ONVIF®

Media2 Configuration Device Test Specification

Version 25.06

June 2025

REVISION HISTORY

Vers.	Date	Description
16.06	Feb 12, 2016	Original publication
16.06	Feb 19, 2016	Step 11 and diagram have been updated in test 4.1.3 updated according to the feedback of Bhetanabottla Sriram
16.06	Feb 25, 2016	The issue in last step in tests MEDIA2-4-2-3 has been fixed
16.06	Mar 8, 2016	The tests 4.1.1 – 4.1.3 have been updated according to the feedback from F2F Tokyo. OSD configuration tests have been added
16.06	Mar 15, 2016	The tests MEDIA2-4-4-1 - MEDIA2-4-4-4 have been updated according to Fredrik's feedback.
16.06	Apr 4, 2016	The response for SetOSD and DeleteOSD requests have been updated according to the notes from Sano Hiroyuki
16.07	Jun 23, 2016	Test numbering and document version have been fixed
16.07	Jul 13, 2016	F2F commnets implemented.
16.07	Jul 14, 2016	Notes from F. Svensson implemented. Tests sequences updated.
16.07	Jul 27, 2016	Review comments implemented
16.07	Aug 8, 2016	Comments from Nicolas Brochu implemented
17.01	Oct 4, 2016	The test MEDIA2-4-4-2 have been updated. Pre-Requisite of the tests MEDIA2-4-4-1 - MEDIA2-4-4-4 have been updated.
17.01	Oct 13, 2016	Test case MEDIA2-4-5-1 SNAPSHOT URI has been added. Annex A.5 Configure Media profile with Video Source Configuration and Video Encoder Configuration has been added.
17.01	Oct 26, 2016	The following test cases and annexes were added in the scope of #1154: MEDIA2-5-1-1 GET VIDEO SOURCE CONFIGURATION OPTIONS MEDIA2-5-1-2 GET VIDEO SOURCE CONFIGURATIONS MEDIA2-5-1-3 VIDEO SOURCE CONFIGURATIONS AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY MEDIA2-5-1-4 PROFILES AND VIDEO SOURCE CONFIGURATIONS CONSISTENCY MEDIA2-5-1-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS MEDIA2-5-1-6 GET VIDEO SOURCE CONFIGURATIONS – INVALID TOKEN A.8 Get Service Capabilities A.7 Get Video Source Configurations List

		A.6 Configure Media profile with Video Source Configuration
17.01	Oct 26, 2016	The following test cases and annexes were added in the scope of #1160: MEDIA2-6-1-1 GET AUDIO SOURCE CONFIGURATION OPTIONS MEDIA2-6-1-2 GET AUDIO SOURCE CONFIGURATIONS MEDIA2-6-1-3 AUDIO SOURCE CONFIGURATIONS AND AUDIO SOURCE CONFIGURATION OPTIONS CONSISTENCY MEDIA2-6-1-4 PROFILES AND AUDIO SOURCE CONFIGURATIONS CONSISTENCY MEDIA2-6-1-5 MODIFY ALL SUPPORTED AUDIO SOURCE CONFIGURATIONS MEDIA2-6-1-6 GET AUDIO SOURCE CONFIGURATIONS – INVALID TOKEN A.9 Get Audio Source Configurations List A.10 Configure Media profile with Audio Source Configuration
17.01	Oct 26, 2016	The following test cases and annexes were added in the scope of #1166: MEDIA2-4-5-1 SNAPSHOT URI A.5 Configure Media profile with Video Source Configuration and Video Encoder Configuration
17.01	Oct 26, 2016	The following test cases and annexes were added in the scope of #1170: MEDIA2-4-1-2 CREATE MEDIA PROFILE WITH PRE-DEFINED CONFIGURATION MEDIA2-4-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION MEDIA2-4-2-4 VIDEO ENCODER CONFIGURATIONS – ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS MEDIA2-4-3-1 G.711 AUDIO ENCODER CONFIGURATION MEDIA2-4-3-2 AAC AUDIO ENCODER CONFIGURATION MEDIA2-6-1-5 MODIFY ALL SUPPORTED AUDIO SOURCE CONFIGURATIONS MEDIA2-5-1-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS The following annexes were added in the scope of #1170: A.11 Delete Media Profile if Max Reached A.12 Crete PullPoint Subscription A.14 Retrieve Profile Changed Event by PullPoint A.13 Delete Subscription A.16 Get Video Encoder Configurations List

		A.17 Get Audio Encoder Configurations List A.15 Retrieve Configuration Changed Event by PullPoint
17.01	Oct 27, 2016	MEDIA2-4-4-2 was updated to get last PositionOption for second create
17.01	Oct 28, 2016	The following test cases and annexes were added in the scope of #1162: MEDIA2-7-1-1 GET AUDIO OUTPUT CONFIGURATION OPTIONS MEDIA2-7-1-2 GET AUDIO OUTPUT CONFIGURATIONS MEDIA2-7-1-3 AUDIO OUTPUT CONFIGURATIONS AND AUDIO OUTPUT CONFIGURATION OPTIONS CONSISTENCY MEDIA2-7-1-4 PROFILES AND AUDIO OUTPUT CONFIGURATIONS CONSISTENCY MEDIA2-7-1-5 MODIFY ALL SUPPORTED AUDIO OUTPUT CONFIGURATIONS MEDIA2-7-1-6 GET AUDIO OUTPUT CONFIGURATIONS – INVALID TOKEN MEDIA2-8-1-1 GET AUDIO DECODER CONFIGURATION OPTIONS MEDIA2-8-1-2 GET AUDIO DECODER CONFIGURATIONS MEDIA2-8-1-3 PROFILES AND AUDIO DECODER CONFIGURATIONS CONSISTENCY MEDIA2-8-1-4 MODIFY ALL SUPPORTED AUDIO DECODER CONFIGURATIONS MEDIA2-8-1-5 GET AUDIO DECODER CONFIGURATIONS – INVALID TOKEN A.18 Get Audio Output Configurations List A.19 Configure Media profile with Audio Output Configuration A.20 Get Audio Decoder Configurations List A.21 Configure Media profile with Audio Output Configuration and Audio Decoder Configuration
17.01	Oct 28, 2016	The following test cases and annexes were added in the scope of #1172: MEDIA2-9-1-1 READY TO USE MEDIA PROFILE FOR PTZ
17.01	Oct 28, 2016	The following test cases and annexes were added in the scope of #1180: MEDIA2-10-1-1 GET VIDEO SOURCE MODES MEDIA2-10-1-2 SET VIDEO SOURCE MODES A.22 Get Video Sources List A.23 Waiting for Reboot

17.01	Nov 13, 2016	The following test cases and annexes were added in the scope of #1156: MEDIA2-4-2-5 VIDEO ENCODER INSTANCES A.24 Find Guaranteed Number of Media Profiles for Video Source Configuration A.25 Configure Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration A.26 Add Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration A.27 Create New or Configure Fixed Media Profiles to Get Guaranteed Number of Media Profiles for Video Source Configuration
17.01	Nov 23, 2016	The following test case was added in the scope of #1174: MEDIA2-9-1-2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ
17.01	Nov 27, 2016	The Annexes format was changed according to comment in #1166.
17.01	Nov 27, 2016	The test MEDIA2-2-4-2 was apdated according comments in #1180: Typos were fixed. The first not enabled will be used for first Set.
17.01	Nov 27, 2016	The following were updated according #1215: Command under test were updated. Step 8 was removed from MEDIA2-2-3-1. Step 5.1 was updated with description in MEDIA2-2-3-3.
17.01	Nov 27, 2016	The format was updated according #1238.
17.01	Nov 27, 2016	Test Structures and test IDs were updated according #1265.
17.01	Nov 27, 2016	The following annexes were updated in the scope of #1260: A.4 OSDConfigurationOptions and OSDConfiguration mapping
17.01	Dec 07, 2016	Annex A.27 were renamed. Annex A.27 Procedure Purpose were updated. Annex A.27 Procedure was fixed to Create Media Profile if maxProfiles was not reached.
17.01	Dec 08, 2016	Fixed typos and link according description in #1162. Fixed typos and link according description in #1166. Fixed typos and link according description in #1161. Fixed typos and link according description in #1170. Fixed typos and link according description in #1180. Fixed responses according in #1178.

17.01	Dec 12, 2016	MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION was updated: subscribtion creation were moved to the loop to prevent receiveing messages after Annex A.1.
17.01	Jan 09, 2017	MEDIA2-2-2-5 was updated according #1294 and #1154: Step 5.9 was updated. Step 5.3 was updated. Some typos was fixed according #1154. Step 5.9 was updated. MEDIA2-4-1-1 was updated according #1174: ONVIF Core Specification Coverage and Pre-Requisite were updated.
17.01	Jan 18, 2017	The test MEDIA2-2-3-4 VIDEO ENCODER CONFIGURATIONS – ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS was apdated according to #1293: GovLength parameter in SetVideoEncoderConfiguration request was updated.
17.06	Jan 26, 2017	The following test cases were added according to #1296: MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES MEDIA2-7-1-2 GET SERVICES AND GET MEDIA2 SERVICE CAPABILITIES CONSISTENCY
17.06	Jan 30, 2017	MEDIA2-3-4-4 MODIFY ALL SUPPORTED AUDIO DECODER CONFIGURATIONS test case was updated according to #1295.
17.06	Feb 09, 2017	MEDIA2-1-1-1 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING test case was updated according to #1284. MEDIA2-2-3-5 VIDEO ENCODER CONFIGURATION OPTIONS test case was added according to #1273.
17.06	Feb 20, 2017	MEDIA2-2-3-1 VIDEO ENCODER CONFIGURATION test case was updated according to #1215.
17.06	Mar 03, 2017	MEDIA2-1-1-1 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING test case was updated according to #1284 and #1345.
17.06	Mar 06, 2017	MEDIA2-4-1-2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ test case was updated according to #1307.
17.06	Mar 22, 2017	MEDIA2-1-1-4 GET PROFILES test case was added according to #1333.
17.06	Mar 24, 2017	Media2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS test case was updated according to #1312.
17.06	Mar 31, 2017	MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES test case was updated according to #1288. The following test cases were updated according to #1367: MEDIA2-3-2-1 G.711 AUDIO ENCODER CONFIGURATION

		MEDIA2-3-2-2 AAC AUDIO ENCODER CONFIGURATION
17.06	Apr 03, 2017	MEDIA2-1-1-5 CREATE MEDIA PROFILE WITH CONFIGURATIONS test case was added according to #1344.
17.06	Apr 24, 2017	MEDIA2-2-1-1 VIDEO ENCODER INSTANCES test case was updated according to #1156. Annex A.28 was added.
17.06	May 24, 2017	MEDIA2-2-3-1 VIDEO ENCODER CONFIGURATION test case was updated according to #1363.
17.12	Jul 13, 2017	MEDIA2-6-1-5 GET OSDS test case was added according to #1347.
17.12	Jul 14, 2017	MEDIA2-1-1-6 REMOVE ALL CONFIGURATIONS FROM MEDIA PROFILE test case was added according to #1404.
17.12	Sept 28, 2017	MEDIA2-2-2-7 PROFILES AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY test case was added according to #1484. MEDIA2-2-3-1 VIDEO ENCODER CONFIGURATION test case was changed according to #1485. The following test cases and annexes were added in the scope of #1488: MEDIA2-3-2-3 GET AUDIO ENCODER CONFIGURATION OPTIONS MEDIA2-3-2-4 AUDIO ENCODER CONFIGURATIONS AND AUDIO ENCODER CONFIGURATION OPTIONS CONSISTENCY
17.12	Oct 24, 2017	MEDIA2-1-1-7 FIXED MEDIA PROFILE CONFIGURATION test case was added according to #1472.
17.12	Oct 31, 2017	The following test case waere added according to #1500: MEDIA2-6-1-6 GET OSD OPTIONS MEDIA2-6-1-7 OSD CONFIGURATIONS AND OSD OPTIONS CONSISTENCY
17.12	Nov 03, 2017	The following test case waere added according to #1487: MEDIA2-4-1-2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ
17.12	Nov 24, 2017	MEDIA2-6-1-6 GET OSD OPTIONS updated according to #1534
17.12	Nov 27, 2017	MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES updated according to #1522
17.12	Nov 28, 2017	MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION was updated according to #1499 Annex A.1 Create Empty Profile was updated according to #1499 MEDIA2-6-1-7 OSD CONFIGURATIONS AND OSD OPTIONS CONSISTENCY was updated according to #1500
17.12	Nov 29, 2017	Media Configuration Test Cases\Video Configuration\General section was removed. Annex A.28 Remove all non-fixed Media Profiles and remove all configurations from fixed Media Profiles was removed.

		Annex A.27 Create New Media Profiles to Get Guaranted Number of Media Profiles for Video Source Configuration
17.12	Dec 01, 2017	MEDIA2-8-1-1 MODIFY ALL SUPPORTED METADATA CONFIGURATIONS was added according to #1486.
18.06	Jan 16, 2018	MEDIA2-6-1-4 SET OSD CONFIGURATION TEXT OVERLAY was changed according to #1519.
18.06	Jan 17, 2018	MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES was changed according to #1521.
18.06	Jan 23, 2018	The following test cases were updated according to #1483: MEDIA2-6-1-6 GET OSD OPTIONS MEDIA2-6-1-2 CREATE OSD CONFIGURATION FOR IMAGE OVERLAY MEDIA2-6-1-3 SET OSD CONFIGURATION IMAGE OVERLAY The following annex was added according to #1483: Annex A.28 OSD Picture File Parameters
18.06	Jan 24, 2018	The following test cases were updated according to #1546: MEDIA2-6-1-1 CREATE OSD CONFIGURATION FOR TEXT OVERLAY MEDIA2-6-1-2 CREATE OSD CONFIGURATION FOR IMAGE OVERLAY
18.06	Jan 24, 2018	MEDIA2-6-1-6 GET OSD OPTIONS was updated according to #1511.
18.06	Jan 24, 2018	The following test cases were updated according to #1575: MEDIA2-8-1-1 MODIFY ALL SUPPORTED METADATA CONFIGURATIONS (note and steps 5.3 and 5.9 were updated)
18.06	Jan 24, 2018	The following test cases were updated according to #1560: MEDIA2-2-3-4 VIDEO ENCODER CONFIGURATIONS – ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS (step 5.3.12 was removed, step 7 was added)
18.06	Mar 20, 2018	The following test cases were updated according to #1581: MEDIA2-6-1-6 GET OSD OPTIONS (step 4.3 and step 5 added) MEDIA2-6-1-7 OSD CONFIGURATIONS AND OSD OPTIONS CONSISTENCY (step 4.6.2 added)
18.06	Mar 21, 2018	The following test cases were updated according to #1547: Annex A.30 Device Configuration to Create OSD with Required Type added Annex A.29 Delete OSD added MEDIA2-6-1-1 CREATE OSD CONFIGURATION FOR TEXT OVERLAY (Pre-Requisite about maximum number of OSD removed, step 4.6, step 4.7, step 4.15, step 4.16, and step 4.25 added)

		MEDIA2-6-1-2 CREATE OSD CONFIGURATION FOR IMAGE OVERLAY (Pre-Requisite about maximum number of OSD removed, step 4.6, and step 4.24 added) MEDIA2-6-1-3 SET OSD CONFIGURATION IMAGE OVERLAY (Pre-Requisite about maximum number of OSD removed, step 4.7, and step 4.19 added) MEDIA2-6-1-4 SET OSD CONFIGURATION TEXT OVERLAY (Pre-Requisite about maximum number of OSD removed, step 4.7, and step 4.18 added)
18.06	Apr 09, 2018	Annex A.30 Device Configuration to Create OSD with Required Type updated according to #1608 Annex A.31 Delete All Text OSDs added according to #1608
18.06	Apr 18, 2018	MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES updated according to #1595 (step 16 added)
18.06	Apr 24, 2018	MEDIA2-2-3-4 SET ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS updated according to #1617
18.06	May 07, 2018	The following were updated according to #1620: MEDIA2-5-1-2 VIDEO ENCODER INSTANCES PER VIDEO SOURCE (added)
18.06	Jun 21, 2018	Reformatting document using new template
18.12	Aug 07, 2018	MEDIA2-7-1-1 MEDIA2 SERVICE CAPABILITIES updated according to #1691
18.06 SR1	Sep 19, 2018	The following were updated according to #1641: MEDIA2-2-3-4 SET ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS (note was updated to remove Quality, RateControl.FrameRateLimit, and RateControl.BitrateLimit from comparison)
18.12	Dec 07, 2018	The following were updated according to #1762: DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ (PTZConfiguration.Name value in 6.3.6 step was updated)
18.12	Dec 21, 2018	Switching Hub description in 'Network Configuration for DUT' section was updated according to #1737
19.06	Jan 10, 2019	The following were updated according to #1762: DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ (step 5 added, step 7.3.3 updated)
19.06	Apr 10, 2019	The following were updated according to #1764: MEDIA2-3-2-2 AAC AUDIO ENCODER CONFIGURATION (step 5.3 and step 5.3.1 updated)
19.06	Apr 26, 2019	The following were updated according to #1813: MEDIA2-4-1-2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ (steps 7.3.6 - 7.3.11 added)
19.12	Sep 23, 2019	The following were updated according to #1926: MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (steps 7- 8 replaced with steps 6.34-6.57)

19.12	Sep 25, 2019	The following were updated according to #1913: MEDIA2-1-1-4 GET PROFILES (steps 16 and 17 added)
19.12	Oct 08, 2019	The following were updated according to #1894: MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (steps with metadata moved under Metadata feature, see step 6.34) MEDIA2-8-1-1 MODIFY ALL SUPPORTED METADATA CONFIGURATIONS (Metadata feature added into Pre-Requisite)
19.12	Oct 09, 2019	The following were done according to #1913: MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (step 7 added) MEDIA2-9-1-1 GET ANALYTICS CONFIGURATIONS (test case added) MEDIA2-9-1-2 PROFILES AND ANALYTICS CONFIGURATIONS CONSISTENCY (test case added) MEDIA2-9-1-3 GET ANALYTICS CONFIGURATIONS – INVALID TOKEN (test case added) A.32 Get Analytics Configurations List (annex case added) MEDIA2-8-1-2 GET METADATA CONFIGURATIONS (test case added) MEDIA2-8-1-3 PROFILES AND METADATA CONFIGURATIONS CONSISTENCY (test case added) MEDIA2-8-1-4 GET METADATA CONFIGURATIONS – INVALID TOKEN (test case added)
19.12	Oct 11, 2019	The following were done according to #1957: MEDIA2-3-2-3 GET AUDIO ENCODER CONFIGURATION OPTIONS(Type parameter in step 7 updated, step 9 updated)
20.06	Mar 03, 2020	Minor changes according to #1913: MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (invalid step ID fixed) MEDIA2-1-1-4 GET PROFILES (Specification coverage section updated)
20.06	May 12, 2020	The following were updated according to #1901 and #1999: MEDIA2-1-1-2 CREATE MEDIA PROFILE WITH PRE-DEFINED CONFIGURATION (steps 5, 8, 17, 18 updated) MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (all steps with calling of A.12, A.14, and A.13 updated) MEDIA2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS (steps 4, 5.5, 5.11, 6 updated) MEDIA2-2-3-4 SET ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS (steps 4, 5.3.3, 5.3.9, 7 updated) MEDIA2-3-1-5 MODIFY ALL SUPPORTED AUDIO SOURCE CONFIGURATIONS (steps 4, 5.5, 6 updated)

		MEDIA2-3-2-1 G.711 AUDIO ENCODER CONFIGURATION (steps 4, 5.3.4, 5.3.11, 6 updated) MEDIA2-3-2-2 AAC AUDIO ENCODER CONFIGURATION (steps 4, 5.3.4, 5.3.11, 6 updated) MEDIA2-3-3-5 MODIFY ALL SUPPORTED AUDIO OUTPUT CONFIGURATIONS (steps 4, 5.5, 5.11, 6 updated) MEDIA2-4-1-2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ (steps 6, 7.3.3, 7.3.14, 7.3.27, 9 updated) MEDIA2-8-1-1 MODIFY ALL SUPPORTED METADATA CONFIGURATIONS (steps 4, 5.5, 5.11, 6 updated)
20.12	Jul 30, 2020	The following were updated according to #2072: MEDIA2-1-1-5 CREATE MEDIA PROFILE WITH CONFIGURATIONS (separated CreateProfile for each configuration)
20.12	Sep 28, 2020	The following were updated according to #2088: MEDIA2-2-3-1-v17.12 VIDEO ENCODER CONFIGURATION (Video feature was added in Pre-Requisite) MEDIA2-2-3-2 VIDEO ENCODER CONFIGURATIONS AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY (Video feature was added in Pre-Requisite) MEDIA2-2-3-3 PROFILES AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY (Video feature was added in Pre-Requisite) MEDIA2-2-3-4-v20.06 SET ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS (Video feature was added in Pre-Requisite) MEDIA2-2-3-5-v17.06 VIDEO ENCODER CONFIGURATION OPTIONS VALIDATION (Video feature was added in Pre-Requisite) MEDIA2-5-1-2 VIDEO ENCODER INSTANCES PER VIDEO SOURCE (Video feature was added in Pre-Requisite) MEDIA2-1-1-3 DYNAMIC MEDIA PROFILE CONFIGURATION (Step 6.10 was changed: steps for Video Encoder were moved under condition that Video is supported)
20.12	Oct 01, 2020	The following were updated according to #2099: MEDIA2-5-1-1 SNAPSHOT URI (step 4 added)
20.12	Nov 09, 2020	The following test case was added according to #2025: MEDIA2-1-1-8 READY TO USE MEDIA PROFILE FOR METADATA STREAMING
20.12	Dec 17, 2020	The following test case was added according to #2120: MEDIA2-1-1-9 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING (PROFILE M)
21.06	Apr 09, 2021	The following annex was updated according to #2124: A.23 Waiting for Reboot (condition with Discovery feature supporting added)

21.06	May 27, 2021	The following test case was updated according to #2216: MEDIA2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS (step 5.3 and 5.9 modified to skip change of Rotate if Reboot=true)
21.12	Aug 31, 2021	The following test case was updated according to #2216: MEDIA2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS (step 7 added)
21.12	Oct 07, 2021	The following test cases and annexes were added in the scope of #2222: MEDIA2-10-1-1 CREATE MASKS MEDIA2-10-1-2 GET MASKS MEDIA2-10-1-3 SET MASKS MEDIA2-10-1-4 GET MASK OPTIONS MEDIA2-10-1-5 MASK CONFIGURATIONS AND MASK OPTIONS CONSISTENCY A.34 Device Configuration For Create Mask A.35 Delete Mask A.36 Create Mask
21.12	Oct 13, 2021	The following test cases and annexes were added in the scope of #2222: MEDIA2-10-1-6 SINGLE COLOR ONLY PARAMETER A.37 Remove all Masks from Video Source Configuration
22.12	Oct 18, 2022	The following test cases were updated in the scope of #33: MEDIA2-10-1-2 GET MASKS (step 3 was added) The following annex was added in the scope of #33: Annex A.38 Device Configuration For Get Masks Test Case
22.12	Nov 15, 2022	The following test cases were updated in the scope of #33: MEDIA2-10-1-1 CREATE MASKS (Note about comparison at step 8 was added, editorial changes) MEDIA2-10-1-3 CREATE MASKS (Note about comparison at step 10 was added, editorial changes)
23.06	Mar 19, 2023	MEDIA2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS updated (according to #81): Steps 7.3.1, 7.3.2, 7.3.3, 7.3.4 added. Step 7.3.5 with SetVideoSourceConfiguration updated to choose rotate mode value different from current mode and to choose degree different from 0.
24.12	Oct 21, 2024	Video Source Configuration feature was added into Pre-Requisite of the following test cases in the scope of #242 ticket:

		MEDIA2-2-2-1 GET VIDEO SOURCE CONFIGURATION OPTIONS MEDIA2-2-2-2 GET VIDEO SOURCE CONFIGURATIONS MEDIA2-2-2-3 VIDEO SOURCE CONFIGURATIONS AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY MEDIA2-2-2-4 PROFILES AND VIDEO SOURCE CONFIGURATIONS CONSISTENCY MEDIA2-2-2-5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS MEDIA2-2-2-6 GET VIDEO SOURCE CONFIGURATIONS – INVALID TOKEN MEDIA2-2-2-7 PROFILES AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY MEDIA2-1-1-2 CREATE MEDIA PROFILE WITH PRE-DEFINED CONFIGURATION
24.12	Nov 4, 2024	Validation of ViewMode value added according of #269 ticket: MEDIA2-2-2-2 GET VIDEO SOURCE CONFIGURATIONS (step 7.1 added)
25.06	Feb 12, 2025	Validation of Fisheye ViewMode value added according of #220 ticket: MEDIA2-2-2-2 GET VIDEO SOURCE CONFIGURATIONS (step 7.1.2 added)

Table of Contents

1 Introduction

The goal of the ONVIF test specification set is to make it possible to realize fully interoperable IP physical security implementation from different vendors. The set of ONVIF test specification describes the test cases need to verify the [ONVIF Network Interface Specs] and [ONVIF Conformance] requirements. In addition, the test cases are to be basic inputs for some Profile specification requirements. It also describes the test framework, test setup, pre-requisites, test policies needed for the execution of the described test cases.

This ONVIF Media2 Test Specification acts as a supplementary document to the [ONVIF Network Interface Specs], illustrating test cases need to be executed and passed. And this specification acts as an input document to the development of test tool, which will be used to test the ONVIF device implementation conformance towards ONVIF standard. This test tool is referred as ONVIF Client hereafter.

1.1 Scope

This ONVIF Media2 Test Specification defines and regulates the conformance testing procedure for the ONVIF conformant devices. Conformance testing is meant to be functional black-box testing. The objective of this specification is to provide test cases to test individual requirements of ONVIF devices according to ONVIF Media2 Service, which is defined in [ONVIF Network Interface Specs].

The principal intended purposes are:

- Provide self-assessment tool for implementations.
- Provide comprehensive test suite coverage for [ONVIF Network Interface Specs].

This specification **does not** address the following:

- Product use cases and non-functional (performance and regression) testing.
- SOAP Implementation Interoperability test i.e. Web Service Interoperability Basic Profile version 2.0 (WS-I BP 2.0).
- Network protocol implementation Conformance test for HTTP, HTTPS, RTP and RTSP protocol.
- Poor streaming performance test (audio/video distortions, missing audio/video frames, incorrect lib synchronization etc.).

  Wi-Fi Conformance test

The set of ONVIF Test Specification will not cover the complete set of requirements as defined in [ONVIF Network Interface Specs]; instead it would cover subset of it. The scope of this specification

is to derive all the normative requirements of [ONVIF Network Interface Specs], which are related to ONVIF Media2 Service and some of the optional requirements.

This ONVIF Media2 Test Specification covers ONVIF Media2 service, which is a functional block of [ONVIF Network Interface Specs]. The following sections describe the brief overview of and scope of each functional block.

1.2 Media Configuration

Media Configuration section covers the test cases needed for the verification of media2 service features as mentioned in [ONVIF Network Interface Specs]. Media2 service is used to configure the media configurations.

Briefly it covers the following things:

- Manage media profiles.
- Manage configuration entities.
- Getting snapshot
- Manage OSD configurationd

The scope of this specification is to cover following configuration entities and Audio/Video media formats:

- Configuration Entities:
  - Video source configuration
  - Audio source configuration
  - Video encoder configuration
  - Audio encoder configuration
- Video Codec:
  - H.264
  - H.265
- Audio Codec:
  - G.711
  - AAC

- OSD:
  - Text Overlay
  - Image Overlay

2 Normative references

- [ONVIF Conformance] ONVIF Conformance Process Specification:

  https://www.onvif.org/profiles/conformance/

- [ONVIF Profile Policy] ONVIF Profile Policy:

  https://www.onvif.org/profiles/

- [ONVIF Network Interface Specs] ONVIF Network Interface Specification documents:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Core Specs] ONVIF Core Specification:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Media2 Spec] ONVIF Media 2 Specification:

  https://www.onvif.org/profiles/specifications/

- [ONVIF Base Test] ONVIF Base Device Test Specification:

  https://www.onvif.org/profiles/conformance/device-test/

- [ISO/IEC Directives, Part 2] ISO/IEC Directives, Part 2, Annex H:

  http://www.iso.org/directives

- [ISO 16484-5] ISO 16484-5:2014-09 Annex P:

  https://www.iso.org/obp/ui/#!iso:std:63753:en

- [SOAP 1.2, Part 1] W3C SOAP 1.2, Part 1, Messaging Framework:

  http://www.w3.org/TR/soap12-part1/

- [XML-Schema, Part 1] W3C XML Schema Part 1: Structures Second Edition:

  http://www.w3.org/TR/xmlschema-1/

- [XML-Schema, Part 2] W3C XML Schema Part 2: Datatypes Second Edition:

  http://www.w3.org/TR/xmlschema-2/

- [WS-Security] "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", OASIS Standard, February 2006.:

http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

- [RFC 2396] "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, MIT/LCS, R. Fielding, U.C. Irvine, L. Masinter, Xerox Corporation, August 1998:

  https://www.ietf.org/rfc/rfc2396.txt

3 Terms and Definitions

3.1 Conventions

The key words "shall", "shall not", "should", "should not", "may", "need not", "can", "cannot" in this specification are to be interpreted as described in [ISO/IEC Directives Part 2].

3.2 Definitions

This section describes terms and definitions used in this document.

Profile	See ONVIF Profile Policy.
ONVIF Device	Computer appliance or software program that exposes one or multiple ONVIF Web Services.
ONVIF Client	Computer appliance or software program that uses ONVIF Web Services.
Configuration Entity	A network video device media abstract component that is used to produce a media stream on the network, i.e. video and/or audio stream.
Media Profile	A media profile maps a video and/or audio source to a video and/or an audio encoder, PTZ and analytics configurations.
SOAP	SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols.
Device Test Tool	ONVIF Device Test Tool that tests ONVIF Device implementation towards the ONVIF Test Specification set.
Media 2 Service	Services to determine the streaming properties of requested media streams.

