvalidParameter' and an`err_code` of '7010', 'Unsupported Parameter'.
3. The retrieved value matches the value set in the`param_settings` element for the first object.

1.17 Set message with allow partial true, non-required parameter fails, multiple parameters

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the `allow_partial` element is set to true, and one of multiple non-required parameters fail.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: true
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
       }
        param_settings {
         param: 'InvalidParameter'
         value: 'IrrelevantValue'
       }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains a single UpdatedObjectResult with an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdatedInstanceResult element.
    a. The UpdatedInstanceResult `affected_path` is equal to 'Device.LocalAgent.Subscription.<instance number>.'.
    b. The UpdatedInstanceResult has a single entry in the `updated_params` map containing 'NotifRetry' as the key.
    c. The UpdatedInstanceResult has a single ParameterError element, with the 'param' field set to 'InvalidParameter', and an `err_code` of '7010', 'Unsupported Parameter'.
3. The retrieved value of NotifRetry matches the value set in the `param_settings` element.

1.18 Set message with unique key addressing in path

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the Controller uses unique key addressing.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the unique keys and their values

are known by the traffic generator.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {

    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.<unique key instance
identifier from test setup>.'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: GET
    }
    body {
      request {
        get {
          param_paths: 'Device.LocalAgent.Subscription.<instance identifier from
    test setup>.NotifRetry'
        }
      }
    }
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains a single UpdatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdateInstanceResult, with the `affected_path` equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
3. The retrieved value matches the value set in the`param_settings` element.

1.19 Set message with wildcard search path, allow partial false, required parameters pass

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the Controller uses a wildcard search path and the requested updates succeed.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Set message to the EUT with the following structure:

   ```
   header {
     msg_id: '<msg_id>'
     msg_type: SET
   }

   body {
     request {
       set {
         allow_partial: false
         update_objs {
           obj_path: 'Device.LocalAgent.Subscription.*.'
           param_settings {
            param: 'NotifRetry'
            value: '<Valid Value>'
            required: true
           }
         }
       }
     }
   }
   ```

2. Allow the EUT to send a SetResp.
3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<first instance identifier
from test setup>.NotifRetry'
      param_paths: 'Device.LocalAgent.Subscription.<second instance
identifier from test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains an UpdatedObjectResults element that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains at least two UpdateInstanceResults, each with the `affected_path` equal to ‘Device.LocalAgent.Subscription.<instance number>.’ of the respective instance, and a single entry in the `updated_params` map containing ‘NotifRetry’ as the key.
3. The retrieved value matches the value set in the`param_settings` element for each object.

1.20 Set message with wildcard search path, allow partial false, required parameters fail

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the Controller uses a wildcard search path, `allow_partial` element is set to false, and required parameters in multiple objects fail.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {

    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.*.'
        param_settings {
         param: 'InvalidParameter'
         value: 'IrrelevantValue'
         required: true
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an Error.
3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends an Error.
2. The Error contains an appropriate error code and at least one ParamError element. The ParamError element contains a `param_path` of ‘Device.LocalAgent.Subscription.<instance identifier of relevant object>.InvalidParameter’ and an appropriate error code.
3. In the GetResp there are no Subscription instances with an ‘InvalidParameter’ parameter.

1.21 Set message with wildcard search path, allow partial true, required parameters fail

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the Controller uses a wildcard search path, the `allow_partial` element is set to true, and a required parameter on multiple objects fails.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: true
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.*.'
        param_settings {
         param: 'Enable'
         value: 'InvalidValue'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.

2. The SetResp contains an UpdatedObjectResults element.
3. The UpdatedObjectResults has an OperationStatus that is an element of type OperationFailure. The OperationFailure contains an `err_code` of ‘7021’, ‘Required Parameter Failed’, and at least one UpdatedInstanceFailure elements. The UpdatedInstanceFailure has an `affected_path` with a value of ‘Device.LocalAgent.Subscription.<instance identifier>.’ for the respective failed instance, and a single ParameterError element. The ParameterError has a `param` element that indicates the Enable parameter, and an `err_code` of ‘7011’ (Invalid Type), or ‘7012’ (Invalid Value).
4. The EUT has no Subscription objects that have an Enable parameter set to ‘InvalidValue’.

1.22 Set message with search expression search path

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the Controller uses a search path.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT with a value for the NotifExpiration that is greater than 0.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.[NotifExpiration>0].'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier from
test setup>.NotifRetry'
    }
  }
}
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a SetResp.

2. The SetResp contains at least one UpdatedObjectResult that has an OperationStatus that is an element of type OperationSuccess. The OperationSuccess contains a single UpdateInstanceResult, with the `affected_path` equal to 'Device.LocalAgent.Subscription.<instance number>.', and a single entry in the `updated_params` map containing 'NotifRetry' as the key.
3. The retrieved value matches the value set in the`param_settings` element.

1.23 Set message with path that matches no objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Set message when the requested path is a search path that does not match any objects, returning an empty `oper_success` element.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: true
      update_objs {
        obj_path: 'Device.LocalAgent.Subscription.
[Recipient=="InvalidValue"].'
        param_settings {
         param: 'NotifRetry'
         value: '<Valid Value>'
         required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send a SetResp.

Test Metrics

1. The EUT sends a SetResp.
2. The SetResp contains one UpdatedObjectResult with an empty `oper_success` element (i.e. `oper_success {}`).

1.24 Delete message with allow partial false, valid object instance

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to false, and the object to be deleted is valid.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends a DeleteResp.
2. The DeleteResp contains a single `deleted_obj_response` with a `requested_path` equal to 'Device.LocalAgent.Subscription.<instance identifier>.' and an `oper_success` element, with one `affected_path` element equal to the path name of the Deleted object.

3. The GetResp does not contain the Subscription instance that was deleted.

1.25 Delete message with allow partial false, object instance doesn’t exist

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to false, and the object instance to be deleted does not exist.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned any existing Subscription objects and their instance identifiers.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: DELETE
     }
    body {
      request {
        delete {
          allow_partial: false
          obj_paths: 'Device.LocalAgent.Subscription.<non-existent instance
identifier>.'
        }
      }
    }
```

2. Allow the EUT to send an DeleteResp.

Test Metrics

1. The EUT sends a DeleteResp containing an empty `oper_success` element.

1.26 Delete message with allow partial false, invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to false, and the object to be deleted is invalid.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.InvalidObject.'
    }
  }
}
```

2. Allow the EUT to send an Error message.

Test Metrics

1. The EUT sends an Error message.

2. The Error contains an `err_code` of 7004, ‘Invalid arguments’, with the`param_errs` element containing a single error with a `param_path` of ‘Device.LocalAgent.InvalidObject.’, and an `err_code` of 7026, ‘InvalidPath’.

1.27 Delete message with allow partial false, multiple objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to false, with multiple valid objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT, and the instance identifiers are known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
 header {
   msg_id: '<msg_id>'
   msg_type: DELETE
  }
 body {
   request {
     delete {
       allow_partial: false
       obj_paths: 'Device.LocalAgent.Subscription.<first instance
identifier>.'
       obj_paths: 'Device.LocalAgent.Subscription.<second instance
identifier>.'
     }
   }
  }
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends a DeleteResp.

2. The DeleteResp contains two `deleted_obj_results`, each with a `requested_path` equal to the `obj_paths` of the Delete message, and an `oper_success` element containing an

`obj_paths` of the Delete message, and an `oper_success` element containing an `affected_path` element equal to the path name of the deleted object.

3. The GetResp does not contain the deleted Subscription objects.

1.28 Delete message with allow partial false, multiple objects, invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to false, and one of the objects to be deleted is invalid.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: DELETE
     }
    body {
      request {
        delete {
          allow_partial: false
          obj_paths: 'Device.LocalAgent.Subscription.<instance identifier.>'
          obj_paths: 'Device.LocalAgent.InvalidObject.'
        }
      }
    }
```

2. Allow the EUT to send an Error message.
3. Send a Get message to the EUT with a request path of `Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends an Error message.
2. The Error contains an `err_code` of 7004, ‘Invalid arguments’, with the `param_errs` element containing a single error with a `param_path` of ‘Device.LocalAgent.InvalidObject.’, and an `err_code` of 7026, ‘InvalidPath’.
3. The GetResp contains the Subscription that was not deleted by step 1.

1.29 Delete message with allow partial true, object instance doesn’t exist

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to true, and the object instance to be deleted does not exist.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
 header {
   msg_id: '<msg_id>'
   msg_type: DELETE
  }
 body {
   request {
     delete {
       allow_partial: true
       obj_paths: 'Device.LocalAgent.Subscription.<invalid instance
identifier>.'
     }
   }
 }
```

2. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT sends a DeleteResp containing an empty `oper_success` element (i.e., `oper_success{}`).

1.30 Delete message with allow partial true, invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to true, and the object is not valid in the Agent's supported data model.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: true
      obj_paths: 'Device.LocalAgent.InvalidObject.'
    }
  }
}
```

2. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT sends a DeleteResp.

2. The DeleteResp contains a single `deleted_obj_result` message with a `requested_path` of 'Device.LocalAgent.InvalidObject.' and an `oper_failure` element, with `err_code` '7026', 'Invalid Path'.

1.31 Delete message with allow partial true, multiple objects, invalid object

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to true, and one of multiple objects is invalid.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
 }
body {
  request {
    delete {
      allow_partial: true
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
      obj_paths: 'Device.LocalAgent.InvalidObject.'
    }
  }
}
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends a DeleteResp.
2. The DeleteResp contains two `deleted_obj_results` elements, one with an `oper_success` element, containing an `affected_path` element with the value ‘Device.LocalAgent.Subscription.<instance identifier>.’, and the other with an `oper_failure` element containing an `err_code` of ‘7026’, ‘InvalidPath’.
3. The GetResp does not contain the Subscription instance deleted in step 1.

1.32 Delete message with allow partial true, multiple objects, object doesn’t exist

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the `allow_partial` element is set to true, and one of multiple objects does not exist in the Agent’s instantiated data model.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the instance identifier is known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
   msg_id: '<msg_id>'
   msg_type: DELETE
  }
body {
  request {
    delete {
      allow_partial: true
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
      obj_paths: 'Device.LocalAgent.Subscription.<invalid instance
identifier>.'
      }
     }
    }
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of `Device.LocalAgent.Subscription.`.

Test Metrics

1. The EUT sends a DeleteResp.
2. The DeleteResp contains two deleted_obj_results elements. One contains an `oper_success` element with an `affected_path` element listing 'Device.LocalAgent.Subscription.<instance identifier>.'. The second contains an empty `oper_success` element (i.e., `oper_success{}`).
3. The GetResp does not contain the Subscription instance deleted in step 1.

1.33 Delete message with unique key addressing

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the Controller uses unique key addressing.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Obtain the unique key values of the Device.LocalAgent. object that correlates with the source of the test USP messages.
3. Ensure that at least one Device.LocalAgent.Controller.{i}.BootParameter. object exists on the EUT, and the instance identifier of the object is known by the traffic generator.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Controller.[EndpointID=="<
EndpointID>"&&Alias=="<Alias if supported>"].BootParameter.<instance
identifier>.'
      }
    }
  }
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Controller.<instance ID>.BootParameter.`

Test Metrics

1. The EUT sends a DeleteResp.
2. The DeleteResp contains a single `deleted_obj_result` with a requested path equal to the path specified in the `obj_path` of the Delete message, containing an`oper_success` element, with one `affected_path` element equal to the path name of the Deleted object.
3. The `affected_path` element uses instance number addressing.
4. The GetResp does not contain the BootParameter deleted in step 1.

1.34 Delete message with wildcard search path, valid objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the Controller uses a wildcard search to delete multiple valid objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.*.'
      }
    }
  }
```

2. Allow the EUT to send a DeleteResp.

3. Send a Get message to the EUT with a requested path of`Device.LocalAgent.Subscription.`

Test Metrics

1. The EUT sends a DeleteResp.

2. The DeleteResp contains a single `deleted_obj_result` with a requested path equal to 'Device.LocalAgent.Subscription.*.' and an `oper_success` element with one or more `affected_path` elements equal to the path names of the Deleted objects.

3. The GetResp does not contain any of the Subscription instances deleted in step 1.

1.35 Delete message with search expression search path

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message when the Controller uses a search expression to delete one or more valid objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the instance identifier of the Controller object that represents the traffic generator is known by the traffic generator.
3. Ensure that at least two Device.LocalAgent.Controller.<instance identifier>.BootParameter. objects exist on the EUT. At least one of these BootParameter objects has a value of 'false' for its 'Enable' parameter, and at least one of these BootParameter objects has a value of 'true' for its 'Enable' parameter.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Controller.<instance
identifier>.BootParameter.[Enable==true].'
      }
    }
  }
```

2. Allow the EUT to send a DeleteResp.
3. Send a Get message to the EUT with a requested path of `Device.LocalAgent.Controller.<instance ID>.BootParameter.`.

Test Metrics

1. The EUT sends a DeleteResp.

2. The DeleteResp contains a single `deleted_obj_results` element, with a `requested_path` equal to 'Device.LocalAgent.Controller.<instance identifier>.BootParameter.[Enable==true]' and an `oper_success` element with the `affected_path` elements equal to the path names of the successfully Deleted objects.
3. The BootParameter whose Enable parameter was equal to 'false' was not deleted.
4. The GetResp does not contain any BootParameter instances where `Enable==true`.

1.36 Get message with full parameter path

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when a single full parameter path is specified.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.EndpointID'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `req_path_results` has no errors, a `requested_path` equal to 'Device.LocalAgent.EndpointID', and contains a single `resolved_path_results` element. The `resolved_path_results` element contains a `resolved_path` equal to 'Device.LocalAgent.' and a single `result_params` element with a key of 'EndpointID' and a value equal to the EUT's EndpointID.

1.37 Get message with multiple full parameter paths, same object

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when multiple full parameter paths are specified within the same object.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.EndpointID'
      param_paths: 'Device.LocalAgent.SoftwareVersion'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains two `req_path_results` elements. The `requested_path_results` have no errors. Each contains a single `resolved_path_results` element. One `resolved_path_result` element contains a `requested_path` equal to 'Device.LocalAgent.EndpointID', a single `resolved_path` equal to 'Device.LocalAgent.', and a single `result_params` element with a key of 'EndpointID' and a value equal to the EUT EndpointID. The other `resolved_path_result` element contains a `requested_path` equal to 'Device.LocalAgent.SoftwareVersion', a single `resolved_path` equal to 'Device.LocalAgent.', and a single `result_params` element with a key of 'SoftwareVersion' with a valid value.

1.38 Get message with multiple full parameter paths, different objects

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when multiple full parameter paths are specified within multiple objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and its instance identifier is known by the traffic generator.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.EndpointID'
      param_paths: 'Device.LocalAgent.Subscription.<instance
identifier>.Enable'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.

2. The GetResp contains two `req_path_results` elements. The `requested_path_results` have no errors. Each contains a single `resolved_path_results` element. One `resolved_path_result` element contains a `requested_path` equal to 'Device.LocalAgent.EndpointID', a single `resolved_path` equal to 'Device.LocalAgent.', and a single `result_params` element with a key of 'EndpointID' and a value equal to the EUT EndpointID. The other `resolved_path_result` element contains a `requested_path` equal to 'Device.LocalAgent.Subscription.<instance identifier>.Enable', a single `resolved_path` equal to 'Device.LocalAgent.Subscription.<instance identifier>.', and a single `result_params` element with a key of 'Enable' with a valid value.

1.39 Get message with object path

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when an object path is specified.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.', and a set of `resolved_path_results` elements. One contains a `resolved_path` of 'Device.LocalAgent.', and a number of `result_params` elements contain keys and values of the parameters of 'Device.LocalAgent.'. Additional `resolved_path_results` exist for each of the sub-objects of Device.LocalAgent., with `result_params` containing the keys and values of each sub-object's parameters.
3. The keys of all `result_params` elements are relative paths.

1.40 Get message with object instance path

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when a path to an object instance is specified.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and its instance identifier is known by the traffic generator.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.<instance identifier>.', and a single `resolved_path_results` element, with a `resolved_path` of 'Device.LocalAgent.Subscription.<instance identifier>.', and a series of `result_params` elements containing the keys and values of the parameters of the instance.
3. The keys of all `result_params` elements are relative paths.

1.41 Get message with invalid parameter

Purpose

The purpose of this test is to ensure the Agent can properly handle a Get message when a single invalid parameter is requested.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.InvalidParameter'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.

2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has a `requested_path` equal to 'Device.LocalAgent.InvalidParameter', and an`err_code` of '7026', 'Invalid Path'.

1.42 Get message with invalid parameter and valid parameter

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when both a valid and invalid parameter are requested.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
       param_paths: 'Device.LocalAgent.EndpointID'
       param_paths: 'Device.LocalAgent.InvalidParameter'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains two `req_path_results` elements. One `requested_path_results` has no errors, and contains a single `resolved_path_results` element. The `resolved_path_results` element contains a `requested_path` equal to 'Device.LocalAgent.EndpointID', a single `resolved_path`equal to 'Device.LocalAgent.', and a single `result_params` element with a key of 'EndpointID' and a value equal to the EUT EndpointID. The other `requested_path_results` has a `requested_path` equal to 'Device.LocalAgent.InvalidParameter', and an `err_code` of '7026', 'Invalid Path'.

1.43 Get message using unique key addressing

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when the requested path uses unique key addressing.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure that at least one Subscription object exists on the EUT, and the unique keys and their values are known by the traffic generator.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.<unique key
identifier>.Enable'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.

2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to ‘Device.LocalAgent.Subscription.<unique key identifier>.Enable’, and a single `resolved_path_results` element, with a `resolved_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.’, and a result_params element containing a key of ‘Enable’ and a valid value.

1.44 Get message using wildcard search path on full parameter

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when the requested path uses a wildcard to retrieve a single parameter from multiple objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.*.Enable'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.

2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.*.Enable', and at least two `resolved_path_results` elements, each with a `resolved_path` of 'Device.LocalAgent.Subscription.<instance identifier>.', and a `result_params` element containing a key of 'Enable' and a valid value.

1.45 Get message using wildcard search path on object path

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when the requested path uses a wildcard to retrieve all parameters from multiple object instances.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure that at least two Subscription objects exist on the EUT.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.*.'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.*.', and a set of `resolved_path_results` elements. Each contains a `resolved_path` of 'Device.LocalAgent.Subscription.<instance identifier>.', and a number of `result_params` elements containing keys and values of the parameters of each Subscription object.
3. The keys of all `result_params` elements are relative paths.

1.46 Get message using search expression search path (equivalence)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when the requested path uses a search path to retrieve objects that that parameters that match a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘true’ for its Enable parameter, and at least one should have a value of ‘false’ for its Enable parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[Enable==true].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to ‘Device.LocalAgent.Subscription.[Enable==true].’, and a set of `resolved_path_results` elements. Each contains a `resolved_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.’, and a number of `result_params` elements containing keys and values of the parameters of each Subscription object where the Enable parameter is ‘true’.

3. The keys of all `result_params` elements are relative paths.
4. The EUT does not return any parameters from Subscription objects whose Enable parameter is ‘false’.

1.47 Get message using search expression search path (non-equivalence)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent’s Instantiated Data Model when the requested path uses a search path that does not match a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘true’ for its Enable parameter, and at least one should have a value of ‘false’ for its Enable parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[Enable!=true].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.[Enable!=true].', and a set of `resolved_path_results` elements. Each contains a `resolved_path` of 'Device.LocalAgent.Subscription.<instance identifier>.', and a number of `result_params` elements containing keys and values of the parameters of each subscription object where the Enable parameter is 'false'.
3. The keys of all `result_params` elements are relative paths.
4. The EUT does not return any parameters from Subscription objects whose Enable parameter is 'true'.

1.48 Get message using search expression search path (exclusive greater comparison)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model when the requested path uses a search path to retrieve objects that that parameters that are greater than a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘10’ for its NotifExpiration parameter, and at least one with a value of ‘20’ for its NotifExpiration parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[NotifExpiration>10].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to ‘Device.LocalAgent.Subscription.[NotifExpiration>10].’, and a set of `resolved_path_results` elements. Each contains a

`resolved_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.’, and a number of `result_params` elements containing keys and values of the parameters of each Subscription object where the NotifExpiration parameter is greater than 10.

3. The keys of all `result_params` elements are relative paths.
4. The EUT does not return any parameters from Subscription objects whose NotifExpiration parameter is equal to or less than 10.

1.49 Get message using search expression search path (exclusive lesser comparison)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent’s Instantiated Data Model when the requested path uses a search path to retrieve objects that that parameters that are less than a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least two Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘10’ for its NotifExpiration parameter, and at least one with a value of ‘5’ for its NotifExpiration parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[NotifExpiration<10].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to ‘Device.LocalAgent.Subscription.[NotifExpiration<10].’, and a set of `resolved_path_results` elements. Each contains a `resolved_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.’, and a number of `result_params` elements contain keys and values of the parameters of each Subscription object where the NotifExpiration parameter is less than 10.
3. The keys of all `result_params` elements are relative paths.
4. The EUT does not return any parameters from Subscription objects whose NotifExpiration parameter is equal to or greater than 10.

1.50 Get message using search expression search path (inclusive greater comparison)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent’s Instantiated Data Model when the requested path uses a search path to retrieve objects that that parameters that are greater than or equal to a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure that at least three Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘10’ for its NotifExpiration parameter, at least one with a value of ‘20’ for its NotifExpiration parameter, and at least one with a value of ‘5’ for its NotifExpiration parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[NotifExpiration>=10].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.

2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to ‘Device.LocalAgent.Subscription.[NotifExpiration>=10].’, and a set of `resolved_path_results` elements. Each contains a

`resolved_path` of ‘Device.LocalAgent.Subscription.<instance identifier>.’, and a number of `result_params` elements containing keys and values of the parameters of each Subscription object where the NotifExpiration parameter is greater than or equal to 10.

3. The keys of all `result_params` elements are relative paths.

4. The EUT does not return any parameters from Subscription objects whose NotifExpiration parameter is less than 10.

1.51 Get message using search expression search path (inclusive lesser comparison)

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent’s Instantiated Data Model when the requested path uses a search path to retrieve objects that that parameters that are less than or equal to a particular value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure that at least three Subscription objects exist on the EUT. At least one of these Subscription objects should have a value of ‘10’ for its NotifExpiration parameter, at least one with a value of ‘20’ for its NotifExpiration parameter, and at least one with a value of ‘5’ for its NotifExpiration parameter.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[NotifExpiration<=10].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.[NotifExpiration<=10].', and a set of `resolved_path_results` elements. Each contains a `resolved_path` of 'Device.LocalAgent.Subscription.<instance identifier>.', and a number of `result_params` elements containing keys and values of the parameters of each Subscription object where the NotifExpiration parameter is less than or equal to 10.
3. The keys of all `result_params` elements are relative paths.
4. The EUT does not return any parameters from Subscription objects whose NotifExpiration parameter is greater than 10.

1.52 Notify - Subscription creation using Value Change

Purpose

The purpose of this test is to ensure that the Agent will create Subscriptions requested by the Controller, and notifies the Controller when the conditions of the subscription are triggered. This test uses the ValueChange event to exercise these functions, validating the behavior of ValueChange in the process.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned the instance identifier of the Device.LocalAgent.Controller. object that represents the Controller simulated by the traffic generator.
3. Set the Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode to an arbitrary value that is not ‘TestValue52’.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ID'
            value: 'notify52'
          }
        param_settings {
            param: 'NotifType'
            value: 'ValueChange'
          }
        param_settings {
            param: 'ReferenceList'
            value: 'Device.LocalAgent.Controller.<instance
identifier>.ProvisioningCode'
            required: true
          }
        param_settings {
            param: 'NotifRetry'
            value: 'true'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp
3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Controller.<instance identifier>.'
        param_settings {
          param: 'ProvisioningCode'
          value: 'TestValue52'
          required: true
        }
      }
    }
  }
}
```