3.3 Abbreviations

This section describes abbreviations used in this document.

HTTP Hyper Text Transport Protocol.

AAC Advanced Audio Coding.

URI Uniform Resource Identifier.

WSDL Web Services Description Language.

XML eXtensible Markup Language.

JPEG Joint Photographic Experts Group.

TTL Time To Live.

4 Test Overview

This section describes about the test setup and prerequisites needed, and the test policies that should be followed for test case execution.

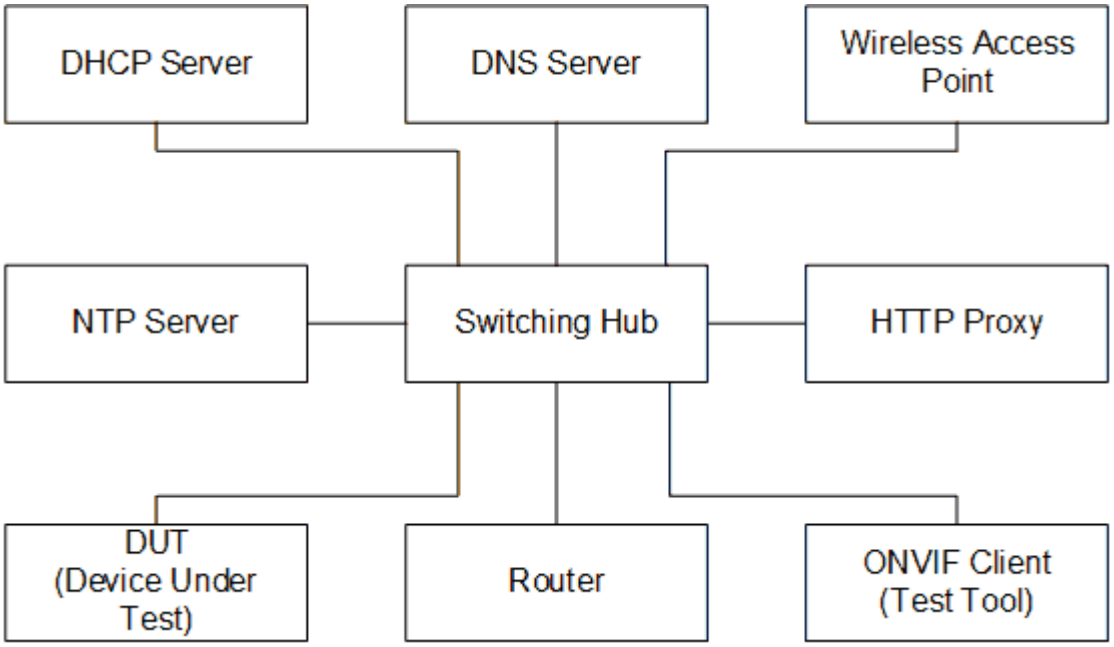
4.1 Test Setup

4.1.1 Network Configuration for DUT

The generic test configuration for the execution of test cases defined in this document is as shown below (Figure 1).

Based on the individual test case requirements, some of the entities in the below setup may not be needed for the execution of those corresponding test cases.

Figure 4.1. Test Configuration for DUT



DUT: ONVIF device to be tested. Hereafter, this is referred to as DUT (Device Under Test).

ONVIF Client (Test Tool): Tests are executed by this system and it controls the behavior of the DUT. It handles both expected and unexpected behavior.

HTTP Proxy: provides facilitation in case of RTP and RTSP tunneling over HTTP.

Wireless Access Point: provides wireless connectivity to the devices that support wireless connection.

DNS Server: provides DNS related information to the connected devices.

DHCP Server: provides IPv4 Address to the connected devices.

NTP Server: provides time synchronization between ONVIF Client and DUT.

Switching Hub: provides network connectivity among all the test equipments in the test environment. All devices should be connected to the Switching Hub. When running multiple test instances in parallel on the same network, the Switching Hub should be configured to use filtering in order to avoid multicast traffic being flooded to all ports, because this may affect test stability.

Router: provides router advertisements for IPv6 configuration.

4.2 Prerequisites

The pre-requisites for executing the test cases described in this Test Specification are:

1. The DUT shall be configured with an IPv4 address.
2. The DUT shall be IP reachable [in the test configuration].
3. The DUT shall be able to be discovered by the Test Tool.
4. The DUT shall be configured with the time i.e. manual configuration of UTC time and if NTP is supported by DUT, then NTP time shall be synchronized with NTP Server.
5. The DUT time and Test tool time shall be synchronized with each other either manually or by common NTP server

4.3 Test Policy

This section describes the test policies specific to the test case execution of each functional block.

The DUT shall adhere to the test policies defined in this section.

4.3.1 Media Configuration

Prior to the execution of Media Configuration test cases, DUT shall be discovered by ONVIF Client using device management service, and it shall demonstrate media capabilities to ONVIF Client using GetServiceCapabilities command.

DUT shall support at least one media profile with Video Configuration. Video Configuration shall include video source and video encoder media entities.

DUT shall support either H.264 or H.265 encoding.

ONVIF Client shall explicitly specify the optional media formats supported by DUT.

ONVIF Client shall explicitly specify if the DUT supports Audio and PTZ.

DUT shall allow creation of at least one media profile by ONVIF Client. In certain test cases, ONVIF Client may create new media configuration (i.e. media profile and media entities). In such cases, the test procedure will delete those modified configurations at the end of the test procedure.

DUT should respond with proper response message for all SOAP actions. Sending fault messages such as "ter:ConfigurationConflict" will be treated as FAILURE of the test cases.

Please refer to Section 5 for Media Configuration Test Cases.

5 Media Configuration Test Cases

5.1 Media Profile

5.1.1 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING

Test Case ID: MEDIA2-1-1-1

Specification Coverage: Video Streaming (Profile T Specification)

Feature Under Test: GetProfiles

WSDL Reference: media2.wsdl, deviceio.wsdl

Test Purpose: To verify that DUT has a ready-to-use Media Service 2.0 Profile for streaming video (either H.264 or H.265) per video source.

Pre-Requisite: Media2 Service is received from the DUT, DeviceIO Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSources** request.
4. The DUT responds with **GetVideoSourcesResponse** message with parameters
   - Token list =: *videoSourceTokenList*
5. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
6. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
7. For each Video Source token *videoSourceToken* in *videoSourceTokenList* repeat the following steps:

7.1. If *profileList* does not contain at least one Media Profile with Configurations.VideoSource.SourceToken value is equal to *videoSourceToken* and with Configurations.VideoEncoder, which Configurations.VideoEncoder.Encoding equals to "H264" or "H265", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourcesResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.1.2 CREATE MEDIA PROFILE WITH PRE-DEFINED CONFIGURATION

Test Case ID: MEDIA2-1-1-2

Specification Coverage: Get media profiles, Create media profile, Delete media profile.

Feature Under Test: GetProfiles, CreateProfile, DeleteProfile

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT can create media profile with populated configuration parameter.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. Subscribtion ONVIF Client deletes Media Profile if Maximum Number of Media Profiles is reached by following the procedure mentioned in Annex A.1.
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfCompleteList* - Video Source Configurations list

5. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ProfileChanged"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
6. ONVIF Client invokes **CreateProfile** request with parameters
   - Name := "testMedia2"
   - Configuration[0].Type := VideoSource
   - Configuration[0].Token = *videoSourceConfCompleteList*[0].@token
7. The DUT responds with **CreateProfileResponse** with parameters
   - Token =: *profileToken*
8. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters
   - in *s* - Subscription reference
   - in *currentTime* - current time for the DUT
   - in *terminationTime* - subscription termination time
   - in *profileToken* - Media Profile token
9. ONVIF Client invokes **GetProfiles** request with parameters
   - Token := *profileToken*
   - Type[0] := VideoSource
10. The DUT responds with **GetProfilesResponse** message with parameters
    - Profiles list =: *profileList*
11. If *profileList* is empty, FAIL the test and skip other steps.

12. If *profileList* contains more than one item, FAIL the test and skip other steps.
13. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.
14. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfCompleteList*[0].@token, FAIL the test and skip other steps.
15. ONVIF Client invokes **DeleteProfile** request with parameters
    - Token := *profileToken*
16. The DUT responds with **DeleteProfileResponse** message.
17. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters
    - in *s* - Subscription reference
    - in *currentTime* - current time for the DUT
    - in *terminationTime* - subscription termination time
    - in *profileToken* - Media Profile token
18. If subscription was created at step 5, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *s* - Subscription reference
19. ONVIF Client invokes **GetProfiles** request with parameters
    - Token := *profileToken*
    - Type skipped
20. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoProfile** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.

- DUT did not send **DeleteProfileResponse** message.
- DUT did not send **CreateProfileResponse** message.
- The DUT did not send the **env:Sender/ter:Action/ter:NoConfig** SOAP 1.2 fault message.

Note: *timeout1* will be taken from Operation Delay field of ONVIF Device Test Tool.

Note: See Annex in [ONVIF Base Test] for Invalid SOAP 1.2 fault message definition.

Note: See Annex A.7 for Name and Token Parameters Length limitations.

5.1.3 DYNAMIC MEDIA PROFILE CONFIGURATION

Test Case ID: MEDIA2-1-1-3

Specification Coverage: Get media profiles, Create media profile, Delete media profile, Add one or more configurations to a profile, Remove one or more configurations from a profile, Get configurations.

Feature Under Test: GetProfiles, CreateProfile, DeleteProfile, AddConfiguration, RemoveConfiguration, GetVideoEncoderConfigurations, GetVideoSourceConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify the behavior of the DUT for dynamic media profile configuration.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client creates new Media Profile or removes all configurations from fixed Media Profile by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *newProfileFlag* - flag that indicates that new Media Profile was created
   - out *profileToken* - empty Media Profile
4. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters

- in **"tns1:Media/ProfileChanged"** - Notification Topic
- out *s* - Subscription Reference
- out *currentTime* - current time for the DUT
- out *terminationTime* - Subscription Termination time

5. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
6. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:
   6.1. ONVIF Client invokes **AddConfiguration** request with parameters
      - ProfileToken = *profileToken*
      - Name skipped
      - Configuration[0].Type = VideoSource
      - Configuration[0].Token = *videoSourceConfiguration*.@token
   6.2. The DUT responds with **AddConfigurationResponse** message.
   6.3. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters
      - in *s* - Subscription reference
      - in *currentTime* - current time for the DUT
      - in *terminationTime* - subscription termination time
      - in *profileToken* - Media Profile token
   6.4. ONVIF Client invokes **GetProfiles** request with parameters
      - Token := *profileToken*
      - Type[0] := VideoSource

6.5. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.6. If *profileList* is empty, FAIL the test and skip other steps.

6.7. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.8. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

6.9. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.10. If DUT supports Video

6.10.1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profileToken*

6.10.2. The DUT responds with compatible video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters

- Configurations list =: *videoEncoderConfigurationList*

6.10.3. If *videoEncoderConfigurationList*.Configurations is skipped or empty, FAIL the test and skip other steps.

6.10.4. Set *videoEncoderConfiguration* := *videoEncoderConfigurationList*.Configurations[0].

6.10.5. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profileToken*
- Name skipped
- Configuration[0].Type := VideoEncoder
- Configuration[0].Token := *videoEncoderConfiguration*.@token

6.10.6. The DUT responds with **AddConfigurationResponse** message.

6.10.7. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged**

event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.10.8. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := VideoSource
- Type[1] := VideoEncoder

6.10.9. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.10.10. If *profileList* is empty, FAIL the test and skip other steps.

6.10.11. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.10.12. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

6.10.13. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.10.14. If *profileList*[0].Configurations.VideoEncoder.@token != *videoEncoderConfiguration*.@token, FAIL the test and skip other steps.

6.10.15. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profileToken*
- Configuration[0].Type := VideoEncoder
- Configuration[0].Token skipped

6.10.16. The DUT responds with **RemoveConfigurationResponse** message.

6.10.17. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged**

event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.10.18. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := VideoSource
- Type[1] := VideoEncoder

6.10.19. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.10.20. If *profileList* is empty, FAIL the test and skip other steps.

6.10.21. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.10.22. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.10.23. If *profileList*[0].Configurations contains VideoEncoder, FAIL the test and skip other steps.

6.11. If the DUT supports Analytics:

6.11.1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profileToken*

6.11.2. The DUT responds with compatible analytics configurations in **GetAnalyticsConfigurationsResponse** with parameters

- Configurations list =: *analyticsConfigurationList*

6.11.3. If *analyticsConfigurationList*.Configurations is skipped or empty, go to the step 6.13 .

6.11.4. Set *analyticsConfiguration* := *analyticsConfigurationList*.Configurations[0].

6.11.5. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profileToken*
- Name skipped
- Configuration[0].Type := Analytics
- Configuration[0].Token := *analyticsConfiguration*.@token

6.11.6. The DUT responds with **AddConfigurationResponse** message.

6.11.7. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.11.8. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := "VideoSource"
- Type[1] := "Analytics"

6.11.9. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.11.10. If *profileList* is empty, FAIL the test and skip other steps.

6.11.11. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.11.12. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

6.11.13. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.11.14.If *profileList*[0].Configurations.Analytics.@token != *analyticsConfiguration*.@token, FAIL the test and skip other steps.

6.11.15.ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profileToken*
- Configuration[0].Type := Analytics
- Configuration[0].Token skipped

6.11.16.The DUT responds with **RemoveConfigurationResponse** message.

6.11.17.If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.11.18.ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := VideoSource
- Type[1] := Analytics

6.11.19.The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.11.20.If *profileList* is empty, FAIL the test and skip other steps.

6.11.21.If *profileList* contains more than one item, FAIL the test and skip other steps.

6.11.22.If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.11.23.If *profileList*[0].Configurations contains Analytics, FAIL the test and skip other steps.

6.12. If the DUT supports Metadata:

6.12.1. ONVIF Client invokes **GetMetadataConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profileToken*

6.12.2. The DUT responds with compatible metadata configurations in **GetMetadataConfigurationsResponse** with parameters

- Configurations list =: *metadataConfigurationList*

6.12.3. If *metadataConfigurationList*.Configurations is skipped or empty, go to the step 6.13 .

6.12.4. Set *metadataConfiguration* := *metadataConfigurationList*.Configurations[0].

6.12.5. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken = *profileToken*
- Name skipped
- Configuration[0].Type = Metadata
- Configuration[0].Token = *metadataConfiguration*.@token

6.12.6. The DUT responds with **AddConfigurationResponse** message.

6.12.7. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.12.8. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*

- Type[0] := VideoSource
- Type[1] := Metadata

6.12.9. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.12.10. If *profileList* is empty, FAIL the test and skip other steps.

6.12.11. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.12.12. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

6.12.13. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.12.14. If *profileList*[0].Configurations.Metadata.@token != *metadataConfiguration*.@token, FAIL the test and skip other steps.

6.12.15. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken = *profileToken*
- Configuration[0].Type = Metadata
- Configuration[0].Token skipped

6.12.16. The DUT responds with **RemoveConfigurationResponse** message.

6.12.17. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.12.18. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*

- Type[0] := VideoSource
- Type[1] := Metadata

6.12.19. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.12.20. If *profileList* is empty, FAIL the test and skip other steps.

6.12.21. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.12.22. If *profileList*[0].ConfigurationsVideoSource.@token != *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

6.12.23. If *profileList*[0].Configurations contains Metadata, FAIL the test and skip other steps.

6.13. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken = *profileToken*
- Configuration[0].Type = VideoSource
- Configuration[0].Token = *videoSourceConfiguration*.@token

6.14. The DUT responds with **RemoveConfigurationResponse** message.

6.15. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

6.16. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := VideoSource

6.17. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

6.18. If *profileList* is empty, FAIL the test and skip other steps.

6.19. If *profileList* contains more than one item, FAIL the test and skip other steps.

6.20. If *profileList*[0].Configurations contains VideoSource, FAIL the test and skip other steps.

7. If *analyticsConfigurationList*.Configurations was skipped or empty for each *videoSourceConfiguration* at step 6.11.3, FAIL the test, restore DUT settings and skip other steps.

8. If the DUT supports Audio:

8.1. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters

- out *audioSourceConfList* - Audio Source Configurations list

8.2. For each Audio Source Configuration *audioSourceConfiguration* in *audioSourceConfList* repeat the following steps:

8.2.1. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken = *profileToken*
- Name skipped
- Configuration[0].Type = AudioSource
- Configuration[0].Token = *audioSourceConfiguration*.@token

8.2.2. The DUT responds with **AddConfigurationResponse** message.

8.2.3. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time

- in *profileToken* - Media Profile token

8.2.4. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioSource

8.2.5. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

8.2.6. If *profileList* is empty, FAIL the test and skip other steps.

8.2.7. If *profileList* contains more than one item, FAIL the test and skip other steps.

8.2.8. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

8.2.9. If *profileList*[0].Configurations.AudioSource.@token != *audioSourceConfiguration*.@token, FAIL the test and skip other steps.

8.2.10. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profileToken*

8.2.11. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters

- Configurations list =: *audioEncoderConfigurationList*

8.2.12. If *audioEncoderConfigurationList*.Configurations is skipped or empty, FAIL the test and skip other steps.

8.2.13. Set *audioEncoderConfiguration* := *audioEncoderConfigurationList*.Configurations[0].

8.2.14. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profileToken*
- Name skipped
- Configuration[0].Type := AudioEncoder

- Configuration[0].Token := *audioEncoderConfiguration*.@token

8.2.15. The DUT responds with **AddConfigurationResponse** message.

8.2.16. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

8.2.17. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioSource
- Type[0] := AudioEncoder

8.2.18. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

8.2.19. If *profileList* is empty, FAIL the test and skip other steps.

8.2.20. If *profileList* contains more than one item, FAIL the test and skip other steps.

8.2.21. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

8.2.22. If *profileList*[0].Configurations.AudioSource.@token != *audioSourceConfiguration*.@token, FAIL the test and skip other steps.

8.2.23. If *profileList*[0].Configurations.AudioEncoder.@token != *audioEncoderConfiguration*.@token, FAIL the test and skip other steps.

8.2.24. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profileToken*
- Configuration[0].Type := AudioEncoder

- Configuration[0].Token skipped

8.2.25. The DUT responds with **RemoveConfigurationResponse** message.

8.2.26. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

8.2.27. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioSource
- Type[1] := AudioEncoder

8.2.28. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

8.2.29. If *profileList* is empty, FAIL the test and skip other steps.

8.2.30. If *profileList* contains more than one item, FAIL the test and skip other steps.

8.2.31. If *profileList*[0].Configurations.AudioSource.@token != *audioSourceConfiguration*.@token, FAIL the test and skip other steps.

8.2.32. If *profileList*[0].Configurations contains AudioEncoder, FAIL the test and skip other steps.

8.2.33. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken = *profileToken*
- Configuration[0].Type = AudioSource
- Configuration[0].Token skipped

8.2.34. The DUT responds with **RemoveConfigurationResponse** message.

8.2.35. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

8.2.36. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioSource

8.2.37. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

8.2.38. If *profileList* is empty, FAIL the test and skip other steps.

8.2.39. If *profileList* contains more than one item, FAIL the test and skip other steps.

8.2.40. If *profileList*[0].Configurations contains AudioSource, FAIL the test and skip other steps.

9. If the DUT supports Audio Output:

9.1. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters

- out *audioOutputConfList* - Audio Output Configurations list

9.2. For each Audio Output Configuration *audioOutputConfiguration* in *audioOutputConfList* repeat the following steps:

9.2.1. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken = *profileToken*
- Name skipped

- Configuration[0].Type = AudioOutput
- Configuration[0].Token = *audioOutputConfiguration*.@token

9.2.2. The DUT responds with **AddConfigurationResponse** message.

9.2.3. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

9.2.4. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioOutput

9.2.5. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

9.2.6. If *profileList* is empty, FAIL the test and skip other steps.

9.2.7. If *profileList* contains more than one item, FAIL the test and skip other steps.

9.2.8. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

9.2.9. If *profileList*[0].Configurations.AudioOutput.@token != *audioOutputConfiguration*.@token, FAIL the test and skip other steps.

9.2.10. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profileToken*

9.2.11. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters

- Configurations list =: *audioDecoderConfigurationList*

9.2.12. If *audioDecoderConfigurationList*.Configurations is skipped or empty, FAIL the test and skip other steps.

9.2.13. Set *audioDecoderConfiguration* := *audioDecoderConfigurationList*.Configurations[0].

9.2.14. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profileToken*
- Name skipped
- Configuration[0].Type := AudioDecoder
- Configuration[0].Token := *audioDecoderConfiguration*.@token

9.2.15. The DUT responds with **AddConfigurationResponse** message.

9.2.16. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

9.2.17. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioOutput
- Type[1] := AudioDecoder

9.2.18. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

9.2.19. If *profileList* is empty, FAIL the test and skip other steps.

9.2.20. If *profileList* contains more than one item, FAIL the test and skip other steps.

9.2.21. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

9.2.22. If *profileList*[0].Configurations.AudioOutput.@token != *audioOutputConfiguration*.@token, FAIL the test and skip other steps.

9.2.23. If *profileList*[0].Configurations.AudioDecoder.@token != *audioDecoderConfiguration*.@token, FAIL the test and skip other steps.

9.2.24. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profileToken*
- Configuration[0].Type := AudioDecoder
- Configuration[0].Token skipped

9.2.25. The DUT responds with **RemoveConfigurationResponse** message.

9.2.26. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

9.2.27. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioOutput
- Type[1] := AudioDecoder

9.2.28. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

9.2.29. If *profileList* is empty, FAIL the test and skip other steps.

9.2.30. If *profileList* contains more than one item, FAIL the test and skip other steps.

9.2.31. If *profileList*[0].Configurations.AudioOutput.@token != *audioOutputConfiguration*.@token, FAIL the test and skip other steps.

9.2.32. If *profileList*[0].Configurations contains AudioDecoder, FAIL the test and skip other steps.

9.2.33. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken = *profileToken*
- Configuration[0].Type = AudioOutput
- Configuration[0].Token skipped

9.2.34. The DUT responds with **RemoveConfigurationResponse** message.

9.2.35. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

9.2.36. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := AudioOutput

9.2.37. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

9.2.38. If *profileList* is empty, FAIL the test and skip other steps.

9.2.39. If *profileList* contains more than one item, FAIL the test and skip other steps.

9.2.40. If *profileList*[0].Configurations contains AudioOutput, FAIL the test and skip other steps.

10. If *newProfileFlag* = true, do the following steps:

    10.1. ONVIF Client invokes **DeleteProfile** request with parameters

    - Token := *profileToken*

    10.2. The DUT responds with **DeleteProfileResponse** message.

    10.3. ONVIF Client invokes **GetProfiles** request with parameters

    - Token := *profileToken*
    - Type skipped

    10.4. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoProfile** SOAP 1.2 fault.

11. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

    - in *s* - Subscription reference

12. ONVIF Client restores DUT configuration if requered.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **DeleteProfileResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT did not send **GetAnalyticsConfigurationsResponse** message.
- DUT did not send **GetAudioEncoderConfigurationsResponse** message.
- DUT did not send **GetAudioDecoderConfigurationsResponse** message.
- DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message.

Note: *timeout1* will be taken from Operation Delay field of ONVIF Device Test Tool.

Note: See Annex in [ONVIF Base Test] for Invalid SOAP 1.2 fault message definition.

Note: See Annex A.7 for Name and Token Parameters Length limitations.

5.1.4 GET PROFILES

Test Case ID: MEDIA2-1-1-4

Specification Coverage: GetProfiles (Media2 Service Specification)

Feature Under Test: GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify the behavior of the DUT for GetProfiles with different Type parameters.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] =: All
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList1*
5. If *profileList1* contains at least two items with the same @token, FAIL the test and skip other steps.
6. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type skipped
7. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList2*

8. If *profileList2* contains at least two items with the same @token, FAIL the test and skip other steps.
9. If amount of profiles in *profileList2* is not equal to amount of profiles in *profileList1*, FAIL the test and skip other steps.
10. For each Profile *profile* in *profileList2* repeat the following steps:
    10.1. If *profileList1* does not contain profile with token equals to *profile*.token, FAIL the test and skip other steps.
    10.2. If *profile* contains not empty Configurations element, FAIL the test and skip other steps.
11. ONVIF Client invokes **GetProfiles** request with parameters
    - Token skipped
    - Type[0] =: VideoSource
12. The DUT responds with **GetProfilesResponse** message with parameters
    - Profiles list =: *profileList3*
13. If *profileList3* contains at least two items with the same @token, FAIL the test and skip other steps.
14. If amount of profiles in *profileList3* is not equal to amount of profiles in *profileList1*, FAIL the test and skip other steps.
15. For each Profile *profile* in *profileList3* repeat the following steps:
    15.1. If *profile* contains at least one configuration in Configurations element differs from VideoSource, FAIL the test and skip other steps.
    15.2. If amount of VideoSource elements in *profile* is not equal to amount of VideoSource elements in *profileList1*[0], where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.
    15.3. If *profile*.Configurations.VideoSource element is not equal to *profileList1*[0].Configurations.VideoSource element, where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.
16. If Analytics feature is supported:
    16.1. ONVIF Client invokes **GetProfiles** request with parameters
        - Token skipped

- Type[0] =: Analytics

16.2. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList4*

16.3. If *profileList4* contains at least two items with the same @token, FAIL the test and skip other steps.

16.4. If amount of profiles in *profileList4* is not equal to amount of profiles in *profileList1*, FAIL the test and skip other steps.

16.5. For each Profile *profile* in *profileList4* repeat the following steps:

16.5.1 If *profile* contains at least one configuration in Configurations element differs from Analytics, FAIL the test and skip other steps.

16.5.2 If amount of Analytics elements in *profile* is not equal to amount of Analytics elements in *profileList1*[0], where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.

16.5.3 If *profile*.Configurations.Analytics element is not equal to *profileList1*[0].Configurations.Analytics element, where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.

17. If Metadata feature is supported:

17.1. ONVIF Client invokes **GetProfiles** request with parameters

- Token skipped
- Type[0] =: Metadata

17.2. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList5*

17.3. If *profileList5* contains at least two items with the same @token, FAIL the test and skip other steps.

17.4. If amount of profiles in *profileList5* is not equal to amount of profiles in *profileList1*, FAIL the test and skip other steps.

17.5. For each Profile *profile* in *profileList5* repeat the following steps:

17.5.1 If *profile* contains at least one configuration in Configurations element differs from Metadata, FAIL the test and skip other steps.

17.5.2 If amount of Metadata elements in *profile* is not equal to amount of Metadata elements in *profileList1*[0], where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.

17.5.3 If *profile*.Configurations.Metadata element is not equal to *profileList1*[0].Configurations.Metadata element, where *profileList1*[0] is profile with token equals to *profile*.token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.

Note: The following fields are compared at step 15.3:

- SourceToken
- Name
- Bounds.x
- Bounds.y
- Bounds.width
- Bounds.height
- Extension.Rotate
- Extension.Rotate.Mode
- Extension.Rotate.Degree
- Extension.Extension.LensDescription list (XFactor will be used as a key)
- Extension.Extension.LensDescription.FocalLength
- Extension.Extension.LensDescription.Offset.x

- Extension.Extension.LensDescription.Offset.y
- Extension.Extension.LensDescription.Projection list (Angle and Radius will be used as key)
- Extension.Extension.LensDescription.Projection.Transmittance

Note: The following fields are compared at step 16.5.3 [56]:

- Name
- AnalyticsEngineConfiguration.AnalyticsModule list (Type and Name will be used as key. Parameters item will not be compared).
- AnalyticsEngineConfiguration.RuleEngineConfiguration list (Type and Name will be used as key. Parameters item will not be compared).

Note: The following fields are compared at step 16.5.3 [57]:

- Name
- CompressionType
- GeoLocation
- PTZStatus.Status
- PTZStatus.Position
- Events.Filter (only field presence will be compared)
- Events.SubscriptionPolicy (only field presence will be compared)
- Analytics
- AnalyticsEngineConfiguration.AnalyticsModule list (Type and Name will be used as key. Parameters item will not be compared).

5.1.5 CREATE MEDIA PROFILE WITH CONFIGURATIONS

Test Case ID: MEDIA2-1-1-5

Specification Coverage: Get media profiles, Create media profile.

Feature Under Test: GetProfiles, CreateProfile

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT can create media profile with video source configuration parameter, audio source configuration parameter and audio output configuration parameter.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. Subscribtion ONVIF Client deletes Media Profile if Maximum Number of Media Profiles is reached by following the procedure mentioned in Annex A.1.
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfCompleteList* - Video Source Configurations list
5. ONVIF Client invokes **CreateProfile** request with parameters
   - Name := "testMedia"
   - Configuration[0].Type := VideoSource
   - Configuration[0].Token = *videoSourceConfCompleteList*[0].@token
6. The DUT responds with **CreateProfileResponse** with parameters
   - Token =: *profileToken*
7. ONVIF Client invokes **GetProfiles** request with parameters
   - Token := *profileToken*
   - Type[0] := All
8. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
9. If *profileList* is empty, FAIL the test and skip other steps.
10. If *profileList* contains more than one item, FAIL the test and skip other steps.
11. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

12. If *profileList*[0].Configurations.VideoSource.@token != *videoSourceConfCompleteList*[0].@token, FAIL the test and skip other steps.

13. ONVIF Client deletes created profile by following the procedure mentioned in Annex A.11 with the following input and output parameters

    - in *profileToken* - Media profile token

14. If Media2 Audio is supported

    14.1. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters

    - out *audioSourceConfCompleteList* - Audio Source Configurations list

    14.2. ONVIF Client invokes **CreateProfile** request with parameters

    - Name := "testMedia"
    - Configuration[1].Type := AudioSource
    - Configuration[1].Token = *audioSourceConfCompleteList*[0].@token

    14.3. The DUT responds with **CreateProfileResponse** with parameters

    - Token =: *profileToken*

    14.4. ONVIF Client invokes **GetProfiles** request with parameters

    - Token := *profileToken*
    - Type[0] := All

    14.5. The DUT responds with **GetProfilesResponse** message with parameters

    - Profiles list =: *profileList*

    14.6. If *profileList* is empty, FAIL the test and skip other steps.

    14.7. If *profileList* contains more than one item, FAIL the test and skip other steps.

    14.8. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

    14.9. If *profileList*[0].Configurations.AudioSource.@token != *audioSourceConfCompleteList*[0].@token, FAIL the test and skip other steps.

    14.10. ONVIF Client deletes created profile by following the procedure mentioned in Annex A.11 with the following input and output parameters

- in *profileToken* - Media profile token

15. If Media2 Audio Outputs is supported

15.1. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters

- out *audioOutputConfCompleteList* - Audio Output Configurations list

15.2. ONVIF Client invokes **CreateProfile** request with parameters

- Name := "testMedia"
- Configuration[0].Type := AudioOutput
- Configuration[0].Token = *audioOutputConfCompleteList*[0].@token

15.3. The DUT responds with **CreateProfileResponse** with parameters

- Token =: *profileToken*

15.4. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := All

15.5. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

15.6. If *profileList* is empty, FAIL the test and skip other steps.

15.7. If *profileList* contains more than one item, FAIL the test and skip other steps.

15.8. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

15.9. If *profileList*[0].Configurations.AudioOutput.@token != *audioOutputConfCompleteList*[0].@token, FAIL the test and skip other steps.

15.10. ONVIF Client deletes created profile by following the procedure mentioned in Annex A.11 with the following input and output parameters

- in *profileToken* - Media profile token

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **CreateProfileResponse** message.

Note: See Annex A.7 for Name and Token Parameters Length limitations.

5.1.6 REMOVE ALL CONFIGURATIONS FROM MEDIA PROFILE

Test Case ID: MEDIA2-1-1-6

Specification Coverage: Remove one or more configurations from a profile (Media 2 Service Specification)

Feature Under Test: RemoveConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify the removal of all configurations from Media Profile with RemoveConfiguration command and Type = All.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList* list repeat the following steps:

5.1. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profile*.@token
- Configuration[0].Type := All
- Configuration[0].Token skipped

5.2. The DUT responds with **RemoveConfigurationResponse** message.

5.3. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profile*.@token
- Type[0] := All

5.4. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList1*

5.5. If *profileList1*[0].Configurations is not empty, FAIL the test and and skip other steps.

5.6. ONVIF Client restores configurations of Media Profile with @token = *profile*.@token.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message(s).
- DUT did not send **RemoveConfigurationResponse** message(s).

5.1.7 FIXED MEDIA PROFILE CONFIGURATION

Test Case ID: MEDIA2-1-1-7

Specification Coverage: Media profiles (Media 2 Service Specification)

Feature Under Test: RemoveConfiguration, AddConfiguration, GetProfiles.

WSDL Reference: media2.wsdl

Test Purpose: To verify the removal of all configurations from fixed Media Profile. To verify adding of configurations to fixed Media Profile.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT.

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList* list with *@fixed* = true repeat the following steps:
   5.1. ONVIF Client invokes **RemoveConfiguration** request with parameters
      - ProfileToken := *profile*.@token
      - Configuration[0].Type := All
      - Configuration[0].Token skipped
   5.2. The DUT responds with **RemoveConfigurationResponse** message.
   5.3. ONVIF Client invokes **GetProfiles** request with parameters
      - Token := *profile*.@token
      - Type[0] := All
   5.4. The DUT responds with **GetProfilesResponse** message with parameters
      - Profiles list =: *profileList1*
   5.5. If *profileList1*[0].Configurations is not empty, FAIL the test and and skip other steps.
   5.6. ONVIF Client invokes **AddConfiguration** request to restore configurations of Media Profile *profile* with parameters
      - ProfileToken := *profile*.@token
      - For each configuration item *configuration* in *profile*.Configurations list:

- Configuration[0].Type := *configuration*
- Configuration[0].Token := *configuration*.@token

5.7. The DUT responds with **AddConfigurationResponse** message.

5.8. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profile*.@token
- Type[0] := All

5.9. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList2*

5.10. If list of configurations in *profileList2*[0].Configurations is not equal to list of configurations in *profileList*[0].Configurations, were *profileList*[0] is profile with token = *profile*.@token, FAIL the test and and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message(s).
- DUT did not send **RemoveConfigurationResponse** message(s).
- DUT did not send **AddConfigurationResponse** message(s).

5.1.8 READY TO USE MEDIA PROFILE FOR METADATA STREAMING

Test Case ID: MEDIA2-1-1-8

Specification Coverage: Metadata streaming (Profile M Specification)

Feature Under Test: GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT has a ready-to-use Media Service 2.0 Profile for streaming metadata.

Pre-Requisite: Media2 Service is received from the DUT. Profile M scope. Metadata feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
6. For each video source configuration *videoSourceConf* in *videoSourceConfList*

   6.1. If *profileList* does not contain at least one Media Profile with Configurations.VideoSource.token = *videoSourceConf*.token and with Configurations.Metadata, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.

5.1.9 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING (PROFILE M)

Test Case ID: MEDIA2-1-1-9

Specification Coverage: Video streaming (Profile M Specification)

Feature Under Test: GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT has a ready-to-use Media Service 2.0 Profile for streaming video.

Pre-Requisite: Media2 Service is received from the DUT. Profile M scope. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
6. For each video source configuration *videoSourceConf* in *videoSourceConfList*

   6.1. If *profileList* does not contain at least one Media Profile with Configurations.VideoSource.token = *videoSourceConf*.token and with Configurations.VideoEncoder, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.

5.2 Video Configuration

5.2.1 Video Source Configuration

5.2.1.1 GET VIDEO SOURCE CONFIGURATION OPTIONS

Test Case ID: MEDIA2-2-2-1

Specification Coverage: Get configuration *options*, Video source configuration.

Feature Under Test: GetVideoSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving Video Source Configuration *options* for specified Video Source Configuration, for specified Profile, generic for the Device.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Media2 Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Media2 Service Capabilities
4. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
5. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters
   - Options =: *options*
6. If *options*.BoundsRange.XRange.Min > *options*.BoundsRange.XRange.Max, FAIL the test and skip other steps.

7. If *options*.BoundsRange.YRange.Min > *options*.BoundsRange.YRange.Max, FAIL the test and skip other steps.

8. If *options*.BoundsRange.WidthRange.Min > *options*.BoundsRange.WidthRange.Max, FAIL the test and skip other steps.

9. If *options*.BoundsRange.HeightRange.Min > *options*.BoundsRange.HeightRange.Max, FAIL the test and skip other steps.

10. If *cap*.@Rotation = true:

    10.1. If *options*.Extension.Rotate is skipped, FAIL the test and skip other steps.

11. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters

    - out *videoSourceConfList* - Video Source Configurations list

12. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters

    - ConfigurationToken := *videoSourceConfList*[0].@token
    - ProfileToken skipped

13. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters

    - Options =: *options*

14. ONVIF Client retrieves Media Profile, which contains Video Source Configuration by following the procedure mentioned in Annex A.12 with the following input and output parameters

    - out *profile* - Media Profile with Video Source Configuration

15. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters

    - ConfigurationToken skipped
    - ProfileToken := *profile*.@token

16. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters

    - Options =: *options*

17. If Media Profile *profile* was changed at step 14, ONVIF Client restores Media Profile.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceConfigurationOptionsResponse** message.

5.2.1.2 GET VIDEO SOURCE CONFIGURATIONS

Test Case ID: MEDIA2-2-2-2

Specification Coverage: Get configurations, Video source configuration.

Feature Under Test: GetVideoSourceConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Video Source Configuration List, Video Source Configuration by Configuration token and compatible Video Source Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
4. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
   - Configurations list =: *videoSourceConfCompleteList*
5. If *videoSourceConfCompleteList* is empty, FAIL the test and skip other steps.
6. If *videoSourceConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.

7. For each *videoSourceConfiguration* in *videoSourceConfCompleteList* repeat the following steps:

   7.1. If *videoSourceConfiguration* contains ViewMode

      7.1.1. If *videoSourceConfiguration*.ViewMode value is not equal to one of value defined in tt:ViewModes list provided in the procedure Annex A.13, FAIL the test and skip other steps.

      7.1.2. If *videoSourceConfiguration*.ViewMode = "tt:Fisheye"

         7.1.2.1. If *videoSourceConfiguration*.Extension does not contain at least one LensDescription element, PASS this step with a WARNING.

   7.2. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

   - ConfigurationToken := *videoSourceConfiguration*.@token
   - ProfileToken skipped

   7.3. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

   - Configurations list =: *videoSourceConfList*

   7.4. If *videoSourceConfList* is empty, FAIL the test and skip other steps.

   7.5. If *videoSourceConfList* contains more than one item, FAIL the test and skip other steps.

   7.6. If *videoSourceConfList* does not contain item with @token = *videoSourceConfiguration*.@token, FAIL the test and skip other steps.

8. ONVIF Client invokes **GetProfiles** request with parameters

   - Token skipped
   - Type[0] := VideoSource

9. The DUT responds with **GetProfilesResponse** message with parameters

   - Profiles list =: *profileList*

10. For each Media Profile *profile* in *profileList* repeat the following steps:

    10.1. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

    - ConfigurationToken skipped
    - ProfileToken := *profile*.@token

10.2. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

- Configurations list =: *videoSourceConfList*

10.3. If *videoSourceConfList* contains at least two items with the same @token, FAIL the test and skip other steps.

10.4. If *videoSourceConfCompleteList* does not contain at least one item with @token from *videoSourceConfList*, FAIL the test and skip other steps.

10.5. If *profile*.Configurations contains VideoSource:

10.5.1. If *videoSourceConfList* does not contain item with @token = *profile*.Configurations.VideoSource.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.2.1.3 VIDEO SOURCE CONFIGURATIONS AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-2-2-3

Specification Coverage: Get configurations, Get configuration *options*, Video source configuration.

Feature Under Test: GetProfiles, GetVideoSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Video Source Configurations are consistent with Video Source Configuration Options.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Media2 Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Media2 Service Capabilities
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
5. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters
      - ConfigurationToken := *videoSourceConfiguration*.@token
      - ProfileToken skipped
   5.2. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters
      - Options =: *options*
   5.3. If *videoSourceConfiguration*.SourceToken is not in *options*.VideoSourceTokensAvailable list, FAIL the test and skip other steps.
   5.4. If *videoSourceConfiguration*.Bounds.x < *options*.BoundsRange.XRange.Min, FAIL the test and skip other steps.
   5.5. If *videoSourceConfiguration*.Bounds.x > *options*.BoundsRange.XRange.Max, FAIL the test and skip other steps.
   5.6. If *videoSourceConfiguration*.Bounds.y < *options*.BoundsRange.YRange.Min, FAIL the test and skip other steps.
   5.7. If *videoSourceConfiguration*.Bounds.y > *options*.BoundsRange.YRange.Max, FAIL the test and skip other steps.

5.8. If *videoSourceConfiguration*.Bounds.width < *options*.BoundsRange.WidthRange.Min, FAIL the test and skip other steps.

5.9. If *videoSourceConfiguration*.Bounds.width > *options*.BoundsRange.WidthRange.Max, FAIL the test and skip other steps.

5.10. If *videoSourceConfiguration*.Bounds.height < *options*.BoundsRange.HeightRange.Min, FAIL the test and skip other steps.

5.11. If *videoSourceConfiguration*.Bounds.height > *options*.BoundsRange.HeightRange.Max, FAIL the test and skip other steps.

5.12. If *cap*.@Rotation = true:

5.12.1. If *options*.Extension.Rotate is skipped, FAIL the test and skip other steps.

5.12.2. If *videoSourceConfiguration*.Extension.Rotate is skipped, FAIL the test and skip other steps.

5.12.3. If *videoSourceConfiguration*.Extension.Rotate.Mode is not in the *options*.Extension.Rotate.Mode list, FAIL the test and skip other steps.

5.12.4. If *options*.Extension.Rotate.DegreeList specified and contains at least one Item:

5.12.4.1. If *videoSourceConfiguration*.Extension.Rotate.Degree is specified and not listed in *options*.Extension.Rotate.DegreeList.Item list, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceConfigurationOptionsResponse** message.

5.2.1.4 PROFILES AND VIDEO SOURCE CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-2-2-4

Specification Coverage: Get configurations, Get media profiles, Video source configuration.

Feature Under Test: GetVideoSourceConfigurations, GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Video Source Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := VideoSource
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList*, which contains Configurations.VideoSource repeat the following steps:
   5.1. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
      - ConfigurationToken := *profile*.Configurations.VideoSource.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
      - Configurations list =: *videoSourceConfList*
   5.3. If *videoSourceConfList*[0] is not equal to *profile*.Configurations.VideoSource, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoSourceConfigurationsResponse** message.

Note: The following fields are compared at step 5.3:

- SourceToken
- Name
- Bounds.x
- Bounds.y
- Bounds.width
- Bounds.height
- Extension.Rotate
- Extension.Rotate.Mode
- Extension.Rotate.Degree
- Extension.Extension.LensDescription list (XFactor will be used as a key)
- Extension.Extension.LensDescription.FocalLength
- Extension.Extension.LensDescription.Offset.x
- Extension.Extension.LensDescription.Offset.y
- Extension.Extension.LensDescription.Projection list (Angle and Radius will be used as key)
- Extension.Extension.LensDescription.Projection.Transmittance

5.2.1.5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS

Test Case ID: MEDIA2-2-2-5

Specification Coverage: Get configurations, Get configuration *options*, Video source configuration, Modify a configuration.

Feature Under Test: GetVideoSourceConfigurationOptions, GetVideoSourceConfigurations, SetVideoSourceConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify whether all supported Video Source Configuration Options can be set.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
5. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters
      - ConfigurationToken := *videoSourceConfiguration*.@token
      - ProfileToken skipped
   5.2. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters
      - Options =: *options*
   5.3. ONVIF Client invokes **SetVideoSourceConfiguration** request with parameters
      - Configuration.@token := *videoSourceConfiguration*.@token
      - Configuration.Name := "TestName1"

- Configuration.SourceToken := first value from *options*.VideoSourceTokensAvailable list
- Configuration.Bounds.x := *options*.BoundsRange.XRange.Min
- Configuration.Bounds.y := *options*.BoundsRange.YRange.Min
- Configuration.Bounds.width := *options*.BoundsRange.WidthRange.Min
- Configuration.Bounds.height := *options*.BoundsRange.HeightRange.Min
- If *options*.Extension.Rotate specified and options.Extension.Rotate.@Reboot != true:
  - Configuration.Extension.Rotate.Mode := first value from *options*.Extension.Rotate.Mode list
- If Configuration.Extension.Rotate.Mode = ON and options.Extension.Rotate.@Reboot != true:
  - If *options*.Extension.Rotate.DegreeList is specified and contains at least one Item:
    - Configuration.Extension.Rotate.Degree := first value from *options*.Extension.Rotate.DegreeList.Item list
  - If *options*.Extension.Rotate.DegreeList is not specified:
    - Configuration.Extension.Rotate.Degree := -180

5.4. DUT responds with **SetVideoSourceConfigurationResponse** message.

5.5. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *videoSourceConfiguration*.@token - Configuration token
- in VideoSource - Configuration Type

5.6. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

- ConfigurationToken := *videoSourceConfiguration*.@token
- ProfileToken skipped

5.7. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

- Configurations list =: *videoSourceConfList*

5.8. If *videoSourceConfList*[0] is not equal to Configuration from step 5.3, FAIL the test and skip other steps.

5.9. ONVIF Client invokes **SetVideoSourceConfiguration** request with parameters

- Configuration.@token := *videoSourceConfiguration*.@token
- Configuration.Name := "TestName2"
- Configuration.SourceToken := last value from *options*.VideoSourceTokensAvailable list
- If *options*.BoundsRange.XRange.Min = *options*.BoundsRange.XRange.Max and *options*.BoundsRange.YRange.Min = *options*.BoundsRange.YRange.Max and *options*.BoundsRange.WidthRange.Min = *options*.BoundsRange.WidthRange.Max and *options*.BoundsRange.HeightRange.Min = *options*.BoundsRange.HeightRange.Max:
  - Configuration.Bounds.x := *options*.BoundsRange.XRange.Min
  - Configuration.Bounds.y := *options*.BoundsRange.YRange.Min
  - Configuration.Bounds.width := *options*.BoundsRange.WidthRange.Min
  - Configuration.Bounds.height := *options*.BoundsRange.HeightRange.Min
- If *options*.BoundsRange.XRange.Min does not equal to *options*.BoundsRange.XRange.Max or *options*.BoundsRange.YRange.Min does not equal to *options*.BoundsRange.YRange.Max or *options*.BoundsRange.WidthRange.Min does not equal to *options*.BoundsRange.WidthRange.Max or *options*.BoundsRange.HeightRange.Min does not equal to *options*.BoundsRange.HeightRange.Max:

- Configuration.Bounds.x := (*options*.BoundsRange.XRange.Max + *options*.BoundsRange.XRange.Min - *options*.BoundsRange.WidthRange.Min) / 2
- Configuration.Bounds.y := (*options*.BoundsRange.YRange.Max + *options*.BoundsRange.YRange.Min - *options*.BoundsRange.HeightRange.Min) / 2
- Configuration.Bounds.width := min{*options*.BoundsRange.WidthRange.Max, *options*.BoundsRange.XRange.Max - Configuration.Bounds.x}
- Configuration.Bounds.height := min{*options*.BoundsRange.HeightRange.Max, *options*.BoundsRange.YRange.Max - Configuration.Bounds.y}

- If *options*.Extension.Rotate specified and options.Extension.Rotate.@Reboot != true:
  - Configuration.Extension.Rotate.Mode := last value from *options*.Extension.Rotate.Mode list
- If Configuration.Extension.Rotate.Mode = ON and options.Extension.Rotate.@Reboot != true:
  - If *options*.Extension.Rotate.DegreeList is specified and contains at least one Item:
    - Configuration.Extension.Rotate.Degree := last value from *options*.Extension.Rotate.DegreeList.Item list
  - If *options*.Extension.Rotate.DegreeList is not specified:
    - Configuration.Extension.Rotate.Degree := 180

5.10. DUT responds with **SetVideoSourceConfigurationResponse** message.

5.11. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *videoSourceConfiguration*.@token - Configuration token

- in VideoSource - Configuration Type

5.12. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

- ConfigurationToken := *videoSourceConfiguration*.@token
- ProfileToken skipped

5.13. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

- Configurations list =: *videoSourceConfList*

5.14. If *videoSourceConfList*[0] is not equal to Configuration from step 5.9, FAIL the test and skip other steps.

5.15. ONVIF Client restores settings of Video Source Configuration with @token = *videoSourceConfiguration*.@token.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

- in *s* - Subscription reference

7. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:

7.1. ONVIF Client invokes **GetVideoSourceConfigurationOptions** request with parameters

- ConfigurationToken := *videoSourceConfiguration*.@token
- ProfileToken skipped

7.2. DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters

- Options =: *options*

7.3. If *options*.Extension.Rotate specified and *options*.Extension.Rotate.@Reboot = true:

7.3.1. If *options*.Extension.Rotate list contains only "OFF" value, skip other steps.

7.3.2. If *options*.Extension.Rotate list contains only "AUTO" value, skip other steps.

7.3.3.If *options*.Extension.Rotate list contains only "ON" value and *options*.Extension.Rotate.DegreeList.Item list contains only 0 value, skip other steps.

7.3.4.If *videoSourceConfiguration* does not contain Extension.Rotate element, FAIL the test and skip other steps.

7.3.5.ONVIF Client invokes **SetVideoSourceConfiguration** request with parameters

- Configuration.@token := *videoSourceConfiguration*.@token
- Configuration.Name := "TestName1"
- Configuration.SourceToken := first value from *options*.VideoSourceTokensAvailable list
- Configuration.Bounds.x := *options*.BoundsRange.XRange.Min
- Configuration.Bounds.y := *options*.BoundsRange.YRange.Min
- Configuration.Bounds.width := *options*.BoundsRange.WidthRange.Min
- Configuration.Bounds.height := *options*.BoundsRange.HeightRange.Min
- Configuration.Extension.Rotate.Mode := first value from *options*.Extension.Rotate.Mode list differs from *videoSourceConfiguration*.Extension.Rotate.Mode (if applicable).
- If Configuration.Extension.Rotate.Mode = ON:
  - If *options*.Extension.Rotate.DegreeList is specified and contains at least one Item:
    - Configuration.Extension.Rotate.Degree := first value from *options*.Extension.Rotate.DegreeList.Item list differs from 0 (if applicable)
  - If *options*.Extension.Rotate.DegreeList is not specified or empty:
    - Configuration.Extension.Rotate.Degree := -180

7.3.6.ONVIF Client waits for the Device reboot by following the procedure mentioned in Annex A.15.

7.3.7.DUT responds with **SetVideoSourceConfigurationResponse** message.

7.3.8. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

- ConfigurationToken := *videoSourceConfiguration*.@token
- ProfileToken skipped

7.3.9. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

- Configurations list =: *videoSourceConfList*

7.3.10 If *videoSourceConfList*[0] is not equal to Configuration from step 7.3.5, FAIL the test and skip other steps.

7.3.11 ONVIF Client restores settings of Video Source Configuration with @token = *videoSourceConfiguration*.@token.

7.3.12 ONVIF Client waits for the Device reboot by following the procedure mentioned in Annex A.15.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceConfigurationsResponse** message.
- DUT did not send **SetVideoSourceConfigurationResponse** message.
- DUT did not send **GetVideoSourceConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.8 and 5.14:

- SourceToken
- Name
- Bounds.x
- Bounds.y
- Bounds.width

- Bounds.height
- Extension.Rotate
- Extension.Rotate.Mode
- Extension.Rotate.Degree
- Extension.Extension.LensDescription list (XFactor will be used as a key)
- Extension.Extension.LensDescription.FocalLength
- Extension.Extension.LensDescription.Offset.x
- Extension.Extension.LensDescription.Offset.y
- Extension.Extension.LensDescription.Projection list (Angle and Radius will be used as key)
- Extension.Extension.LensDescription.Projection.Transmittance

Note: The following fields are compared at step 7.3.7:

- Extension.Rotate
- Extension.Rotate.Mode
- Extension.Rotate.Degree

5.2.1.6 GET VIDEO SOURCE CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-2-2-6

Specification Coverage: Get configurations, Video source configuration.

Feature Under Test: GetVideoSourceConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of GetVideoSourceConfigurations with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
4. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
   - ConfigurationToken := other than listed in *videoSourceConfList*
   - ProfileToken skipped
5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message

5.2.1.7 PROFILES AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-2-2-7

Specification Coverage: None.

Feature Under Test: GetProfiles, GetVideoSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To check that GetProfiles command and GetVideoSourceConfigurationOptions command are consistent.

Pre-Requisite: Media2 Service is received from the DUT. Video Source Configuration is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters:
   - Type = VideoSource
4. The DUT responds with **GetProfilesResponse** message with parameters:
   - Profiles List =: *profilesList*
5. For each media profile *profile1* from **profilesList** containing Configurations.VideoSource element do the following steps:
   5.1. ONVIF Client invokes **GetVideoSourceConfigurationOptions**request with parameters:
      - ConfigurationToken=*profile1*.Configurations.VideoSource.@token
      - ProfileToken =: *profile1*.@token
   5.2. The DUT responds with **GetVideoSourceConfigurationOptionsResponse** message with parameters:
      - Options =: *configurationOptions*
   5.3. If *profile1*.Configurations.VideoSource.SourceToken is not equal to one of *configurationOptions*.VideoSourceTokensAvailable items, FAIL the test and skip other steps.
   5.4. If *profile1*.Configurations.VideoSource.Bounds.@x < *configurationOptions*.BoundsRange.XRange.Min, FAIL the test and skip other steps.
   5.5. If *profile1*.Configurations.VideoSource.Bounds.@x > *configurationOptions*.BoundsRange.XRange.Max, FAIL the test and skip other steps.
   5.6. If *profile1*.Configurations.VideoSource.Bounds.@y < *configurationOptions*.BoundsRange.YRange.Min, FAIL the test and skip other steps.

5.7. If *profile1*.Configurations.VideoSource.Bounds.@y > *configurationOptions*.BoundsRange.YRange.Max, FAIL the test and skip other steps.

5.8. If *profile1*.Configurations.VideoSource.Bounds.@width < *configurationOptions*.BoundsRange.WidthRange.Min, FAIL the test and skip other steps.

5.9. If *profile1*.Configurations.VideoSource.Bounds.@width > *configurationOptions*.BoundsRange.WidthRange.Max, FAIL the test and skip other steps.

5.10. If *profile1*.Configurations.VideoSource.Bounds.@height < *configurationOptions*.BoundsRange.HeightRange.Min, FAIL the test and skip other steps.

5.11. If *profile1*.Configurations.VideoSource.Bounds.@height > *configurationOptions*.BoundsRange.HeightRange.Max, FAIL the test and skip other steps.

5.12. If *profile1*.Configurations.VideoSource contains Extension element:

5.12.1. If *profile1*.Configurations.VideoSource.Extension contains Rotate element:

5.12.1.1. If *profile1*.Configurations.VideoSource.Extension.Rotate.Mode is not equal to one of *configurationOptions*.Extension.Rotate.Mode items, FAIL the test and skip other steps.

5.12.1.2. If *profile1*.Configurations.VideoSource.Extension.Rotate contains Degree element and *profile1*.Configurations.VideoSource.Extension.Rotate.Degree is not equal to one of *configurationOptions*.Extension.Rotate.DegreeList.Items items, FAIL the test and skip other steps.

5.12.2. If *profile1*.Configurations.VideoSource.Extension contains Extension element:

5.12.2.1. If *profile1*.Configurations.VideoSource.Extension.Extension contains SceneOrientation element and *profile1*.Configurations.VideoSource.Extension.Extension.SceneOrientation.Mode is not equal to one of *configurationOptions*.Extension.Extension.SceneOrientationMode items, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoSourceConfigurationOptionsResponse** message.

5.2.2 Video Encoder Configuration

5.2.2.1 VIDEO ENCODER CONFIGURATION

Test Case ID: MEDIA2-2-3-1

Specification coverage: Get media profiles (Media2 Service), Get configurations (Media2 Service)

Feature under test: GetProfiles, GetVideoEncoderConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify DUT sends All Video Encoder Configurations, specific Video Encoder Configuration, and Video Encoder Configurations compatible with specific profile.

Pre-Requisite: Media2 Service is received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
   - ConfigurationToken - skipped
   - ProfileToken - skipped
4. The DUT responds with all video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters
   - Configurations list =: *videoEncoderConfCompleteList1*
5. If *videoEncoderConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
6. ONVIF Client invokes **GetProfiles** request with parameters

- Token - skipped
- Type list - skipped

7. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList1*
8. For each Media Profile *profile1* in *profileList1* repeat the following steps:
   8.1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
      - ConfigurationToken - skipped
      - ProfileToken := *profile1*.@token
   8.2. The DUT responds with compatible video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters
      - Configurations list =: *videoEncoderConfList1*
   8.3. If *videoEncoderConfList1* contains at least two items with the same @token, FAIL the test and skip other steps.
   8.4. If *profile1*.Configurations contains VideoEncoder:
      8.4.1. If *videoEncoderConfList1* does not contain item with @token = *profile1*.Configurations.VideoEncoder.@token, FAIL the test and skip other steps.
   8.5. For each Video Encoder Configuration *videoEncoderConf1* in *videoEncoderConfList1* repeat the following steps:
      8.5.1. If *videoEncoderConfCompleteList1* does not contain item with @token = *videoEncoderConf1*.@token, FAIL the test and skip other steps.
9. For each Video Encoder Configuration *configuration1* in *videoEncoderConfCompleteList1* repeat the following steps:
   9.1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
      - ConfigurationToken := *configuration1*.@token
   9.2. The DUT responds with requested video encoder configuration in **GetVideoEncoderConfigurationsResponse** with parameters
      - Configurations list =: *videoEncoderConfList2*

9.3. If *videoEncoderConfList2* does not contain only one item with the same @token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.

5.2.2.2 VIDEO ENCODER CONFIGURATIONS AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-2-3-2

Specification Coverage: None.

Feature Under Test: GetVideoEncoderConfigurations, GetVideoEncoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all video encoder configurations are consistent with video encoder configurations options.

Pre-Requisite: Media2 Service is received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoEncoderConfigurations** to get all video encoder configurations from DUT.
4. DUT sends the list of video encoder configurations.
5. ONVIF Client verifies the list of video encoder configurations sent by DUT.
6. For each Video Encoder Configuration in **GetVideoEncoderConfigurationsResponse**, ONVIF Client saves this configuration in *Configuration1* and runs the following steps:

6.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** with *Configuration1* token as input argument.

6.2. DUT sends **GetVideoEncoderConfigurationOptionsResponse** with the list of video encoder configuration options available for *Configuration1*.

6.3. ONVIF Client verifies that *Configuration1* parameters are consistent with at least one option from **GetVideoEncoderConfigurationOptionsResponse**. See details of fields mapping in Annex A.16 VideoEncoderConfigurationOptions and VideoEncoderConfiguration mapping.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT failed consistency check according to Annex A.16.

5.2.2.3 PROFILES AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-2-3-3

Specification Coverage: None.

Feature Under Test: GetProfiles, GetVideoEncoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To check that **GetProfiles** command and GetVideoEncoderConfigurationOptions command are consistent.