4. Allow the EUT to send a Notify message.

5. Send a NotifyResp to the EUT.

Test Metrics

1. The EUT sends a successful AddResp.

2. The EUT sends a Notify message with a `subscription_id` field equal to ‘Notify52’, and an event element of `value_change` with a `param_path` of ‘Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode’ and a `param_value` of ‘TestValue52’.

1.53 Notify - Subscription Deletion Using Value Change

Purpose

The purpose of this test is to ensure that the Agent will remove and terminate a Subscription when the Controller uses the Delete message.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned the instance identifier of the Device.LocalAgent.Controller. object that represents the Controller simulated by the traffic generator.
3. Set the Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode to an arbitrary value that is not ‘TestValue53’.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ID'
            value: 'notify53'
          }
        param_settings {
            param: 'NotifType'
            value: 'ValueChange'
          }
        param_settings {
            param: 'ReferenceList'
            value: 'Device.LocalAgent.Controller.<instance
identifier>.ProvisioningCode'
            required: true
          }
        param_settings {
            param: 'NotifRetry'
            value: 'true'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp, and store the instance identifier of the Subscription object.
3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Controller.<instance identifier>.'
        param_settings {
          param: 'ProvisioningCode'
          value: 'TestValue53'
          required: true
        }
      }
    }
  }
}
```

4. Allow the EUT to send a Notify message.

5. Send a NotifyResp to the EUT.

6. Send a Delete message with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier>.'
      }
    }
  }
```

7. Allow the EUT to send a DeleteResp.

8. Repeat step 3, changing the value of ProvisioningCode to ‘notify53-2’.

9. Wait 20 seconds.

Test Metrics

1. The EUT sends a successful DeleteResp.
2. The EUT does not send a Notify message based on the activity in the ProvisioningCode parameter.

1.54 Notification Retry using Value Change

Purpose

The purpose of this test is to ensure that the Agent will attempt to resend Notify messages when the NotifRetry parameter in a Subscription object is set to true and the Controller does not send a NotifyResp.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned the instance identifier of the Device.LocalAgent.Controller. object that represents the Controller simulated by the traffic generator.
3. Set the Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode to an arbitrary value that is not ‘TestValue54’.
4. Ensure that the Device.LocalAgent.Controller.<instance identifier>.USPNotifRetryMinimumWaitInterval is set to its default value (5).

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ID'
            value: 'notify54'
          }
        param_settings {
            param: 'NotifType'
            value: 'ValueChange'
          }
        param_settings {
            param: 'ReferenceList'
            value: 'Device.LocalAgent.Controller.<instance
identifier>.ProvisioningCode'
            required: true
          }
        param_settings {
            param: 'NotifRetry'
            value: 'true'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp
3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Controller.<instance identifier>.'
        param_settings {
          param: 'ProvisioningCode'
          value: 'TestValue54'
          required: true
        }
      }
    }
  }
}
```

4. Allow the EUT to send a Notify message.
5. Do not send a NotifyResp to the EUT.
6. Wait 10 seconds to allow the EUT to send a Notify message.
7. Do not send a NotifyResp to the EUT.
8. Wait 20 seconds to allow the EUT to send a Notify message.
9. Send a NotifyResp to the EUT.

Test Metrics

1. The EUT retries the Notify message.
2. The first retry occurs within 5-10 seconds. The second retry occurs within 10-20 seconds.

1.55 Subscription Expiration using Value Change

Purpose

The purpose of this test is to ensure that the Agent removes a Subscription from the Subscription table after its TimeToLive has expired.

Functionality Tag

Conditionally Mandatory (Supports TimeToLive in Device.LocalAgent.Subscription.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned the instance identifier of the Device.LocalAgent.Controller. object that represents the Controller simulated by the traffic generator.
3. Set the Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode to an arbitrary value that is not ‘TestValue55’.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ID'
            value: 'notify55'
          }
        param_settings {
            param: 'NotifType'
            value: 'ValueChange'
          }
        param_settings {
            param: 'ReferenceList'
            value: 'Device.LocalAgent.Controller.<instance
identifier>.ProvisioningCode'
            required: true
          }
        param_settings {
            param: 'NotifRetry'
            value: 'true'
          }
        param_settings {
            param: 'TimeToLive'
            value: '20'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp

3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Controller.<instance identifier>.'
        param_settings {
          param: 'ProvisioningCode'
          value: 'TestValue55'
          required: true
        }
      }
    }
  }
}
```

4. Allow the EUT to send a Notify message.

5. Send a NotifyResp to the EUT.

6. Wait 20 seconds.

7. Send a GetInstances message to the EUT with the following structure:

```
header {
      msg_id: '<msg_id>'
      msg_type: GET_INSTANCES
    }
body {
      request {
        get_instances {
          obj_paths: 'Device.LocalAgent.Subscription.'
        }
      }
    }
```

8. Allow the EUT to send a GetInstancesResponse.
9. Repeat step 3 with a value of ‘TestValue55-2’.
10. Wait 10 seconds.

Test Metrics

1. The EUT sends a Notify message after step 3.
2. The GetInstancesResponse does not list the instance of the Subscription object created in step 1.
3. The EUT does not send a Notify message after step 9.

1.56 Notification Retry Expiration using Value Change

Purpose

The purpose of this test is to ensure that the Agent will cease attempts to retry Notify messages after an amount of time specified in value of the NotifExpiration parameter in the Subscription object has passed.

Functionality Tag

Conditional Mandatory (supports Subscription.{i}.NotifExpiration parameter).

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the traffic generator has learned the instance identifier of the Device.LocalAgent.Controller. object that represents the Controller simulated by the traffic generator.
3. Set the Device.LocalAgent.Controller.<instance identifier>.ProvisioningCode to an arbitrary value that is not ‘TestValue56’.

4. Ensure that the Device.LocalAgent.Controller.<instance identifier>.USPNotifRetryMinimumWaitInterval is set to its default value (5).

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: ADD
    }
    body {
      request {
        add {
          allow_partial: false
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'notify56'
              }
            param_settings {
                param: 'NotifType'
                value: 'ValueChange'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.Controller.<instance
    identifier>.ProvisioningCode'
                required: true
              }
            param_settings {
                param: 'NotifRetry'
                value: 'true'
              }
            param_settings {
                param: 'NotifExpiration'
                value: '20'
              }
            }
          }
        }
      }
```

2. Allow the EUT to send an AddResp
3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.LocalAgent.Controller.<instance identifier>.'
        param_settings {
          param: 'ProvisioningCode'
          value: 'TestValue56'
          required: true
        }
      }
    }
  }
}
```

4. Allow the EUT to send a Notify message.
5. Do not send a NotifyResp to the EUT.
6. Wait 10 seconds to allow the EUT to send a Notify message.
7. Do not send a NotifyResp to the EUT.
8. Wait 20 seconds to allow the EUT to send a Notify message.
9. Do not send a Notify Response to the EUT.
10. Wait 30 seconds.

Test Metrics

1. The EUT retries the Notify message within 20 seconds.
2. The EUT does not retry the Notify message after 20 seconds.

1.57 ObjectCreation Notification

Purpose

The purpose of this test is to ensure that the Agent will send a Notify message to the Controller when the Controller is Subscribed to the ObjectCreation event.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: ADD
    }
    body {
      request {
        add {
          allow_partial: false
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'notify57'
              }
            param_settings {
                param: 'NotifType'
                value: 'ObjectCreation'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.Subscription.'
                required: true
              }
            param_settings {
                param: 'NotifRetry'
                value: 'true'
              }
            }
          }
        }
      }
```

2. Allow the EUT to send an AddResp
3. Send an Add message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: ADD
    }
    body {
      request {
        add {
          allow_partial: false
          create_objs {
            obj_path: 'Device.LocalAgent.Subscription.'
            param_settings {
                param: 'Enable'
                value: 'true'
              }
            param_settings {
                param: 'ID'
                value: 'notify57-2'
              }
            param_settings {
                param: 'NotifType'
                value: 'ValueChange'
              }
            param_settings {
                param: 'ReferenceList'
                value: 'Device.LocalAgent.Controller.<instance
    identifier>.ProvisioningCode'
                required: true
              }
            param_settings {
                param: 'NotifRetry'
                value: 'true'
              }
            }
          }
        }
      }
```

4. Allow the EUT to send an AddResp

5. Allow the EUT to send a Notify message.

6. Send a NotifyResp to the EUT.

Test Metrics

1. The EUT sends a successful AddResp.
2. The EUT sends a Notify message with a `subscription_id` field equal to 'Notify57', and an event element of `obj_creation` with a `obj_path` of 'Device.LocalAgent.Subscription.<instance number>.' and a map element of `unique_keys` with values of 'ID, 'Notify57-2' and 'Recipient, Device.LocalAgent.Controller.<instance identifier>.'.

1.58 ObjectDeletion Notification

Purpose

The purpose of this test is to ensure that the Agent will send a Notify message to the Controller when the Controller is Subscribed to the ObjectDeletion event.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that at least one Subscription object exists on the EUT, and the unique keys and their values are known by the traffic generator.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
            param: 'Enable'
            value: 'true'
          }
        param_settings {
            param: 'ID'
            value: 'notify58'
          }
        param_settings {
            param: 'NotifType'
            value: 'ObjectDeletion'
          }
        param_settings {
            param: 'ReferenceList'
            value: 'Device.LocalAgent.Subscription.'
            required: true
          }
        param_settings {
            param: 'NotifRetry'
            value: 'true'
          }
        }
      }
    }
  }
```

2. Allow the EUT to send an AddResp
3. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier from
test setup 2>.'
      }
    }
  }
```

4. Allow the EUT to send a DeleteResp
5. Allow the EUT to send a Notify message.
6. Send a NotifyResp to the EUT.

Test Metrics

1. The EUT sends a successful AddResp.
2. The EUT sends a Notify message with a `subscription_id` field equal to ‘Notify58’, and an event element of `obj_deletion` with a `obj_path` of ‘Device.LocalAgent.Subscription.<instance number>.’

1.59 Event Notification using Periodic!

Purpose

The purpose of this test is to ensure that the Agent will send a Notify message to the Controller when the Controller is Subscribed to an Event notification that correlates with an event defined in its supported data model.

Functionality Tag

Conditional Mandatory (supports Controller:1 profile and Device.LocalAgent.Controller.{i}.PeriodicNotifTime parameter)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
          allow_partial: false
            update_objs {
                obj_path: 'Device.LocalAgent.Controller.<Controller ID>.'
                param_settings {
                    param: 'PeriodicNotifInterval'
                    value: '60'
                }
                param_settings {
                    param: 'PeriodicNotifTime'
                    value: '2019-01-01T00:00:00Z'
                }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: ADD
    }
    body {
        request {
            add {
              allow_partial: false
              create_objs {
                  obj_path: 'Device.LocalAgent.Subscription.'
                  param_settings {
                          param: 'Enable'
                          value: 'true'
                      }
                  param_settings {
                          param: 'ID'
                          value: 'sub-103'
                      }
                  param_settings {
                          param: 'NotifType'
                          value: 'Event'
                      }
                  param_settings {
                          param: 'ReferenceList'
                          value: 'Device.LocalAgent.Periodic!'
                      }
                  }
            }
        }
    }
```

3. Wait for a Notification from the EUT.
4. Wait for a Notification from the EUT.

Test Metrics

1. The EUT sends a SetResponse with an `oper_success` after step 1.
2. The EUT sends an AddResponse with an `oper_success` after step 2.
3. The EUT sends a Notification with an Periodic! event element.
4. A second Periodic event is sent by the EUT 60 (+/- 4) seconds after the first.

1.60 OnBoardRequest Notification

Purpose

The purpose of this test is to ensure that the Agent will send a Notify message to the Controller when the Controller initiates a SendOnBoardRequest() operation.

Functionality Tag

Conditional Mandatory (supports Device.LocalAgent.Controller.{i}.SendOnBoardRequest() command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: OPERATE
}
body {
  request {
    operate {
      command: 'Device.LocalAgent.Controller.<instance identifier of traffic
generator>.SendOnBoardRequest()'
      command_key: 'test60'
      send_resp: false
    }
  }
}
```

2. Allow the EUT to send a Notify message.

3. Send a NotifyResp to the EUT.

Test Metrics

1. The EUT sends a Notify message with (at minimum) a`subscription_id` field set to an empty string, and an event element of `on_board_req` with appropriate values for the`oui`, `product_class`, `serial_number`, and `agent_supported_protocol_versions` fields.

1.61 Operate message using Reboot() with send_resp true

Purpose

The purpose of this test is to ensure that the Agent will correctly process an Operate message using the Reboot() operation as a trigger when `send_resp` is true.

Functionality Tag

Conditional Mandatory (supports Reboot:1 or any other command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: OPERATE
}
body {
  request {
    operate {
      command: 'Device.Reboot()'
      command_key: 'test61'
      send_resp: true
    }
  }
}
```

2. Allow the EUT to send an OperateResp
3. Allow the EUT to reboot.

Test Metrics

1. The EUT sends an OperateResp message with a single `operation_results` element containing an `executed_command` of ‘Device.Reboot()’ and a `req_output_args` element containing an empty `output_args` element.
2. The EUT reboots and resumes connectivity with the test system.

1.62 Operate message using Reboot() with send_resp false

Purpose

The purpose of this test is to ensure that the Agent will correctly process an Operate message using the Reboot() operation as a trigger when `send_resp` is false.

Functionality Tag

Conditional Mandatory (supports Reboot:1 or any other command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: OPERATE
}
body {
  request {
    operate {
      command: 'Device.Reboot()'
      command_key: 'test62'
      send_resp: false
    }
  }
}
```

2. Allow the EUT to reboot.

Test Metrics

1. The EUT reboots and resumes connectivity with the test system.

1.63 Operate message using input arguments (DEPRECATED by 1.79)

The purpose of this test is to ensure that the Agent will correctly process an Operate message with input arguments.

Note: as of TP-469 Amendment 1, this test has been deprecated to sync with the deprecation of Device.LocalAgent.Controller.{i}.ScheduleTimer() in Device:2.14. The command was replaced with Device.ScheduleTimer() and is covered by test 1.79

Functionality Tag

Conditional Mandatory (supports Device.LocalAgent.Controller.{i}.ScheduleTimer() command or at least one operation that contains input arguments)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT, subscribed to the Timer! event.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.LocalAgent.Controller.<Controller
instance>.ScheduleTimer()'
            command_key: 'test63'
            send_resp: true
            input_args {
                key: 'DelaySeconds'
                value: '30'
            }
        }
    }
}
```

2. Allow the EUT to send a Timer! event.

Test Metrics

1. The EUT sends an OperateResp message with a single `operation_results` element containing

an executed_command of 'Device.LocalAgent.Controller.<Controller instance>.ScheduleTimer()' and a `req_output_args` element containing an empty `output_args` element.
2. The EUT sends a Notify message containing a Event message with`obj_path` of 'Device.LocalAgent.Controller.<Controller instance>.ScheduleTimer()'.

1.64 Asynchronous operation with send_resp true

Purpose

The purpose of this test is to ensure that the Agent will correctly process an Operate message where the operation is asynchronous and `send_resp` is set to true.

Functionality Tag

Conditional Mandatory (supports the TraceRoute:1 profile or at least one other asynchronous operation)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT that is subscribed to the OperationComplete notification for TraceRoute().

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.IP.Diagnostics.TraceRoute()'
            command_key: 'test64'
            send_resp: true
            input_args {
               key: 'Host'
               value: '<remote host IP>'
            }
            input_args {
               key: 'ProtocolVersion'
               value: 'Any'
            }
        }
    }
}
```

2. Allow the EUT to send an OperateResp message with an`executed_command` which matches the command sent in the Operate message.
3. Allow the EUT to send a Notify message with an inner OperationComplete message with a `obj_path` element matching the command sent in the Operate Message.

Test Metrics

1. The EUT sends an OperateResp message with a single`operation_results` element containing an `executed_command` of 'Device.IP.Diagnostics.TraceRoute()' and a`req_obj_path` field containing a path name to the Request object created by the EUT.
2. The EUT sends a Notify message with an`obj_path` of 'Device.IP.Diagnostics.', a`command_name` of 'TraceRoute()', and a `command_key` of 'test64'.

1.65 Asynchronous operation with send_resp false

Purpose

The purpose of this test is to ensure that the Agent will correctly process an Operate message where the operation is asynchronous and `send_resp` is set to false.

Functionality Tag

Conditional Mandatory (supports the TraceRoute:1 profile or at least one other asynchronous operation)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT that is subscribed to the OperationComplete notification for TraceRoute().

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.IP.Diagnostics.TraceRoute()'
            command_key: 'test65'
            send_resp: false
            input_args {
                key: 'Host'
                value: '<remote host IP>'
            }
        }
    }
}
```

2. Allow the EUT to send a Notify message containing an OperationComplete message with an `obj_path` and `command_name` matching the command sent in the Operate Request.

Test Metrics

1. The EUT does not send an OperateResp message.
2. The EUT sends a Notify message containing an OperationComplete message with an`obj_path` of ‘Device.IP.Diagnostics.’, a `command_name` of ‘TraceRoute()’, and a `command_key` of ‘test65’.

1.66 GetInstances using a single object, first_level_only true

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetInstances message on a single object when first_level_only is true.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the Agent has at least one entry in the Device.LocalAgent.Controller.{i}. table and supports at least one multi-instance sub-object (e.g., .MTP.{i}., etc.)

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_INSTANCES
}
body {
    request {
        get_instances {
            obj_paths: 'Device.LocalAgent.Controller.'
            first_level_only: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetInstancesResp with one `req_path_results` elements containing a `requested_path` of 'Device.LocalAgent.Controller.' and at least one `cur_insts` element.
2. All `instantiated_obj_path` elements in the GetInstancesResp only contain 'Device.LocalAgent.Controller.' instances.

1.67 GetInstances using a single object, first_level_only false

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetInstances message on a single object when `first_level_only` is false.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.
2. Ensure that the Agent has at least one entry in the Device.LocalAgent.Controller.{i}. table and supports at least one multi-instance sub-object (e.g., .MTP.{i}., etc.)

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_INSTANCES
}
body {
    request {
        get_instances {
            obj_paths: 'Device.LocalAgent.Controller.'
            first_level_only: false
        }
    }
}
```

Test Metrics

1. The EUT sends a GetInstancesResp with one `req_path_results` elements containing a `requested_path` of ‘Device.LocalAgent.Controller.’, and lists all instances of the Controller object, plus any instances of all sub-objects.

1.68 GetInstances with multiple objects

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetInstances message on multiple objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_INSTANCES
}
body {
    request {
        get_instances {
            obj_paths: 'Device.LocalAgent.Controller.'
            obj_paths: 'Device.LocalAgent.MTP.'
            first_level_only: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetInstancesResp with two `req_path_results` elements containing a `requested_path` of ‘Device.LocalAgent.Controller.’ and ‘Device.LocalAgent.MTP.’
2. Both `req_path_results` and each having at least one `cur_insts` element.

1.69 DELETED

Note: This test was formerly named “GetInstances with root object” and was invalid. It has been removed from this version of the test plan and exists only as a placeholder for numeric consistency.

1.70 GetInstances with wildcard search path

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetInstances message when a wildcard search path is used.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_INSTANCES
}
body {
    request {
        get_instances {
            obj_paths: 'Device.LocalAgent.Controller.*.MTP.'
            first_level_only: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetInstancesResp with at least one `req_path_results` element containing a ‘Device.LocalAgent.Controller.{i}.MTP.’ instance.

1.71 GetInstances with search expression search path

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetInstances message when a search expression search path is used.

Functionality Tag

Conditional Mandatory (Supports least one nested multi-instance object)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure there is at least one BootParameter for the Controller instance used for testing.
3. Ensure the Alias of the Controller used for testing is known.

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_INSTANCES
}

body {
    request {
        get_instances {
            obj_paths: 'Device.LocalAgent.Controller.[Alias=="<Controller
alias>"].BootParameter.'
            first_level_only: false
        }
    }
}
```

Test Metrics

1. The EUT sends a GetInstancesResp with at least one `req_path_results` element containing a 'Device.LocalAgent.Controller.<Controller instance>.BootParameter.' instance.

1.72 GetSupportedDM using a single object, first_level_only false, all options

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetSupportedDM message using a single object, when `first_level_only` is false and all options are true.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
header {
    msg_id: '<msg id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths: 'Device.LocalAgent.'
            first_level_only: false
            return_commands: true
            return_events: true
            return_params: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetSupportedDMResp.
2. Every `req_obj_results` element contains all parameters, events, and commands below the specified partial path, plus the supported data model information of all sub-objects.
3. Each SupportedParamResult field contains the `param_name`, `access`, `value_type`, and `value_change` fields with valid information, if the element is a parameter.
4. Each SupportedCommandResult field contains the `command_name` field, `command_type` field, and a set of `input_arg_names` and `output_arg_names` fields with valid information, if the element is a command.
5. Each SupportedEventResult field contains the `event_name` field and a set of `arg_names` fields with valid information, if the element is an event.

1.73 GetSupportedDM using a single object, first_level_only true, all options

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetSupportedDM message using a single object, when `first_level_only` is true and all options are true.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
header {
    msg_id: '<msg id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths: 'Device.LocalAgent.'
            first_level_only: true
            return_commands: true
            return_events: true
            return_params: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetSupportedDMResp containing `req_object_results` elements for the specified object and each immediate child object.
2. Only the `req_obj_results` element of the object specified in`obj_paths` contains parameters, events, and commands.
3. Each SupportedParamResult field contains the`param_name`, `access`, `value_type`, and `value_change` fields with valid information, if applicable.
4. Each SupportedCommandResult field contains the`command_name` field, `command_type` field, and a set of `input_arg_names` and `output_arg_names` fields with valid information, if applicable.

5. Each SupportedEventResult field contains the `event_name` field and a set of `arg_names` fields with valid information, if applicable.

1.74 GetSupportedDM using a single object, first_level_only true, no options

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetSupportedDM message using a single object, when `first_level_only` is true and all options are false.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
    header {
        msg_id: '<msg id>'
        msg_type: GET_SUPPORTED_DM
    }
    body {
        request {
            get_supported_dm {
                obj_paths: 'Device.LocalAgent.'
                first_level_only: true
                return_commands: false
                return_events: false
                return_params: false
            }
        }
    }
```

Test Metrics

1. The EUT sends a GetSupportedDMResp containing `req_object_results` elements for the specified object and each immediate child object.
2. None of the `req_obj_results` elements contain any commands, events, or params.

1.75 GetSupportedDM using multiple objects, first_level_only true, all options

Purpose

The purpose of this test is to ensure that the Agent will correctly process a GetSupportedDM message using multiple objects, when `first_level_only` is true and all options are true.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
header {
    msg_id: '<msg id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths: 'Device.LocalAgent.Controller.'
            obj_paths: 'Device.LocalAgent.MTP.'
            first_level_only: true
            return_commands: true
            return_events: true
            return_params: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetSupportedDMResp containing `req_object_results` elements for the specified objects and each immediate child object.
2. Only the `req_obj_results` element of the object specified in`obj_paths` contains parameters, events, and commands.
3. Each SupportedParamResult field contains the`param_name`, `access`, `value_type`, and `value_change` fields with valid information, if applicable.
4. Each SupportedCommandResult field contains the`command_name` field, `command_type` field, and a set of `input_arg_names` and `output_arg_names` fields with valid information, if applicable.
5. Each SupportedEventResult field contains the`event_name` field and a set of`arg_names` fields with valid information, if applicable.

1.76 GetSupportedDM on root object, all options

Purpose

The purpose of this test is to ensure the Agent will correctly process a GetSupportedDM message when the requested path is the root of the data model.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
header {
    msg_id: '<msg id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths:'Device.'
            first_level_only: false
            return_commands: true
            return_events: true
            return_params: true
        }
    }
}
```

Test Metrics

1. The EUT sends a GetSupportedDMResp message with one or more `req_obj_results` specifying its entire supported data model, listing commands, parameters, and events.
2. Each SupportedParamResult field contains the `param_name`, `access`, `value_type`, and `value_change` fields with valid information, if applicable.
3. Each SupportedCommandResult field contains the `command_name` field, `command_type` field, and a set of `input_arg_names` and `output_arg_names` fields with valid information, if applicable.
4. Each SupportedEventResult field contains the `event_name` field and a set of `arg_names` fields with valid information, if applicable.

1.77 GetSupportedDM on unsupported object

Procedure

The purpose of this test is to ensure the Agent will correctly process a GetSupportedDM message when the requested path is an unsupported object.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedDM to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths:'Device.LocalAgent.UnsupportedObject.'
            first_level_only: false
            return_commands: true
            return_events: true
            return_params: true
        }
    }
}
```

Test Metrics

1. The EUT returns a GetSupportedDMResp with a single `req_obj_results` with a `err_code` of ‘7026’.

1.78 Removal of subscriptions that have no associated controller

Purpose

According to the Device.LocalAgent.Subscription.{i}.Recipient parameter:

```
The value MUST be the Path Name of the Controller instance that will receive the
Notification associated with this Subscription. If the referenced object is
deleted, this instance MUST also be deleted (so the parameter value will never be
an empty string).
```

This test validates that if a Controller is removed from the Agent’s Device.LocalAgent.Controller.{i}. table, any associated Subscription objects are also removed.

Functionality Tag

Conditional Mandatory (supports Controller:1 profile with the ability to create instances of the Device.LocalAgent.Controller. object)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the EUT has two Controller instances in its Device.LocalAgent. Controller.{i}. table, and that both Controllers can be simulated by the test equipment. Consider one to be the primary Controller, and the other to be the secondary Controller. Record the secondary Controller's instance identifier.
3. Ensure that there is at least one Subscription object in the EUT Device.LocalAgent.Subscription.{i}. table created by the secondary Controller.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Controller.<instance identifier of
secondary Controller>.'
    }
  }
}
```

2. Allow the EUT to send an DeleteResp.
3. Send a Get message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: GET
    }
    body {
      request {
        get {
          param_paths: 'Device.LocalAgent.Subscription.'
        }
      }
    }
```

4. Allow the EUT to send a GetResp.

Test Metrics

1. The Subscription table does not contain the Subscription object outlined in test setup step 3.

1.79 Operate message using input arguments

Purpose

The purpose of this test is to ensure that the Agent will correctly process an Operate message with input arguments.

Functionality Tag

Conditional Mandatory (supports Device.ScheduleTimer() command or at least one operation that contains input arguments)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT with NotifType OperationComplete on Device.ScheduleTimer().

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: OPERATE
}
body {
  request {
      operate {
          command: 'Device.ScheduleTimer()'
          command_key: 'test79'
          send_resp: true
          input_args {
              key: 'DelaySeconds'
              value: '30'
          }
      }
  }
}
```

2. Wait at least 30 seconds.
3. Allow the EUT to send a Notify message.

Test Metrics

1. The EUT sends an OperateResp message with the `req_obj_path` field, containing an Object Instance Path to the Request Object created as a result of this asynchronous operation.
2. The EUT sends a Notify message containing a valid OperationComplete event, with a `command_name` of ‘ScheduleTimer()’, an `obj_path` of ‘Device.’, and a `command_key` of ‘test79’.

1.80 GetSupportedProtocol

Purpose

The purpose of this test is to ensure the Controller can learn the supported USP protocol version(s) of the EUT.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetSupportedProtocol message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET_SUPPORTED_PROTO
}
body {
  request {
    get_supported_protocol {
      controller_supported_protocol_versions: '<comma-separated list of USP
specification versions>'
    }
  }
}
```

2. Allow the EUT to send a GetSupportedProtocolResponse.

Test Metrics

1. The EUT sends a GetSupportedProtocolResponse.

2. The `agent_supported_protocol_versions` element contains a comma-separated list of supported USP specification versions.

1.81 Automatic unique key generation

Purpose

The purpose of this test is to validate that the EUT assigns unique keys which are not supplied in the Add message.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. If the EUT has a limit on the number of instances of the Subscription object, ensure that the number of existing Subscription object instances is less than the maximum supported.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
    request {
      add {
        allow_partial: false
          create_objs {
              obj_path: 'Device.LocalAgent.Subscription.'
                param_settings {
                    param: 'Enable'
                    value: 'true'
                  }
                param_settings {
                    param: 'NotifType'
                    value: 'ValueChange'
                  }
                param_settings {
                    param: 'ReferenceList'
                    value: 'Device.LocalAgent.SoftwareVersion'
                  }
              }
          create_objs {
              obj_path: 'Device.LocalAgent.Subscription.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
                param_settings {
                  param: 'NotifType'
                  value: 'ValueChange'
                }
                param_settings {
                  param: 'ReferenceList'
                  value: 'Device.LocalAgent.EndpointID'
                }
            }
        }
    }
  }
```

2. Allow the EUT to send an AddResp.

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Clean-up: Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
}
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier 1>.'
      obj_paths: 'Device.LocalAgent.Subscription.<instance identifier 2>.'
    }
  }
}
```

5. Allow the EUT to send a DeleteResp.

Test Metrics

1. The EUT AddResp is valid.

2. The AddResp contains two CreatedObjectResults that each have an OperationStatus of OperationSuccess. The OperationSuccess elements contains no parameter errors and 3 elements in the unique key map: Alias, Recipient, and ID. The values of Alias and ID must differ between the two CreatedObjectResults, and the values of Recipient must be identical. Alternatively, the OperationSuccess contains 2 elements in the unique key map if the Alias parameter is not supported: ‘Recipient’, and ‘ID’. In this case the values of ID must differ between the two CreatedObjectResults, and the values of Recipient must be identical.

1.82 Get message with unmatched search expression

Purpose

The purpose of this test is to verify that the EUT sends a successful empty response when a Get request using a search expression returns no objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that no Subscription objects exist on the EUT.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[Enable==true].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT sends a GetResp.
2. The GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Subscription.[Enable==true].', and an empty `resolved_path_results` element (i.e., `resolved_path_results{}`).

1.83 GetInstances message with unmatched search expression

Purpose

The purpose of this test is to verify that the EUT sends a successful empty response when a GetInstances request using a search expression returns no objects.

Functionality Tag

Conditional Mandatory (supports least one nested multi-instance object)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a GetInstances message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET_INSTANCES
}
body {
  request {
    get_instances {
      obj_paths: 'Device.LocalAgent.Controller.[Alias=="<non-existent
alias>"].BootParameter.'
    }
  }
}
```

2. Allow the EUT to send a GetInstancesResp.

Test Metrics

1. The EUT sends a GetInstancesResp.
2. The GetInstancesResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.Controller.[Alias=="<non-existent alias>"].BootParameter.', and an empty `curr_insts` element (i.e., `curr_insts{}`).

1.84 Notification - Subscription using search paths

Purpose

The purpose of this test is to ensure that the Agent will create and acknowledge Subscriptions containing a search path, and notifies the Controller when the conditions of the subscription are triggered.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure there are at least 2 enabled BootParameter instances under the Controller used for testing.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: "<msg_id>"
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: "Device.LocalAgent.Subscription."
        param_settings: {
            param: "Enable"
            value: "true"
        }
        param_settings: {
            param: "ID"
            value: "notify84"
        }
        param_settings: {
            param: "NotifType"
            value: "ValueChange"
        }
        param_settings: {
            param: "ReferenceList"
            value: "Device.LocalAgent.Controller.<instance identifier of
Controller>.BootParameter.*.Enable"
            required: true
        }
      }
    }
  }
}
```

2. Allow the EUT to send an AddResp
3. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: "<msg_id>"
  msg_type: SET
}

body {
  request {

    set {
      allow_partial: false
      update_objs {
        obj_path: "Device.LocalAgent.Controller.<instance identifier of
Controller>.BootParameter.<valid BootParameter instance>"
        param_settings: {
         param: "Enable"
         value: "false"
         required: true
        }
      }
    }
  }
}
```

Test Metrics

1. The EUT sends an AddResp.
2. The EUT sends a notification for the created subscription containing the BootParameter modified in step 3.

1.85 (For future work) Get message with unresolved instances - addressing by instance number

Purpose

This test was left out of version 1.2 of this document. There is some ambiguity in TR-369 Amendment 2 (USP 1.2) with regards to how paths that use instance number addressing should be treated when they address non-existant objects. This will be clarified in a future release.

1.86 Get message with unresolved instances - using a search path

Purpose

The purpose of this test is to ensure that the Agent successfully responds to a Get request when the requested Instantiated Object Path is valid but does not resolve to an existing object when using a search path.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that no enabled Subscription objects exist on the EUT.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.[Enable==true].'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT’s sends a GetResp.
2. The GetResp contains no errors.
3. The GetResp contains a single `req_path_results` element. The `requested_path_results` element is empty (i.e. `requested_path_results{}`).

1.87 Get message with unresolved instances - using an object path

Purpose

The purpose of this test is to ensure that the Agent successfully responds to a Get request when the requested Instantiated Object Path is valid but does not resolve to an existing object when using an Object Path.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that no Subscription objects exist on the EUT.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.Subscription.'
    }
  }
}
```

2. Allow the EUT to send a GetResp.

Test Metrics

1. The EUT's sends a GetResp.
2. The GetResp contains no errors.
3. The GetResp contains a single `req_path_results` element. The `requested_path_results` element is empty (i.e. `requested_path_results{}`).