Pre-Requisite: Media2 Service is received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.

3. ONVIF Client invokes **GetProfiles** with Type=VideoEncoder as input parameter.
4. DUT sends the list of existing media profiles in **GetProfilesResponse** message.
5. For each media profile from **GetProfilesResponse**, ONVIF Client saves this profile in *Profile1* variable, saves *Profile1* Configurations VideoEncoder configuration in *Configuration1* variable and runs the following steps:
   5.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** with ConfigurationToken=*Configuration1* and ProfileToken=*Profile1* token as input arguments.
   5.2. DUT sends **GetVideoEncoderConfigurationOptionsResponse** with the list of configuration *options*.
   5.3. ONVIF Client verifies that *Configuration1* parameters are consistent with at least one option from **GetVideoEncoderConfigurationOptionsResponse**. See details of fields mapping in Annex A.16 VideoEncoderConfigurationOptions and VideoEncoderConfiguration mapping. ONVIF Client saves this option in *Option1* variable.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT failed consistency check according to Annex A.16

5.2.2.4 SET ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS

Test Case ID: MEDIA2-2-3-4

Specification Coverage: Get configurations, Get configuration *options*, Video encoder configuration, Modify a configuration

Feature Under Test: GetVideoEncoderConfigurationOptions, GetVideoEncoderConfigurations, SetVideoEncoderConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify whether all supported *options* can be set.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Encoder Configurations list by following the procedure mentioned in Annex A.17 with the following input and output parameters
   - out *videoEncoderConfList* - Video Encoder Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
5. For each Video Encoder Configuration *videoEncoderConfiguration* in *videoEncoderConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters
      - ConfigurationToken := *videoEncoderConfiguration*.@token
      - ProfileToken skipped
   5.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters
      - Options list =: *optionsList*
   5.3. For each *options* in *optionsList* repeat the following steps:

5.3.1. ONVIF Client invokes **SetVideoEncoderConfiguration** request with parameters

- Configuration.@token := *videoEncoderConfiguration*.@token
- Configuration.Name := "TestName1"
- Configuration.@GovLength := if *options*.GovLengthRange is specified, set minimum value from *options*.GovLengthRange list, otherwise, skip the parameter
- Configuration.@Profile := if *videoEncoderConfiguration*.@Profile is specified and *options*.@ProfilesSupported is specified and contains at least one item, *options*.@ProfilesSupported[0], otherwise, skip the parameter
- Configuration.Encoding := *options*.Encoding
- Configuration.Resolution := *options*.ResolutionsAvailable[0]
- Configuration.RateControl.@ConstantBitRate skipped
- Configuration.RateControl.@FrameRateLimit := minimum value from *options*.FrameRatesSupported list
- Configuration.RateControl.@BitrateLimit := *options*.BitrateRange.Min
- Configuration.Multicast := *videoEncoderConfiguration*.Multicast
- Configuration.Quality := *options*.QualityRange.Min

5.3.2. DUT responds with **SetVideoEncoderConfigurationResponse** message.

5.3.3. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *videoEncoderConfiguration*.@token - Configuration token

- in VideoEncoder - Configuration Type

5.3.4. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters

- ConfigurationToken := *videoEncoderConfiguration*.@token
- ProfileToken skipped

5.3.5. The DUT responds with requested video encoder configuration in **GetVideoEncoderConfigurationsResponse** with parameters

- Configurations list =: *videoEncoderConfList*

5.3.6. If *videoEncoderConfList*[0] is not equal to Configuration from step 5.3.1, FAIL the test and skip other steps.

5.3.7. ONVIF Client invokes **SetVideoEncoderConfiguration** request with parameters

- Configuration.@token := *videoEncoderConfiguration*.@token
- Configuration.Name := "TestName2"
- Configuration.@GovLength := if *options*.GovLengthRange is specified, set maximum value from *options*.GovLengthRange list, otherwise, skip the parameter
- Configuration.@Profile := if *videoEncoderConfiguration*.@Profile is specified and *options*.@ProfilesSupported is specified and contains at least one item, *options*.@ProfilesSupported[last], otherwise, skip the parameter
- Configuration.Encoding := *options*.Encoding
- Configuration.Resolution := *options*.ResolutionsAvailable[last]
- Configuration.RateControl.@ConstantBitRate skipped
- Configuration.RateControl.@FrameRateLimit := maximum value from *options*.FrameRatesSupported list
- Configuration.RateControl.@BitrateLimit := *options*.BitrateRange.Max
- Configuration.Multicast := *videoEncoderConfiguration*.Multicast
- Configuration.Quality := *options*.QualityRange.Max

5.3.8. DUT responds with **SetVideoEncoderConfigurationResponse** message.

5.3.9. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *videoEncoderConfiguration*.@token - Configuration token
- in VideoEncoder - Configuration Type

5.3.10. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters

- ConfigurationToken := *videoEncoderConfiguration*.@token
- ProfileToken skipped

5.3.11. The DUT responds with requested video encoder configuration in **GetVideoEncoderConfigurationsResponse** with parameters

- Configurations list =: *videoEncoderConfList*

5.3.12. If *videoEncoderConfList*[0] is not equal to Configuration from step 5.3.7, FAIL the test and skip other steps.

6. ONVIF Client restores settings for all Video Encoder Configurations.
7. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

- in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT did not send **SetVideoEncoderConfigurationsResponse** message.
- DUT did not send **SetVideoEncoderConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.3.6 and 5.3.12:

- Name
- Encoding
- Resolution.Height
- Resolution.Width
- GovLength

5.2.2.5 VIDEO ENCODER CONFIGURATION OPTIONS

Test Case ID: MEDIA2-2-3-5

Specification Coverage: Video Encoder Configuration Options.

Feature Under Test: GetVideoEncoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To validate video encoder configurations options.

Pre-Requisite: Media2 Service is received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped

4. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** with parameters
   - Options list =: *vecOptionsList*
5. For each Options (*options*) in (*vecOptionsList*) repeat the following steps:
   5.1. If *options* contains GovLengthRange, ONVIF Client checks the following:
      5.1.1. If *options*.GovLengthRange list does not contain two values, FAIL the test and skip other steps.
      5.1.2. If *options*.GovLengthRange list contains more than two values, FAIL the test and skip other steps.
      5.1.3. If the first value in *options*.GovLengthRange list is greater than the second value, FAIL the test and skip other steps.
   5.2. If *options* contains FrameRatesSupported and *options*.FrameRatesSupported list is not sorted with descending sort order, FAIL the test and skip other steps.
6. ONVIF Client invokes **GetVideoEncoderConfigurations** with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
7. The DUT responds with all video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters
   - Configurations list =: *vecList*
8. If *vecList* is empty, FAIL the test and skip other steps.
9. For each Video Encoder Configuration (*vec*) in *vecList* repeat the following steps:
   9.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters
      - ConfigurationToken := *vec*.@token
      - ProfileToken skipped
   9.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** with parameters
      - Options list =: *vecOptionsList*
   9.3. For each Options (*options*) in (*vecOptionsList*) repeat the following steps:

9.3.1. If *options* contains GovLengthRange, ONVIF Client checks the following:

9.3.1.1. If *options*.GovLengthRange list does not contain two values, FAIL the test and skip other steps.

9.3.1.2. If *options*.GovLengthRange list contains more than two values, FAIL the test and skip other steps.

9.3.1.3. If the first value in *options*.GovLengthRange list is greater than the second value, FAIL the test and skip other steps.

9.3.2. If *options* contains FrameRatesSupported and *options*.FrameRatesSupported list is not sorted with descending sort order, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.

5.2.3 Video Source

5.2.3.1 GET VIDEO SOURCE MODES

Test Case ID: MEDIA2-2-4-1

Specification Coverage: Video source mode, GetVideoSourceModes.

Feature Under Test: GetVideoSourceModes

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving supported Video Source Modes for specified Video Source.

Pre-Requisite: Media2 Service is received from the DUT. Device IO Service is received from the DUT. Video sources modes is supported by Device as indicated by the VideoSourceMode=true capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Sources list by following the procedure mentioned in Annex A.18 with the following input and output parameters
   - out *videoSourcesList* - Video Sources list
4. For each Video Source token *videoSourceToken* from *videoSourcesList* repeat the following steps:
   4.1. ONVIF Client invokes **GetVideoSourceModes** request with parameters
      - VideoSourceToken := *videoSourceToken*
   4.2. DUT responds with **GetVideoSourceModesResponse** message with parameters
      - VideoSourceModes list =: *videoSourceModesList*
   4.3. If *videoSourceModesList* contains at least two items with the same @token, FAIL the test and skip other steps.
   4.4. If *videoSourceModesList* contains at least two items with Enabled=true, FAIL the test and skip other steps.
   4.5. If *videoSourceModesList* contains no items with Enabled=true, FAIL the test and skip other steps.
   4.6. If *videoSourceModesList* contains at least one item with empty Encodings list, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceModesResponse** message.

5.2.3.2 SET VIDEO SOURCE MODES

Test Case ID: MEDIA2-2-4-2

Specification Coverage: Video source mode, GetVideoSourceModes, SetVideoSourceModes.

Feature Under Test: GetVideoSourceModes, SetVideoSourceModes

WSDL Reference: media2.wsdl, deviceio.wsld

Test Purpose: To verify change of Video Source Mode for specified Video Source.

Pre-Requisite: Media2 Service is received from the DUT. Device IO Service is received from the DUT. Video sources modes is supported by Device as indicated by the VideoSourceMode=true capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Sources list by following the procedure mentioned in Annex A.18 with the following input and output parameters
   - out *videoSourcesList* - Video Sources list
4. For each Video Source token *videoSourceToken* from *videoSourcesList* repeat the following steps:
   4.1. ONVIF Client invokes **GetVideoSourceModes** request with parameters
      - VideoSourceToken := *videoSourceToken*
   4.2. DUT responds with **GetVideoSourceModesResponse** message with parameters
      - VideoSourceModes list =: *videoSourceModesList*
   4.3. Set *modeToEnable* := videoSourceModesList[*first*].@token, where *first* is the index number of the first item on the *videoSourceModesList* list that has no @Enabled or @Enabled = false (if any).
   4.4. ONVIF Client invokes **SetVideoSourceMode** request with parameters
      - VideoSourceToken := *videoSourceToken*
      - VideoSourceModeToken := *modeToEnable*
   4.5. DUT responds with **SetVideoSourceModeResponse** message with parameters
      - Reboot =: *rebootFlag*
   4.6. If *rebootFlag* = true, ONVIF Client waits for the Device reboot by following the procedure mentioned in Annex A.15.

4.7. ONVIF Client invokes **GetVideoSourceModes** request with parameters

- VideoSourceToken := *videoSourceToken*

4.8. DUT responds with **GetVideoSourceModesResponse** message with parameters

- VideoSourceModes list =: *videoSourceModesList*

4.9. If *videoSourceModesList* contains at least two items with @Enabled=true, FAIL the test and skip other steps.

4.10. If *videoSourceModesList* item with @token = *modeToEnable* has no @Enabled or @Enabled = false for it, FAIL the test and skip other steps.

4.11. If *videoSourceModesList* have only one item, go to step 4.20

4.12. Set *modeToEnable* := videoSourceModesList[*last*].@token, where *last* is the index number of the last item on the *videoSourceModesList* list.

4.13. ONVIF Client invokes **SetVideoSourceMode** request with parameters

- VideoSourceToken := *videoSourceToken*
- VideoSourceModeToken := *modeToEnable*

4.14. DUT responds with **SetVideoSourceModeResponse** message with parameters

- Reboot =: *rebootFlag*

4.15. If *rebootFlag* = true, ONVIF Client waits for the Device reboot by following the procedure mentioned in Annex A.15.

4.16. ONVIF Client invokes **GetVideoSourceModes** request with parameters

- VideoSourceToken := *videoSourceToken*

4.17. DUT responds with **GetVideoSourceModesResponse** message with parameters

- VideoSourceModes list =: *videoSourceModesList*

4.18. If *videoSourceModesList* contains at least two items with @Enabled=true, FAIL the test and skip other steps.

4.19. If *videoSourceModesList* item with @token = *modeToEnable* has no @Enabled or @Enabled = false for it, FAIL the test and skip other steps.

4.20. ONVIF Client restores settings for *videoSourceModesList*.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceModesResponse** message.
- DUT did not send **SetVideoSourceModesResponse** message.

5.3 Audio Configuration

5.3.1 Audio Source Configuration

5.3.1.1 GET AUDIO SOURCE CONFIGURATION OPTIONS

Test Case ID: MEDIA2-3-1-1

Specification Coverage: Get configuration options, Audio source configuration.

Feature Under Test: GetAudioSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving Audio Source Configuration options for specified Audio Source Configuration, for specified Profile, generic for the Device.

Pre-Requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioSourceConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped

4. DUT responds with **GetAudioSourceConfigurationOptionsResponse** message with parameters
   - Options =: *options*
5. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters
   - out *audioSourceConfCompleteList* - Audio Source Configurations list
6. ONVIF Client invokes **GetAudioSourceConfigurationOptions** request with parameters
   - ConfigurationToken := *audioSourceConfigurationList*[0].@token
   - ProfileToken skipped
7. DUT responds with **GetAudioSourceConfigurationOptionsResponse** message with parameters
   - Options =: *options*
8. ONVIF Client retrieves Media Profile, which contains Audio Source Configuration by following the procedure mentioned in Annex A.19 with the following input and output parameters
   - out *profile* - Media Profile with Audio Source Configuration
9. ONVIF Client invokes **GetAudioSourceConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken := *profile*.@token
10. DUT responds with **GetAudioSourceConfigurationOptionsResponse** message with parameters
    - Options =: *options*
11. If Media Profile *profile* was changed at step 8, ONVIF Client restores Media Profile.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationOptionsResponse** message.

5.3.1.2 GET AUDIO SOURCE CONFIGURATIONS

Test Case ID: MEDIA2-3-1-2

Specification Coverage: Get configurations, Audio source configuration.

Feature Under Test: GetAudioSourceConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Audio Source Configuration List, Audio Source Configuration by Configuration token and compatible Audio Source Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
4. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters
   - Configurations list =: *audioSourceConfCompleteList*
5. If *audioSourceConfCompleteList* is empty, FAIL the test and skip other steps.
6. If *audioSourceConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
7. For each Audio Source Configuration *audioSourceConfiguration* in *audioSourceConfCompleteList* repeat the following steps:
   7.1. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
      - ConfigurationToken := *audioSourceConfiguration*.@token
      - ProfileToken skipped
   7.2. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters

- Configurations list =: *audioSourceConfList*

7.3. If *audioSourceConfList* is empty, FAIL the test and skip other steps.

7.4. If *audioSourceConfList* contains more than one item, FAIL the test and skip other steps.

7.5. If *audioSourceConfList* does not contain item with @token = *audioSourceConfiguration*.@token, FAIL the test and skip other steps.

8. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioSource
9. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
10. For each Media Profile *profile* in *profileList* repeat the following steps:

10.1. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken := *profile*.@token

10.2. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters

- Configurations list =: *audioSourceConfList*

10.3. If *audioSourceConfList* contains at least two items with the same @token, FAIL the test and skip other steps.

10.4. If *audioSourceConfCompleteList* does not contain at least one item with @token from *audioSourceConfList*, FAIL the test and skip other steps.

10.5. If *profile*.Configurations contains AudioSource:

10.5.1. If *audioSourceConfList* does not contain item with @token = *profile*.Configurations.AudioSource.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.3.1.3 AUDIO SOURCE CONFIGURATIONS AND AUDIO SOURCE CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-3-1-3

Specification Coverage: Get configurations, Get configuration options, Audio source configuration.

Feature Under Test: GetProfiles, GetAudioSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Audio Source Configurations are consistent with Audio Source Configuration Options.

Pre-Requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters
   - out *audioSourceConfList* - Audio Source Configurations list
4. For each Audio Source Configuration *audioSourceConfiguration* in *audioSourceConfList* repeat the following steps:
   4.1. ONVIF Client invokes **GetAudioSourceConfigurationOptions** request with parameters
      - ConfigurationToken := *audioSourceConfiguration*.@token
      - ProfileToken skipped

4.2. DUT responds with **GetAudioSourceConfigurationOptionsResponse** message with parameters

- Options =: *options*

4.3. If *audioSourceConfiguration*.SourceToken is not in *options*.InputTokensAvailable list, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationOptionsResponse** message.

5.3.1.4 PROFILES AND AUDIO SOURCE CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-3-1-4

Specification Coverage: Get configurations, Get media profiles, Audio source configuration.

Feature Under Test: GetAudioSourceConfigurations, GetAudioSourceConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Audio Source Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped

- Type[0] := AudioSource

4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList* which contains Configurations.AudioSource repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
      - ConfigurationToken := *profile*.Configurations.AudioSource.@token ProfileToken skipped
   5.2. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters
      - Configurations list =: *audioSourceConfList*
   5.3. If *audioSourceConfList*[0] is not equal to *profile*.Configurations.AudioSource, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetAudioSourceConfigurationsResponse** message.

5.3.1.5 MODIFY ALL SUPPORTED AUDIO SOURCE CONFIGURATIONS

Test Case ID: MEDIA2-3-1-5

Specification Coverage: Get configurations, Get configuration options, Audio source configuration, Modify a configuration

Feature Under Test: GetAudioSourceConfigurationOptions, GetAudioSourceConfigurations, SetAudioSourceConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify whether all supported Audio Source Configuration Options can be set.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters
   - out *audioSourceConfList* - Audio Source Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
5. For each Audio Source Configuration *audioSourceConfiguration* in *audioSourceConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioSourceConfigurationOptions** request with parameters
      - ConfigurationToken := *audioSourceConfiguration*.@token
      - ProfileToken skipped
   5.2. DUT responds with **GetAudioSourceConfigurationOptionsResponse** message with parameters
      - Options =: *options*
   5.3. ONVIF Client invokes **SetAudioSourceConfiguration** request with parameters

- Configuration.@token := *audioSourceConfiguration*.@token
- Configuration.Name := "TestName1"
- Configuration.SourceToken := other then current value (if possible) from *options*.InputTokensAvailable list

5.4. DUT responds with **SetAudioSourceConfigurationResponse** message.

5.5. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioSourceConfiguration*.@token - Configuration token
- in AudioSource - Configuration Type

5.6. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters

- ConfigurationToken := *audioSourceConfiguration*.@token
- ProfileToken skipped

5.7. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters

- Configurations list =: *audioSourceConfList*

5.8. If *audioSourceConfList*[0] is not equal to Configuration from step 5.3, FAIL the test and skip other steps.

5.9. ONVIF Client restores settings of Audio Source Configuration with @token = *audioSourceConfiguration*.@token.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

- in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationsResponse** message.
- DUT did not send **SetAudioSourceConfigurationResponse** message.
- DUT did not send **GetAudioSourceConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.8:

- SourceToken
- Name

5.3.1.6 GET AUDIO SOURCE CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-3-1-6

Specification Coverage: Get configurations, Audio source configuration.

Feature Under Test: GetAudioSourceConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of **GetAudioSourceConfigurations** with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.9 with the following input and output parameters

- out *audioSourceConfCompleteList* - Audio Source Configurations list

4. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
   - ConfigurationToken := other than listed in *audioSourceConfCompleteList*
   - ProfileToken skipped
5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message.

5.3.2 Audio Encoder Configuration

5.3.2.1 G.711 AUDIO ENCODER CONFIGURATION

Test Case ID: MEDIA2-3-2-1

Specification Coverage: Get configurations, Get configuration options, Audio encoder configuration, Modify a configuration.

Feature Under Test: GetAudioEncoderConfigurations, GetAudioEncoderConfigurationOptions, SetAudioEncoderConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT changes audio configuration with G.711 Encoding properly.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Media2_G.711 feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client retrieves Audio Encoder Configurations list by following the procedure mentioned in Annex A.20 with the following input and output parameters
   - out *audioEncoderConfList* - Audio Encoder Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
5. For each Audio Encoder Configuration *audioEncoderConfiguration* in *audioEncoderConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters
      - ConfigurationToken := *audioEncoderConfiguration*.@token
      - ProfileToken skipped
   5.2. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters
      - Options list =: *optionsList*
   5.3. If *optionsList* contains an item with Encoding = PCMU, do the following steps:
      5.3.1. Set *options* := first item from *optionsList* with Encoding = PCMU.
      5.3.2. ONVIF Client invokes **SetAudioEncoderConfiguration** request with parameters
         - Configuration.@token := *audioEncoderConfiguration*.@token
         - Configuration.Name := string different from *audioEncoderConfiguration*.Name value (length shall be less than or equal to 64 characters and it shall contain only readable characters)

- Configuration.Encoding := *options*.Encoding
- Configuration.Bitrate := minimum value from *options*.BitrateList.Items list if *options*.BitrateList.Items list contains items, otherwise, *audioEncoderConfiguration*.Bitrate
- Configuration.SampleRate := minimum value from *options*.SampleRateList.Items list if *options*.SampleRateList.Items list contains items, otherwise, *audioEncoderConfiguration*.SampleRate
- Configuration.Multicast := *audioEncoderConfiguration*.Multicast

5.3.3. DUT responds with **SetAudioEncoderConfigurationResponse** message.

5.3.4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioEncoderConfiguration*.@token - Configuration token
- in AudioEncoder - Configuration Type

5.3.5. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters

- ConfigurationToken := *audioEncoderConfiguration*.@token
- ProfileToken skipped

5.3.6. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters

- Configurations list =: *audioEncoderConfList*

5.3.7. If *audioEncoderConfList* contains more items with @token = *audioEncoderConfiguration*.@token than 1, FAIL the test and skip other steps.

5.3.8. If *audioEncoderConfList*[0] is not equal to Configuration from step 5.3.2, FAIL the test and skip other steps.

5.3.9. ONVIF Client invokes **SetAudioEncoderConfiguration** request with parameters

- Configuration.@token := *audioEncoderConfiguration*.@token
- Configuration.Name := *audioEncoderConfiguration*.Name
- Configuration.Encoding := *options*.Encoding
- Configuration.Bitrate := maximum value from *options*.BitrateList.Items list if *options*.BitrateList.Items list contains items, otherwise, *audioEncoderConfiguration*.Bitrate
- Configuration.SampleRate := maximum value from *options*.SampleRateList.Items list if *options*.SampleRateList.Items list contains items, otherwise, *audioEncoderConfiguration*.SampleRate
- Configuration.Multicast := *audioEncoderConfiguration*.Multicast

5.3.10. DUT responds with **SetAudioEncoderConfigurationResponse** message.

5.3.11. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioEncoderConfiguration*.@token - Configuration token
- in AudioEncoder - Configuration Type

5.3.12. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters

- ConfigurationToken := *audioEncoderConfiguration*.@token
- ProfileToken skipped

5.3.13. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters

- Configurations list =: *audioEncoderConfList*

5.3.14. If *audioEncoderConfList* contains more items with @token = *audioEncoderConfiguration*.@token than 1, FAIL the test and skip other steps.

5.3.15. If *audioEncoderConfList*[0] is not equal to Configuration from step 5.3.9, FAIL the test and skip other steps.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

   - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.3.8 and 5.3.15:

- @token
- Name
- Encoding
- Multicast.Address.Type
- Multicast.Address.IPv4Address
- Multicast.Address.IPv6Address
- Multicast.Port
- Multicast.TTL
- Multicast.AutoStart
- Bitrate
- SampleRate

5.3.2.2 AAC AUDIO ENCODER CONFIGURATION

Test Case ID: MEDIA2-3-2-2

Specification Coverage: Get configurations, Get configuration options, Audio encoder configuration, Modify a configuration.

Feature Under Test: GetAudioEncoderConfigurations, GetAudioEncoderConfigurationOptions, SetAudioEncoderConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT changes audio configuration with AAC Encoding properly.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Media2_AAC feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Encoder Configurations list by following the procedure mentioned in Annex A.20 with the following input and output parameters
   - out *audioEncoderConfList* - Audio Encoder Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
5. For each Audio Encoder Configuration *audioEncoderConfiguration* in *audioEncoderConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters
      - ConfigurationToken := *audioEncoderConfiguration*.@token

- ProfileToken skipped

5.2. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters

- Options list =: *optionsList*

5.3. If *optionsList* contains an item with Encoding = MP4A-LATM or MPEG4-GENERIC, do the following steps:

5.3.1. Set *options* := first item from *optionsList* with Encoding = MP4A-LATM or MPEG4-GENERIC.

5.3.2. ONVIF Client invokes **SetAudioEncoderConfiguration** request with parameters

- Configuration.@token := *audioEncoderConfiguration*.@token
- Configuration.Name := string different from *audioEncoderConfiguration*.Name value (length shall be less than or equal to 64 characters and it shall contain only readable characters)
- Configuration.Encoding := *options*.Encoding
- Configuration.Bitrate := minimum value from *options*.BitrateList.Items list if *options*.BitrateList.Items list contains items, otherwise, *audioEncoderConfiguration*.Bitrate
- Configuration.SampleRate := minimum value from *options*.SampleRateList.Items list if *options*.SampleRateList.Items list contains items, otherwise, *audioEncoderConfiguration*.SampleRate
- Configuration.Multicast := *audioEncoderConfiguration*.Multicast

5.3.3. DUT responds with **SetAudioEncoderConfigurationResponse** message.

5.3.4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT

- in *terminationTime* - subscription termination time
- in *audioEncoderConfiguration*.@token - Configuration token
- in AudioEncoder - Configuration Type

5.3.5. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters

- ConfigurationToken := *audioEncoderConfiguration*.@token
- ProfileToken skipped

5.3.6. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters

- Configurations list =: *audioEncoderConfList*

5.3.7. If *audioEncoderConfList* contains more items with @token = *audioEncoderConfiguration*.@token than 1, FAIL the test and skip other steps.

5.3.8. If *audioEncoderConfList*[0] is not equal to Configuration from step 5.3.2, FAIL the test and skip other steps.

5.3.9. ONVIF Client invokes **SetAudioEncoderConfiguration** request with parameters

- Configuration.@token := *audioEncoderConfiguration*.@token
- Configuration.Name := *audioEncoderConfiguration*.Name
- Configuration.Encoding := *options*.Encoding
- Configuration.Bitrate := maximum value from *options*.BitrateList.Items list if *options*.BitrateList.Items list contains items, otherwise, *audioEncoderConfiguration*.Bitrate
- Configuration.SampleRate := maximum value from *options*.SampleRateList.Items list if *options*.SampleRateList.Items list contains items, otherwise, *audioEncoderConfiguration*.SampleRate
- Configuration.Multicast := *audioEncoderConfiguration*.Multicast

5.3.10. DUT responds with **SetAudioEncoderConfigurationResponse** message.

5.3.11. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioEncoderConfiguration*.@token - Configuration token
- in AudioEncoder - Configuration Type

5.3.12. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters

- ConfigurationToken := *audioEncoderConfiguration*.@token
- ProfileToken skipped

5.3.13. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters

- Configurations list =: *audioEncoderConfList*

5.3.14. If *audioEncoderConfList* contains more items with @token = *audioEncoderConfiguration*.@token than 1, FAIL the test and skip other steps.

5.3.15. If *audioEncoderConfList*[0] is not equal to Configuration from step 5.3.9, FAIL the test and skip other steps.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

- in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.3.8 and 5.3.15:

- @token
- Name
- Encoding
- Multicast.Address.Type
- Multicast.Address.IPv4Address
- Multicast.Address.IPv6Address
- Multicast.Port
- Multicast.TTL
- Multicast.AutoStart
- Bitrate
- SampleRate

5.3.2.3 GET AUDIO ENCODER CONFIGURATION OPTIONS

Test Case ID: MEDIA2-3-2-3

Specification Coverage: Get configuration options.

Feature Under Test: GetAudioEncoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving Audio Encoder Configuration Options for specified Audio Encoder Configuration, for specified Profile, generic for the Device.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters
   - ConfigurationToken skipped

- ProfileToken skipped

4. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters
   - Options =: *options*
5. ONVIF Client retrieves Audio Encoder Configurations list by following the procedure mentioned in Annex A.20 with the following input and output parameters
   - out *audioEncoderConfList* - Audio Encoder Configurations list
6. For each Audio Encoder Configuration (*audioEncoderConf*) in *audioEncoderConfList*
   6.1. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters
      - ConfigurationToken := *audioEncoderConf*.@token
      - ProfileToken skipped
   6.2. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters
      - Options =: *options*
7. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type : = AudioSource
8. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
9. For each Profile (*profile*) in *profileList* that contains Configurations.AudioSource
   9.1. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters
      - ConfigurationToken skipped
      - ProfileToken := *profile*.@token
   9.2. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters

- Options =: *options*

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioEncoderConfigurationOptionsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.3.2.4 AUDIO ENCODER CONFIGURATIONS AND AUDIO ENCODER CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-3-2-4

Specification Coverage: Get configurations, Get configuration options, Audio encoder configuration.

Feature Under Test: GetProfiles, GetAudioEncoderConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify all Audio Encoder Configurations are consistent with Audio Encoder Configuration Options.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Encoder Configurations list by following the procedure mentioned in Annex A.20 with the following input and output parameters
   - out *audioEncoderConfList* - Audio Encoder Configurations list
4. For each Audio Encoder Configuration *audioEncoderConfiguration* in *audioEncoderConfList* repeat the following steps:

4.1. ONVIF Client invokes **GetAudioEncoderConfigurationOptions** request with parameters

- ConfigurationToken := *audioEncoderConfiguration*.@token
- ProfileToken skipped

4.2. DUT responds with **GetAudioEncoderConfigurationOptionsResponse** message with parameters

- Options List=: *optionsList*

4.3. If *optionsList* does not contain at least one Options.Encoding element with value is equal to *audioEncoderConfiguration*.Encoding, FAIL the test and skip other steps.