1.88 Add message fails when unique key is invalid

Purpose

The purpose of this test is to validate that the EUT will deliver an error after it receives an Add message that includes a unique key parameter that is not required but is set to an invalid value.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}

body {
  request {
    add {
      allow_partial: false
      create_objs {
        obj_path: 'Device.LocalAgent.Subscription.'
        param_settings {
           param: 'Enable'
           value: 'True'
         }
        param_settings {
           param: 'ID'
           value: ''
         }
        param_settings {
           param: 'NotifType'
           value: 'ValueChange'
         }
        param_settings {
           param: 'ReferenceList'
           value: 'Device.LocalAgent.SoftwareVersion'
         }
        }
      }
    }
  }
```

2. Allow the EUT to send an Error message.

Test Metrics

1. The EUT sends an Error message containing an appropriate error code.
2. The EUT does not create the new Subscription object.

1.89 Get message using max_depth

Purpose

The purpose of this test is to ensure the Controller can retrieve the values of parameters in the Agent's Instantiated Data Model using the `max_depth` field to limit the tree depth of`result_params`.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.'
      max_depth: 1
    }
  }
}
```

2. Allow the EUT to send a GetResp.

3. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.'
      max_depth: 2
    }
  }
}
```

4. Allow the EUT to send a GetResp.

5. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.LocalAgent.'
      max_depth: 0
    }
  }
}
```

6. Allow the EUT to send a GetResp.

Test Metrics

1. The first GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.', and a set of `resolved_path_results` elements. One contains a `resolved_path` of 'Device.LocalAgent.', and a number of `result_params` elements ONLY containing keys and values of the parameters of 'Device.LocalAgent.'. No additional `resolved_path_results` elements are included.

2. The second GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.', and a set of `resolved_path_results` elements. One contains a `resolved_path` of 'Device.LocalAgent.', and a number of `result_params` elements contain keys and values of the parameters of 'Device.LocalAgent.'. Additional `resolved_path_results` exist for each of the immediate-child sub-objects of Device.LocalAgent., with `result_params` containing the keys and values of each sub-object's parameters.

3. The third GetResp contains a single `req_path_results` element. The `requested_path_results` has no errors, has a `requested_path` equal to 'Device.LocalAgent.', and a set of `resolved_path_results` elements. One contains a `resolved_path` of 'Device.LocalAgent.', and a number of `result_params` elements contain keys and values of the parameters of 'Device.LocalAgent.'. Additional `resolved_path_results` exist for each of the sub-objects of Device.LocalAgent., and their sub-objects, with `result_params` containing the keys and values of each sub-object's parameters.

4. The keys of all `result_params` elements are Relative Paths.

1.90 Delete message with search expression that matches no objects

Purpose

The purpose of this test is to validate that the EUT properly handles a Delete message using a search path that matches no objects.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that no Subscription objects exist in the Agent's Instantiated Data Model with the Enable parameter set to 'false'.

Test Procedure

1. Send a Delete message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: DELETE
 }
body {
  request {
    delete {
      allow_partial: false
      obj_paths: 'Device.LocalAgent.Subscription.[Enable==false].'
    }
  }
}
```

2. Allow the EUT to send an DeleteResp.

Test Metrics

1. The EUT sends a DeleteResp containing an empty `oper_success` element.

1.91 Unknown arguments in an Operate message

Purpose

The purpose of this test is to ensure the Agent ignores unknown arguments that are included in an operate message, using Device.ScheduleTimer() as an example.

Functionality Tags

Conditional Mandatory (supports Device.ScheduleTimer() command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.ScheduleTimer()'
            send_resp: true
            input_args {
                key: 'DelaySeconds'
                value: '10'
            }
            input_args {
                key: 'InvalidArgument'
                value: '2'
            }
        }
    }
}
```

2. Wait for the EUT to send an OperateResp.

Test Metrics

1. The EUT sends a successful OperateResponse with 'ScheduleTimer()' in the`executed_command` element.

1.92 Agent uses default values for Operate arguments

Purpose

The purpose of this test is to ensure that the Agent will correctly use default values for non-mandatory command arguments that include defaults defined in the data model.

Functionality Tag

Conditional Mandatory (supports the TraceRoute:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.IP.Diagnostics.TraceRoute()'
            command_key: 'test92'
            send_resp: true
            input_args {
               key: 'Host'
               value: '<remote host IP>'
            }
            input_args {
               key: 'ProtocolVersion'
               value: 'Any'
            }
        }
    }
}
```

2. Allow the EUT to send an Operate Response message with an`executed_command` which matches the command sent in the Operate message.

3. Do not reply to the TraceRoute ICMP requests.

Test Metrics

1. The EUT sends an OperateResp message with a single `operation_results` element containing an `executed_command` of 'Device.IP.Diagnostics.TraceRoute()' and a `req_obj_path` field containing a path name to the Request object created by the EUT.
2. The EUT attempts a TraceRoute diagnostic against the supplied host name. It retries 3 times within a 1% range of a 5000 millisecond delay between each retry, which are the default values.

2 Authentication and Access Control Test Cases

2.1 Agent does not accept messages from its own Endpoint ID

Purpose

The purpose of this test is to ensure the EUT does not respond to a USP message when the `from_id` is the EUT endpoint ID.

Functionality Tag

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Steps

1. Send a message to the EUT with the following record structure:

```
        to_id: '<EUT_ID>'
        from_id: '<EUT_ID>'

        session_context {
            # ...
        }
```

Test Metrics

1. The EUT does not respond to the message.

2.2 Agent rejects messages that do not contain its to_id in the USP Record

Purpose

The purpose of this test is to ensure the EUT does not respond to a USP message when the USP record doesn't contain a the EUT `to_id`.

Functionality Tags

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Steps

1. Send a message to the EUT with the following record structure:

```
        to_id: '<invalid ID>'
        from_id: '<EUT_ID>'

        # ...
```

Test Metrics

1. The EUT does not respond to the USP message.

2.3 Agent does not process messages without 's certificate information - DEPRECATED

Purpose

The purpose of this test is to ensure that the EUT doesn't process a USP message when the EUT does not possess the Controller's certificate information.

Functionality Tags

Deprecated

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Simulate a second Controller whose credentials are signed by an untrusted certificate authority.
3. Ensure that the UntrustedRole feature is either unsupported or disabled in the EUT.

Test Procedure

1. Send a Get message from the second simulated Controller to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}

body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

Test Metrics

1. Ensure the EUT does not respond to the Get message.

2.4 Agent rejects messages from Endpoint IDs that are not in subjectAltName - DEPRECATED

Purpose

The purpose of this test is to ensure that the EUT rejects a message from an Endpoint ID that doesn’t match the subjectAltName in the provided certificate.

Functionality Tags

Deprecated

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT using a certificate with a subjectAltName that does not match the Controller's Endpoint ID.

Test Metrics

1. The EUT does not respond to the Get message.

2.5 Agent use of self-signed certificates - DEPRECATED

Purpose

The purpose of this test is to ensure the EUT can handle self-signed certificates.

Functionality Tags

Deprecated

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the is configured to use a self-signed certificate and Endpoint ID that the EUT has not seen.

Test Procedure

1. Send a Get message to the EUT using a self-signed cert with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

Test Metrics

1. The EUT responds to the Get with a GetResponse containing a ‘Device.LocalAgent.ControllerTrust.{i}.Alias’ parameter.

2.6 Connecting without absolute time

Purpose

The purpose of this test is to ensure the EUT can communicate with a Controller if it cannot obtain an absolute time.

Functionality Tags

Conditional Mandatory (Supports USP Session Context)

Test Setup

1. The EUT is booted into a test environment where it cannot resolve absolute time.
2. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
3. Ensure the Controller is configured to use an expired certificate.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header{
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

Test Metrics

1. The EUT responds to the Get message with a GetResponse, ignoring the expired dates on the certificate.

2.7 Agent ignores unsigned or invalid Record signatures

Purpose

The purpose of this test is to ensure the EUT will ignore a USP record when the signature field is invalid.

Functionality Tags

Conditional Mandatory (Supports USP Session Context)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Get message to the EUT with an invalid signature value.

Test Metrics

1. The EUT does not respond to the Get message.

2.8 Agent ignores invalid TLS certificate

Purpose

The purpose of this test is to ensure the EUT rejects TLS connections when an Endpoint’s TLS certificate is invalid.

Functionality Tags

Conditional Mandatory (Supports USP Session Context)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the EUT has obtained an absolute time reference.

Test Procedure

1. Send a Get message to the EUT with an expired TLS certificate.

Test Metrics

1. The EUT doesn’t respond to the Get message.

2.9 Use of the Untrusted role

Purpose

The purpose of this test is to ensure the EUT correctly assigns new a Role of Untrusted.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Using a secondary Controller, connect to the EUT and send an Get message.
2. Using the primary trusted Controller send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.'
        }
    }
}
```

Test Metrics

1. Ensure the `Device.LocalAgent.Controller.<secondary Controller instance>.AssignedRole` matches the value of

`Device.LocalAgent.ControllerTrust.UntrustedRole`.

2.10 Adding a Role

Purpose

The purpose of this test is to ensure that the Add message can be used to add new Roles to the EUT data model.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a Add message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: ADD
    }
    body {
        request {
            add {
                allow_partial: false
                create_objs {
                    obj_path: 'Device.LocalAgent.ControllerTrust.Role.'
                    param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                    param_settings {
                        param: 'Name'
                        value: 'Trusted'
                    }
                }
            }
        }
    }
```

Test Metrics

1. The EUT correctly sent an AddResponse with a new Role instance.

2.11 Permissions - Object Creation Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to allow the creation of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.'
                    }
                param_settings {
                        param: 'Obj'
                        value: 'rw--'
                    }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.Subscription.'
            }
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a oper_success element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an AddResponse with a oper_success element containing a new Device.LocalAgent.Subscription. object in step 2.

2.12 Permissions - Object Creation Not Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to restrict the creation of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.'
                    }
                param_settings {
                        param: 'Obj'
                        value: 'r---'
                    }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.Subscription.'
            }
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a oper_success element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an Error containing type ‘7006’ - Permission Denied.

2.13 Permissions - Object Deletion Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to allow the deletion of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.

2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is one or more Subscription object that can be deleted.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.'
                    }
                param_settings {
                        param: 'InstantiatedObj'
                        value: 'rw--'
                    }
            }
        }
    }
}
```

2. Send an Delete message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: DELETE
    }
    body {
        request {
            delete {
                allow_partial: false
                obj_paths: 'Device.LocalAgent.Subscription.<subscription to
delete>'
            }
        }
    }
```

Test Metrics

1. The EUT sends an AddResponse with a `oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an DeleteResponse with a `oper_success` element containing the Device.LocalAgent.Subscription. object in step 2.

2.14 Permissions - Object Deletion Not Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to restrict the deletion of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is one or more Subscription object that can be deleted.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.'
                    }
                param_settings {
                        param: 'InstantiatedObj'
                        value: 'r---'
                    }
            }
        }
    }
}
```

2. Send an Delete message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: DELETE
}
body {
    request {
        delete {
            allow_partial: false
            obj_paths: 'Device.LocalAgent.Subscription.<subscription to
delete>'
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a`oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an Error containing type ‘7006’ - Permission Denied.

2.15 Permissions - Parameter Update Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to allow the update of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is one or more Subscription object that can be edited.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.<instance that
can be edited>.'
                    }
                param_settings {
                        param: 'Param'
                        value: 'rw--'
                    }
            }
        }
    }
}
```

2. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            allow_partial: false
            update_objs {
                    obj_path: 'Device.LocalAgent.Subscription.<instance that
can be edited>.'
                    param_settings {
                            param: 'Enable'
                            value: '<negation of previous value>'
                            required: true
                        }
                }
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a `oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends a SetResponse with a `oper_success` element containing Device.LocalAgent.Subscription.{i}.Enable in step 2.

2.16 Permissions - Parameter Update Not Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to restrict the update of a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is one or more Subscription object that can be edited.

Test Steps

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Subscription.<instance that
can be edited>.'
                    }
                param_settings {
                        param: 'Param'
                        value: 'r---'
                    }
            }
        }
    }
}
```

2. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            allow_partial: false
            update_objs {
                    obj_path: 'Device.LocalAgent.Subscription.<instance that
can be edited>.'
                    param_settings {
                            param: 'Enable'
                            value: '<negation of previous value>'
                            required: true
                        }
                }
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a`oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an Error containing type ‘7006’ - Permission Denied.

2.17 Permissions - Operation Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to allow the invocation of commands on a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Steps

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.Reboot()'
                    }
                param_settings {
                        param: 'CommandEvent'
                        value: 'r-x-'
                    }
            }
        }
    }
}
```

2. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.Reboot()'
            send_resp: true
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a `oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an OperateResponse with a `req_output_args` element in step 2.

2.18 Permissions - Operation Not Allowed

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to restrict the invocation of commands on a particular object.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.

2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Steps

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.Reboot()'
                    }
                param_settings {
                        param: 'CommandEvent'
                        value: 'r---'
                    }
            }
        }
    }
}
```

2. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.Reboot()'
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse with a`oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an Error message containing an appropriate error code.

2.19 Permissions - Value Change Notification Allowed on Parameter

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to allow a Controller to subscribe to the ValueChange notification of a particular parameter.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Steps

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Controller.<Controller
instance id>.PeriodicNotifInterval'
                    }
                param_settings {
                        param: 'Param'
                        value: 'rw-n'
                    }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: ADD
    }
    body {
        request {
            add {
                allow_partial: false
                create_objs {
                    obj_path: 'Device.LocalAgent.Subscription.'
                    param_settings {
                            param: 'Enable'
                            value: 'true'
                        }
                    param_settings {
                            param: 'NotifType'
                            value: 'ValueChange'
                        }
                    param_settings {
                            param: 'ReferenceList'
                            value: 'Device.LocalAgent.Controller.<Controller
    instance id>.PeriodicNotifInterval'
                        }
                }
            }
        }
      }
```

3. Send a Set message to the EUT, setting Device.LocalAgent.Controller.<Controller instance id>.PeriodicNotifInterval to a new value.

4. Wait for a Notification from the EUT.

Test Metrics

1. The EUT sends an AddResponse with an `oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an AddResponse with an `oper_success` element containing a new Device.LocalAgent.Subscription. object in step 2.
3. The EUT sends a SetResponse with an `oper_success` element with the path ‘Device.LocalAgent.Controller.<Controller instance id>.PeriodicNotifInterval’.

4. The EUT sends a Notify message with a`value_change` element pointing to 'Device.LocalAgent.Controller.<Controller instance>.PeriodicNotifInterval'.

2.20 Permissions - Value Change Notification Not Allowed on Parameter

Purpose

The purpose of this test is to ensure the EUT adheres to permissions set to restrict a Controller from subscribing to the ValueChange notification of a particular parameter.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.

Test Steps

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.Role.<Controller
Role instance>.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Controller.<Controller
instance id>.PeriodicNotifInterval'
                    }
                param_settings {
                        param: 'Param'
                        value: 'rw--'
                    }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.Subscription.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'NotifType'
                        value: 'ValueChange'
                    }
                param_settings {
                        param: 'ReferenceList'
                        value: 'Device.LocalAgent.Controller.<Controller
instance id>.PeriodicNotifInterval'
                    }
            }
        }
    }
}
```

3. Send a Set message to the EUT, setting Device.LocalAgent.Controller.<Controller instance id>.PeriodicNotifInterval to a new value.
4. Wait 30 seconds.

Test Metrics

1. The EUT sends an AddResponse with an `oper_success` element containing a new Device.LocalAgent.ControllerTrust.Role.{i}.Permission. object in step 1.
2. The EUT sends an AddResponse with an `oper_success` element containing a new Device.LocalAgent.Subscription. object in step 2.
3. The EUT sends a SetResponse with an `oper_success` element with the path ‘Device.LocalAgent.Controller.<Controller instance id>.PeriodicNotifInterval’.

4. The EUT does not send a Notify message with a value_change element pointing to ‘Device.LocalAgent.Controller.<Controller instance>.PeriodicNotifInterval’.