4.4. If *optionsList* does not contain at least one Options.Encoding element with value is equal to *audioEncoderConfiguration*.Encoding and with at least one Options.BitrateList.Items element with value is equal to *audioEncoderConfiguration*.Bitrate, FAIL the test and skip other steps.

4.5. If *optionsList* does not contain at least one Options.Encoding element with value is equal to *audioEncoderConfiguration*.Encoding and with at least one Options.SampleRateList.Items element with value is equal to *audioEncoderConfiguration*.SampleRate, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioEncoderConfigurationOptionsResponse** message.

5.3.3 Audio Output Configuration

5.3.3.1 GET AUDIO OUTPUT CONFIGURATION OPTIONS

Test Case ID: MEDIA2-3-3-1

Specification Coverage: Get configuration options, Audio output configuration.

Feature Under Test: GetAudioOutputConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving Audio Output Configuration options for specified Audio Output Configuration, for specified Profile, generic for the Device.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioOutputConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
4. DUT responds with **GetAudioOutputConfigurationOptionsResponse** message with parameters
   - Options =: *options*
5. If *options*.OutputLevelRange.Min > *options*.OutputLevelRange.Max, FAIL the test and skip other steps.
6. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters
   - out *audioOutputConfList* - Audio Output Configurations list
7. ONVIF Client invokes **GetAudioOutputConfigurationOptions** request with parameters
   - ConfigurationToken := *audioOutputConfList*[0].@token
   - ProfileToken skipped
8. DUT responds with **GetAudioOutputConfigurationOptionsResponse** message with parameters
   - Options =: *options*
9. ONVIF Client retrieves Media Profile, which contains Audio Output Configuration by following the procedure mentioned in Annex A.21 with the following input and output parameters
   - out *profile* - Media Profile with Audio Output Configuration

10. ONVIF Client invokes **GetAudioOutputConfigurationOptions** request with parameters
    - ConfigurationToken skipped
    - ProfileToken := *profile*.@token
11. DUT responds with **GetAudioOutputConfigurationOptionsResponse** message with parameters
    - Options =: *options*
12. If Media Profile *profile* was changed at step 9, ONVIF Client restores Media Profile.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationOptionsResponse** message.

5.3.3.2 GET AUDIO OUTPUT CONFIGURATIONS

Test Case ID: MEDIA2-3-3-2

Specification Coverage: Get configurations, Audio output configuration.

Feature Under Test: GetAudioOutputConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Audio Output Configuration List, Audio Output Configuration by Configuration token and compatible Audio Output Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
    - ConfigurationToken skipped

- ProfileToken skipped

4. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
   - Configurations list =: *audioOutputConfCompleteList*
5. If *audioOutputConfCompleteList* is empty, FAIL the test and skip other steps.
6. If *audioOutputConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
7. For each Audio Output Configuration *audioOutputConfiguration* in *audioOutputConfCompleteList* repeat the following steps:
   7.1. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
      - ConfigurationToken := *audioOutputConfiguration*.@token
      - ProfileToken skipped
   7.2. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
      - Configurations list =: *audioOutputConfList*
   7.3. If *audioOutputConfList* is empty, FAIL the test and skip other steps.
   7.4. If *audioOutputConfList* contains more than one item, FAIL the test and skip other steps.
   7.5. If *audioOutputConfList* does not contain item with @token = *audioOutputConfiguration*.@token, FAIL the test and skip other steps.
8. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioOutput
9. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
10. For each Media Profile *profile* in *profileList* repeat the following steps:
    10.1. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
      - ConfigurationToken skipped
      - ProfileToken := *profile*.@token

10.2. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters

- Configurations list =: *audioOutputConfList*

10.3. If *audioOutputConfList* contains at least two items with the same @token, FAIL the test and skip other steps.

10.4. If *audioOutputConfCompleteList* does not contain at least one item with @token from *audioOutputConfList*, FAIL the test and skip other steps.

10.5. If *profile*.Configurations contains AudioOutput:

10.5.1. If *audioOutputConfList* does not contain item with @token = *profile*.Configurations.AudioOutput.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationOptionsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.3.3.3 AUDIO OUTPUT CONFIGURATIONS AND AUDIO OUTPUT CONFIGURATION OPTIONS CONSISTENCY

Test Case ID: MEDIA2-3-3-3

Specification Coverage: Get configurations, Get configuration options, Audio output configuration.

Feature Under Test: GetProfiles, GetAidioOutputConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Audio Output Configurations are consistent with Audio Output Configuration Options.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters
   - out *audioOutputConfList* - Audio Output Configurations list
4. For each Audio Output Configuration *audioOutputConfiguration* in *audioOutputConfList* repeat the following steps:
   4.1. ONVIF Client invokes **GetAudioOutputConfigurationOptions** request with parameters
      - ConfigurationToken := *audioOutputConfiguration*.@token
      - ProfileToken skipped
   4.2. DUT responds with **GetAudioOutputConfigurationOptionsResponse** message with parameters
      - Options =: *options*
   4.3. If *audioOutputConfiguration*.OutputToken is not in *options*.OutputTokensAvailable list, FAIL the test and skip other steps.
   4.4. If *audioOutputConfiguration*.SendPrimacy specified:
      4.4.1. If *audioOutputConfiguration*.SendPrimacy is not in *options*.SendPrimacyOptions, FAIL the test and skip other steps
   4.5. If *audioOutputConfiguration*.OutputLevel < *options*.OutputLevelRange.Min, FAIL the test and skip other steps.
   4.6. If *audioOutputConfiguration*.OutputLevel > *options*.OutputLevelRange.Max, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationOptionsResponse** message.

5.3.3.4 PROFILES AND AUDIO OUTPUT CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-3-3-4

Specification Coverage: Get configurations, Get media profiles, Audio output configuration.

Feature Under Test: GetAudioOutputConfigurations, GetAudioOutputConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Audio Output Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioOutput
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList* which contains Configurations.AudioOutput repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
      - ConfigurationToken := *profile*.Configurations.AudioOutput.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
      - Configurations list =: *audioOutputConfList*
   5.3. If *audioOutputConfList*[0] is not equal to *profile*.Configurations.AudioOutput, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

Note: The following fields are compared at step 5.3:

- Name
- OutputToken
- SendPrimacy
- OutputLevel

5.3.3.5 MODIFY ALL SUPPORTED AUDIO OUTPUT CONFIGURATIONS

Test Case ID: MEDIA2-3-3-5

Specification Coverage: Get configurations, Get configuration options, Audio output configuration, Modify a configuration.

Feature Under Test: GetAudioOutputConfigurationOptions, GetAudioOutputConfigurations, SetAudioOutputConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify whether all supported Audio Output Configuration Options can be set.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters

- out *audioOutputConfList* - Audio Output Configurations list

4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ConfigurationChanged"** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
5. For each Audio Output Configuration *audioOutputConfiguration* in *audioOutputConfList* repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioOutputConfigurationOptions** request with parameters
      - ConfigurationToken := *audioOutputConfiguration*.@token
      - ProfileToken skipped

   5.2. DUT responds with **GetAudioOutputConfigurationOptionsResponse** message with parameters
      - Options =: *options*

   5.3. ONVIF Client invokes **SetAudioOutputConfiguration** request with parameters
      - Configuration.@token := *audioOutputConfiguration*.@token
      - Configuration.Name := "TestName1"
      - Configuration.OutputToken := first value from *options*.OutputTokensAvailable list
      - Configuration.SendPrimacy := if *options*.SendPrimacyOptions is specified, set first value from options.SendPrimacyOptions list, otherwise, set *audioOutputConfiguration*.SendPrimacy
      - Configuration.OutputLevel := *options*.OutputLevelRange.Min

   5.4. DUT responds with **SetAudioOutputConfigurationResponse** message.

   5.5. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged**

event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioOutputConfiguration*.@token - Configuration token
- in AudioOutput - Configuration Type

5.6. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters

- ConfigurationToken := *audioOutputConfiguration*.@token
- ProfileToken skipped

5.7. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters

- Configurations list =: *audioOutputConfList*

5.8. If *audioOutputConfList*[0] is not equal to Configuration from step 5.3, FAIL the test and skip other steps.

5.9. ONVIF Client invokes **SetAudioOutputConfiguration** request with parameters

- Configuration.@token := *audioOutputConfiguration*.@token
- Configuration.Name := "TestName2"
- Configuration.OutputToken := last value from *options*.OutputTokensAvailable list
- Configuration.SendPrimacy := if *options*.SendPrimacyOptions is specified, set last value from options.SendPrimacyOptions list, otherwise, set *audioOutputConfiguration*.SendPrimacy
- Configuration.OutputLevel := *options*.OutputLevelRange.Max

5.10. DUT responds with **SetAudioOutputConfigurationResponse** message.

5.11. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioOutputConfiguration*.@token - Configuration token
- in AudioOutput - Configuration Type

5.12. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters

- ConfigurationToken := *audioOutputConfiguration*.@token
- ProfileToken skipped

5.13. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters

- Configurations list =: *audioOutputConfList*

5.14. If *audioOutputConfList*[0] is not equal to Configuration from step 5.9, FAIL the test and skip other steps.

5.15. ONVIF Client restores settings of Audio Output Configuration with @token = *audioOutputConfiguration*.@token.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters

   - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationsResponse** message.
- DUT did not send **SetAudioOutputConfigurationResponse** message.
- DUT did not send **GetAudioOutputConfigurationOptionsResponse** message.

Note: The following fields are compared at steps 5.8 and 5.14:

- OutputToken
- Name
- SendPrimacy
- OutputLevel

5.3.3.6 GET AUDIO OUTPUT CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-3-3-6

Specification Coverage: Get configurations, Audio output configuration.

Feature Under Test: GetAudioOutputConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of **GetAudioOutputConfigurations** with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Output Configurations list by following the procedure mentioned in Annex A.10 with the following input and output parameters
   - out *audioOutputConfList* - Audio Output Configurations list
4. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
   - ConfigurationToken := other than listed in *audioOutputConfList*
   - ProfileToken skipped
5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message.

5.3.4 Audio Decoder Configuration

5.3.4.1 GET AUDIO DECODER CONFIGURATION OPTIONS

Test Case ID: MEDIA2-3-4-1

Specification Coverage: Get configuration options, Audio decoder configuration.

Feature Under Test: GetAudioDecoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving Audio Decoder Configuration options for specified Audio Decoder Configuration, for specified Profile, generic for the Device.

Pre-Requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioDecoderConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped

4. DUT responds with **GetAudioDecoderConfigurationOptionsResponse** message with parameters
   - Options list =: *optionsList*
5. ONVIF Client retrieves Audio Decoder Configurations list by following the procedure mentioned in Annex A.22 with the following input and output parameters
   - out *audioDecoderConfList* - Audio Decoder Configurations list
6. ONVIF Client invokes **GetAudioDecoderConfigurationOptions** request with parameters
   - ConfigurationToken := *audioDecoderConfList*[0].@token
   - ProfileToken skipped
7. DUT responds with **GetAudioDecoderConfigurationOptionsResponse** message with parameters
   - Options list =: *optionsList*
8. ONVIF Client configures Media Profile containing Audio Output Configuration and Audio Decoder Configuration by following the procedure mentioned in Annex A.23 with the following input and output parameters
   - out *profile* - Media Profile containing Audio Output Configuration and Audio Decoder Configuration
9. ONVIF Client invokes **GetAudioDecoderConfigurationOptions** request with parameters
   - ConfigurationToken skipped
   - ProfileToken := *profile*.@token
10. DUT responds with **GetAudioDecoderConfigurationOptionsResponse** message with parameters
    - Options list =: *optionsList*
11. If Media Profile *profile* was changed at step 8, ONVIF Client restores Media Profile.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioDecoderConfigurationOptionsResponse** message.

5.3.4.2 GET AUDIO DECODER CONFIGURATIONS

Test Case ID: MEDIA2-3-4-2

Specification Coverage: Get configurations, Audio decoder configuration.

Feature Under Test: GetAudioDecoderConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Audio Decoder Configuration List, Audio Decoder Configuration by Configuration token and compatible Audio Decoder Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Audio decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
4. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters
   - Configurations list =: *audioDecoderConfCompleteList*
5. If *audioDecoderConfCompleteList* is empty, FAIL the test and skip other steps.
6. If *audioDecoderConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
7. For each Audio Decoder Configuration *audioDecoderConfiguration* in *audioDecoderConfCompleteList* repeat the following steps:

   7.1. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters

- ConfigurationToken := *audioDecoderConfiguration*.@token
- ProfileToken skipped

7.2. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters

- Configurations list =: *audioDecoderConfList*

7.3. If *audioDecoderConfList* is empty, FAIL the test and skip other steps.

7.4. If *audioDecoderConfList* contains more than one item, FAIL the test and skip other steps.

7.5. If *audioDecoderConfList* does not contain item with @token = *audioDecoderConfiguration*.@token, FAIL the test and skip other steps.

8. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioDecoder
9. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
10. For each Media Profile *profile* in *profileList* repeat the following steps:

10.1. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken := *profile*.@token

10.2. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters

- Configurations list =: *audioDecoderConfList*

10.3. If *audioDecoderConfList* contains at least two items with the same @token, FAIL the test and skip other steps.

10.4. If *audioDecoderConfCompleteList* does not contain at least one item with @token from *audioDecoderConfList*, FAIL the test and skip other steps.

10.5. If *profile*.Configurations contains AudioDecoder:

10.5.1. If *audioDecoderConfList* does not contain item with @token = *profile*.Configurations.AudioDecoder.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioDecoderConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.3.4.3 PROFILES AND AUDIO DECODER CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-3-4-3

Specification Coverage: Get configurations, Get media profiles, Audio decoder configuration.

Feature Under Test: GetAudioDecoderConfigurations, GetAudioDecoderConfigurationOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Audio Decoder Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioDecoder

4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList* which contains Configurations.AudioDecoder repeat the following steps:
   5.1. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters **Note:** The following fields are compared at step 5.3:
      - ConfigurationToken := *profile*.Configurations.AudioDecoder.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters **Note:** The following fields are compared at step 5.3:
      - Configurations list =: *audioDecoderConfList*
   5.3. If *audioDecoderConfList*[0] is not equal to *profile*.Configurations.AudioDecoder, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioDecoderConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

Note: The following fields are compared at step 5.3:

- Name

5.3.4.4 MODIFY ALL SUPPORTED AUDIO DECODER CONFIGURATIONS

Test Case ID: MEDIA2-3-4-4

Specification Coverage: Get configurations, Audio decoder configuration, Modify a configuration

Feature Under Test: GetAudioDecoderConfigurations, SetAudioDecoderConfiguration

WSDL Reference: media2.wsdl

Test Purpose: To verify change of Audio Decoder Configuration.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Decoder Configurations list by following the procedure mentioned in Annex A.22 with the following input and output parameters
   - out *audioDecoderConfList* - Audio Decoder Configurations list
4. For each Audio Decoder Configuration *audioDecoderConfiguration* in *audioDecoderConfList* repeat the following steps:
   4.1. ONVIF Client invokes **SetAudioDecoderConfiguration** request with parameters
      - Configuration.@token := *audioDecoderConfiguration*.@token
      - Configuration.Name := "TestName1"

   4.2. DUT responds with **SetAudioDecoderConfigurationResponse** message.

   4.3. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters
      - ConfigurationToken := *audioDecoderConfiguration*.@token
      - ProfileToken := *audioDecoderConfiguration*

   4.4. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters
      - Configurations list =: *audioDecoderConfList*

   4.5. If *audioDecoderConfList*[0] is not equal to Configuration from step 4.1, FAIL the test and skip other steps.

   4.6. ONVIF Client restores settings of Audio Decoder Configuration with @token = *audioDecoderConfiguration*.@token.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioDecoderConfigurationsResponse** message.
- DUT did not send **SetAudioDecoderConfigurationResponse** message.

Note: The following fields are compared at step 4.5:

- Name

5.3.4.5 GET AUDIO DECODER CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-3-4-5

Specification Coverage: Get configurations, Audio decoder configuration.

Feature Under Test: GetAudioDecoderConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of **GetAudioDecoderConfigurations** with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Audio Decoder Configurations list by following the procedure mentioned in Annex A.22 with the following input and output parameters
   - out *audioDecoderConfList* - Audio Decoder Configurations list

4. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters
   - ConfigurationToken := other than listed in *audioDecoderConfList*
   - ProfileToken skipped
5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message.

5.4 PTZ Configuration

5.4.1 READY TO USE MEDIA PROFILE FOR PTZ

Test Case ID: MEDIA2-4-1-1

Specification Coverage: Absolute PTZ Move (Profile T) or Continuous PTZ Move (Profile T), Media profiles (Media 2), PTZ Configuration (Media 2)

Feature Under Test: GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT has a ready-to-use Media Service 2.0 Profile for PTZ.

Pre-Requisite: Media2 Service is received from the DUT. PTZ Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters

- Token skipped
- Type[0] := VideoSource
- Type[1] := PTZ

4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. If *profileList* contains no Media Profiles with both Configurations.VideoSource and Configurations.PTZ, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.

5.4.2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ

Test Case ID: MEDIA2-4-1-2

Specification Coverage: Get media profiles, Create media profile, Delete media profile, Add one or more configurations to a profile, Remove one or more configurations from a profile, Get configurations, GetConfigurations (PTZ Service), GetCompatibleConfigurations (PTZ Service).

Feature Under Test: GetProfiles, AddConfiguration for PTZ Configuration, RemoveConfiguration for PTZ Configuration, GetVideoSourceConfigurations, GetConfigurations, GetCompatibleConfigurations, SetConfiguration, tns1:Media/ProfileChanged, tns1:Media/ConfigurationChanged

WSDL Reference: media2.wsdl

Test Purpose: To verify the behavior of the DUT for dynamic media profile configuration with PTZ.

Pre-Requisite: Media2 Service is received from the DUT. Event Service was received from the DUT. PTZ Service was received from the DUT. GetCompatibleConfigurations is supported by Device as indicated by the GetCompatibleConfigurations = true capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := VideoSource
   - Type[1] := PTZ
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList*, which contains Configurations.VideoSource repeat the following steps:
   5.1. If *profile*.Configurations contains PTZ repeat the following steps:
      5.1.1. ONVIF Client invokes **RemoveConfiguration** request with parameters
         - ProfileToken := *profileToken*
         - Configuration[0].Type := PTZ
         - Configuration[0].Token skipped
      5.1.2. The DUT responds with **RemoveConfigurationResponse** message.
6. If DUT supports Pull-Point Notification feature and (Profile Changed Notification or Configuration Changed Notification feature), ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Media/ProfileChanged|tns1:Media/ConfigurationChanged"** - Notification Topic if both Profile Changed Notification and Configuration Changed Notification features are supported. Otherwise only supported topic is used as Notification Topic.
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT

- out *terminationTime* - Subscription termination time

7. For each Media Profile *profile* in *profileList*, which contains Configurations.VideoSource repeat the following steps:

7.1. ONVIF Client invokes **GetCompatibleConfigurations** request with parameters

- ProfileToken := *profileToken*

7.2. The DUT responds with **GetCompatibleConfigurationsResponse** request with parameters

- PTZConfiguration list =: *ptzConfigurationList*

7.3. For each PTZ Configuration *ptzConfiguration* in *ptzConfigurationList* repeat the following steps:

7.3.1. ONVIF Client invokes **AddConfiguration** request with parameters

- ProfileToken := *profileToken*
- Name skipped
- Configuration[0].Type := PTZ
- Configuration[0].Token := *ptzConfiguration*.@token

7.3.2. The DUT responds with **AddConfigurationResponse** message.

7.3.3. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

7.3.4. ONVIF Client invokes **GetConfigurationOptions** request with parameters

- ConfigurationToken := *ptzConfiguration*.@token

7.3.5. DUT responds with **GetConfigurationOptionsResponse** message with parameters

- PTZConfigurationOptions =: *ptzOptions*

7.3.6. If *ptzOptions*.Spaces contains at least two AbsolutePanTiltPositionSpace elements with equal URI values, FAIL the test and skip other steps.

7.3.7. If *ptzOptions*.Spaces contains at least two AbsoluteZoomPositionSpace elements with equal URI values, FAIL the test and skip other steps.

7.3.8. If *ptzOptions*.Spaces contains at least two RelativePanTiltTranslationSpace elements with equal URI values, FAIL the test and skip other steps.

7.3.9. If *ptzOptions*.Spaces contains at least two RelativeZoomTranslationSpace elements with equal URI values, FAIL the test and skip other steps.

7.3.10. If *ptzOptions*.Spaces contains at least two ContinuousPanTiltVelocitySpace elements with equal URI values, FAIL the test and skip other steps.

7.3.11. If *ptzOptions*.Spaces contains at least two ContinuousZoomVelocitySpace elements with equal URI values, FAIL the test and skip other steps.

7.3.12. ONVIF Client invokes **SetConfiguration** request with parameters

- PTZConfiguration.@token := *ptzConfiguration*.@token
- PTZConfiguration.Name := "TestNameN" (N is number that increases for each iteration of cycle beginning from 0)
- PTZConfiguration.UseCount := *ptzConfiguration*.UseCount
- PTZConfiguration.MoveRamp skipped
- PTZConfiguration.PresetRamp skipped
- PTZConfiguration.PresetTourRamp skipped
- PTZConfiguration.NodeToken := *ptzConfiguration*.NodeToken
- If *ptzOptions*.Spaces contains AbsolutePanTiltPositionSpace:
  - PTZConfiguration.DefaultAbsolutePantTiltPositionSpace := *ptzOptions*.Spaces.AbsolutePanTiltPositionSpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.AbsolutePanTiltPositionSpace list

otherwise, PTZConfiguration.DefaultAbsolutePantTiltPositionSpace skipped

- If *ptzOptions*.Spaces contains AbsoluteZoomPositionSpace:
  - PTZConfiguration.DefaultAbsoluteZoomPositionSpace := *ptzOptions*.Spaces.AbsoluteZoomPositionSpace.URI from first other then current (if possible) item form *ptzOptions*.Spaces.AbsoluteZoomPositionSpace list

  otherwise, PTZConfiguration.DefaultAbsoluteZoomPositionSpace skipped

- If *ptzOptions*.Spaces contains RelativePanTiltTranslationSpace:
  - PTZConfiguration.DefaultRelativePanTiltTranslationSpace := *ptzOptions*.Spaces.RelativePanTiltTranslationSpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.RelativePanTiltTranslationSpace list

  otherwise, PTZConfiguration.DefaultRelativePanTiltTranslationSpace skipped

- If *ptzOptions*.Spaces contains RelativeZoomTranslationSpace:
  - PTZConfiguration.DefaultRelativeZoomTranslationSpace := *ptzOptions*.Spaces.RelativeZoomTranslationSpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.RelativeZoomTranslationSpace list

  otherwise, PTZConfiguration.DefaultRelativeZoomTranslationSpace skipped

- If *ptzOptions*.Spaces contains ContinuousPanTiltVelocitySpace:
  - PTZConfiguration.DefaultContinuousPanTiltVelocitySpace := *ptzOptions*.Spaces.ContinuousPanTiltVelocitySpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.ContinuousPanTiltVelocitySpace list

  otherwise, PTZConfiguration.DefaultContinuousPanTiltVelocitySpace skipped

- If *ptzOptions*.Spaces contains ContinuousZoomVelocitySpace:

- PTZConfiguration.DefaultContinuousZoomVelocitySpace := *ptzOptions*.Spaces.ContinuousZoomVelocitySpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.ContinuousZoomVelocitySpace list

otherwise, PTZConfiguration.DefaultContinuousZoomVelocitySpace skipped

- If *ptzOptions*.Spaces contains PanTiltSpeedSpace or ZoomSpeedSpace:
  - If *ptzOptions*.Spaces contains PanTiltSpeedSpace:
    - PTZConfiguration.DefaultPTZSpeed.PanTilt.space := *ptzOptions*.Spaces.PanTiltSpeedSpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.PanTiltSpeedSpace
    - PTZConfiguration.DefaultPTZSpeed.PanTilt.x := *ptzOptions*.Spaces.PanTiltSpeedSpace.XRange.Max from the same item from *ptzOptions*.Spaces.PanTiltSpeedSpace as was used for PTZConfiguration.DefaultPTZSpeed.PanTilt.space
    - PTZConfiguration.DefaultPTZSpeed.PanTilt.y := *ptzOptions*.Spaces.PanTiltSpeedSpace.XRange.Min from the same item from *ptzOptions*.Spaces.PanTiltSpeedSpace as was used for PTZConfiguration.DefaultPTZSpeed.PanTilt.space

    otherwise, PTZConfiguration.DefaultPTZSpeed.PanTilt skipped
  - If *ptzOptions*.Spaces contains ZoomSpeedSpace:
    - PTZConfiguration.DefaultPTZSpeed.Zoom.space := *ptzOptions*.Spaces.ZoomSpeedSpace.URI from first other then current (if possible) item from *ptzOptions*.Spaces.ZoomSpeedSpace
    - PTZConfiguration.DefaultPTZSpeed.Zoom.x := *ptzOptions*.Spaces.ZoomSpeedSpace.XRange.Max from the same item from *ptzOptions*.Spaces.ZoomSpeedSpace as was used for PTZConfiguration.DefaultPTZSpeed.Zoom.space

    otherwise, PTZConfiguration.DefaultPTZSpeed.Zoom skipped

  otherwise, PTZConfiguration.DefaultPTZSpeed skipped
- PTZConfiguration.DefaultPTZTimeout =: PTZTimeout.Min

- PanTiltLimits skipped
- ZoomLimits skipped
- PTZConfiguration.PTControlDirection skipped

7.3.13. DUT responds with **SetConfigurationResponse** message.

7.3.14. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *ptzConfiguration*.@token - Configuration token
- in PTZ - Configuration Type

7.3.15. ONVIF Client invokes **GetConfiguration** request with parameters

- PTZConfigurationToken := *ptzConfiguration*.@token

7.3.16. The DUT responds with **GetConfigurationResponse** with parameters

- PTZConfiguration =: *ptzConfiguration*

7.3.17. If *ptzConfiguration* is not equal to PTZConfiguration from step 6.3.6, FAIL the test and skip other steps.

7.3.18. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := All

7.3.19. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

7.3.20. If *profileList* is empty, FAIL the test and skip other steps.

7.3.21. If *profileList* contains more than one item, FAIL the test and skip other steps.

7.3.22. If *profileList*[0].@token != *profileToken*, FAIL the test and skip other steps.

7.3.23. If *profileList*[0].Configurations.PTZ.@token != *ptzConfiguration*.@token, FAIL the test and skip other steps.

7.3.24. If *profileList*[0].Configurations.PTZ is not equal to PTZConfiguration from step 6.3.6, FAIL the test and skip other steps.

7.3.25. ONVIF Client invokes **RemoveConfiguration** request with parameters

- ProfileToken := *profileToken*
- Configuration[0].Type := PTZ
- Configuration[0].Token skipped

7.3.26. The DUT responds with **RemoveConfigurationResponse** message.

7.3.27. If DUT supports Pull-Point Notification feature and Profile Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *profileToken* - Media Profile token

7.3.28. ONVIF Client invokes **GetProfiles** request with parameters

- Token := *profileToken*
- Type[0] := PTZ

7.3.29. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

7.3.30. If *profileList* is empty, FAIL the test and skip other steps.

7.3.31. If *profileList* contains more than one item, FAIL the test and skip other steps.

7.3.32. If *profileList*[0].Configurations contains PTZ, FAIL the test and skip other steps.

8. ONVIF Client restores Media Profiles and PTZ Configurations that were changed.
9. If subscription was created at step 6, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters
   - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.
- DUT did not send **GetCompatibleConfigurationsResponse** message.
- DUT did not send **SetConfigurationResponse** message.

Note: *timeout1* will be taken from Operation Delay field of ONVIF Device Test Tool.

Note: The following fields are compared at steps 6.3.18 and 6.3.11:

- Name
- NodeToken
- DefaultAbsolutePantTiltPositionSpace
- DefaultAbsoluteZoomPositionSpace
- DefaultRelativePanTiltTranslationSpace
- DefaultRelativeZoomTranslationSpace
- DefaultContinuousPanTiltVelocitySpace
- DefaultContinuousZoomVelocitySpace
- DefaultPTZSpeed.PanTilt.x
- DefaultPTZSpeed.PanTilt.y

- DefaultPTZSpeed.PanTilt.space
- DefaultPTZSpeed.Zoom.x
- DefaultPTZSpeed.Zoom.space
- DefaultPTZTimeout

5.5 Media Streaming

5.5.1 SNAPSHOT URI

Test Case ID: MEDIA2-5-1-1

Specification Coverage: Get media profiles, Request snapshot URI

Feature Under Test: GetSnapshotUri

WSDL Reference: media2.wsdl

Test Purpose: To retrieve snapshot URI of DUT for given media profile

Pre-Requisite: SnapshotUri feature for Media2 service is supported by the DUT. Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. If DUT supports Video feature
   - 3.1. ONVIF Client configures Media Profile containing Video Source Configuration and Video Encoder Configuration by following the procedure mentioned in Annex A.24 with the following input and output parameters
     - out *profile* - Media Profile containing Video Source Configuration and Video Encoder Configuration
4. If DUT does not support Video feature
   - 4.1. ONVIF Client configures Media Profile containing Video Source Configuration by following the procedure mentioned in Annex A.12 with the following input and output parameters

- out *profile* - Media Profile containing Video Source Configuration

5. ONVIF Client invokes **GetSnapshotUri** request with parameters
   - ProfileToken := *profile*.@token
6. DUT responds with **GetSnapshotUriResponse** message with parameters
   - Uri =: *snapshotUri*
7. ONVIF Client invokes **HTTP GET** request on the *snapshotUri*.
8. DUT responds with **HTTP 200 OK** message and the single shot JPEG image data.
9. ONVIF Client verifies the JPEG image sent by the DUT.
10. If media profile profile was changed at step 3, ONVIF Client restores media profile.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetSnapshotUriResponse** message.
- DUT did not send **200 OK** message to **HTTP GET** request.
- DUT did not send valid JPEG image data.

5.5.2 VIDEO ENCODER INSTANCES PER VIDEO SOURCE

Test Case ID: MEDIA2-5-1-2

Specification Coverage: GetVideoEncoderInstances

Feature Under Test: GetVideoEncoderInstances

WSDL Reference: media2.wsdl, deviceio.wsdl

Test Purpose: To verify that for each video source there is at least one video source configuration for which the GetVideoEncoderInstances returns a Total greater than 0.

Pre-Requisite: Media2 Service is received from the DUT. Device IO Service is received from the DUT. Video feature is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
4. ONVIF Client retrieves Video Sources list by following the procedure mentioned in Annex A.18 with the following input and output parameters
   - out *videoSourcesList1* - Video Sources list
5. For each Video Source *videoSource1* from *videoSourcesList* repeat the following steps:
   5.1. Set *totalGreaterThanZeroFlag* := false.
   5.2. For each Video Source Configuration *videoSourceConfig1* with SourceToken = *videoSource1* from *videoSourceConfList1* repeat the following steps:
      5.2.1. ONVIF Client invokes **GetVideoEncoderInstances** request with parameters
         - ConfigurationToken := *videoSourceConfig1*.token
      5.2.2. DUT responds with **GetVideoEncoderInstancesResponse** message with parameters
         - Info = *info1*
      5.2.3. If *info1*.Total > 0:
         5.2.3.1. Set *totalGreaterThanZeroFlag* := true.
         5.2.3.2. Go to step 5.3.
   5.3. If *totalGreaterThanZeroFlag* = false, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourcesResponse** message.

- DUT did not send **GetVideoEncoderInstancesResponse** message.

5.6 OSD Configuration

5.6.1 CREATE OSD CONFIGURATION FOR TEXT OVERLAY

Test Case ID: MEDIA2-6-1-1

Specification Coverage: None

Feature Under Test: GetVideoSourceConfigurations, GetOSDs, GetOSDOptions, CreateOSD, DeleteOSD

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT creates OSD Configuration.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSourceConfigurations** message to retrieve a list of existing Video Source Configurations on the DUT.
4. Verify that the DUT returns at least one Video Source Configuration in the **GetVideoSourceConfigurationsResponse** message. For each Video Source Configuration from **GetVideoSourceConfigurationsResponse**, ONVIF Client saves the token of this configuration in *VideoSourceConfigurationToken1* variable and runs the following steps:

   4.1. ONVIF Client invokes **GetOSDs** request with parameters

   - OSDToken - skipped
   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.2. The DUT responds with **GetOSDsResponse** with parameters

   - OSDs list =: *OSDConfigurationsList1*

   4.3. ONVIF Client invokes **GetOSDOptions** request with parameters

- ConfigurationToken := *VideoSourceConfigurationToken1*

4.4. DUT responds with **GetOSDOptionsResponse** message with parameters

- OSDOptions =: *OSDOptions1*

4.5. ONVIF Client checks if current Video Source Configuration supports OSD configurations of the Type = Text. Otherwise, skip steps 4.6-4.25 and go to the next Video Source Configuration.

4.6. Set *osdTextStringType1* := *OSDOptions1*.TextOption.Type[0], where Type[0] is the 1st in Type list from {Plain, Date, Time, DateAndTime} values

4.7. ONVIF Client configures Device to have free space to create OSD with Text type and with required TextString Type by following the procedure mentioned in Annex A.25 with the following input and output parameters

- in *OSDConfigurationsList1* - OSD Configurations List
- in *osdOptions* - OSD Options
- in "Text" - OSD Type
- out *OSDConfigurationsList1* - updated OSD Configurations List

4.8. ONVIF Client invokes **CreateOSD** with token = "testOSD" (note: this token can be ignored by DUT), VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Text" and the rest of parameter are populated using values from *OSDOptions1*. For this step, if the value has a range of available values, then minimum value (or the first value from list if list of available values is provided by DUT) from *OSDOptions1* should be used. See details of fields mapping in Annex A.28.

4.9. DUT creates OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to OSDConfigurationToken1 variable.

4.10. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken1*.

4.11. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.12. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken1* and configuration fields equal to the fields set at 4.8 step.

4.13. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken1* as input parameter.

4.14. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.15. Set *osdTextStringType2* := *OSDOptions1*.TextOption.Type[0], where Type[0] is the last in Type list from {Plain, Date, Time, DateAndTime} values

4.16. ONVIF Client invokes **CreateOSD** with token = "testOSD", VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Text" and the rest of parameters are populated using values from *OSDOptions1*. For this step, if the value has a range of available values, then maximum value (or the last value from list if list of available values is provided by DUT) from *OSDOptions1* should be used. See details of fields mapping in Annex A.28.

4.17. DUT created OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to *OSDConfigurationToken2* variable.

4.18. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken2*.

4.19. ONVIF Client invokes **GetOSDs** message with configuration token = *OSDConfigurationsList1* as input parameter to retrieve the list of OSD Configurations, which includes newly created OSD Configuration.

4.20. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken2* and the fields of this configuration equal to the fields set at step 4.15.

4.21. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken2* as input parameter.

4.22. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.23. ONVIF Client invokes **GetOSDs** message with configuration token = *VideoSourceConfigurationToken1* as input parameter to retrieve the list of OSD Configurations, which includes newly created OSD Configuration.

4.24. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client verifies that response does not contain OSD Configuration with token = *OSDConfigurationToken2*.

4.25. If OSD Configuration was deleted at 4.7 [159] step, ONVIF Client restores OSD Configuration.

Test Result:

PASS –

- DUT passes all assertions.
- DUT did not send at least one text overlay (Text Option is skipped) in **GetOSDOptionsResponse**

FAIL –

- The DUT did not send **GetVideoSourceConfigurationsResponse** message.
- The **GetVideoSourceConfigurationsResponse** does not contain at least one Video Source configuration.
- The DUT did not send **GetOSDsResponse** message.
- The DUT did not send **GetOSDOptionsResponse** message.
- The DUT did not send **CreateOSDResponse** message.
- The DUT did not send **GetOSDResponse** message.
- The DUT did not send **DeleteOSDResponse** message.
- The **GetOSDsResponse** contains deleted OSD Configuration

5.6.2 CREATE OSD CONFIGURATION FOR IMAGE OVERLAY

Test Case ID: MEDIA2-6-1-2

Specification Coverage: None

Feature Under Test: GetVideoSourceConfigurations, GetOSDs, GetOSDOptions, CreateOSD, DeleteOSD

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT creates OSD Configuration.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSourceConfigurations** message to retrieve a list of existing Video Source Configurations on the DUT.
4. Verify that the DUT returns at least one Video Source Configuration in the **GetVideoSourceConfigurationsResponse** message. For each Video Source Configuration from **GetVideoSourceConfigurationsResponse**, ONVIF Client saves the token of this configuration in VideoSourceConfigurationToken1 variable and runs steps 4.1 - 4.24:

   4.1. ONVIF Client invokes **GetOSDs** request with parameters

   - OSDToken - skipped
   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.2. The DUT responds with **GetOSDsResponse** with parameters

   - OSDs list =: *OSDConfigurationsList1*

   4.3. ONVIF Client invokes **GetOSDOptions** request with parameters

   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.4. DUT responds with **GetOSDOptionsResponse** message with parameters

   - OSDOptions =: *OSDOptions1*

   4.5. ONVIF Client checks if current Video Source Configuration supports OSD configurations of the Type = Image. Otherwise, skip steps 4.6 – 4.24 and go to the next Video Source Configuration.

4.6. ONVIF Client configures Device to have free space to create OSD with Image type by following the procedure mentioned in Annex A.25 with the following input and output parameters

- in *OSDConfigurationsList1* - OSD Configurations List
- in *osdOptions* - OSD Options
- in "Image" - OSD Type
- out *OSDConfigurationsList1* - updated OSD Configurations List

4.7. If *osdOptions*.ImageOption contains FormatsSupported attribute:

4.7.1. Set *imagePath1* := *osdOptions*.ImageOption.ImagePath[0]

4.7.2. ONVIF client invokes **HTTP POST** to *imagePath1* with parameters

- HTTP Header [Content-Type] := "image/png"
- HTTP Body := PNG picture (see Annex A.29 for details)

4.7.3. The DUT responds with **HTTP 200 OK** message.

4.7.4. ONVIF Client waits *timeout1*.

4.7.5. If *osdOptions*.ImageOption contains more than one ImagePath:

4.7.5.1. Set *imagePath2* := *osdOptions*.ImageOption.ImagePath[last]

4.7.5.2. ONVIF client invokes **HTTP POST** to *imagePath2* with parameters

- HTTP Header [Content-Type] := "image/png"
- HTTP Body := PNG picture (see Annex A.29 for details)

4.7.5.3. The DUT responds with **HTTP 200 OK** message.

4.7.5.4. ONVIF Client waits *timeout1*.

4.8. ONVIF Client invokes **CreateOSD** with token = "testOSD", VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Image", Position. Type should be set to the first item from *OSDOptions1*.PositionOption list and Image.ImgPath should be set to the first item from *OSDOptions1*.ImageOption.ImagePath list.

4.9. DUT creates OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to OSDConfigurationToken1 variable.

4.10. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken1*.

4.11. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.12. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken1* and configuration fields equal to the fields set at step in 4.6.

4.13. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken1* as input parameter.

4.14. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.15. ONVIF Client invokes **CreateOSD** with token = "testOSD", VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Image", Position. Type should be set to the last item from *OSDOptions1*.PositionOption list and Image.ImgPath should be set to the last item from *OSDOptions1*.ImageOption.ImagePath list.

4.16. DUT created OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to *OSDConfigurationToken2* variable.

4.17. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken2*.

4.18. ONVIF Client invokes **GetOSDs** message with configuration token = *OSDConfigurationToken2* as input parameter to retrieve the list of OSD Configurations, which includes newly created OSD Configuration.

4.19. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken2* and configuration fields equal to the fields set at step in 4.16.

4.20. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken2* as input parameter.

4.21. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.22. ONVIF Client invokes **GetOSDs** message with configuration token = *VideoSourceConfigurationToken1* as input parameter to retrieve the list of OSD Configurations, which includes newly created OSD Configuration.

4.23. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client verifies that response does not contain OSD Configuration with token = *OSDConfigurationToken2*.

4.24. If OSD Configuration was deleted at 4.6 [163] step, ONVIF Client restores OSD Configuration.

Test Result:

PASS –

- DUT passes all assertions.
- DUT did not send at least one image overlay (Image Option is skipped) in **GetOSDOptionsResponse**

FAIL –

- The DUT did not send **GetVideoSourceConfigurationsResponse** message.
- The **GetVideoSourceConfigurationsResponse** does not contain at least one Video Source configuration.
- The DUT did not send **GetOSDsResponse** message.
- The DUT did not send **GetOSDOptionsResponse** message.
- The DUT did not send **CreateOSDResponse** message.
- The DUT did not send **GetOSDResponse** message.
- The DUT did not send **DeleteOSDResponse** message.
- The **GetOSDsResponse** contains deleted OSD Configuration.

5.6.3 SET OSD CONFIGURATION IMAGE OVERLAY

Test Case ID: MEDIA2-6-1-3

Specification Coverage: None

Feature Under Test: GetVideoSourceConfigurations, GetOSDs, GetOSDOptions, CreateOSD, DeleteOSD, SetOSD

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT changes OSD Configuration for Image Overlay.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSourceConfigurations** message to retrieve a list of existing Video Source Configurations on the DUT.
4. Verify that the DUT returns at least one Video Source Configuration in the **GetVideoSourceConfigurationsResponse** message. For each Video Source Configuration from **GetVideoSourceConfigurationsResponse**, ONVIF Client saves this configuration in VideoSourceConfigurationToken1 variable and runs the following steps:

   4.1. ONVIF Client invokes **GetOSDOptions** request with parameters

   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.2. DUT responds with **GetOSDOptionsResponse** message with parameters

   - OSDOptions =: *osdOptions1*

   4.3. If **GetOSDOptionsResponse** contains at least one item in Image Option (image overlay), then ONVIF Client saves the options from **GetOSDOptionsResponse** in *OSDOptions1* variable and goes to step 4.5.

   4.4. If *OSDOptions1* Image Option list is empty, then PASS the test.

   4.5. ONVIF Client invokes **GetOSDs** request with parameters

   - OSDToken - skipped
   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.6. The DUT responds with **GetOSDsResponse** with parameters

   - OSDs list =: *OSDConfigurationsList1*

4.7. ONVIF Client configures Device to have free space to create OSD with Image type and with required TextString Type by following the procedure mentioned in Annex A.25 with the following input and output parameters

- in *OSDConfigurationsList1* - OSD Configurations List
- in *osdOptions* - OSD Options
- in "Image" - OSD Type
- out *OSDConfigurationsList1* - updated OSD Configurations List

4.8. If *osdOptions*.ImageOption contains FormatsSupported attribute:

4.8.1. Set *imagePath1* := *osdOptions*.ImageOption.ImagePath[0]

4.8.2. ONVIF client invokes **HTTP POST** to *imagePath1* with parameters

- HTTP Header [Content-Type] := "image/png"
- HTTP Body := PNG picture (see Annex A.29 for details)

4.8.3. The DUT responds with **HTTP 200 OK** message.

4.8.4. ONVIF Client waits *timeout1*.

4.9. ONVIF Client invokes **CreateOSD** with token = "testOSD", VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Image", Position.Type should be set to the first item from *OSDOptions1*.PositionOption list and Image.ImgPath should be set to the first item from *OSDOptions1*.ImageOption.ImagePath list.

4.10. DUT creates OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to OSDConfigurationToken1 variable.

4.11. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.12. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken1* and saves this configuration in OSDConfigurations1 variable.

4.13. ONVIF Client changes position parameter in OSDConfigurations1 variable. The Position.Type should be changed to the last item from *OSDOptions1*.PositionOption list.

4.14. ONVIF Client invokes **SetOSD** with OSD = OSDConfigurations1 as input parameter. DUT applies the changes and sends **SetOSDResponse**.

4.15. ONVIF Client invokes **GetOSDs** message with OSD Token = OSDConfigurations1 token as input parameter.

4.16. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = OSDConfigurations1 token and Position.Type is set to the value, which has been set at step 4.11.

4.17. If new OSD Configuration has been created at step 4.5, then ONVIF Client invokes **DeleteOSD** with OSD Token = OSDConfigurations1 token as input parameter.

4.18. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.19. If OSD Configuration was deleted at 4.7 [167] step, ONVIF Client restores OSD Configuration.

Test Result:

PASS –

- DUT passes all assertions.
- DUT did not send at least one image overlay (Image Option list is empty) in **GetOSDOptionsResponse**

FAIL –

- The DUT did not send **GetVideoSourceConfigurationsResponse** message.
- The **GetVideoSourceConfigurationsResponse** does not contain at least one Video Source configuration.
- The DUT did not send **GetOSDsResponse** message.
- The DUT did not send **GetOSDOptionsResponse** message.
- The DUT did not send **CreateOSDResponse** message.
- The DUT rejected create OSD Request.
- The DUT did not send **GetOSDResponse** message.

- The DUT did not send **DeleteOSDResponse** message.
- The DUT did not send **DeleteOSDResponse** message.

5.6.4 SET OSD CONFIGURATION TEXT OVERLAY

Test Case ID: MEDIA2-6-1-4

Specification Coverage: None

Feature Under Test: GetVideoSourceConfigurations, GetOSDs, GetOSDOptions, CreateOSD, DeleteOSD, SetOSD

WSDL Reference: media2.wsdl

Test Purpose: To verify that DUT changes OSD Configuration for Test Overlay.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetVideoSourceConfigurations** message to retrieve a list of existing Video Source Configurations on the DUT.
4. Verify that the DUT returns at least one Video Source Configuration in the **GetVideoSourceConfigurationsResponse** message. For each Video Source Configuration from **GetVideoSourceConfigurationsResponse**, ONVIF Client saves this configuration in VideoSourceConfigurationToken1 variable and runs steps 4.1 - 4.18:

   4.1. ONVIF Client invokes **GetOSDOptions** request with parameters

   - ConfigurationToken := *VideoSourceConfigurationToken1*

   4.2. DUT responds with **GetOSDOptionsResponse** message with parameters

   - OSDOptions =: *OSDOptions1*

   4.3. ONVIF Client invokes **GetOSDs** request with parameters

   - OSDToken - skipped
   - ConfigurationToken := *VideoSourceConfigurationToken1*

4.4. The DUT responds with **GetOSDsResponse** with parameters

- OSDs list =: *OSDConfigurationsList1*

4.5. Onvif client checks if DUT supports OSD configurations of the type=text. Otherwise, skip all steps and pass the test.

4.6. Set *osdTextStringType* := *OSDOptions1*.TextOption.Type[0], where Type[0] is the 1st in Type list from {Plain, Date, Time, DateAndTime} values

4.7. ONVIF Client configures Device to have free space to create OSD with Text type and with required TextString Type by following the procedure mentioned in Annex A.25 with the following input and output parameters

- in *OSDConfigurationsList1* - OSD Configurations List
- in *osdOptions* - OSD Options
- in "Text" - OSD Type
- out *OSDConfigurationsList1* - updated OSD Configurations List

4.8. ONVIF Client invokes **CreateOSD** with token = "testOSD", VideoSourceConfigurationToken = *VideoSourceConfigurationToken1*, Type="Text" and the rest of parameters are populated using values from *OSDOptions1*. For this step, if the value has a range of available values, then minimum value (or the first value from list if list of available values is provided by DUT) from *OSDOptions1* should be used. See details of fields mapping in Annex A.28.

4.9. DUT creates OSD Configuration and sends **CreateOSDResponse** message. ONVIF Client receives **CreateOSDResponse** with token of newly created OSD Configuration and saves this token to OSDConfigurationToken1 variable.

4.10. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.11. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurationToken1* and saves this configuration in *OSDConfigurations1* variable.

4.12. ONVIF Client changes the fields value of *OSDConfigurations1* variable to the maximum available values. If the value has a range of available values, then maximum value (or the last value from list if list of available values is provided by DUT) from

OSDOptions1 should be used. OSD.TextString.Type shall not be changed. See details of fields mapping in Annex A.28.

4.13. ONVIF Client invokes **SetOSD** with OSD = *OSDConfigurations1* as input parameter. DUT applies the changes and sends **SetOSDResponse**.

4.14. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurations1* token as input parameter.

4.15. DUT sends OSD Configuration for the given token. ONVIF Client verifies that response contains OSD Configuration with token = *OSDConfigurations1* token and configuration fields equal to the fields set at step 4.9.

4.16. If new OSD Configuration has been created at step 4.5, then ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurations1* token as input parameter.

4.17. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.18. If OSD Configuration was deleted at 4.7 [170] step, ONVIF Client restores OSD Configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetVideoSourceConfigurationsResponse** message.
- The **GetVideoSourceConfigurationsResponse** does not contain at least one Video Source configuration.
- The DUT did not send **GetOSDsResponse** message.
- The DUT did not send **GetOSDOptionsResponse** message.
- The DUT did not send **CreateOSDResponse** message.
- The DUT rejected create OSD Request.
- The DUT did not send **GetOSDResponse** message.
- The DUT did not send **SetOSDResponse** message.

- The DUT did not send **DeleteOSDResponse** message.

5.6.5 GET OSDS

Test Case ID: MEDIA2-6-1-5

Specification coverage: GetOSDs (Media 2 Service Specification), Get configurations (Media 2 Service Specification)

Feature under test: GetOSDs

WSDL Reference: media2.wsdl

Test Purpose: To verify DUT sends complete OSDs list and list of OSDs, which are compatible with specific Video Source Configuration.

Pre-Requisite: Media2 Service is received from the DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetOSDs** request with parameters
   - OSDToken - skipped
   - ConfigurationToken - skipped
4. The DUT responds with **GetOSDsResponse** with parameters
   - OSDs list =: *osdConfCompleteList1*
5. If *osdConfCompleteList1* contains at least two items with the same @token, FAIL the test and skip other steps.
6. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
7. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:

7.1. ONVIF Client invokes **GetOSDs** request with parameters

- OSDToken - skipped
- ConfigurationToken := *videoSourceConf1*.@token

7.2. The DUT responds with **GetOSDsResponse** with parameters

- OSDs list =: *osdConfList1*

7.3. If *osdConfList1* contains at least two items with the same @token, FAIL the test and skip other steps.

7.4. If *osdConfCompleteList1* does not contain at least one item from *osdConfList1* list, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetOSDsResponse** message(s).

5.6.6 GET OSD OPTIONS

Test Case ID: MEDIA2-6-1-6

Specification Coverage: GetOSDOptions

Feature Under Test: GetOSDOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify consistency in OSD Options.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
4. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:
   4.1. ONVIF Client invokes **GetOSDOptions** request with parameters
      - ConfigurationToken := *videoSourceConfiguration*.@token
   4.2. DUT responds with **GetOSDOptionsResponse** message with parameters
      - OSDOptions =: *osdOptions*
   4.3. If *osdOptions*.MaximumNumberOfOSDs.Total > 0:
      4.3.1. If *osdOptions* has Type = "Text" and does not have TextOption element, FAIL the test and skip other steps.
      4.3.2. If *osdOptions* has TextOption element and does not have Type = "Text" , FAIL the test and skip other steps.
      4.3.3. If *osdOptions* has Type = "Image" and does not have ImageOption element, FAIL the test and skip other steps.
      4.3.4. If *osdOptions* has ImageOption element and does not have Type = "Image" , FAIL the test and skip other steps.
      4.3.5. If *osdOptions*.MaximumNumberOfOSDs has @PlainText with value > 0 and *osdOptions* does not have Type = "Text", FAIL the test and skip other steps.
      4.3.6. If *osdOptions*.MaximumNumberOfOSDs has @Image with value > 0 and *osdOptions* does not have Type = "Image", FAIL the test and skip other steps.
      4.3.7. If *osdOptions*.ImageOption contains FormatsSupported attribute:
         4.3.7.1. If *osdOptions*.ImageOption.FormatsSupported does not contain "image/png" item, FAIL the test and skip other steps.
         4.3.7.2. If *osdOptions*.ImageOption does not contain MaxSize attribute or it does not contain both MaxWidth attribute and MaxHeight attribute, FAIL the test and skip other steps.

4.3.7.3. If *osdOptions*.ImageOption contains MaxSize attribute with value less than 1024, FAIL the test and skip other steps.

4.3.7.4. If *osdOptions*.ImageOption contains MaxWidth attribute with value less than 16, FAIL the test and skip other steps.

4.3.7.5. If *osdOptions*.ImageOption contains MaxHeight attribute with value less than 16, FAIL the test and skip other steps.

4.3.8. If *osdOptions*.TextOption has Type = "Date" and does not have at least one DateFormat element, FAIL the test and skip other steps.

4.3.9. If *osdOptions*.TextOption has Type = "Time" and does not have at least one TimeFormat element, FAIL the test and skip other steps.

4.3.10. If *osdOptions*.TextOption has Type = "DateAndTime" and does not have at least one DateFormat element and at least one TimeFormat element, FAIL the test and skip other steps.

5. If there was no at least one *osdOptions* with MaximumNumberOfOSDs.Total > 0 at step 4.3 [174], FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetOSDOptionsResponse** message(s).

5.6.7 OSD CONFIGURATIONS AND OSD OPTIONS CONSISTENCY

Test Case ID: MEDIA2-6-1-7

Specification Coverage: GetOSDOptions, GetOSDs

Feature Under Test: GetOSDOptions, GetOSDs

WSDL Reference: media2.wsdl

Test Purpose: To verify all OSD configurations are consistent with OSD Options.

Pre-Requisite: Media2 Service feature is supported by DUT. OSD feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
4. For each Video Source Configuration *videoSourceConfiguration* in *videoSourceConfList* repeat the following steps:
   4.1. ONVIF Client invokes **GetOSDs** request with parameters
      - OSDToken skipped
      - ConfigurationToken := *videoSourceConfiguration*.@token
   4.2. DUT responds with **GetOSDsResponse** message with parameters
      - OSDs =: *osdList*
   4.3. If *osdList* is empty, skip other steps.
   4.4. ONVIF Client invokes **GetOSDOptions** request with parameters
      - ConfigurationToken := *videoSourceConfiguration*.@token
   4.5. DUT responds with **GetOSDOptionsResponse** message with parameters
      - OSDOptions =: *osdOptions*
   4.6. For each OSD *osd* in *osdList* repeat the following steps:
      4.6.1. If *osd*.VideoSourceConfigurationToken is not equal to *videoSourceConfiguration*.@token, FAIL the test and skip other steps.
      4.6.2. If *osdOptions*.MaximumNumberOfOSDs.Total is not greater than 0, FAIL the test and skip other steps.
      4.6.3. If *osdOptions* does not contain Type element with value is equal to *osd*.Type, FAIL the test and skip other steps.

4.6.4. If *osdOptions* does not contain PositionOption element with value is equal to *osd*.Position.Type, FAIL the test and skip other steps.

4.6.5. If *osd*.TextString specified:

4.6.5.1. If *osdOptions* does not contain TextOption element with Type element with value is equal to *osd*.TextString.Type, FAIL the test and skip other steps.

4.6.5.2. Set *osdOptions*.TextOption =: *textOption1*.

4.6.6. If *osd*.TextString.DateFormat specified:

4.6.6.1. If *textOption1* does not contain at least one DateFormat element with value is equal to *osd*.TextString.DateFormat, FAIL the test and skip other steps.

4.6.7. If *osd*.TextString.TimeFormat specified:

4.6.7.1. If *textOption1* does not contain at least one TimeFormat element with value is equal to *osd*.TextString.TimeFormat, FAIL the test and skip other steps.

4.6.8. If *osd*.TextString.FontSize specified:

4.6.8.1. If *textOption1*.FontSizeRange.Min > *osd*.TextString.FontSize, FAIL the test and skip other steps.

4.6.8.2. If *textOption1*.FontSizeRange.Max < *osd*.TextString.FontSize, FAIL the test and skip other steps.

4.6.9. If *osd*.TextString.FontColor specified:

4.6.9.1. If *textOption1* does not contain FontColor.Color element, FAIL the test and skip other steps.

4.6.9.2. If *textOption1*.FontColor.Color has at least one ColorList element:

4.6.9.2.1. If *textOption1*.FontColor does not contain ColorList element with @X = *osd*.TextString.FontColor.Color.@X, and with @Y = *osd*.TextString.FontColor.Color.@Y, and with @Z = *osd*.TextString.FontColor.Color.@Z, and with @Colorspace =

osd.TextString.FontColor.Color.@Colorspace, FAIL the test and skip other steps.

4.6.9.3. If *textOption1*.FontColor.Color has at least one ColorspaceRange element:

4.6.9.3.1. If *osd*.TextString.FontColor.Color does not have @Colorspace element, FAIL the test and skip other steps.

4.6.9.3.2. Set *textOption1*.FontColor.Color.ColorspaceRange =: *colorspaceRange1*, where *textOption1*.FontColor.Color.ColorspaceRange is the first ColorspaceRange element that corresponds the following requirements:

- *colorspaceRange1*.@Colorspace = *osd*.TextString.FontColor.Color.@Colorspace.
- *colorspaceRange1*.X.Min <= *osd*.TextString.FontColor.Color.@X
- *colorspaceRange1*.X.Max >= *osd*.TextString.FontColor.Color.@X
- *colorspaceRange1*.Y.Min <= *osd*.TextString.FontColor.Color.@Y
- *colorspaceRange1*.Y.Max >= *osd*.TextString.FontColor.Color.@Y
- *colorspaceRange1*.Z.Min <= *osd*.TextString.FontColor.Color.@Z
- *colorspaceRange1*.Z.Max >= *osd*.TextString.FontColor.Color.@Z

4.6.9.3.3. If *colorspaceRange1* is empty, FAIL the test and skip other steps.

4.6.10. If *osd*.TextString.FontColor.@Transparent specified:

4.6.10.1. If *textOption1* does not contain FontColor.Transparent element, FAIL the test and skip other steps.

4.6.10.2. If *textOption1*.FontColor.Transparent.Min > *osd*.TextString.FontColor.@Transparent, FAIL the test and skip other steps.

4.6.10.3. If *textOption1*.FontColor.Transparent.Max < *osd*.TextString.FontColor.@Transparent, FAIL the test and skip other steps.

4.6.11. If *osd*.TextString.BackgroundColor specified:

4.6.11.1. If *textOption1* does not contain BackgroundColor.Color element, FAIL the test and skip other steps.

4.6.12. If *textOption1*.BackgroundColor.Color has at least one ColorList element:

4.6.12.1. If *textOption1*.BackgroundColor does not contain ColorList element with @X = *osd*.TextString.BackgroundColor.Color.@X, and with @Y = *osd*.TextString.BackgroundColor.Color.@Y, and with @Z = *osd*.TextString.BackgroundColor.Color.@Z, and with @Colorspace = *osd*.TextString.BackgroundColor.Color.@Colorspace, FAIL the test and skip other steps.

4.6.13. If *textOption1*.BackgroundColor.Color has at least one ColorspaceRange element:

4.6.13.1. If *osd*.TextString.BackgroundColor.Color does not have @Colorspace element, FAIL the test and skip other steps.