2.21 Permissions - Overlapping Permissions

Purpose

The purpose of this test is to ensure the EUT allows for the creation of Permission instances, and when Permissions overlap the EUT behaves correctly.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is at least one BootParameter configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.<Controller
id>.Role.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Controller.<Controller
instance id>.BootParameter.<boot parameter instance>.'
                    }
                param_settings {
                        param: 'Param'
                        value: '----'
                    }
            }
            create_objs {
                obj_path: 'Device.LocalAgent.ControllerTrust.<Controller
id>.Role.Permission.'
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'Targets'
                        value: 'Device.LocalAgent.Controller.<Controller
instance id>.BootParameter.<boot parameter instance>.'
                    }
                param_settings {
                        param: 'Param'
                        value: 'rw--'
                    }
                param_settings {
                        param: 'Order'
                        value: '1'
                    }
            }
        }
    }
}
```

2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.<Controller instance
ID>.BootParameter.'
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse message after step 1. The message contains two`oper_success` elements, one for each added permission.
2. The EUT sends a GetResponse with a `result_params` element containing parameters of the specified BootParameter instance.

2.22 Using Get when no read permissions are available on some parameters

Purpose

The purpose of this test is to ensure the EUT correctly returns parameters that are readable while ignoring parameters that do not have read permissions.

Functionality Tags

Conditional Mandatory (supports the ControllerTrust:1 profile with at least one role that allows object creation, or supports writable parameters in Device.LocalAgent.ControllerTrust.{i}.Role.{i}.)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the Controller used for testing has an assigned Role that is writable.
3. Ensure there is at least one BootParameter configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            allow_partial: false
            create_objs {
                    obj_path: 'Device.LocalAgent.ControllerTrust.Role.
<Controller trust instance>.Permission.'
                    param_settings {
                            param: 'Enable'
                            value: 'true'
                        }
                    param_settings {
                            param: 'Targets'
                            value: 'Device.LocalAgent.Controller.<Controller
instance ID>.BootParameter.<known instance>.ParameterName'
                        }
                    param_settings {
                            param: 'Param'
                            value: '----'
                        }
                }
        }
    }
}
```

2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.<Controller instance
ID>.BootParameter.<known instance>.'
        }
    }
}
```

Test Metrics

1. The EUT sends an AddResponse message after step 1. The message contains a`oper_success` element for the added Permission.
2. The EUT sends a GetResponse with a `result_params` element containing parameters of the specified BootParameter instance, with the exception of the 'ParameterName' parameter.

3 USP Record Test Cases

3.1 Bad request outside a session context

Purpose

The purpose of this test is to ensure the EUT correctly responds to a bad request outside a session context.

Functionality Tags

Mandatory

Test Setup

1. Ensure the EUT is configured to not use a session context.
2. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send a malformed USP Record to the EUT.

Test Metrics

1. The EUT either ignores the malformed record or sends a USP Record Error.

3.2 Agent Verifies Non-Payload Field Integrity

Purpose

The purpose of this test is to ensure the EUT verifies the integrity of the non-payload fields in a USP record.

Functionality Tags

‘Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)’

Test Setup

1. Ensure the relevant equipment are configured to NOT provide integrity protection at the MTP layer.
2. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.

Test Procedure

1. Send a Get message to the EUT with a`payload_security` of PLAINTEXT.

Test Metrics

1. After the EUT receives the USP record, it exhibits the expected 'bad request' behavior for the applicable MTP.

3.3 Agent rejects invalid signature starting a session context

Purpose

The purpose of this test is to ensure the EUT handles an attempt to start a session context with an invalid `mac_signature`.

Functionality Tags

'Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)'

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.

Test Procedure

1. Send a TLS 'client hello' to the EUT to begin a session context as described inEnd to End Message Exchange' in TR-369 with an invalid`mac_signature`.

Test Metrics

1. After the EUT receives the USP record, it exhibits the expected 'bad request' behavior for the applicable MTP.

3.4 Using TLS for USP Record Integrity

Purpose

The purpose of this test is to ensure the EUT uses TLS to validate the integrity of USP records when the `payload_security` is TLS and the TLS handshake has completed.

Functionality Tags

‘Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)’

Test Setup

1. Ensure the EUT and controller are configured to secure the USP record payload with TLS.

Test Procedure

1. Start a E2E session with the EUT using TLS to secure the payload.

2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

Test Metrics

1. In the GetResponse sent by the EUT, the`mac_signature` in the USP Record secures the non-payload fields via the MAC mechanism.
2. The `mac_signature` in the USP record sent by the EUT validates the integrity of the non-payload fields.

3.5 Failure to Establish TLS

Purpose

The purpose of this test is to ensure the EUT behaves correctly when the TLS session used to encapsulate the payload cannot be established.

Functionality Tags

‘Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)’

Test Setup

1. Configure the controller to use TLS12 as `apayload_security`.
2. Ensure `PeriodicNotifInterval` is ‘60’, and the controller used for testing is subscribed to Periodic Event Notification.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.<controller
instance>.E2ESession.'
        }
    }
}
```

2. Attempt to establish a new secure session with the EUT using TLS payload encapsulation.

3. Configure the controller to send TLS alerts during the TLS handshake process.

4. Wait for the EUT to attempt to start a session with the controller.

5. Allow the controller to send a TLS alert to the EUT and for the session to terminate.

6. Configure the controller to not send a TLS alert.

7. Wait for the EUT to retry establishing a E2E session.

Test Metrics

1. After sending the client certificate to the EUT, the EUT sends a TLS alert, terminating the session.
2. After step 5, the EUT waits before retrying the session in accordance with the 'SessionRetry' parameters found in step 1.

3.6 Agent does not accept TLS renegotiation for E2E message exchange

Purpose

The purpose of this test is to ensure the EUT does not accept TLS renegotiation. frames during a E2E message exchange.

Functionality Tags

'Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)'

Test Setup

1. Ensure both the EUT and the controller are configured to use TLS payload security.

Test Procedure

1. Establish a E2E session with the EUT.

2. Send a request to renegotiate TLS in place of the payload.

3. Send a Get message to the EUT with the following structure:

   ```
   header {
       msg_id: '<msg_id>'
       msg_type: GET
   }
   body {
       request {
           get {
               param_paths: 'Device.DeviceInfo.'
           }
       }
   }
   ```

4. Wait for a GetResponse from the EUT.

Test Metrics

1. Between sending the TLS renegotiation request and receiving the GetResponse, the EUT either sends no records, or sends a TLS alert of type `no_renegotiation(100)`.

3.7 Use of X.509 Certificates

Purpose

The purpose of this test is to ensure the EUT correctly uses X.509 certificates to authenticate other endpoints, and in turn provides a X.509 certificate for the purpose of authentication.

Functionality Tags

‘Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)’

Test Setup

1. Ensure the EUT and controller are configured to use TLS payload security.

Test Procedure

1. Configure the controller to provide a X.509 certificate with a`subjectAltName` that does not match the controller's USP endpoint ID.

2. Attempt to start a session with the EUT and send a Get message with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

Test Metrics

1. During the TLS handshake the EUT provides a X.509 certificate with a`subjectAltName` that matches the endpoint ID of the EUT.
2. During the TLS handshake the EUT requests a X.509 certificate from the controller.
3. The EUT rejects the controller's certificate.

3.8 Establishing a Session Context

Purpose

The purpose of this test is to ensure the EUT can use a session context to exchange USP messages.

Functionality Tag

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to establish a connection and exchange USP messages.
2. Ensure at the start of the test there is no existing session context between the EUT and controller.

Test Procedure

1. Start a session context with the EUT and send a Get message with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

Test Metrics

1. After step 1, the EUT responds with a USP record containing a session context, a `sequence_number` of 1 and a `session_id` that matched the session identifier sent to the EUT.

3.9 Receipt of a Record out of a Session Context

Purpose

The purpose of this test is to ensure the EUT correctly handles the receiving of a USP record outside of a session context.

Functionality Tags

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to establish a session and exchange USP messages.

Test Procedure

1. Start a session with the EUT using a session context.
2. Send a Get message to the EUT for`Device.DeviceInfo.` using a USP Record with the following structure:

```
Record {
      session_context {
        session_id: <new_session_id>
        sequence_id:  1
        expected_id:  1
        payload {
            #  ...
          }
        }
      }
```

Test Metrics

1. The EUT sends the GetResponse in a USP Record using the new`session_id` and a `sequence_id` of 1.

3.10 Session Context Expiration

Purpose

The purpose of this test is to ensure the EUT correctly adheres to the`SessionExpiration` parameter.

Functionality Tags

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information required to start a session and exchange USP records.
2. Ensure the controller is subscribed to Periodic! event.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.LocalAgent.Controller.<controller
instance>.E2ESession.'
                    param_settings {
                            param: 'SessionExpiration'
                            value: '60'
                        }
                }
            update_objs {
                    obj_path: 'Device.LocalAgent.Controller.<controller
instance>.'
                    param_settings {
                            param: 'PeriodicNotifInterval'
                            value: '10'
                        }
                }
        }
    }
}
```

2. Wait for 3 Notify messages from the EUT containing a Periodic! event.

3. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.LocalAgent.Controller.<controller
instance>.E2ESession.'
                    param_settings: {
                            param: 'SessionExpiration'
                            value: '5'
                        }
                }
            update_objs {
                    obj_path: 'Device.LocalAgent.Controller.<controller
instance>.'
                    param_settings {
                            param: 'PeriodicNotifInterval'
                            value: '10'
                        }
                }
        }
    }
}
```

4. Wait for 3 Notify messages from the EUT containing a Periodic! event.

Test Metrics

1. All three Notify messages received in step 2 use the same session context.
2. None of the three Notify messages received in step 4 shared the same session context.

3.11 Use of Sequence ID and Expected ID

Purpose

The purpose of this test is to ensure the EUT correctly uses the`sequence_id` and `expected_id` attributes found in a session context.

Functionality Tags

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to start a session and exchange USP messages.
2. Ensure the controller is not subscribed to any events on the EUT.

Test Procedure

1. Start a new session by sending a Get message to the EUT with`sequence_id` and `expected_id` set to 1 for 'Device.DeviceInfo.ModelNumber'.
2. Send a Get message to the EUT with the`sequence_id` and `expected_id` set to 4 for 'Device.DeviceInfo.SoftwareVersion'.
3. Send a Get message to the EUT with the`sequence_id` and `expected_id` set to 2 for 'Device.DeviceInfo.HardwareVersion'.
4. Send a Get message to the EUT with the`sequence_id` and `expected_id` set to 3 for 'Device.DeviceInfo.HardwareVersion'.

Test Metrics

1. After step 1, the EUT returns a GetResponse with a`sequence_id` of 1 containing the parameter 'Device.DeviceInfo.ModelNumber'.
2. The EUT buffers the Get message sent in step 2 and does not immediately respond.
3. After step 3, The EUT sends a GetResponse with a`sequence_id` of 2 containing the parameter 'Device.DeviceInfo.HardwareVersion'.
4. After step 4, the EUT sends a GetResponse with a`sequence_id` of 3 containing the parameter 'Device.DeviceInfo.HardwareVersion'. The EUT then sends a GetResponse for the buffered Get message from step 2 with a `sequence_id` of 4 containing the parameter 'Device.DeviceInfo.SoftwareVersion'.

3.12 Preservation of USP Records

The purpose of this test is to ensure the EUT preserves a sent record in the event the receiving endpoint requests a retransmission.

Functionality Tags

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to start a session an exchange USP messages.

Test Procedures

1. Start a new session.
2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

3. Wait 60 seconds.
4. Send a USP record to the EUT with a`retransmit_id` set to the`expected_id` value in the record sent in step 1.

Test Metrics

1. The EUT sends the same GetResponse twice, once after step 2 and once after step 4.

3.13 Agent Rejects Records with Different Payload Security than the Established Context

Purpose

The purpose of this test is to ensure the EUT does not accept USP Records that have a different `payload_security` value than the that of the established session context.

Functionality Tags

Conditional Mandatory (supports Secure Message Exchange using TLS for USP Record Integrity)

Test Setup

1. Ensure the EUT and controller have the necessary information to start a session and exchange USP messages.
2. Ensure the EUT and controller have the necessary information to secure the USP record payload using TLS.

Test Procedure

1. Starts a session with the EUT using`payload_security` TLS12.
2. After the session is established, send the following Get message for any valid parameter using `payload_security` PLAINTEXT and a plaintext payload.

Test Metrics

1. The EUT does not send a GetResponse.
2. The EUT starts a new session after step 2.

3.14 Use of retransmit_id

Purpose

The purpose of this test is to ensure the EUT correctly uses the`retransmit_id` value in a USP record and adheres to the related parameters in the data model.

Functionality Tags

Conditionality Mandatory (supports session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to start a session and exchange USP messages.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.LocalAgent.Controller.<controller
instance>.E2ESession.'
                    param_settings {
                            param: 'MaxRetransmitTries'
                            value: '2'
                        }
                }
        }
    }
}
```

2. Wait for a SetResponse

3. Send a USP record with a`retransmit_id` set to the value of the`sequence_id` found in the

SetResponse in step 2.

4. Repeat steps 2 and 3 twice more.

Test Metrics

1. The first three SetResponse messages are sent in the same session context.
2. On the third retransmit request, the EUT doesn't send a SetResponse and instead starts a new session with the controller.

3.15 Handling Duplicate Records

Purpose

The purpose of this test is to ensure the EUT can correctly handle receiving duplicate records.

Functionality Tags

Conditional Mandatory (supports USP session context)

Test Setup

1. Ensure the EUT and controller have the necessary information to start session and exchange USP messages.

Test Procedure

1. Start a session with the EUT.
2. Send a Get message to the EUT requesting a parameter that is known to exist.
3. Retransmit the same USP record sent in step 2 to the EUT, using the same non-payload USP record field values.
4. Repeat step 3 twice more.

Test Metrics

1. The EUT send only one GetResponse.

4 General MTP Test Cases

4.1 Use of X.509 certificates at the MTP layer

Purpose

The purpose of this test is to ensure the EUT can use X.509 certificates to secure communication at the MTP layer.

Functionality Tags

Mandatory

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the use of MTP layer encryption is enabled on the EUT.

Test Procedure

1. Send a GetSupportedProtocol message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_SUPPORTED_PROTO
}
body {
    request {
        get_supported_protocol {
            controller_supported_protocol_versions: '<comma separated list of
test controller versions>'
        }
    }
}
```

Test Metrics

1. The EUT processes the certificate and establishes a secure TLS connection at the MTP layer.

5 CoAP Test Cases (DEPRECATED)

5.1 Mapping a USP Record to a CoAP message (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT can properly use CoAP to transport USP Records.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

2. Wait for a GetResponse

Test Metrics

1. The GetResponse is encapsulated in a CoAP message.
2. The CoAP message used transport the GetResponse uses 'application/octet-stream' for`Content-Format`.

5.2 USP Records that exceed CoAP message size (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT properly segments large USP records and transports them using block encapsulation.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.'
        }
    }
}
```

2. Wait for a GetResponse

Test Metrics

1. The EUT sends the GetResponse message using multiple block encapsulated CoAP messages.

5.3 Successful CoAP exchange (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT correctly sends a 2.04 Changed response to CoAP messages.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

Test Metrics

1. After the transmission of the Get message the EUT sends a 2.04 Changed message.

5.4 Failed CoAP exchange - timeout (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT behaves correctly when a timeout occurs at the MTP layer when using CoAP.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Configure the to not send 2.04 Changed responses to CoAP messages

2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

3. Wait for a GetResponse message from the EUT.

4. Prevent the Controller from sending a 2.04 Changed CoAP response.

5. Wait for EUT to retry sending the GetResponse.

6. Allow the Controller to send a 2.04 Changed CoAP response.

Test Metrics

1. The EUT attempts to retransmit the GetResponse message after not receiving a 2.04 Changed from the Controller.

5.5 Failed CoAP Exchange - Invalid Method (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT correctly responds when it receives a CoAP message with an invalid method.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.
2. The EUT the Controller are configured to communicate over CoAP.

Test Procedure

1. Send a USP record to the EUT using a CoAP message with method code`0x06`.
2. Wait up to 60 seconds for the EUT to send a CoAP response.

Test Metrics

1. The EUT sends a reply to the CoAP message with an invalid method code.
2. The EUT CoAP response uses code `4.05` to indicate an invalid CoAP method.

5.6 Failed CoAP Exchange - Invalid Content-Format (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT properly responds to CoAP messages that feature invalid Content-Format options.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Send a USP record to the EUT using a CoAP message with Content-Format option`0x113a`.
2. Wait up to 60 second for the EUT to respond.

Test Metrics

1. The EUT sends a reply to the CoAP message with an invalid Content-Format.
2. The EUT CoAP response uses code `4.15` to indicate an invalid Content-Format.

5.7 Failed CoAP Exchange - Invalid USP Record (DEPRECATED)

Purpose

The purpose of this is to ensure the EUT properly responds to a CoAP message containing a malformed USP record.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.
2. The EUT and Controller are configured to communicate over CoAP.

Test Procedure

1. Send a malformed USP record to the EUT in a CoAP message.
2. Wait up to 60 seconds for the EUT to send a CoAP reply.

Test Metrics

1. The EUT sends a reply to the CoAP message with the malformed USP record.
2. The EUT CoAP response uses code `4.00` to indicate the USP record is invalid or not understandable.

5.8 Use of DTLS (DEPRECATED)

Purpose

The purpose of this test is to ensure the EUT can establish secure communication with another CoAP endpoint at the CoAP layer.

Functionality Tags

Conditional Mandatory (supports the CoAP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.
2. The EUT and Controller are configured to communicate over CoAP using DTLS.
3. The EUT and Controller have the necessary information about one another to establish an encrypted channel of communication.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

2. Wait for the EUT to send a GetResponse.

Test Metrics

1. The Controller is able to establish a DTLS session with the EUT.
2. The EUT established a DTLS session and sends a GetResponse.

6 STOMP Test Cases

6.1 Support of Required Profiles

Purpose

The purpose of this test is to ensure the EUT supports the required STOMP profiles.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.

Test Procedure

1. Send a GetSupportedDM message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths: 'Device.STOMP.'
            obj_paths: 'Device.LocalAgent.'
            return_params: true
            first_level_only: false
        }
    }
}
```

2. Wait for the GetSupportedDMResponse.

Test Metrics

1. The EUT sends a GetSupportedDMResponse.
2. The GetSupportedDMResponse from the EUT contains the following parameters: * Device.LocalAgent.Controller.{i}.MTP.{i}.STOMP.Reference * Device.LocalAgent.Controller.{i}.MTP.

{i}.STOMP.Destination * Device.STOMP.ConnectionNumberOfEntries * Device.STOMP.Connection.{i}.Alias * Device.STOMP.Connection.{i}.Enable * Device.STOMP.Connection.{i}.Status * Device.STOMP.Connection.{i}.Host * Device.STOMP.Connection.{i}.Port * Device.STOMP.Connection.{i}.VirtualHost * Device.STOMP.Connection.{i}.ServerRetryInitialInterval * Device.STOMP.Connection.{i}.ServerRetryIntervalMultiplier * Device.STOMP.Connection.{i}.ServerRetryMaxInterval

6.2 STOMP session establishment

Purpose

The purpose of this test is to ensure the EUT can properly start a STOMP session.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure that the EUT is configured to use a STOMP server that exists in the test environment.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the STOMP server and subscribe to a destination.

Test Metrics

1. The EUT sends a STOMP frame to the STOMP server to initiate the STOMP session.

6.3 STOMP Connection Retry

Purpose

The purpose of this test is to ensure the EUT properly enters a retry state when it fails to connect to the STOMP server.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure that the EUT is configured to use a STOMP server that exists in the test environment.

Test Procedure

1. Send a Get message to the EUT with the following structure

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.STOMP.Connection.'
        }
    }
}
```

2. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.Reboot()'
        }
    }
}
```

3. Disable the STOMP server.
4. Allow the EUT to attempt to start a STOMP session with the STOMP server.
5. Reenable the STOMP server after the EUT fails to connect to the STOMP server twice.

Test Metrics

1. The EUT retries connecting to the STOMP server within the`ServerRetryInitialInterval` of the connection instance.
2. The EUT retries a second time in accordance with`ServerRetryInitialInterval` and `ServerRetryIntervalMultiplier`.

6.4 Successful USP message over STOMP with required headers

Purpose

The purpose of this test is to ensure the EUT can communicate over STOMP using the correct headers.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure that the EUT is configured to use a STOMP server that exists in the test environment.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

2. Allow the EUT to send a GetResponse.

Test Metrics

1. In the STOMP frame transporting the GetResponse the`content-length` header is present and contains the length of the included body of the message.
2. In the STOMP frame transporting the GetResponse the`content-type` header is present and contains `application/vnd.bbf.usp.msg`.
3. In the STOMP frame transporting the GetResponse the`reply-to-dest` header is present and contains the STOMP destination of the EUT.