4.6.13.2. Set *textOption1*.BackgroundColor.Color.ColorspaceRange =: *colorspaceRange1*, where *textOption1*.BackgroundColor.Color.ColorspaceRange is the first ColorspaceRange element that corresponds the following requirements:

- *colorspaceRange1*.@Colorspace = *osd*.TextString.BackgroundColor.Color.@Colorspace.
- *colorspaceRange1*.X.Min <= *osd*.TextString.BackgroundColor.Color.@X
- *colorspaceRange1*.X.Max >= *osd*.TextString.BackgroundColor.Color.@X

- *colorspaceRange1*.Y.Min <= *osd*.TextString.BackgroundColor.Color.@Y
- *colorspaceRange1*.Y.Max >= *osd*.TextString.BackgroundColor.Color.@Y
- *colorspaceRange1*.Z.Min <= *osd*.TextString.BackgroundColor.Color.@Z
- *colorspaceRange1*.Z.Max >= *osd*.TextString.BackgroundColor.Color.@Z

4.6.13.3. If *colorspaceRange1* is empty, FAIL the test and skip other steps.

4.6.14. If *osd*.TextString.BackgroundColor.@Transparent specified:

4.6.14.1. If *textOption1* does not contain BackgroundColor.Transparent element, FAIL the test and skip other steps.

4.6.14.2. If *textOption1*.BackgroundColor.Transparent.Min > *osd*.TextString.BackgroundColor.@Transparent, FAIL the test and skip other steps.

4.6.14.3. If *textOption1*.BackgroundColor.Transparent.Max < *osd*.TextString.BackgroundColor.@Transparent, FAIL the test and skip other steps.

4.6.15. If *osd*.Image specified:

4.6.15.1. If *osdOptions* does not contain ImageOption element, FAIL the test and skip other steps.

4.6.15.2. If *osdOptions*.ImageOption does not contain ImagePath element with value is equal to *osd*.Image.ImgPath, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetOSDsResponse** message(s).

- The DUT did not send **GetOSDOptionsResponse** message(s).

5.7 Capabilities

5.7.1 MEDIA2 SERVICE CAPABILITIES

Test Case ID: MEDIA2-7-1-1

Specification Coverage: Capabilities (ONVIF Media2 Service Specification), GetServiceCapabilities command (ONVIF Media2 Service Specification)

Feature Under Test: GetServiceCapabilities (for Media2 Service)

WSDL Reference: media2.wsdl

Test Purpose: To verify DUT Media2 Service Capabilities.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServiceCapabilities**.
4. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*
5. If *cap*.ProfileCapabilities does not contain MaximumNumberOfProfiles attribute, FAIL the test.
6. If *cap*.ProfileCapabilities does not contain ConfigurationsSupported attribute, FAIL the test.
7. If *cap*.ProfileCapabilities.ConfigurationsSupported contains no items, FAIL the test.
8. If *cap*.ProfileCapabilities.ConfigurationsSupported contains 'All' item, FAIL the test.
9. If DUT supports Analytics Service and *cap*.ProfileCapabilities.ConfigurationsSupported does not contain 'Analytics' item, FAIL the test.
10. If DUT *cap*.ProfileCapabilities.ConfigurationsSupported contains 'Analytics' item and DUT does not support Analytics Service, FAIL the test.

11. If DUT supports PTZ Service and *cap*.ProfileCapabilities.ConfigurationsSupported does not contain 'PTZ' item, FAIL the test.
12. If DUT *cap*.ProfileCapabilities.ConfigurationsSupported contains 'PTZ' item and DUT does not support PTZ Service, FAIL the test.
13. If DUT supports Media2/Video feature and *cap*.ProfileCapabilities.ConfigurationsSupported does not contain 'VideoSource' item, FAIL the test.
14. If DUT supports Media2/Audio feature and *cap*.ProfileCapabilities.ConfigurationsSupported does not contain 'AudioSource' item, FAIL the test.
15. If DUT supports Media2/Audio outputs feature and *cap*.ProfileCapabilities.ConfigurationsSupported does not contain 'AudioDecoder' item, FAIL the test.
16. If *cap*.StreamingCapabilities contains RTSPWebSocketUri
    16.1. Set *rtspWebSocketUri* := *cap*.StreamingCapabilities.RTSPWebSocketUri
    16.2. If scheme component of *rtspWebSocketUri* is not equal to **ws** or **wss**, FAIL the test and skip other steps.
    16.3. If hierarchical component (hier_part in [rfc2396]) of *rtspWebSocketUri* is not absolute path construction (abs_path in [rfc2396]), FAIL the test and skip other steps.
17. If *cap*.ProfileCapabilities.ConfigurationsSupported contains 'PTZ' item
    17.1. ONVIF Client invokes **GetConfigurations** request.
    17.2. The DUT responds with **GetConfigurationsResponse** message with parameters
        - PTZConfiguration list =: *ptzConfigurationList*
    17.3. If *ptzConfigurationList* is empty, FAIL the test and skip other steps.
18. If *cap*.ProfileCapabilities.ConfigurationsSupported contains 'Analytics' item
    18.1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters
        - ConfigurationToken skipped
        - ProfileToken skipped
    18.2. The DUT responds with **GetAnalyticsConfigurationsResponse** message with parameters
        - Configurations list =: *analyticsConfigurationList*

18.3. If *analyticsConfigurationList* is empty, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetServiceCapabilitiesResponse** message.
- The DUT did not send **GetConfigurationsResponse** message.
- The DUT did not send **GetAnalyticsConfigurationsResponse** message.

5.7.2 GET SERVICES AND GET MEDIA2 SERVICE CAPABILITIES CONSISTENCY

Test Case ID: MEDIA2-7-1-2

Specification Coverage: Capability exchange (ONVIF Core Specification), Capabilities (ONVIF Media2 Service Specification), GetServiceCapabilities command (ONVIF Media2 Service Specification)

Feature Under Test: GetServices, GetServiceCapabilities (for Media2 Service)

WSDL Reference: devicemgmt.wsdl, media2.wsdl

Test Purpose: To verify Get Services and Media2 Service Capabilities consistency.

Pre-Requisite: Media2 Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServices** with parameters
   - IncludeCapability := true
4. The DUT responds with a **GetServicesResponse** message with parameters

- Services list =: *servicesList*

5. ONVIF Client selects Service with Service.Namespace = “http://www.onvif.org/ver20/media/wsdl”:
   - Services list [Namespace = “http://www.onvif.org/ver20/media/wsdl”] =: *media2Service*
6. ONVIF Client invokes **GetServiceCapabilities** message to retrieve media2 service capabilities of the DUT.
7. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*
8. If *cap* differs from *media2Service*.Capabilities.Capabilities, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetServicesResponse** message.
- The DUT did not send **GetServiceCapabilitiesResponse** message.

Note: The following fields are compared at step 6:

- SnapshotUri
- Rotation
- VideoSourceMode
- OSD
- Mask
- ProfileCapabilities.MaximumNumberOfProfiles
- ProfileCapabilities.ConfigurationsSupported
- StreamingCapabilities.RTSPStreaming
- StreamingCapabilities.RTPMulticast

- StreamingCapabilities.RTP_RTSP_TCP
- StreamingCapabilities.NonAggregateControl
- StreamingCapabilities.RTSPWebSocketUri

5.8 Metadata Configuration

5.8.1 Metadata Configuration

5.8.1.1 MODIFY ALL SUPPORTED METADATA CONFIGURATIONS

Test Case ID: MEDIA2-8-1-1

Specification Coverage: Get configurations, Get configuration options, Metadata Configuration, Modify a configuration, Configuration Change

Feature Under Test: GetMetadataConfigurationOptions, GetMetadataConfigurations, SetMetadataConfiguration, Media Configuration Changed Event.

WSDL Reference: media2.wsdl

Test Purpose: To verify whether all supported Metadata Configuration Options can be set. To verify tns1:Media/ConfigurationChanged event generation when Metadata Configuration changes.

Pre-Requisite: Media2 Service is received from the DUT. Metadata feature under Media2 Service is supported by the DUT. Event Service was received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Metadata Configurations list by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - out *metadataConfList* - Metadata Configurations list
4. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters

- in **"tns1:Media/ConfigurationChanged"** - Notification Topic
- out *s* - Subscription reference
- out *currentTime* - current time for the DUT
- out *terminationTime* - Subscription termination time

5. For each Metadata Configuration *metadataConfiguration* in *metadataConfList* repeat the following steps:

5.1. ONVIF Client invokes **GetMetadataConfigurationOptions** request with parameters

- ConfigurationToken := *metadataConfiguration*.@token
- ProfileToken skipped

5.2. DUT responds with **GetMetadataConfigurationOptionsResponse** message with parameters

- Options =: *options*

5.3. ONVIF Client invokes **SetMetadataConfiguration** request with parameters

- Configuration.@token := *metadataConfiguration*.@token
- Configuration.Name := "TestName1"
- Configuration.UseCount := *metadataConfiguration*.UseCount
- If *options* does not contain at least one Extension.CompressionType element:
  - Configuration.CompressionType skipped
- Otherwise:
  - Configuration.CompressionType := *options*.Extension.CompressionType[0]
- If DUT does not support PTZ service:
  - Configuration.PTZStatus skipped
- If DUT supports PTZ service:
  - Configuration.PTZStatus.Status := (*options*.PTZStatusFilterOptions.PanTiltStatusSupported or *options*.PTZStatusFilterOptions.ZoomStatusSupported)

- If *options*.PTZStatusFilterOptions.PanTiltPositionSupported and *options*.PTZStatusFilterOptions.ZoomPositionSupported are skipped:
  - Configuration.PTZStatus.Position := false
- Otherwise:
  - Configuration.PTZStatus.Position := (*options*.PTZStatusFilterOptions.ZoomPositionSupported) or *options*.PTZStatusFilterOptions.PanTiltPositionSupported)

- Configuration.Events.Filter.TopicExpression := "tns1:Media/ConfigurationChanged"
- Configuration.Events.Filter.TopicExpression.Dialect := "http://www.onvif.org/ver10/tev/topicExpression/ConcreteSet"
- If DUT does not support Analytics service:
  - Configuration.Analytics skipped
- If DUT supports Analytics service:
  - Configuration.Analytics := true
- Configuration.Multicast := *metadataConfiguration*.Multicast
- Configuration.SessionTimeout := *metadataConfiguration*.SessionTimeout
- Configuration.AnalyticsEngineConfiguration skipped

5.4. DUT responds with **SetMetadataConfigurationResponse** message.

5.5. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *metadataConfiguration*.@token - Configuration token
- in Metadata - Configuration Type

5.6. ONVIF Client invokes **GetMetadataConfigurations** request with parameters

- ConfigurationToken := *metadataConfiguration*.@token
- ProfileToken skipped

5.7. The DUT responds with **GetMetadataConfigurationsResponse** with parameters

- Configurations list =: *metadataConfList*

5.8. If *metadataConfList*[0] is not equal to Configuration from step 5.3, FAIL the test and skip other steps.

5.9. ONVIF Client invokes **SetMetadataConfiguration** request with parameters

- Configuration.@token := *metadataConfiguration*.@token
- Configuration.Name := "TestName2"
- Configuration.UseCount := *metadataConfiguration*.UseCount
- If *options* does not contain at least one Extension.CompressionType element:
  - Configuration.CompressionType skipped
- Otherwise:
  - Configuration.CompressionType := if *options*.Extension.CompressionType contains at least two items *options*.Extension.CompressionType[1], otherwise *options*.Extension.CompressionType[0]
- If DUT does not support PTZ service:
  - Configuration.PTZStatus skipped
- If DUT supports PTZ service:
  - Configuration.PTZStatus.Status := false
  - Configuration.PTZStatus.Position := false
- Configuration.Events skipped
- If DUT does not support Analytics service:
  - Configuration.Analytics skipped

- If DUT supports Analytics service:
  - Configuration.Analytics := false
- Configuration.Multicast := *metadataConfiguration*.Multicast
- Configuration.SessionTimeout := *metadataConfiguration*.SessionTimeout
- Configuration.AnalyticsEngineConfiguration skipped

5.10. DUT responds with **SetMetadataConfigurationResponse** message.

5.11. If DUT supports Pull-Point Notification feature and Configuration Changed Notification feature, ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.14 with the following input and output parameters

- in *s* - Subscription reference
- in *currentTime* - current time for the DUT
- in *terminationTime* - subscription termination time
- in *audioSourceConfiguration*.@token - Configuration token
- in Metadata - Configuration Type

5.12. ONVIF Client invokes **GetMetadataConfigurations** request with parameters

- ConfigurationToken := *metadataConfiguration*.@token
- ProfileToken skipped

5.13. The DUT responds with **GetMetadataConfigurationsResponse** with parameters

- Configurations list =: *metadataConfList*

5.14. If *metadataConfList*[0] is not equal to Configuration from step 5.9, FAIL the test and skip other steps.

6. If subscription was created at step 4, ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.6 with the following input and output parameters
   - in *s* - Subscription reference
7. ONVIF Client restores Metadata Configurations.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetMetadataConfigurationsResponse** message.
- DUT did not send **SetMetadataConfigurationResponse** message.
- DUT did not send **GetMetadataConfigurationOptionsResponse** message.

Note: The following fields are compared at step 5.8 and 5.14:

- token
- Name
- CompressionType
- PTZStatus
- Events
- Analytics
- Multicast

5.8.1.2 GET METADATA CONFIGURATIONS

Test Case ID: MEDIA2-8-1-2

Specification Coverage: Get configurations, Metadata configuration.

Feature Under Test: GetMetadataConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Metadata Configuration List, Metadata Configuration by Configuration token and compatible Metadata Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Metadata feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Metadata Configurations list by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - out *metadataConfCompleteList* - Metadata Configurations list
4. If *metadataConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
5. For each *metadataConfiguration* in *metadataConfCompleteList* repeat the following steps:
   5.1. ONVIF Client invokes **GetMetadataConfigurations** request with parameters
      - ConfigurationToken := *metadataConfiguration*.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetMetadataConfigurationsResponse** with parameters
      - Configurations list =: *metadataConfList*
   5.3. If *metadataConfList* is empty, FAIL the test and skip other steps.
   5.4. If *metadataConfList* contains more than one item, FAIL the test and skip other steps.
   5.5. If *metadataConfList* does not contain item with @token = *metadataConfiguration*.@token, FAIL the test and skip other steps.
6. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := Metadata
7. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
8. For each Media Profile *profile* in *profileList* repeat the following steps:
   8.1. ONVIF Client invokes **GetMetadataConfigurations** request with parameters
      - ConfigurationToken skipped

- ProfileToken := *profile*.@token

8.2. The DUT responds with **GetMetadataConfigurationsResponse** with parameters

- Configurations list =: *metadataConfList*

8.3. If *metadataConfList* contains at least two items with the same @token, FAIL the test and skip other steps.

8.4. If *metadataConfCompleteList* does not contain at least one item with @token from *metadataConfList*, FAIL the test and skip other steps.

8.5. If *profile*.Configurations contains Metadata:

8.5.1. If *metadataConfList* does not contain item with @token = *profile*.Configurations.Metadata.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetMetadataConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.8.1.3 PROFILES AND METADATA CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-8-1-3

Specification Coverage: Get configurations, Get media profiles, Metadata configuration.

Feature Under Test: GetMetadataConfigurations, GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Metadata Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Metadata feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := Metadata
4. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
5. For each Media Profile *profile* in *profileList*, which contains Configurations.Metadata repeat the following steps:
   5.1. ONVIF Client invokes **GetMetadataConfigurations** request with parameters
      - ConfigurationToken := *profile*.Configurations.Metadata.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetMetadataConfigurationsResponse** with parameters
      - Configurations list =: *metadataConfList*
   5.3. If *metadataConfList*[0] is not equal to *profile*.Configurations.Metadata, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetMetadataConfigurationsResponse** message.

Note: The following fields are compared at step 5.3:

- Name
- CompressionType

- GeoLocation
- PTZStatus.Status
- PTZStatus.Position
- Events.Filter (only field presence will be compared)
- Events.SubscriptionPolicy (only field presence will be compared)
- Analytics
- AnalyticsEngineConfiguration.AnalyticsModule list (Type and Name will be used as key. Parameters item will not be compared).

5.8.1.4 GET METADATA CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-8-1-4

Specification Coverage: Get configurations, Metadata configuration.

Feature Under Test: GetMetadataConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of GetMetadataConfigurations with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Metadata feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Metadata Configurations list by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - out *metadataConfList* - Metadata Configurations list
4. ONVIF Client invokes **GetMetadataConfigurations** request with parameters
   - ConfigurationToken := other than listed in *metadataConfList*

- ProfileToken skipped

5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message

5.9 Analytics Configuration

5.9.1 GET ANALYTICS CONFIGURATIONS

Test Case ID: MEDIA2-9-1-1

Specification Coverage: Get configurations, Analytics configuration.

Feature Under Test: GetAnalyticsConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify retrieving complete Analytics Configuration List, Analytics Configuration by Configuration token and compatible Analytics Configuration by Profile token.

Pre-Requisite: Media2 Service is received from the DUT. Analytics feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Analytics Configurations list by following the procedure mentioned in Annex A.31 with the following input and output parameters
   - out *analyticsConfCompleteList* - Analytics Configurations list

4. If *analyticsConfCompleteList* contains at least two items with the same @token, FAIL the test and skip other steps.
5. For each *analyticsConfiguration* in *analyticsConfCompleteList* repeat the following steps:
   5.1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters
      - ConfigurationToken := *analyticsConfiguration*.@token
      - ProfileToken skipped
   5.2. The DUT responds with **GetAnalyticsConfigurationsResponse** with parameters
      - Configurations list =: *analyticsConfList*
   5.3. If *analyticsConfList* is empty, FAIL the test and skip other steps.
   5.4. If *analyticsConfList* contains more than one item, FAIL the test and skip other steps.
   5.5. If *analyticsConfList* does not contain item with @token = *analyticsConfiguration*.@token, FAIL the test and skip other steps.
6. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := Analytics
7. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
8. For each Media Profile *profile* in *profileList* repeat the following steps:
   8.1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters
      - ConfigurationToken skipped
      - ProfileToken := *profile*.@token
   8.2. The DUT responds with **GetAnalyticsConfigurationsResponse** with parameters
      - Configurations list =: *analyticsConfList*
   8.3. If *analyticsConfList* contains at least two items with the same @token, FAIL the test and skip other steps.
   8.4. If *analyticsConfCompleteList* does not contain at least one item with @token from *analyticsConfList*, FAIL the test and skip other steps.

8.5. If *profile*.Configurations contains Analytics:

8.5.1. If *analyticsConfList* does not contain item with @token = *profile*.Configurations.Analytics.@token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAnalyticsConfigurationsResponse** message.
- DUT did not send **GetProfilesResponse** message.

5.9.2 PROFILES AND ANALYTICS CONFIGURATIONS CONSISTENCY

Test Case ID: MEDIA2-9-1-2

Specification Coverage: Get configurations, Get media profiles, Analytics configuration.

Feature Under Test: GetAnalyticsConfigurations, GetProfiles

WSDL Reference: media2.wsdl

Test Purpose: To verify all Media Profiles are consistent with Analytics Configurations.

Pre-Requisite: Media2 Service is received from the DUT. Analytics feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := Analytics
4. The DUT responds with **GetProfilesResponse** message with parameters

- Profiles list =: *profileList*

5. For each Media Profile *profile* in *profileList*, which contains Configurations.Analytics repeat the following steps:

   5.1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters

   - ConfigurationToken := *profile*.Configurations.Analytics.@token
   - ProfileToken skipped

   5.2. The DUT responds with **GetAnalyticsConfigurationsResponse** with parameters

   - Configurations list =: *analyticsConfList*

   5.3. If *analyticsConfList*[0] is not equal to *profile*.Configurations.Analytics, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetAnalyticsConfigurationsResponse** message.

Note: The following fields are compared at step 5.3:

- Name
- AnalyticsEngineConfiguration.AnalyticsModule list (Type and Name will be used as key. Parameters item will not be compared).
- AnalyticsEngineConfiguration.RuleEngineConfiguration list (Type and Name will be used as key. Parameters item will not be compared).

5.9.3 GET ANALYTICS CONFIGURATIONS – INVALID TOKEN

Test Case ID: MEDIA2-9-1-3

Specification Coverage: Get configurations, Analytics configuration.

Feature Under Test: GetAnalyticsConfigurations

WSDL Reference: media2.wsdl

Test Purpose: To verify SOAP 1.2 Fault receiving in case of GetAnalyticsConfigurations with invalid token.

Pre-Requisite: Media2 Service is received from the DUT. Analytics feature under Media2 Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Analytics Configurations list by following the procedure mentioned in Annex A.31 with the following input and output parameters
   - out *analyticsConfList* - Analytics Configurations list
4. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters
   - ConfigurationToken := other than listed in *analyticsConfList*
   - ProfileToken skipped
5. The DUT returns **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:NoConfig** SOAP 1.2 fault message

5.10 Masks Configuration

5.10.1 CREATE MASKS

Test Case ID: MEDIA2-10-1-1

Specification Coverage: CreateMask, DeleteMask (ONVIF Media2 Service Specification)

Feature Under Test: CreateMask, DeleteMask

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT creates and removes Mask.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client configures device for adding of new mask by following the procedure mentioned in Annex A.32 with the following input and output parameters
   - out *maskOptions* - Options to create mask with
   - out *vscToken* - Token of Video Source Configuration to be used
   - out *maskToRestore* - Mask to restore (if a Mask was removed)
4. ONVIF Client invokes **CreateMask** request with parameters
   - token := "TestMask" (note: this token can be ignored by DUT)
   - ConfigurationToken := *vscToken*
   - If *maskOptions*.RectangleOnly is false or skipped:
     - Polygon := Point[0].@x="-0.4", Point[0].@y="-0.2", Point[1].@x="-0.2", Point[1].@y="0.3", Point[2].@x="0.1", Point[2].@y="0.4", Point[3].@x="0.3", Point[3].@y="-0.3"
   - If *maskOptions*.RectangleOnly = true:
     - Polygon := Point[0].@x="-0.5", Point[0].@y="-0.5", Point[1].@x="-0.5", Point[1].@y="0.5", Point[2].@x="0.5", Point[2].@y="0.5", Point[3].@x="0.5", Point[3].@y="-0.5"
   - Type := first value from *maskOptions*.Types list
   - If Type = "Color":

- If *maskOptions*.Color.ColorList is specified:
  - Color := first value that are listed in *maskOptions*.Color.ColorList
- If *maskOptions*.Color.ColorspaceRange is specified:
  - Color.@Colorspace := Colorspace value of the first item that is listed in *maskOptions*.Color.ColorspaceRange
  - Color.@X := value from the range [X.Min,X.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
  - Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
  - Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
- Enabled := false

5. DUT responds with **CreateMaskResponse** message with parameters
   - Token of the created mask =: *maskToken*
6. ONVIF Client invokes **GetMasks** request with parameters
   - Token := *maskToken*
   - ConfigurationToken skipped
7. DUT responds with **GetMasksResponse** message with parameters
   - Masks =: *maskList*
8. If *maskList*[0] is not equal to Mask from step 4, FAIL the test and skip other steps.
9. ONVIF Client invokes **DeleteMask** request with parameters
   - Token := *maskToken*
10. DUT responds with **DeleteMaskResponse** message
11. ONVIF Client invokes **GetMasks** request with parameters
    - Token skipped
    - ConfigurationToken := *videoSourceConf1*[0].@Token

12. DUT responds with **GetMasksResponse** message with parameters
    - Masks =: *maskList2*
13. If *maskList2* contains Mask with @Token = *maskToken*, FAIL the test and skip other steps.
14. ONVIF Client restores the DUT state.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetMasksResponse** message(s).
- The DUT did not send **CreateMaskResponse** message.
- The DUT did not send **DeleteMaskResponse** message.

Note: The following fields are compared at step 8:

- ConfigurationToken
- Polygon, only the following will be checked:
  - the field is specified
  - list contains at least 3 points
- Type
- Color (if Type = "Color")
- Enabled

5.10.2 GET MASKS

Test Case ID: MEDIA2-10-1-2

Specification Coverage: GetMasks (ONVIF Media2 Service Specification)

Feature Under Test: GetMasks

WSDL Reference: media2.wsdl

Test Purpose: To verify DUT sends complete Masks list and list of Masks, which are compatible with specific Video Source Configuration.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client configures device to add mask if no masks exists by following the procedure mentioned in Annex A.34 with the following input and output parameters:
   - out *maskToRemove* - Mask to remove (if a Mask was created)
4. ONVIF Client invokes **GetMasks** request with parameters
   - Token skipped
   - ConfigurationToken skipped
5. DUT responds with **GetMasksResponse** message with parameters
   - Masks =: *maskConfCompleteList1*
6. If *maskConfCompleteList1* is empty, FAIL the test and skip other steps.
7. If *maskConfCompleteList1* contains at least two items with the same @token, FAIL the test and skip other steps.
8. For each Mask *mask* in *maskConfCompleteList1* repeat the following steps:
   8.1. ONVIF Client invokes **GetMasks** request with parameters
      - Token := *mask*.@token
      - ConfigurationToken skipped
   8.2. DUT responds with **GetMasksResponse** message with parameters
      - Masks =: *maskList1*
   8.3. If *maskList1* is empty, FAIL the test and skip other steps.
   8.4. If *maskList1* contains more than one item, FAIL the test and skip other steps.

8.5. If *maskList1*[0].@token != *mask*.@token, FAIL the test and skip other steps.

8.6. If *maskList1*[0] is not equal to *mask*, FAIL the test and skip other steps.

9. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
10. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:

    10.1. ONVIF Client invokes **GetMasks** request with parameters
    - Token skipped
    - ConfigurationToken := *videoSourceConf1*.@Token

    10.2. DUT responds with **GetMasksResponse** message with parameters
    - Masks =: *maskConfList1*

    10.3. If *maskConfList1* contains at least two items with the same @token, FAIL the test and skip other steps.

    10.4. If *maskConfCompleteList1* does not contain at least one item with @token from *maskConfList1* list, FAIL the test and skip other steps.

11. ONVIF Client restores the DUT state if required.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetMasksResponse** message(s).

Note: The following fields are compared at step 8.6:

- token
- ConfigurationToken
- Polygon

- Type
- Color
- Enabled

5.10.3 SET MASKS

Test Case ID: MEDIA2-10-1-3

Specification Coverage: SetMask, DeleteMask (ONVIF Media2 Service Specification)

Feature Under Test: SetMask

WSDL Reference: media2.wsdl

Test Purpose: To verify the DUT changes and removes Mask.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client configures device for adding of new mask by following the procedure mentioned in Annex A.32 with the following input and output parameters
   - out *maskOptions* - Options to create mask with
   - out *vscToken* - Token of Video Source Configuration to be used
   - out *maskToRestore* - Mask to restore (if a Mask was removed)
4. Set *maskType* := first value from *maskOptions*.Types list
5. ONVIF Client creates mask by following the procedure mentioned in Annex A.35 with the following input and output parameters
   - in *vscToken* - Video Source configuration token
   - in *maskOptions* - Options of the mask
   - in *maskType* - Mask type

- out *maskToken* - Token of the created mask

6. ONVIF Client invokes **SetMask** request with parameters
   - token := *maskToken*
   - ConfigurationToken := *vscToken*
   - If *maskOptions*.RectangleOnly is false or skipped:
     - Polygon := Point[0].@x="-0.5", Point[0].@y="-0.3", Point[1].@x="-0.3", Point[1].@y="0.4", Point[2].@x="0.2", Point[2].@y="0.5", Point[3].@x="0.4", Point[3].@y="-0.4"
   - If *maskOptions*.RectangleOnly = true:
     - Polygon := Point[0].@x="-0.6", Point[0].@y="-0.6", Point[1].@x="-0.6", Point[1].@y="0.6", Point[2].@x="0.6", Point[2].@y="0.6", Point[3].@x="0.6", Point[3].@y="-0.6"
   - Type := last value from *maskOptions*.Types list
   - If Type = "Color":
     - If *maskOptions*.Color.ColorList is specified:
       - Color := last value that are listed in *maskOptions*.Color.ColorList
     - If *maskOptions*.Color.ColorspaceRange is specified:
       - Color.@Colorspace := Colorspace value of the last item that is listed in *maskOptions*.Color.ColorspaceRange
       - Color.@X := value from the range [X.Min,X.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
       - Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
       - Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
   - Enabled := true
7. DUT responds with **SetMaskResponse** message
8. ONVIF Client invokes **GetMasks** request with parameters

- Token := *maskToken*
- ConfigurationToken skipped

9. DUT responds with **GetMasksResponse** message with parameters
   - Masks =: *maskList*
10. If *maskList*[0] is not equal to Mask from step 6, FAIL the test and skip other steps.
11. ONVIF Client invokes **DeleteMask** request with parameters
    - Token := *maskToken*
12. DUT responds with **DeleteMaskResponse** message
13. ONVIF Client invokes **GetMasks** request with parameters
    - Token skipped
    - ConfigurationToken := *videoSourceConf1*[0].@Token
14. DUT responds with **GetMasksResponse** message with parameters
    - Masks =: *maskList2*
15. If *maskList2* contains Mask with @Token = *maskToken*, FAIL the test and skip other steps.
16. ONVIF Client restores the DUT state.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **SetMaskResponse** message.
- The DUT did not send **GetMasksResponse** message(s).
- The DUT did not send **DeleteMaskResponse** message.

Note: The following fields are compared at step 10:

- ConfigurationToken

- Polygon, only the following will be checked:
  - the field is specified
  - list contains at least 3 points
- Type
- Color (if Type = "Color")
- Enabled

5.10.4 GET MASK OPTIONS

Test Case ID: MEDIA2-10-1-4

Specification Coverage: GetMaskOptions (ONVIF Media2 Service Specification)

Feature Under Test: GetMaskOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify consistency in Mask Options.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
4. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:
   4.1. ONVIF Client invokes **GetMaskOptions** request with parameters
      - ConfigurationToken := *videoSourceConf1*.@token
   4.2. DUT responds with **GetMaskOptionsResponse** message with parameters

- MaskOptions =: *maskOptions*

4.3. If *maskOptions*.MaxMasks < 0, FAIL the test and skip other steps.

4.4. If *maskOptions*.MaxPoints < 3, FAIL the test and skip other steps.

4.5. If *maskOptions* does not have Types element, FAIL the test and skip other steps.

4.6. If *maskOptions* has Color.ColorspaceRange element:

4.6.1. If *maskOptions*.Color.ColorspaceRange.X.Min > *maskOptions*.Color.ColorspaceRange.X.Max, FAIL the test and skip other steps.