6.5 STOMP destination - provided in subscribe-dest

Purpose

The purpose of this test is to ensure the EUT correct subscribe to a destination found in the`subscribe-dest` header in a CONNECTED frame.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure the EUT is configured to use a STOMP server that is part of the test environment.

Test Procedure

1. Configure the STOMP server to send an unused destination via the`subscribe-dest` header in the CONNECTED frames.
2. Reboot the EUT.
3. Allow the EUT to reconnect to the STOMP server.
4. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.'
        }
    }
}
```

5. Allow the EUT to respond to the Get message.

Test Metrics

1. The EUT subscribes to the destination configured in step 1.
2. The STOMP frame containing the GetResponse has a `reply-to-dest` header which matches the destination configured in step 1.

6.6 STOMP destination - configured in USP data model

Purpose

The purpose of this test is to ensure the EUT can use the`Device.LocalAgent.MTP.{i}.STOMP.Destination` parameter to select a STOMP destination.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Steps

1. Ensure the EUT is configured to use a STOMP server that is part of the test environment.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                obj_path: 'Device.LocalAgent.MTP.<active MTP
instance>.STOMP.'
                param_settings {
                        param: 'Destination'
                        value: '<new unused destination>'
                    }
            }
        }
    }
}
```

2. Reboot the EUT.
3. Wait for the EUT to reconnect to the STOMP server.
4. Allow the STOMP server to send a CONNECTED frame that does NOT include the subscribe-dest field.
5. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

6. Wait for a GetResponse from the EUT.

Test Metrics

1. The EUT subscribes to the destination configured in step 1.
2. The STOMP frame containing the GetResponse has a `reply-to-dest` header which contains the STOMP destination configured in step 1.

6.7 STOMP Destination - terminates unconfigured session

Purpose

The purpose of this test is to ensure the EUT terminates a STOMP session when no destination id configured.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. The EUT is configured to use a STOMP server which exists in the test environment.
2. Configure the STOMP server to not provide a `subscribe-dest` header in the CONNECTED frame.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.LocalAgent.MTP.<active MTP
instance>.STOMP.'
                    param_settings {
                            param: 'Destination'
                            value: ''
                        }
                }
        }
    }
}
```

2. Reboot the EUT.
3. Wait for the EUT to attempt to reconnect to the STOMP server.

Test Metrics

1. The EUT terminates the STOMP session after the STOMP server sends a CONNECTION to the EUT.

6.8 Use of STOMP heartbeat mechanism

Purpose

The purpose of this test is to ensure the EUT can correctly implements the STOMP heartbeat mechanism and the relevant parameters in the data model.

Functionality Tags

Conditional Mandatory (supports STOMPHeartbeat:1 profile)

Test Setup

1. The EUT is configured to use a STOMP server which exists in the test environment.
2. Ensure the STOMP server supports heartbeats.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.STOMP.Connection.<STOMP server in
use>.'
                    param_settings {
                            param: 'EnableHeartbeats'
                            value: 'true'
                        }
                    param_settings {
                            param: 'IncomingHeartbeat'
                            value: '10000'
                        }
                    param_settings {
                            param: 'OutgoingHeartbeat'
                            value: '15000'
                        }
                }
        }
    }
}
```

2. Reboot the EUT.

3. Wait for the EUT to reconnect to the STOMP server.

4. Wait for 60 seconds

Test Metrics

1. In the STOMP frame sent to the STOMP server during step 2, the `heart-beat` header sent by the EUT contains '15000, 10000'.
2. After the EUT is connected to the STOMP server, the EUT sends heartbeat messages every 15 seconds.

6.9 Error Handling - Unprocessed Record

Purpose

The purpose of this test is to ensure the EUT will correctly send an ERROR STOMP frame when a malformed USP record is received.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure the EUT is configured to use a STOMP server that exists in the test environment.

Test Procedure

1. Send a malformed USP record to the EUT.
2. Wait 60 seconds for the EUT to send a response.

Test Metrics

1. The EUT either ignores the malformed record or sends a USP Record Error.

6.10 Agent’s STOMP destination is changed

Purpose

The purpose of this test is to ensure that when the EUT destination is altered it properly unsubscribes and subscribes to the new destination.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure the EUT is configured to use a STOMP server that exists in the test environment.
2. Ensure the STOMP server is configured to not provide a destination via the`subscribe-dest` header.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.LocalAgent.MTP.<active MTP
instance>.STOMP.'
                    param_settings {
                            param: 'Destination'
                            value: '<new destination>'
                        }
                }
        }
    }
}
```

Test Metrics

1. After the STOMP destination was changed the EUT sent an UNSUBSCRIBE message message to the STOMP server.
2. After the EUT sent an UNSUBSCRIBE to the STOMP server it sent a SUBSCRIBE message with the new destination to the STOMP server.

6.11 STOMP - Use of TLS

Purpose

The purpose of this test is to ensure the EUT can establish secure STOMP communication via TLS.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure the EUT is configured to the use a STOMP server that exists in the test environment.
2. Ensure the EUT and STOMP server are configured with the appropriate certificates to communicate over TLS.

Test Procedure

1. Reboot the EUT

2. Wait for the EUT to reconnect to the STOMP server

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

4. Wait for the EUT to send a GetResponse

Test Metrics

1. All communication between the EUT and STOMP server after step 1 are encrypted using TLS

6.12 STOMP - Use of Connect Record

Purpose

The purpose of this test is to ensure the EUT correctly sends a Connect record after it has established a communications channel to the controller.

Functionality Tags

Conditional Mandatory (supports the STOMP MTP)

Test Setup

1. Ensure the EUT is configured to use a STOMP server that exists in the test environment.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the STOMP server.

Test Metrics

1. After reconnecting to the STOMP server the EUT transmits a STOMPConnectRecord to the test controller within 30 seconds.
2. The version in the received STOMPConnectRecord matches the expected version.
3. The subscribed_destination string in the received STOMPConnectRecord matches the destination the EUT is subscribed to.

7 WebSocket Test Cases

7.1 Session Establishment

Purpose

The purpose of this test is to ensure the EUT can establish a session using WebSocket as the MTP.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket and to communicate to the controller that exists in the test environment.

Test Procedure

1. Reboot the EUT.

2. Wait for the EUT to reconnect to the controller.

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

4. Wait for a GetResponse from the EUT

Test Metrics

1. The EUT is able to establish a WebSocket connection to the controller
2. The EUT sends a GetResponse to the Get message sent in step 3

7.2 Use of only one session

Purpose

The purpose of this test is to ensure the EUT maintains only one WebSocket connection to a controller at a time.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP with requirement R-WS.6)

Test Setup

1. Ensure the EUT is configured to use WebSocket and to communicate to the controller that exists in the test environment.

Test Procedure

1. Send a Get message to the EUT using the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

2. Attempt to open a second WebSocket connection to the EUT.

3. Send the Get message outlined in step 1 over the new connection.

Test Metrics

1. Either the first WebSocket connection closes when the second is opened, or the second connection is rejected.

7.3 Agent session acceptance from Controller

Purpose

This test has been DEPRECATED as of version 1.0.1 of this test plan.

Functionality Tags

N/A

Test Setup

N/A

Test Procedure

N/A

Test Metrics

N/A

7.4 Closing a WebSocket Connection

Purpose

The purpose of this test is to ensure the EUT correctly implements the procedure to close a WebSocket connection.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket.
2. Ensure there is an active WebSocket connection between the EUT and the controller that was initiated by the EUT.

Test Procedure

1. Send a `Close` WebSocket control frame to the EUT.
2. Wait for the EUT to close the underlying TCP session.

Test Metrics

1. The EUT sends a WebSocket Close frame.

7.5 Rejection of Session Establishment

Purpose

The purpose of this test is to ensure the EUT will correctly reject WebSocket sessions.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket.
2. Configure the controller to reject WebSocket connections from the EUT.

Test Procedure

1. Configure the controller to not include the`Sec-WebSocket-Protocol` when opening new WebSocket connections.
2. Reboot the EUT
3. Attempt to start a WebSocket connection to the EUT.

Test Metrics

1. The EUT rejects the WebSocket connection with the missing`Sec-WebSocket-Protocol` header.

7.6 Error Handling - Unprocessed Records

Purpose

The purpose of this test is to ensure the EUT correctly closes the WebSocket connection when a malformed USP Record is received.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket
2. Ensure there is an active WebSocket connection between the EUT and controller.

Test Procedure

1. Send a malformed USP record to the EUT.

Test Metrics

1. After step 1 the EUT closes the WebSocket connection with a WebSocket`Close` control frame containing status code 1003.

7.7 Use of Ping and Pong frames

Purpose

The purpose of this test is to ensure the EUT correctly uses`Ping` and `Pong` control frames to keep the WebSocket session alive.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket
2. Ensure there is an active WebSocket session between the EUT and the Controller.

Test Procedure

1. Send a `Ping` control frame to the EUT.
2. Wait up to 60 seconds for a `Pong` control frame from the EUT.
3. Send a `Pong` control frame to the EUT.

Test Metrics

1. The EUT sends a `Pong` control frame in response to the`Ping` control frame.
2. The EUT doesn’t terminate the WebSocket connection after receiving an unsolicited`Pong` control frame.

7.8 WebSocket Session Retry

Purpose

The purpose of this test is to ensure the EUT will correctly attempt to reestablish a WebSocket session if a session is unexpectedly closed.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket.
2. Ensure there is an active WebSocket connection between the EUT and controller.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.<test controller
instance>.MTP.<active MTP instance>.'
        }
    }
}
```

2. Configure the controller to reject new WebSocket connections.
3. Terminate the underlying TCP connection on the existing WebSocket connection.
4. Wait for the EUT to attempt to establish a WebSocket connection.
5. Configure the controller to accept new WebSocket connections.
6. Wait for the EUT to attempt to establish a WebSocket connection.

Test Metrics

1. The EUT attempts to start a new WebSocket connection in conformance with the `SessionRetryMinimumWaitInterval` parameter.
2. The EUT makes a second attempt to start a new WebSocket connection in conformance with the `SessionRetryMinimumWaitInterval` and `SessionRetryIntervalMultiplier` parameters.

7.9 Use of TLS

Purpose

The purpose of this test is to ensure the EUT can establish and use a secure WebSocket connection.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to use WebSocket.
2. Ensure the EUT and controller both have the required certificates to secure a websocket connection.

Test Procedure

1. Reboot the EUT.

2. Wait for the EUT to connect to the controller.

3. Send a Get message to the EUT with the following structure:

   ```
   header {
       msg_id: '<msg_id>'
       msg_type: GET
   }
   body {
       request {
           get {
               param_paths: 'Device.DeviceInfo.'
           }
       }
   }
   ```

4. Wait for GetResponse from the EUT.

Test Metrics

1. The EUT starts a WebSocket connection with the controller using TLS.
2. The EUT sends a GetResponse in step 4.

7.10 WebSocket - Use of Connect Record

Purpose

The purpose of this test is to ensure the EUT correctly sends a Connect Record after it has established a WebSocket connection to the Controller.

Functionality Tags

Conditional Mandatory (supports the WebSocket MTP)

Test Setup

1. Ensure the EUT is configured to connect to the test controller using WebSocket.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the test controller.

Test Metrics

1. After reconnecting to the test controller the EUT transmits a WebSocketConnectRecord to the test controller within 30 seconds.

8 Discovery Test Cases

8.1 DHCP Discovery - Agent Request Requirements

Purpose

The purpose of this test is to ensure the EUT correctly requests controller information via DHCP.*Note: this test can be run over DHCPv4 or DHCPv6, depending on the deployment model of the EUT.*

Functionality Tags

Conditional Mandatory (supports discovery via DHCP Options)

Test Setup

1. Ensure the EUT is configured to request controller DHCP information.
2. Ensure the EUT is configured to acquire an address via DHCP.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to request an address via DHCP.

Test Metrics

1. The EUT includes a Vendor Class option with Enterprise Number 3561 and vendor-class-data “usp” in the DHCP request.

8.2 DHCP Discovery - Agent handling of received options

Purpose

The purpose of this test is to ensure the EUT can properly handle the USP options provided by a DHCP server.

Functionality Tags

Conditional Mandatory (supports discovery via DHCP Options)

Test Setup

1. Ensure the EUT is configured to request controller DHCP information
2. Ensure the EUT is configured to acquire an address via DHCP.
3. Ensure the EUT ProvisioningCode parameter is set to a value other than that which will be set during the test procedure.

Test Procedure

1. Configure the DHCP server to provide a null terminated provisioning code.

2. Reboot the EUT.

3. Wait for the EUT to request an address via DHCP.

4. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Controller.<test controller
instance>.'
        }
    }
}
```

5. Wait for the GetResponse from the EUT.

Test Metrics

1. The ProvisioningCode parameter found in the GetResponse matches the provisioning code configured on the DHCP server.

8.3 DHCP Discovery - FQDN Leads to DNS Query

Purpose

The purpose of this test is to ensure the EUT correctly uses DNS to retrieve additional controller information upon receiving a FQDN of a controller.

Functionality Tags

Conditional Mandatory (supports discovery via DHCP Options)

Test Setup

1. Ensure the EUT is configured to request controller information via DHCP.
2. Ensure the EUT is configured to acquire an address via DHCP.

Test Procedure

1. Configure the DHCP server to provide a controller URL with a FQDN.
2. Reboot the EUT.
3. Wait for the EUT to request an address.
4. Wait for the EUT to query the DNS with the FQDN.
5. Wait for the EUT to connect to the controller.

Test Metrics

1. After the EUT receives a FQDN in the DHCP Offer, the EUT uses DNS to retrieve additional information about the controller.

8.4 mDNS

Purpose

The purpose of this test is to ensure the EUT correctly implements mDNS.

Functionality Tags

Conditional Mandatory (supports discovery via mDNS, supports the Reboot:1 profile)

Test Setup

1. Ensure the EUT has mDNS enabled.
2. Ensure the controller exists on the same network as the EUT.
3. Ensure that the EUT has the Controller's URL, which contains ".local." is preconfigured on the EUT.
4. Ensure that a Subscription exists for the Boot! event on the EUT with the test Controller as the Recipient.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to send a mDNS request for the FQDN.
3. Allow the controller to respond to the mDNS request.

Test Metrics

1. After the EUT receives a FQDN via DHCP containing ".local." the EUT uses mDNS to resolve it.

8.5 mDNS and Message Transfer Protocols

Purpose

The purpose of this test is to ensure the EUT correctly advertises the MTP it supports.

Functionality Tags

Conditional Mandatory (supports discovery via mDNS)

Test Setup

1. Ensure the EUT has mDNS enabled.
2. Ensure the Controller exists on the same network as the EUT.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to acquire an address.
3. Wait for the EUT to send an unsolicited mDNS response.

Test Metrics

1. The EUT sends an unsolicited multicast DNS response containing correct SRV and TXT records that convey the DNS-SD Service Instance Name for each supported MTP.

8.6 DNS - DNS Record Requirements

Purpose

The purpose of this test is to ensure the EUT provides valid DNS-SD records.

Functionality Tags

Conditional Mandatory (supports discovery via mDNS)

Test Setup

1. Ensure mDNS is enabled on the EUT.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to acquire a new address.
3. Wait for to the EUT to send a multicast mDNS advertisement.

Test Metrics

1. The EUT sends a multicast mDNS advertisement containing a TXT record for every supported MTP.

2. Every TXT record in the mDNS advertisement has a “path” and “name” attribute.

8.7 mDNS request response

Purpose

The purpose of this test is to ensure the EUT will respond to mDNS requests.

Functionality Tags

Conditional Mandatory (supports discovery via mDNS)

Test Setup

1. Ensure that the EUT is configured to listen for mDNS requests.

Test Procedure

1. Reboot the EUT.
2. Send an mDNS query to the multicast domain that includes the EUT.
3. Wait for an mDNS response from the EUT.

Test Metrics

1. The EUT responds to the mDNS query with the proper information.

9 Functionality Test Cases

9.1 Use of the Timer! Event (DEPRECATED by 9.11)

Purpose

The purpose of this test is to ensure the Timer! event can be configured, and the EUT correctly triggers the event.

Functionality Tags

Conditional Mandatory (supports Device.LocalAgent.Controller.{i}.ScheduleTimer() command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.LocalAgent.Controller.<Controller
ID>.ScheduleTimer()'
            input_args {
                key: 'DelaySeconds'
                value: '60'
            }
        }
    }
}
```

2. Wait for the EUT to send a Notification.

Test Metrics

1. The EUT sends an OperateResponse with 'ScheduleTimer()' in the`executed_command` element.
2. The EUT sends a Notify message with an`event` element containing Timer!

9.2 Use of Device.LocalAgent.AddCertificate()

Purpose

The purpose of this test is to ensure the AddCertificate() operation on the EUT functions correctly.

Functionality Tags

Conditional Mandatory (supports Device.LocalAgent.AddCertificate() command)

Test Setups

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Have an alternate certificate that the EUT hasn't seen.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.LocalAgent.AddCertificate()'
            send_resp: true
            input_args {
                key: 'Alias'
                value: 'addedCert'
            }
            input_args {
                key: 'Certificate'
                value: '<new certificate>'
            }
        }
    }
}
```

2. Reconfigure the Controller to use the new certificate.

3. Reestablish a connection to the EUT.

4. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Certificate.'
        }
    }
}
```

5. Send an Operate message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: OPERATE
    }
    body {
        request {
            operate {
                command: 'Device.LocalAgent.Certificate.<cert instance>.Delete()'
            }
        }
    }
```

Test Metrics

1. The EUT sends an OperateResponse after step 1.
2. The EUT accepts the connection after the Controller has been reconfigured to use the new certificate.
3. The EUT returns a GetResponse after step 4 which contains an instance with an Alias which matches the certificate added in step 1.
4. The EUT sends an OperateResponse after step 5.

9.3 Upgraded the Agent’s Firmware - Autoactivate enabled

Purpose

The purpose of this test is to ensure the EUT can download firmware and automatically activate it using the AutoActivate parameter.

Functionality Tags

Conditional Mandatory (supports Firmware:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the EUT has a Subscription to the TransferComplete! and Boot! events with the Recipient being the Controller instance used for testing.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command:  'Device.DeviceInfo.FirmwareImage.<inactive
instance>.Download()'
            input_args {
                key: 'AutoActivate'
                value: 'true'
            }
            input_args {
                key: 'URL'
                value: '<firmware URL>'
            }
            input_args {
                key: 'Username'
                value: '<optional username>'
            }
            input_args {
                key: 'Password'
                value:'<optional password>'
            }
            input_args {
                key: 'FileSize'
                value: '<file size>'
            }
        }
    }
}
```

2. Wait for the EUT to send a Notification
3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.'
        }
    }
}
```

Test Metrics

1. The EUT sends a Notify message after step 1 containing `aoper_complete` element with a `command_name` of 'Download()'
2. The EUT sends a Notify message with a TransferComplete! event.
3. The EUT sends a Notify message with a Boot! event, with the 'FirmwareUpdated' argument set to true.
4. The EUT sends a GetResponse message after step 3 which shows that Device.DeviceInfo.ActiveFirmwareImage matches the FirmwareImage instance on which the Download() operation was called; also that Device.DeviceInfo.SoftwareVersion matches the expected version.

9.4 Upgrading the Agent's Firmware - Using TimeWindow, Immediate

Purpose

The purpose of this test is to ensure the EUT can activate a firmware image when a TimeWindow object is used with Immediately mode.

Functionality Tags

Conditional Mandatory (supports Firmware:1 profile with Activate() operation)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the EUT has a FirmwareImage instance containing inactive firmware.
3. Ensure the EUT has a Subscription instance for Boot! with the Controller used for testing set as the Recipient.
4. Ensure that the EUT has a Subscription instance for the OperationComplete notification with a NotifType equal to 'OperationComplete' and a ReferenceList that matches the path of the 'Activate()' command with the Controller used for testing set as the Recipient.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: "Device.DeviceInfo.FirmwareImage.<instance>.Activate()"
            input_args: {
                key: 'TimeWindow.1.Start'
                value: '1'
            }
            input_args: {
                key: 'TimeWindow.1.End'
                value: '100'
            }
            input_args: {
                key: 'TimeWindow.1.Mode'
                value: 'Immediately'
            }
        }
    }
}
```

2. Wait for Notify message from the EUT.

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.SoftwareVersion'
        }
    }
}
```

Test Metrics

1. The EUT sends a Notify message within 5 seconds with an OperationComplete element with a `command_name` of ‘Activate()’.
2. The EUT sends a Notify message with a Boot! event and a FirmwareUpdated argument set to true.
3. The EUT responds to the Get message with a GetResponse containing a SoftwareVersion element with the expected software version.

9.5 Upgrading the Agent’s Firmware - Using TimeWindow, AnyTime

Purpose

The purpose of this test is to ensure the EUT can activate a firmware image when a TimeWindow instance used with the AnyTime mode.

Functionality Tags

Conditionally Mandatory (implements Firmware:1 and Activate() operation)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the EUT has a FirmwareImage instance containing inactive firmware.
3. Ensure the EUT has a Subscription to the Boot! event with the Controller used for testing set as the Recipient.
4. Ensure that the EUT has a Subscription instance for the OperationComplete notification with a NotifType equal to 'OperationComplete' and a ReferenceList that matches the path of the 'Activate()' command with the Controller used for testing set as the Recipient.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.DeviceInfo.FirmwareImage.<inactive
instance>.Activate()'
            input_args: {
                key: 'TimeWindow.1.Start'
                value: '0'
            }
            input_args: {
                key: 'TimeWindow.1.End'
                value: '120'
            }
            input_args: {
                key: 'TimeWindow.1.Mode'
                value: 'AnyTime'
            }
        }
    }
}
```

2. Wait for a Notify message from the EUT.

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.SoftwareVersion'
        }
    }
}
```

Test Metrics

1. The EUT sends a Notify message within 2 minutes after step 1.
2. The Notify message has a OperationComplete element.
3. The EUT sends a Notify message with a Boot! event and a FirmwareUpdated argument set to true.
4. The EUT sends a GetResponse after step 3 with a SoftwareVersion parameter that matches the expected version.

9.6 Upgrading the Agent's Firmware - Validated Firmware

Purpose

The purpose of this test is to ensure the EUT can validate the integrity of downloaded firmware.

Functionality Tags

Conditional Mandatory (supports Firmware:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

2. Ensure that the EUT has Subscriptions to the TransferComplete! event notification with the Controller used for testing set as the Recipient.

Test Procedure

1. Send an Operate message to the EUT with the following structure using an invalid checksum:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.DeviceInfo.FirmwareImage.<inactive
slot>.Download()'
            input_args {
                key: 'URL'
                value: '<firmware URL>'
            }
            input_args {
                key: 'CheckSumAlgorithm'
                value: 'SHA-1'
            }
            input_args {
                key: 'CheckSum'
                value: '<invalid checksum>'
            }
        }
    }
}
```

2. Wait for a Notify message from the EUT.

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo.FirmwareImage.<previously used
instance>.'
        }
    }
}
```

Test Metrics

1. The EUT sends a Notify message with a TransferComplete! event.
2. The EUT sends a Get response with a Status parameter of ValidationFailed.

9.7 Upgrading the Agent’s Firmware - Download to Active Bank

Purpose

The purpose of this test is to ensure the EUT is capable downloading and installing new firmware for EUTs that may support only the active firmware bank.

Functionality Tags

Conditional Mandatory (supports Firmware:1 profile)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that the EUT has a Subscription to the TransferComplete! event with the recipient being the

instance used for testing.
3. Ensure the EUT has a Subscription to the Boot! event and the Controller used for testing is set as the Recipient.
4. Record the number of firmware banks the EUT supports.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.DeviceInfo.FirmwareImage.<active firmware
slot>.Download()'
            input_args {
                key: 'URL'
                value: '<firmware URL>'
            }
            input_args {
                key: 'AutoActivate'
                value: 'true'
            }
        }
    }
}
```

2. Wait for a Notify message from the EUT.

Test Metrics

1. The EUT sends a Notify message with an OperationComplete, if the EUT supports multiple firmware banks the OperationComplete may contain an error indicating that downloading to an active firmware slot is not allowed.
2. If an OperationComplete Notification is sent without an error, the EUT sends a Notify message with a Boot! event and a FirmwareUpdated argument set to true.

9.8 Upgrading the Agent's Firmware - Cancelling a request using the Cancel() command

Purpose

The purpose of this test is to ensure the EUT can correctly cancel a Download() operation.

Functionality Tags

Conditional Mandatory (supports Firmware:1 profile and Device.LocalAgent.Request.{i}.Cancel() operation)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure the EUT has inactive firmware in one of FirmwareImage slots.
3. Ensure the EUT has a subscription to the Boot! event with the Controller used for testing set as the Recipient.

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.DeviceInfo.FirmwareImage.<valid
instance>.Activate()'
            input_args {
                key: 'TimeWindow.1.Start'
                value: '120'
            }
            input_args {
                key: 'TimeWindow.1.End'
                value: '500'
            }
            input_args {
                key: 'TimeWindow.1.Mode'
                value: 'AnyTime'
            }
            send_resp: true
        }
    }
}
```

2. Send an message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.LocalAgent.Request.<returned in step
1>.Cancel()'
        }
    }
}
```

3. Wait up to 500 seconds for a Boot! event from the EUT.

4. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.LocalAgent.Request.'
        }
    }
}
```

Test Metrics

1. The EUT sends a OperationResponse after step 1 with a`executed_command` element of 'Activate()' and a `req_obj_path` referencing an entry in the Device.LocalAgent.Request table.
2. The EUT never sends a Boot! event.
3. In the GetResponse from the EUT after step 4, the Request instance is either non-existent or the Status parameter of the relevant request is either Cancelled or Cancelling.

9.9 Adding a New Controller - OnBoardRequest

Purpose

The purpose of this test is to ensure the EUT can handle the manual adding of a new Controller.

Functionality Tags

Conditional Mandatory (supports Controller:1 profile with the ability to create instances of the Device.LocalAgent.Controller. object, supports SendOnBoardRequest())

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP

Records to each other.

2. A valid role instance is configured on the EUT for use with the new certificate.
3. A valid certificate instance is configured on the EUT
4. A secondary Controller is configured and ready to communicate with another endpoint.

Test Procedure

1. Send an Add message to the EUT with the following structure.

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            create_objs {
                obj_path: 'Device.LocalAgent.Controller.'
                param_settings {
                        param: 'Alias'
                        value: 'usp-111-Controller'
                    }
                param_settings {
                        param: 'EndpointID'
                        value: '<new Controller endpoint ID>'
                    }
                param_settings {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings {
                        param: 'AssignedRole'
                        value: '<valid role instance>'
                    }
            }
        }
    }
}
```

2. Send an Add message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: ADD
    }
    body {
        request {
            add {
              create_objs {
                obj_path: 'Device.LocalAgent.Controller.<new Controller
    instance>.MTP.'
                param_settings: {
                        param: 'Enable'
                        value: 'true'
                    }
                param_settings: {
                        param: 'Protocol'
                        value: '<supported MTP>'
                    }
               }
                  #  .
                  #  .
                  #  <further parameters to configure supported MTP>
                  #  .
                  #  .
            }
        }
    }
```

3. Send an Operate message to the EUT with the following structure:

```
    header {
        msg_id: '<msg_id>'
        msg_type: OPERATE
    }
    body {
        request {
            operate {
                command: 'Device.LocalAgent.Controller.<new
    instance>.SendOnBoardRequest()'
            }
        }
    }
```

4. Allow the secondary Controller to receive the OnBoardRequest() and send a NotifyResponse.

Test Metrics

1. The EUT is able to start a session with the secondary Controller.
2. The EUT sends a Notify message to the secondary Controller containing an OnBoardRequest element.

9.10 Use of the Boot! event and BootParameters

Purpose

The purpose of this test is to ensure the EUT correctly triggers the Boot! event and correctly includes the configured BootParameters.

Functionality Tags

Conditional Mandatory (supports Reboot:1 profile, supports Device.DeviceInfo.BootFirmwareImage)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
            create_objs {
                    obj_path: 'Device.LocalAgent.Subscription.'
                    param_settings {
                            param: 'NotifType'
                            value: 'Event'
                        }
                    param_settings {
                            param: 'ReferenceList'
                            value: 'Device.Boot!'
                        }
                    param_settings {
                            param: 'Enable'
                            value: 'true'
                        }
                }
            create_objs {
                    obj_path: 'Device.LocalAgent.Controller.<Controller
instance>.BootParameter.'
                    param_settings {
                            param: 'Enable'
                            value: 'true'
                        }
                    param_settings {
                            param: 'ParameterName'
                            value: 'Device.DeviceInfo.BootFirmwareImage'
                        }
                }
        }
    }
}
```

2. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.Reboot()'
        }
    }
}
```

3. Wait for a Notify message from the EUT.

Test Metrics

1. After step 2 the EUT sends a Notify message with an event element containing a ParameterMap argument with ‘Device.DeviceInfo.BootFirmwareImage’

9.11 Use of the Timer! Event

Purpose

The purpose of this test is to ensure the Timer! event can be configured, and the EUT correctly triggers the event.

Functionality Tags

Conditional Mandatory (supports Device.ScheduleTimer() command)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that a Subscription object exists on the EUT with NotifType OperationComplete on

Device.ScheduleTimer().

Test Procedure

1. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.ScheduleTimer()'
            send_resp: true
            input_args {
                key: 'DelaySeconds'
                value: '60'
            }
        }
    }
}
```

2. Wait for the EUT to send a Notification.

Test Metrics

1. The EUT sends an OperateResponse with ScheduleTimer() in the executed_command element.
2. The EUT sends an OperationComplete Notify message with an event element containing ScheduleTimer().

10 Bulk Data Collection Test Cases

10.1 Use BulkData collection using HTTP and JSON

Purpose

The purpose of this test is to verify that EUT supports JSON BulkData collection over HTTP.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols, “JSON” ∈ Device.BulkData.EncodingTypes)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured to support receiving JSON BulkData transfers.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: 'JSON'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

6. Wait up to 130 ((`ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTP endpoint within 130 ((`ReportingInterval * 2) + 10`) seconds.
2. The encoding of the BulkData transfer is JSON and is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name ‘UpTime’.
4. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are encoded as URI parameters in the request.

10.2 Use BulkData collection using HTTPS and JSON

Purpose

The purpose of this test is to verify that EUT supports JSON BulkData collection over HTTPS.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols, “JSON” ∈ Device.BulkData.EncodingTypes)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTPS endpoint that is accessible by the EUT is configured to support receiving JSON BulkData transfers.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: 'JSON'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

6. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTPS endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the BulkData transfer is JSON and is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name 'UpTime'.
4. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are encoded as URI parameters in the request.

10.3 Use BulkData collection using HTTP and CSV

Purpose

The purpose of this test is to verify that EUT supports CSV BulkData collection over HTTP.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols, “CSV” ∈ Device.BulkData.EncodingTypes)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured to support receiving CSV BulkData transfers.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: 'CSV'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: SET
    }
    body {
      request {
        set {
            allow_partial: false
            update_objs {
                obj_path: 'Device.BulkData.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
            update_objs {
                obj_path: 'Device.BulkData.Profile.<instance identifier>.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
        }
      }
    }
```

6. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTP endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the BulkData transfer is CSV and is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name ‘UpTime’.
4. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are encoded as URI parameters in the request.

10.4 Use BulkData collection using HTTPS and CSV

Purpose

The purpose of this test is to verify that EUT supports CSV BulkData collection over HTTPS.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols, “CSV” ∈ Device.BulkData.EncodingTypes)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTPS endpoint that is accessible by the EUT is configured to support receiving CSV BulkData transfers.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: 'CSV'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

6. Wait up to 130 ((`ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTPS endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the BulkData transfer is CSV and is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name ‘UpTime’.
4. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are encoded as URI parameters in the request.

10.5 Use BulkData collection using HTTP with URI Parameters

Purpose

The purpose of this test is to verify that EUT supports BulkData collection over HTTP with extra URI parameters

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: '<Supported Encoding type>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.HTTP.RequestURIParameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.UpTime'
           }
        }
    }
  }
}
```

6. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

7. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTP endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The EUT includes the EUT UpTime encoded into the URI using the name ‘UpTime’
3. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are also encoded as URI parameters in the request.

10.6 Use BulkData collection using HTTPS with URI Parameters

Purpose

The purpose of this test is to verify that EUT supports BulkData collection over HTTPS with extra URI parameters

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTPS endpoint that is accessible by the EUT is configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: '<Supported Encoding type>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of https endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.HTTP.RequestURIParameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.UpTime'
           }
        }
    }
  }
}
```

6. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

7. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTPS endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The EUT includes the EUT UpTime encoded into the URI using the name ‘UpTime’
3. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are also encoded as URI parameters in the request.

10.7 BulkData collection retry mechanism over HTTP

Purpose

The purpose of this test is to verify that EUT supports BulkData collection retry mechanism.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: '<Supported Encoding type>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
```

```
              }
             param_settings {
                  param: 'HTTP.URL'
                  value: '<URL of http endpoint>'
              }
             param_settings {
                  param: 'HTTP.RetryEnable'
                  value: 'true'
              }
             param_settings {
                  param: 'HTTP.RetryMinimumWaitInterval'
                  value: '5'
              }
             param_settings {
                  param: 'HTTP.RetryIntervalMultiplier'
                  value: '2000'
              }
           }
       }
     }
   }
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Disable the HTTP BulkData collection endpoint.

6. Send a Set message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: SET
    }
    body {
      request {
        set {
            allow_partial: false
            update_objs {
                obj_path: 'Device.BulkData.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
            update_objs {
                obj_path: 'Device.BulkData.Profile.<instance identifier>.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
        }
      }
    }
```

7. Allow the EUT to attempt to send a BulkData transfer.
8. Wait for 13 seconds for the EUT to retry the BulkData transfer.
9. Wait for 23 seconds for the EUT to retry the BulkData transfer.
10. Enable the HTTP BulkData collection endpoint
11. Wait for 43 seconds for the EUT to retry the BulkData transfer.
12. Wait for the EUT to send a BulkData transfer

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTP endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.

2. The EUT retries sending the BulkData transfer 3 times with each time occurring within the expected retry interval.
3. The EUT does not attempt to retry the BulkData transfer after receiving a positive response in step 9.
4. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are encoded as URI parameters in each request.

10.8 Use BulkData collection using HTTP with wildcard parameter

Purpose

The purpose of this test is to verify that EUT supports BulkData collection over HTTP with a wildcarded parameter

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured.
3. Ensure there are at least 2 BootParameters configured for the test controller.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: '<Supported Encoding type>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'Enabled'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.Controller.*.BootParameter.*.Enable'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

6. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTP endpoint within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. Ensure at least two "Enable" are in the BulkData transfer with the parameter names matching the following expression: `Enabled\.[1-9][0-9]*\.[1-9][0-9]*`
3. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are also encoded as URI parameters in the request.

10.9 Use BulkData collection using HTTP with Object Path

Purpose

The purpose of this test is to verify that EUT supports BulkData collection over HTTP with an Object Path

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “HTTP” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a HTTP endpoint that is accessible by the EUT is configured.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'HTTP'
           }
          param_settings {
               param: 'EncodingType'
               value: '<Supported Encoding type>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'HTTP.URL'
               value: '<URL of http endpoint>'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'TestController'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.Controller.<controller instance>.'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: SET
    }
    body {
      request {
        set {
            allow_partial: false
            update_objs {
                obj_path: 'Device.BulkData.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
            update_objs {
                obj_path: 'Device.BulkData.Profile.<instance identifier>.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
        }
      }
    }
```

6. Wait up to 130 ((`ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfers to the HTTPS endpoint within 130 ((`ReportingInterval * 2) + 10`) seconds.
2. Ensure that all the parameters included in the BulkData transfer match`TestController.<parameter>` and that all expected parameters are present.
3. Ensure the Manufacturer OUI, Product Class and Serial Number or the USP Endpoint ID are also encoded as URI parameters in the request.

10.10 Use BulkData collection Push event

Purpose

The purpose of this test is to verify that EUT supports BulkData collection via the Push event.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “USPEventNotif” $\in$ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'USPEventNotif'
           }
          param_settings {
               param: 'EncodingType'
               value: '<JSON or CSV>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.LocalAgent.Subscription.'
          param_settings {
               param: 'NotifType'
               value: 'Event'
           }
          param_settings {
               param: 'ReferenceList'
               value: 'Device.BulkData.Profile.<instance identifier>.Push!'
           }
          param_settings {
               param: 'Enable'
               value: 'true'
           }
        }
    }
  }
}
```

6. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

7. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfer Push! events to the controller within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the Data parameter in the Push! notification is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name 'UpTime'.

10.11 Use BulkData collection Push event with Wildcard path

Purpose

The purpose of this test is to verify that EUT supports BulkData collection via the Push event using a wildcard path.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “USPEventNotif” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure that there are at least 2 BootParameters configured for the test Controller.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'USPEventNotif'
           }
          param_settings {
               param: 'EncodingType'
               value: '<JSON or CSV>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'Enabled'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.Controller.*.BootParameter.*.Enable'
           }
        }
    }
  }
}
```

5. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.LocalAgent.Subscription.'
          param_settings {
               param: 'NotifType'
               value: 'Event'
           }
          param_settings {
               param: 'ReferenceList'
               value: 'Device.BulkData.Profile.<instance identifier>.Push!'
           }
          param_settings {
               param: 'Enable'
               value: 'true'
           }
        }
    }
  }
}
```

6. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

7. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfer Push! events to the controller within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the Data parameter in the Push! notification is well formed (parsable).
3. The BulkData transfer contains at least 2 parameters, one for each of the configured BootParameters and the name of the parameters match the expected name `Enabled\.[1-9][0-9]*\.[1-9][0-9]*`.

10.12 Use BulkData collection Push event with Object path

Purpose

The purpose of this test is to verify that EUT supports BulkData collection via the Push event using an object path.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “USPEventNotif” ∈ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'USPEventNotif'
           }
          param_settings {
               param: 'EncodingType'
               value: '<JSON or CSV>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse

3. Record the instance identifiers of the created objects as reported by the EUT.

4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'Controller'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.LocalAgent.Controller.'
           }
        }
    }
  }
}
```

5. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.LocalAgent.Subscription.'
          param_settings {
               param: 'NotifType'
               value: 'Event'
           }
          param_settings {
               param: 'ReferenceList'
               value: 'Device.BulkData.Profile.<instance identifier>.Push!'
           }
          param_settings {
               param: 'Enable'
               value: 'true'
           }
        }
    }
  }
}
```

6. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}
body {
  request {
    set {
        allow_partial: false
        update_objs {
            obj_path: 'Device.BulkData.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
        update_objs {
            obj_path: 'Device.BulkData.Profile.<instance identifier>.'
            param_settings {
              param: 'Enable'
              value: 'true'
            }
          }
    }
  }
}
```

7. Wait up to 130 (`(ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT sent 2 BulkData transfer Push! events to the controller within 130 (`(ReportingInterval * 2) + 10`) seconds.
2. The encoding of the Data parameter in the Push! notification is well formed (parsable).
3. The BulkData transfer contains at parameters with names using the prefix “Controller”.

10.13 Use BulkData collection over MQTT

Purpose

The purpose of this test is to verify that EUT supports BulkData collection via MQTT.

Functionality Tags

Conditional Mandatory (supports BulkDataColl:1, “MQTT” $\in$ Device.BulkData.Protocols)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP Records to each other.
2. Ensure a MQTT endpoint that is accessible by the EUT is enabled in the test environment.
3. Ensure there is a Device.MQTT.Client. entry in the EUT’s data model for the MQTT endpoint mentioned in step 2 of the test setup.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.'
          param_settings {
               param: 'Protocol'
               value: 'MQTT'
           }
          param_settings {
               param: 'EncodingType'
               value: '<JSON or CSV>'
           }
          param_settings {
               param: 'ReportingInterval'
               value: 'max(60, Device.BulkData.MinReportingInterval)'
           }
          param_settings {
               param: 'MQTT.Reference'
               value: '<MQTT.Client instance from step 3 of test setup>'
           }
          param_settings {
               param: 'MQTT.PublishTopic'
               value: 'mqtt-bulkdata'
           }
        }
    }
  }
}
```

2. Allow the EUT to send an AddResponse
3. Record the instance identifiers of the created objects as reported by the EUT.
4. Send a Add message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: ADD
}
body {
  request {
    add {
      allow_partial: false
      create_objs {
          obj_path: 'Device.BulkData.Profile.<instance
identifier>.Parameter.'
          param_settings {
               param: 'Name'
               value: 'UpTime'
           }
          param_settings {
               param: 'Reference'
               value: 'Device.DeviceInfo.UpTime'
           }
        }
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
    header {
      msg_id: '<msg_id>'
      msg_type: SET
    }
    body {
      request {
        set {
            allow_partial: false
            update_objs {
                obj_path: 'Device.BulkData.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
            update_objs {
                obj_path: 'Device.BulkData.Profile.<instance identifier>.'
                param_settings {
                  param: 'Enable'
                  value: 'true'
                }
              }
        }
      }
    }
```

6. Wait up to 130 ((`ReportingInterval * 2) + 10`) seconds

Test Metrics

1. After enabling the BulkData profile the EUT published 2 BulkData transfers to the MQTT server using using the configured topic within 130 ((`ReportingInterval * 2) + 10`) seconds.
2. The encoding of the Data parameter in the MQTT frame notification is well formed (parsable).
3. The BulkData transfer contains the one parameter configured in step 4 and the name of the parameter matches the expected name ‘UpTime’.

11 MQTT Test Cases

11.1 Support of Required MQTT Profiles

Purpose

The purpose of this test is to ensure the EUT supports the required MQTT profiles.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure that the EUT and test equipment have the necessary information to send and receive USP records to each other.

Test Procedure

1. Send a GetSupportedDM message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET_SUPPORTED_DM
}
body {
    request {
        get_supported_dm {
            obj_paths: 'Device.MQTT.'
            obj_paths: 'Device.LocalAgent.'
            return_params: true
            first_level_only: false
        }
    }
}
```

2. Wait for the GetSupportedDMResponse.

Test Metrics

1. The EUT sends a GetSupportedDMResponse.
2. The GetSupportedDMResponse from the EUT contains all required parameters in the MQTTClientCon:1, MQTTClientSubscribe:1, MQTTAgent:1, and MQTTController:1 data model profiles. The parameter `Device.MQTT.Client.{i}.MessageRetryTime` is not required to be supported for EUTs that only implement MQTT version 5.0.

11.2 MQTT session establishment using a CONNECT packet

Purpose

The purpose of this test is to ensure the EUT can properly start an MQTT session using an MQTT CONNECT packet.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.
2. Ensure that the EUT data model is configured with `.MQTT.Client.{i}.Username`, `.MQTT.Client.{i}.Password` values.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the MQTT server.

Test Metrics

1. The EUT sends an MQTT CONNECT packet to the MQTT server.
2. The MQTT CONNECT packet Version is either 5.0 or 3.1.1.
3. If the EUT uses MQTT 5.0, the MQTT CONNECT packet contains a User Property name-value pair

with name of “usp-endpoint-id” and value of the EUT USP Endpoint ID.

4. The EUT includes User Name and Password fields in the MQTT CONNECT packet.

11.3 MQTT Use of TLS

Purpose

The purpose of this test is to ensure the EUT can establish secure MQTT communication via TLS.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.
2. Ensure the EUT and MQTT server are configured with the appropriate certificates to communicate over TLS.

Test Procedure

1. Reboot the EUT

2. Wait for the EUT to reconnect to the MQTT server

3. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

4. Wait for the EUT to send a GetResponse

Test Metrics

1. All communication between the EUT and MQTT server after step 1 are encrypted using TLS 1.2 or later.

11.4 MQTT 5.0 ClientID

Purpose

The purpose of this test is to ensure the EUT properly sets the ClientID field in MQTT packets.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP, version 5.0)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: SET
}

body {
  request {
    set {
      allow_partial: false
      update_objs {
        obj_path: 'Device.MQTT.Client.<active MQTT client instance>.'
        param_settings {
         param: 'ClientID'
         value: ''
         required: true
        }
      }
    }
  }
}
```

2. Reboot the EUT.

3. Wait for the EUT to reconnect to the MQTT server.

4. The MQTT server sends an MQTT CONNACK packet with an Assigned Client Identifier.

5. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.MQTT.Client.<active MQTT client
instance>.ClientID'
    }
  }
}
```

Test Metrics

1. The EUT MQTT CONNECT packet must include a ClientID set to an empty string.
2. The retrieved value of ClientID matches the Assigned Client Identifier from step 4.

11.5 MQTT ClientID Persistence

Purpose

The purpose of this test is to ensure the MQTT ClientID field persists after successful connection with an MQTT server.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.MQTT.Client.<active MQTT client
instance>.ClientID'
    }
  }
}
```

2. Reboot the EUT.
3. Wait for the EUT to reconnect to the MQTT server.

Test Metrics

1. The EUT uses the same ClientID in the subsequent MQTT CONNECT packet.

11.6 MQTT Message Retry

Purpose

The purpose of this test is to ensure the EUT properly enters a retry state when it fails to connect to the MQTT server.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that is part of the test environment.

Test Procedure

1. Send a Get message to the EUT with the following structure

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.MQTT.Client.'
        }
    }
}
```

2. Send an Operate message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: OPERATE
}
body {
    request {
        operate {
            command: 'Device.Reboot()'
        }
    }
}
```

3. Disable the MQTT server.
4. Allow the EUT to attempt to start an MQTT session with the MQTT server.
5. Reenable the MQTT server after the EUT fails to connect to the MQTT server twice.

Test Metrics

1. The EUT retries connecting to the MQTT server within the`ConnectRetryTime` of the connection instance.
2. The EUT retries a second time in accordance with`ConnectRetryTime` and `ConnectRetryIntervalMultiplier`.

11.7 MQTT Keep Alive

Purpose

The purpose of this test is to ensure the EUT can correctly implement the MQTT keep alive mechanism and the relevant parameters in the data model.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP, version 5.0)

Test Setup

1. The EUT is configured to use an MQTT server which exists in the test environment.

Test Procedure

1. Send a Set message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: SET
}
body {
    request {
        set {
            update_objs {
                    obj_path: 'Device.MQTT.Client.<active MQTT client
instance>.'
                    param_settings {
                            param: 'KeepAliveTime'
                            value: '60'
                        }
                }
        }
    }
}
```

2. Reboot the EUT.

3. Wait for the EUT to reconnect to the MQTT server.

4. The MQTT server sends an MQTT CONNACK packet with a Keep Alive value of 30 seconds.
5. Wait 45 seconds.

Test Metrics

1. The EUT sends an MQTT CONNECT packet on boot with the Keep Alive property set to 60 seconds.
2. The EUT sends an MQTT PINGREQ packet within 45 seconds of the last MQTT message. This represents 1.5 times the Keep Alive value.

11.8 MQTT SUBSCRIBE Packet

Purpose

The purpose of this test is to ensure the EUT includes the correct fields in an MQTT SUBSCRIBE packet.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP, version 5.0)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the MQTT server.
3. The MQTT server sends an MQTT CONNACK packet containing a User Property of subscribe-topic.
4. For an EUT using MQTT 5.0, the MQTT CONNACK packet contains a Response Information property.
5. Wait for the EUT to send an MQTT SUBSCRIBE packet.

Test Metrics

1. The EUT sends an MQTT CONNECT packet to the MQTT server. If the EUT uses MQTT 5.0, the Request Response Information property must be set to 1.
2. The EUT sends an MQTT SUBSCRIBE packet to the MQTT server. The MQTT SUBSCRIBE packet includes the subscribe-topic sent in step 3.
3. If the EUT uses MQTT 5.0, the MQTT SUBSCRIBE packet includes the Response Information property sent in step 4.

11.9 MQTT New Subscription

Purpose

The purpose of this test is to ensure the EUT sends an MQTT SUBSCRIBE packet when a new `Device.MQTT.Client.{i}.Subscription.{i}.` object is added.

Functionality Tags

Conditional Mandatory (supports MQTT MTP)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.

Test Metrics

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
          allow_partial: false
          create_objs {
              obj_path: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.'
              param_settings {
                      param: 'Enable'
                      value: 'true'
                  }
              param_settings {
                      param: 'Topic'
                      value: 'newTopic-11-9'
                  }
              }
        }
    }
}
```

2. Wait for the EUT to send an MQTT SUBSCRIBE packet.
3. The MQTT server sends an MQTT SUBACK packet indicating the QoS level that was granted for the subscription.
4. Send a Get message with topic `newTopic-11-9` with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.<instance identifier from step 1>.'
    }
  }
}
```

5. Send a Set message to the EUT with the following structure:

```
header {
  msg_id: "<msg_id>"
  msg_type: SET
}

body {
  request {

    set {
      allow_partial: false
      update_objs {
        obj_path: "Device.MQTT.Client.<active MQTT client
instance>.Subscription.<instance identifier from step 1>"
        param_settings: {
         param: "Enable"
         value: "false"
         required: true
        }
      }
    }
  }
}
```

6. The MQTT Server sends an MQTT UNSUBACK packet indicating success.

Test Metrics

1. The EUT sends an MQTT SUBSCRIBE packet that includes the new Topic from step 1.
2. The EUT sends a GetResp for the Subscription.
3. The EUT sends an MQTT UNSUBSCRIBE packet for the configured Topic after Enable is set to false.

11.10 MQTT No Topic in CONNACK

Purpose

The purpose of this test is to ensure the EUT will disconnect from the MQTT server if it receives no subscribe-topic.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP, version 5.0)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.
2. Ensure that all instances of `Device.MQTT.Client.<active MQTT client instance>.Subscription.` are removed from the EUT.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to reconnect to the MQTT server.
3. The MQTT server sends an MQTT CONNACK packet that does not include a subscribe-topic User Property.

Test Metrics

1. The EUT sends an MQTT DISCONNECT packet to the MQTT server.

11.11 MQTT Failure to Subscribe

Purpose

The purpose of this test is to ensure the EUT will disconnect from the MQTT server if it is unable to subscribe to a Topic.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure that the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to send an MQTT SUBSCRIBE packet.
3. The MQTT server sends an MQTT SUBACK packet that includes an error for each Topic in the SUBSCRIBE packet.

Test Metrics

1. The EUT sends an MQTT DISCONNECT packet to the MQTT server.

11.12 MQTT PUBLISH Packet

Purpose

The purpose of this test is to ensure the EUT can send a properly formatted an MQTT PUBLISH packet.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Reboot the EUT.

2. Wait for the EUT to reconnect to the MQTT server.

3. The MQTT server sends a CONNACK packet.

4. For an EUT using MQTT 5.0, the CONNACK packet contains a Response Information property.

5. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

6. Wait for the EUT to send a GetResponse

Test Metrics

1. The EUT sends an MQTT PUBLISH packet containing a GetResponse.
2. If the EUT uses MQTT 5.0, the Response Topic is set to the Response Information property from step 4.
3. If the EUT uses MQTT 3.1.1, the “reply to” is included after`/reply-to=` at the end of the PUBLISH Topic Name, with any `/` character in the Topic replaced by`%2F`.
4. If the EUT uses MQTT 5.0, the Content Type property is set to`usp.msg`

11.13 MQTT QoS

Purpose

The purpose of this test is to ensure the EUT supports at least MQTT QoS levels 0 and 1.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
          allow_partial: false
          create_objs {
              obj_path: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.'
              param_settings {
                      param: 'Enable'
                      value: 'true'
                  }
              param_settings {
                      param: 'Topic'
                      value: 'newTopic-11-13-QoS0'
                  }
              }
        }
    }
}
```

2. Wait for the EUT to send an MQTT SUBSCRIBE packet.

3. The MQTT server sends an MQTT SUBACK packet indicating a QoS level of 0 for the subscription.

4. Send an Add message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: ADD
}
body {
    request {
        add {
          allow_partial: false
          create_objs {
              obj_path: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.'
              param_settings {
                      param: 'Enable'
                      value: 'true'
                  }
              param_settings {
                      param: 'Topic'
                      value: 'newTopic-11-13-QoS1'
                  }
              param_settings {
                      param: 'QoS'
                      value: '1'
                  }
              }
        }
    }
}
```

5. The MQTT server sends an MQTT SUBACK packet indicating a QoS level of 1 for the subscription.

6. Send a Get message with topic ‘newTopic-11-13-QoS0’ to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.<instance from step 1>.'
    }
  }
}
```

7. Send a Get message with topic ‘newTopic-11-13-QoS1’ to the EUT with the following structure:

```
header {
  msg_id: '<msg_id>'
  msg_type: GET
}
body {
  request {
    get {
      param_paths: 'Device.MQTT.Client.<active MQTT client
instance>.Subscription.<instance from step 4>.'
    }
  }
}
```

Test Metrics

1. The EUT sends a GetResp for the Get message sent to topic ‘newTopic-11-13-QoS0’.
2. The EUT sends a GetResp for the Get message sent to topic ‘newTopic-11-13-QoS1’.

11.14 MQTT Reply to Topic

Purpose

The purpose of this test is to ensure the EUT can process and set the “reply to” Topic in MQTT PUBLISH packets.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

2. Wait for the EUT to send a GetResponse.

Test Metrics

1. If the EUT uses MQTT 5.0, the EUT must send an MQTT PUBLISH packet that includes a GetResponse. The Topic Name must be set to the “reply to” Topic from the controller’s MQTT PUBLISH packet Response Topic property. The EUT must include a Response Topic property that has a “reply to” Topic set.
2. If the EUT uses MQTT 3.1.1, the EUT must send an MQTT PUBLISH packet that includes a GetResponse. The Topic must be set to the reply to topic at the end of the Topic Name in the Controller’s MQTT PUBLISH packet. All instances of `%2F` must be replaced by `/` in the Topic Name.

The EUT must set a reply to topic by including it at the end of the Topic Name after `/reply-to=` and replacing all instances of `/` with `%2F`.

11.15 MQTT 5.0 Content Type

Purpose

The purpose of this test is to ensure the EUT can accept valid values of the MQTT Content Type property.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP, version 5.0)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Configure the Controller to include an MQTT Content Type property of`usp.msg` in its MQTT packets.
2. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

3. Wait for the EUT to send a GetResponse.
4. Configure the Controller to include an MQTT Content Type property of `application/vnd.bbf.usp.msg` in its MQTT packets.
5. Send a Get message to the EUT with the following structure:

```
header {
    msg_id: '<msg_id>'
    msg_type: GET
}
body {
    request {
        get {
            param_paths: 'Device.DeviceInfo'
        }
    }
}
```

6. Wait for the EUT to send a GetResponse.

Test Metrics

1. The EUT must send a GetResponse for both Get messages, indicating that it processed the Controller’s MQTT PUBLISH packets.

11.16 MQTT Connection Retry

Purpose

The purpose of this test is to ensure the EUT retries its connection with the MQTT server after the server terminates the connection.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Send an MQTT DISCONNECT packet to the EUT.
2. Allow the EUT to start a new MQTT session with the MQTT server.

Test Metrics

1. The EUT retries connecting to the MQTT server between ConnectRetryTime of the connection instance and ConnectRetryTime*(ConnectRetryIntervalMultiplier/1000).

11.17 MQTT - Use of Connect Record

Purpose

The purpose of this test is to ensure the EUT correctly sends a Connect record after it has established a communications channel to the controller.

Functionality Tags

Conditional Mandatory (supports the MQTT MTP)

Test Setup

1. Ensure the EUT is configured to use an MQTT server that exists in the test environment.

Test Procedure

1. Reboot the EUT.
2. Wait for the EUT to establish an MQTT session with the MQTT server.

Test Metrics

1. After reconnecting to the MQTT server, the EUT transmits an MQTTConnectRecord within 30 seconds. The EUT includes the MQTTVersion field set to the correct MQTT version and the subscribed_topic field set to a Topic that the EUT is subscribed to.