4.6.2. If *maskOptions*.Color.ColorspaceRange.Y.Min > *maskOptions*.Color.ColorspaceRange.Y.Max, FAIL the test and skip other steps.

4.6.3. If *maskOptions*.Color.ColorspaceRange.Z.Min > *maskOptions*.Color.ColorspaceRange.Z.Max, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetMaskOptionsResponse** message(s).

5.10.5 MASK CONFIGURATIONS AND MASK OPTIONS CONSISTENCY

Test Case ID: MEDIA2-10-1-5

Specification Coverage: GetMasks, GetMaskOptions (ONVIF Media2 Service Specification)

Feature Under Test: GetMasks, GetMaskOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify all Mask configurations are consistent with Mask Options.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
4. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* repeat the following steps:
   4.1. ONVIF Client invokes GetMasks request with parameters
      - Token skipped
      - ConfigurationToken := *videoSourceConf1*.@token
   4.2. DUT responds with **GetMasksResponse** message with parameters
      - Masks =: *maskConfList1*
   4.3. ONVIF Client invokes **GetMaskOptions** request with parameters
      - ConfigurationToken := *videoSourceConf1*.@token
   4.4. DUT responds with **GetMaskOptionsResponse** message with parameters
      - MaskOptions := *maskOptions*
   4.5. Set *colorMask* := empty
   4.6. For each Mask *mask* in *maskConfList1* repeat the following steps:
      4.6.1. If *mask*.ConfigurationToken is not equal to *videoSourceConf1*.@token, FAIL the test and skip other steps.
      4.6.2. If *maskOptions* does not contain Types element with value is equal to *mask*.Type, FAIL the test and skip other steps.
      4.6.3. If *maskOptions*.RectangleOnly = true and *mask*.Polygon does not have four points, FAIL the test and skip other steps.

4.6.4. If number of points of *mask*.Polygon > *maskOptions*.MaxPoints, FAIL the test and skip other steps.

4.6.5. If *mask*.Color is specified:

4.6.5.1. If *mask*.Color@Colorspace is skipped:

4.6.5.1.1. Set *mask*.Color@Colorspace = http://www.onvif.org/ver10/colorspace/YCbCr

4.6.5.2. If *maskOptions*.Color has at least one ColorList element:

4.6.5.2.1. For each ColorList element *colorList* in *maskOptions*.Color repeat the following steps:

4.6.5.2.1.1. If *colorList*@Colorspace is skipped:

4.6.5.2.1.1.1. Set *colorList*@Colorspace = http://www.onvif.org/ver10/colorspace/YCbCr

4.6.5.2.1.2. If *maskOptions*.Color does not contain ColorList element with @X = *mask*.Color.@X, and with @Y = *mask*.Color.@Y, and with @Z = *mask*.Color.@Z, and with @Colorspace = *mask*.Color@Colorspace, FAIL the test and skip other steps.

4.6.5.3. If *maskOptions*.Color has at least one ColorspaceRange element:

4.6.5.3.1. Set *maskOptions*.Color.ColorspaceRange =: *colorspaceRange1*, where *maskOptions*.Color.ColorspaceRange is the first ColorspaceRange element that corresponds the following requirements:

4.6.5.3.1.1. *colorspaceRange1*.@Colorspace = *mask*.Color.@Colorspace.

4.6.5.3.1.2. *colorspaceRange1*.X.Min <= *mask*.Color.@X

4.6.5.3.1.3. *colorspaceRange1*.X.Max >= *mask*.Color.@X

4.6.5.3.1.4. *colorspaceRange1*.Y.Min <= *mask*.Color.@Y

4.6.5.3.1.5. *colorspaceRange1*.Y.Max >= *mask*.Color.@Y

4.6.5.3.1.6. *colorspaceRange1*.Z.Min <= *mask*.Color.@Z

4.6.5.3.1.7. *colorspaceRange1*.Z.Max >= *mask*.Color.@Z

4.6.5.3.2. If *colorspaceRange1* is empty, FAIL the test and skip other steps.

4.6.5.4. If *MaskOptions*.SingleColorOnly = true

4.6.5.4.1. If *colorMask* is empty:

4.6.5.4.1.1. *colorMask* := *mask*.Color

4.6.5.4.1.2. GoTo step 4.6

4.6.5.4.2. If *colorMask* is not equal to *mask*.Color, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetMasksResponse** message(s).
- The DUT did not send **GetMaskOptionsResponse** message(s).

5.10.6 SINGLE COLOR ONLY PARAMETER

Test Case ID: MEDIA2-10-1-6
Specification Coverage: GetMaskOptions (ONVIF Media2 Service Specification)

Feature Under Test: GetMaskOptions

WSDL Reference: media2.wsdl

Test Purpose: To verify Masks Color consistency if SingleColorOnly is present and true in Mask Options.

Pre-Requisite: Media2 Service feature is supported by DUT. Mask feature is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
4. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:
   4.1. ONVIF Client invokes **GetMaskOptions** request with parameters
      - ConfigurationToken := *videoSourceConf1*.@Token
   4.2. DUT responds with **GetMaskOptionsResponse** message with parameters
      - MaskOptions =: *maskOptions*
   4.3. If *maskOptions*.SingleColorOnly != true or *maskOptions*.Types list does not contain "Color" or *maskOptions*.MaxMasks < 2 go to step 4.
   4.4. ONVIF Client removes all Masks from Video Source Configuration by following the procedure mentioned in Annex A.36 with the following input and output parameters
      - in *videoSourceConf1*.@token - Video Source Configuration token
   4.5. Set *maskType* := “Color”
   4.6. ONVIF Client creates mask by following the procedure mentioned in Annex A.35 with the following input and output parameters
      - in *vscToken* - Video Source configuration token
      - in *maskOptions* - Options of the mask

- in *maskType* - Mask type
- out *maskToken1* - Token of the created mask

4.7. ONVIF Client invokes **CreateMask** request with parameters

- token := "TestMask" (note: this token can be ignored by DUT)
- ConfigurationToken := *videoSourceConf1*.@token
- Polygon := Point[0].@x="-0.4", Point[0].@y="-0.2", Point[1].@x="-0.2", Point[1].@y="0.3", Point[2].@x="0.1", Point[2].@y="0.4", Point[3].@x="0.3", Point[3].@y="-0.3" if *maskOptions*.RectangleOnly is false or skipped. Else Polygon := Point[0].@x="-0.5", Point[0].@y="-0.5", Point[1].@x="-0.5", Point[1].@y="0.5", Point[2].@x="0.5", Point[2].@y="0.5", Point[3].@x="0.5", Point[3].@y="-0.5"
- Type := “Color”
- Enabled := false

4.8. DUT responds with **CreateMaskResponse** message with parameters

- Token of the created mask =: *maskToken2*

4.9. ONVIF Client invokes **GetMasks** request with parameters

- Token := *maskToken1*
- ConfigurationToken skipped

4.10. DUT responds with **GetMasksResponse** message with parameters

- Masks =: *maskList1*

4.11. ONVIF Client invokes **GetMasks** request with parameters

- Token := *maskToken2*
- ConfigurationToken skipped

4.12. DUT responds with **GetMasksResponse** message with parameters

- Masks =: *maskList2*

4.13. If *maskList2*[0].Color is not equal to *maskList1*[0].Color, FAIL the test and skip other steps.

4.14. ONVIF Client invokes **SetMask** request with parameters

- token := *maskToken2*
- ConfigurationToken := *vscToken*
- Polygon := Point[0].@x="-0.5", Point[0].@y="-0.3", Point[1].@x="-0.3", Point[1].@y="0.4", Point[2].@x="0.2", Point[2].@y="0.5", Point[3].@x="0.4", Point[3].@y="-0.4" if *maskOptions*.RectangleOnly is false or skipped. Else Polygon := Point[0].@x="-0.6", Point[0].@y="-0.6", Point[1].@x="-0.6", Point[1].@y="0.6", Point[2].@x="0.6", Point[2].@y="0.6", Point[3].@x="0.6", Point[3].@y="-0.6"
- Type := "Color"
- If *maskOptions*.Color.ColorList is specified:
  - Color := last value that are listed in *maskOptions*.Color.ColorList
- If *maskOptions*.Color.ColorspaceRange is specified:
  - Color.@Colorspace := Colorspace value of the last item that is listed in *maskOptions*.Color.ColorspaceRange
  - Color.@X := value from the range [X.Min,X.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
  - Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
  - Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
- Enabled := true

4.15. DUT responds with **SetMaskResponse** message

4.16. ONVIF Client invokes **GetMasks** request with parameters

- Token := *maskToken1*
- ConfigurationToken skipped

4.17. DUT responds with **GetMasksResponse** message with parameters

- Masks =: *maskList3*

4.18. If *maskList3*[0].Color is not equal to Color value from step 4.14, FAIL the test and skip other steps.

4.19. ONVIF Client restores the DUT state.

5. PASS the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **CreateMaskResponse** message(s).
- The DUT did not send **SetMaskResponse** message(s).
- The DUT did not send **GetMasksResponse** message(s).
- The DUT did not send **GetMaskOptionsResponse** message(s).

Annex A Helper Procedures and Additional Notes

A.1 Delete Media Profile if Max Reached

Name: HelperDeleteMediaProfileWhenMaxProfiles

Procedure Purpose: Helper procedure to delete Media Profile if maximum number of Media Profiles is reached.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: None.

Procedure:

1. ONVIF Client retrieves Media2 Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Media2 Service Capabilities
2. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
3. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
4. If number of items in *profileList* = *cap*.ProfileCapabilities.MaximumNumberOfProfiles:
   4.1. If *profileList* does not contain items with @fixed = false, FAIL the test and skip other steps.
   4.2. ONVIF Client invokes **DeleteProfile** request with parameters
      - Token := @token of item with @fixed = false from *profileList*
   4.3. The DUT responds with **DeleteProfileResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **DeleteProfileResponse** message.

A.2 Get Service Capabilities

Name: HelperGetServiceCapabilities

Procedure Purpose: Helper procedure to retrieve Media2 Service Capabilities.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Media2 Service Capabilities (*cap*).

Procedure:

1. ONVIF Client invokes **GetServiceCapabilities** request.
2. The DUT responds with **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetServiceCapabilitiesResponse** message.

A.3 Get Video Source Configurations List

Name: HelperGetVideoSourceConfigurationsList

Procedure Purpose: Helper procedure to retrieve Video Source Configurations List.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Video Source Configurations list (*videoSourceConfList*).

Procedure:

1. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
   - Configurations list =: *videoSourceConfList*
3. If *videoSourceConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourceConfigurationsResponse** message.

A.4 Create Pull Point Subscription

Name: HelperCreatePullPointSubscription

Procedure Purpose: Helper procedure to create PullPoint Subscription with specified Topic.

Pre-requisite: Event Service is received from the DUT.

Input: Notification Topic (*topic*).

Returns: Subscription reference (*s*), current time for the DUT (*ct*), subscription termination time (*tt*).

Procedure:

1. ONVIF Client invokes **CreatePullPointSubscription** request with parameters
   - Filter.TopicExpression := *topic*
   - Filter.TopicExpression.@Dialect := "http://www.onvif.org/ver10/tev/topicExpression/ConcreteSet"
2. The DUT responds with **CreatePullPointSubscriptionResponse** message with parameters

- SubscriptionReference =: *s*
- CurrentTime =: *ct*
- TerminationTime =: *tt*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **CreatePullPointSubscriptionResponse** message.

A.5 Retrieve Profile Changed Event by PullPoint

Name: HelperPullProfileChanged

Procedure Purpose: Helper procedure to retrieve and check tns1:Media/ProfileChanged event with PullMessages.

Pre-requisite: Event Service is received from the DUT.

Input: Subscription reference (*s*), current time for the DUT (*ct*), Subscription termination time (*tt*) and Media Profile token (*profileToken*).

Returns: None

Procedure:

1. Until *timeout1* timeout expires, repeat the following steps:
   1.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
   1.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
      - Timeout := PT60S
      - MessageLimit := 1
   1.3. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*

- NotificationMessage list =: *notificationMessageList*

1.4. If *notificationMessageList* is not empty and the Token source simple item in *notificationMessageList* is equal to *profileToken*, skip other steps and finish the procedure.

1.5. If *timeout1* timeout expires for step 1 without Notification with Token source simple item equal to *profileToken*, FAIL the test and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **PullMessagesResponse** message.

Note: *timeout1* will be taken from Operation Delay field of ONVIF Device Test Tool.

A.6 Delete Subscription

Name: HelperDeleteSubscription

Procedure Purpose: Helper procedure to delete supscribtion.

Pre-requisite: Event Service is received from the DUT.

Input: Subscription reference (*s*)

Returns: None

Procedure:

1. ONVIF Client sends an **Unsubscribe** to the subscription endpoint s.
2. The DUT responds with **UnsubscribeResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **UnsubscribeResponse** message.

A.7 Name Parameters

There are the following limitations on maximum length of the Name parameters that shall be used during tests by ONVIF Device Test Tool to prevent faults from DUT:

- Name shall be less than or equal to 64 characters (only readable characters accepted).
- UTF-8 character set shall be used for Name.

Note: these limitations will not be used, if ONVIF Device Test Tool reuses values that were received from the DUT.

A.8 Create Empty Profile

Name: HelperCreateEmptyProfile

Procedure Purpose: Helper procedure to find, create or configure empty Media Profile (without Configurations).

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Empty Media Profile token (*profileToken*). Flag that indicates that new Media Profile was created (*newProfileFlag*).

Procedure:

1. ONVIF Client retrieves Media2 Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Media2 Service Capabilities
2. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := All
3. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
4. If number of items in *profileList* >= *cap*.ProfileCapabilities.MaximumNumberOfProfiles:

4.1. If *profileList* does not contain items with @fixed = false, go to step 9.

4.2. ONVIF Client invokes **DeleteProfile** request with parameters

- Token := @token of item with @fixed = false from *profileList*

4.3. The DUT responds with **DeleteProfileResponse** message.

5. ONVIF Client invokes **CreateProfile** request with parameters
   - Name := "testMedia2"
   - Configuration list - skipped
6. DUT responds with **CreateProfileResponse** message with parameters
   - Token =: *profileToken*
7. Set *newProfileFlag* := true
8. Skip other steps of procedure.
9. ONVIF Client invokes **RemoveConfiguration** request with parameters
   - ProfileToken := *profileList*[0].@token
   - Configuration[0].Type := All
   - Configuration[0].Token skipped
10. The DUT responds with **RemoveConfigurationResponse** message.
11. Set *profileToken* := *profileList*[0].@token
12. Set *newProfileFlag* := false

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **DeleteProfileResponse** message.
- DUT did not send **CreateProfileResponse** message.

- DUT did not send **RemoveConfigurationResponse** message.

Note: See Annex A.7 for Name and Token Parameters Length limitations.

A.9 Get Audio Source Configurations List

Name: HelperGetAudioSourceConfigurationsList

Procedure Purpose: Helper procedure to retrieve Audio Source Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Input: None.

Returns: Audio Source Configurations List (*audioSourceConfList*).

Procedure:

1. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters
   - Configurations list =: *audioSourceConfList*
3. If *audioSourceConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioSourceConfigurationsResponse** message.

A.10 Get Audio Output Configurations List

Name: HelperGetAudioOutputConfigurationsList

Procedure Purpose: Helper procedure to retrieve Audio Output Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Input: None.

Returns: Audio Output Configurations List (*audioOutputConfList*).

Procedure:

1. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
   - Configurations list =: *audioOutputConfList*
3. If *audioOutputConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioOutputConfigurationsResponse** message.

A.11 Delete Media Profile

Name: HelperDeleteMediaProfile

Procedure Purpose: Helper procedure to delete Media Profile.

Pre-requisite: Media2 Service is received from the DUT.

Input: Media profile token (*profileToken*).

Returns: None.

Procedure:

1. ONVIF Client invokes **DeleteProfile** request with parameters
   - Token = *profileToken*

2. The DUT responds with **DeleteProfileResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **DeleteProfileResponse** message.

A.12 Configure Media profile with Video Source Configuration

Name: HelperConfigureMediaProfileWithVideoSource

Procedure Purpose: Helper procedure to configure Media Profile to contain Video Source Configuration.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Media Profile (*profile*) containing Video Source Configuration.

Procedure:

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := VideoSource
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList* is empty, FAIL the test and skip other steps.
4. If *profileList* contains Media Profile with Configurations.VideoSource:
   4.1. Set *profile* := the first profile with Configurations.VideoSource from the *profileList*.
   4.2. Skip other steps.
5. Set *profile* := *profileList*[0].
6. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters

- ConfigurationToken skipped
- ProfileToken = *profile*.@token

7. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
   - Configurations list =: *videoSourceConfigurationList*
8. If *videoSourceConfigurationList* is empty, FAIL the test and skip other steps.
9. ONVIF Client invokes **AddConfiguration** request with parameters
   - ProfileToken := *profile*.@token
   - Name skipped
   - Configuration[0].Type := VideoSource
   - Configuration[0].Token := *videoSourceConfigurationList*[0]
10. The DUT responds with **AddConfigurationResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoSourceConfigurationsResponse** message.
- DUT did not send **AddConfigurationResponse** message.

A.13 View Modes List

Source view modes supported by device defined in tt:ViewModes

- tt:Fisheye
- tt:360Panorama
- tt:180Panorama
- tt:Quad

- tt:Original
- tt:LeftHalf
- tt:RightHalf
- tt:Dewarp

A.14 Retrieve Configuration Changed Event by PullPoint

Name: HelperPullConfigurationChanged

Procedure Purpose: Helper procedure to retrieve and check tns1:Media/ConfigurationChanged event with PullMessages.

Pre-requisite: Event Service is received from the DUT.

Input: Subscription reference (*s*), current time for the DUT (*ct*), Subscription termination time (*tt*), configuration token (*confToken*) and configuration type (*confType*).

Returns: None

Procedure:

1. Until *timeout1* timeout expires, repeat the following steps:
   1.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
   1.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
      - Timeout := PT60S
      - MessageLimit := 1
   1.3. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*
      - NotificationMessage list =: *notificationMessageList*
   1.4. If *notificationMessageList* is not empty and source simple item Token = *confToken* and Type = *confType*, skip other steps and finish the procedure.
   1.5. If *timeout1* timeout expires for step 1 without Notification with source simple item Token = *confToken* and Type = *confType*, FAIL the test and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **PullMessagesResponse** message.

Note: *timeout1* will be taken from Operation Delay field of ONVIF Device Test Tool.

A.15 Waiting for Reboot

Name: HelperWaitingReboot

Procedure Purpose: Helper procedure to wait until the Device becomes available after reboot.

Pre-requisite: None.

Input: None.

Returns: None.

Procedure:

1. If DUT supports Discovery:

   1.1. Until *timeout1* timeout expires, repeat the following steps:

      1.1.1. The DUT will send Multicast **Hello** message after it is successfully rebooted with parameters:

      - EndpointReference.Address equal to unique endpoint reference of the DUT
      - Types list
      - Scopes list
      - XAddrs list := *xAddrsList*
      - MetadataVersion

      1.1.2. If *xAddrsList* contains URI address with IPv4 address other than LinkLocal from ONVIF Client subnet, go to step 3.

   1.2. If *timeout1* timeout expires for the step 1.1 without **Hello** with URI address with not a LinkLocal IPv4 address from ONVIF Client subnet, FAIL the test and skip other steps.

1.3. ONVIF client waits for 5 seconds after **Hello** was received.

2. If DUT does not support Discovery:

2.1. ONVIF Client waits during *rebootTimeout*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **Hello** message.

Note: *timeout1* will be taken from Reboot Timeout field of ONVIF Device Test Tool.

A.16 VideoEncoderConfigurationOptions and VideoEncoderConfiguration mapping

Table A.1. VideoEncoderConfigurationOptions and VideoEncoderConfiguration mapping

VideoEncoderConfigurationOptions field	VideoEncoderConfiguration field
Encoding	Encoding
QualityRange.Min-QualityRange.Max	Quality
Resolutions Available contains list of Height and Width pairs	Height and Width
GovLengthRange.Min-GovLengthRange.Max	GovLength
FrameRatesSupported contains list of available values	RateControl.FrameRateLimit
BitrateRange.Min-BitrateRange.Max	RateControl.BitrateLimit
ProfilesSupported the list of string values listed in tt:VideoEncodingProfiles	Profile

A.17 Get Video Encoder Configurations List

Name: HelperGetVideoEncoderConfigurationsList

Procedure Purpose: Helper procedure to retrieve Video Encoder Configurations List.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Video Encoder Configurations List (*videoEncoderConfList*).

Procedure:

1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with all video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters
   - Configurations list =: *videoEncoderConfList*
3. If *videoEncoderConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationsResponse** message.

A.18 Get Video Sources List

Name: HelperGetVideoSourcesList

Procedure Purpose: Helper procedure to retrieve Video Sources List.

Pre-requisite: Media2 Service is received from the DUT. DeviceIO Service is received from the DUT.

Input: None.

Returns: Video Sources List (*videoSourcesList*).

Procedure:

1. ONVIF Client invokes **GetVideoSources** request.
2. The DUT responds with **GetVideoSourcesResponse** with parameters

- Token list =: *videoSourcesList*

3. If *videoSourcesList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSourcesResponse** message.

A.19 Configure Media profile with Audio Source Configuration

Name: HelperConfigureMediaProfileWithAudioSource

Procedure Purpose: Helper procedure to configure Media Profile to contain Audio Source Configuration.

Pre-requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Input: None.

Returns: Media Profile (*profile*) containing Audio Source Configuration.

Procedure:

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioSource
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList* is empty, FAIL the test and skip other steps.
4. If *profileList* contains Media Profile with Configurations.AudioSource:
   4.1. Set *profile* := the first profile with Configurations.AudioSource from the *profileList*.
   4.2. Skip other steps.

5. Set *profile* := *profileList*[0].
6. ONVIF Client invokes **GetAudioSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken = *profile*.@token
7. The DUT responds with **GetAudioSourceConfigurationsResponse** with parameters
   - Configurations list =: *audioSourceConfigurationList*
8. If *audioSourceConfigurationList* is empty, FAIL the test and skip other steps.
9. ONVIF Client invokes **AddConfiguration** request with parameters
   - ProfileToken := *profile*.@token
   - Name skipped
   - Configuration[0].Type := AudioSource
   - Configuration[0].Token := *audioSourceConfigurationList*[0]
10. The DUT responds with **AddConfigurationResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetAudioSourceConfigurationsResponse** message.
- DUT did not send **AddConfigurationResponse** message.

A.20 Get Audio Encoder Configurations List

Name: HelperGetAudioEncoderConfigurationsList

Procedure Purpose: Helper procedure to retrieve Audio Encoder Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Audio configuration is supported by the DUT as indicated by receiving the GetAudioEncoderConfigurationOptionsResponse.

Input: None.

Returns: Audio Encoder Configurations List (*audioEncoderConfList*).

Procedure:

1. ONVIF Client invokes **GetAudioEncoderConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetAudioEncoderConfigurationsResponse** with parameters
   - Configurations list =: *audioEncoderConfList*
3. If *audioEncoderConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioEncoderConfigurationsResponse** message.

A.21 Configure Media profile with Audio Output Configuration

Name: HelperConfigureMediaProfileWithAudioOutput

Procedure Purpose: Helper procedure to configure Media Profile to contain Audio Output Configuration.

Pre-requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability.

Input: None.

Returns: Media Profile (*profile*) containing Audio Output Configuration.

Procedure:

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioOutput

2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList* is empty, FAIL the test and skip other steps.
4. If *profileList* contains Media Profile with Configurations.AudioOutput:
   4.1. Set *profile* := the first profile with Configurations.AudioOutput from the *profileList*.
   4.2. Skip other steps.
5. Set *profile* := *profileList*[0].
6. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken = *profile*.@token
7. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
   - Configurations list =: *audioOutputConfigurationList*
8. If *audioOutputConfigurationList* is empty, FAIL the test and skip other steps.
9. ONVIF Client invokes **AddConfiguration** request with parameters
   - ProfileToken := *profile*.@token
   - Name skipped
   - Configuration[0].Type := AudioOutput
   - Configuration[0].Token := *audioOutputConfigurationList*[0]
10. The DUT responds with **AddConfigurationResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetAudioOutputConfigurationsResponse** message.

- DUT did not send **AddConfigurationResponse** message.

A.22 Get Audio Decoder Configurations List

Name: HelperGetAudioDecoderConfigurationsList

Procedure Purpose: Helper procedure to retrieve Audio Decoder Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Input: None.

Returns: Audio Decoder Configurations List (*audioDecoderConfList*).

Procedure:

1. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters
   - Configurations list =: *audioDecoderConfList*
3. If *audioDecoderConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAudioDecoderConfigurationsResponse** message.

A.23 Configure Media profile with Audio Output Configuration and Audio Decoder Configuration

Name: HelperConfigureMediaProfileWithAudioBackCh

Procedure Purpose: Helper procedure to configure Media Profile to contain Audio Output Configuration and Audio Decoder Configuration.

Pre-requisite: Media2 Service is received from the DUT. Audio Outputs is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioOutput capability. Audio Decoder is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = AudioDecoder capability.

Input: None.

Returns: Media Profile (*profile*) containing Audio Output Configuration and Audio Decoder Configuration.

Procedure:

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := AudioOutput
   - Type[1] := AudioDecoder
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList* is empty, FAIL the test and skip other steps.
4. If *profileList* contains Media Profile with Configurations.AudioOutput and Configurations.AudioDecoder:
   4.1. Set *profile* := the first Media Profile with Configurations.AudioOutput and Configurations.AudioDecoder from the *profileList*.
   4.2. Skip other steps of procedure.
5. If *profileList* contains Media Profile with Configurations.AudioOutput:
   5.1. Set *profile* := the first Media Profile with Configurations.AudioOutput from the *profileList*.
   5.2. Go to step 13
6. Set *profile* := *profileList*[0].
7. ONVIF Client invokes **GetAudioOutputConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken := *profile*.@token

8. The DUT responds with **GetAudioOutputConfigurationsResponse** with parameters
   - Configurations list =: *audioOutputConfigurationList*
9. If *audioOutputConfigurationList* is empty, FAIL the test and skip other steps.
10. ONVIF Client invokes **AddConfiguration** request with parameters
    - ProfileToken := *profile*.@token
    - Name skipped
    - Configuration[0].Type := AudioOutput
    - Configuration[0].Token := *audioOutputConfigurationList*[0]
11. The DUT responds with **AddConfigurationResponse** message.
12. ONVIF Client invokes **GetAudioDecoderConfigurations** request with parameters
    - ConfigurationToken skipped
    - ProfileToken = *profile*.@token
13. The DUT responds with **GetAudioDecoderConfigurationsResponse** with parameters
    - Configurations list =: *audioDecoderConfigurationList*
14. If *audioDecoderConfigurationList* is empty, FAIL the test and skip other steps.
15. ONVIF Client invokes **AddConfiguration** request with parameters
    - ProfileToken := *profileList*.Profiles[0].@token
    - Name skipped
    - Configuration[0].Type := AudioDecoder
    - Configuration[0].Token := *audioDecoderConfigurationList*[0]
16. The DUT responds with **AddConfigurationResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetAudioOutputConfigurationsResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **GetAudioDecoderConfigurationsResponse** message.

A.24 Configure Media profile with Video Source Configuration and Video Encoder Configuration

Name: HelperConfigureMediaProfileWithVideo

Procedure Purpose: Helper procedure to configure Media Profile to contain Video Source Configuration and Video Encoder Configuration.

Pre-requisite: Media2 Service is received from the DUT.

Input: None.

Returns: Media Profile (*profile*) containing Video Source Configuration and Video Encoder Configuration.

Procedure:

1. ONVIF Client invokes **GetProfiles** request with parameters
   - Token skipped
   - Type[0] := VideoSource
   - Type[1] := VideoEncoder
2. The DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList*
3. If *profileList* is empty, FAIL the test and skip other steps.
4. If *profileList* contains Media Profile with Configurations.VideoSource and Configurations.VideoEncoder:
   4.1. Set *profile* := the first Media Profile with Configurations.VideoSource and Configurations.VideoEncoder from the *profileList*.
   4.2. Skip other steps of procedure.

5. If *profileList* contains Media Profile with Configurations.VideoSource:
   5.1. Set *profile* := the first Media Profile with Configurations.VideoSource from the *profileList*.
   5.2. Go to step 13
6. Set *profile* := *profileList*[0].
7. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken := *profile*.@token
8. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters
   - Configurations list =: *videoSourceConfigurationList*
9. If *videoSourceConfigurationList* is empty, FAIL the test and skip other steps.
10. ONVIF Client invokes **AddConfiguration** request with parameters
    - ProfileToken := *profile*.@token
    - Name skipped
    - Configuration[0].Type := VideoSource
    - Configuration[0].Token := *videoSourceConfigurationList*[0]
11. The DUT responds with **AddConfigurationResponse** message.
12. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
    - ConfigurationToken skipped
    - ProfileToken = *profile*.@token
13. The DUT responds with compatible video encoder configurations in **GetVideoEncoderConfigurationsResponse** with parameters
    - Configurations list =: *videoEncoderConfigurationList*
14. If *videoEncoderConfigurationList* is empty, FAIL the test and skip other steps.
15. ONVIF Client invokes **AddConfiguration** request with parameters
    - ProfileToken := *profileList*.Profiles[0].@token

- Name skipped
- Configuration[0].Type := VideoEncoder
- Configuration[0].Token := *videoEncoderConfigurationList*[0]

16. The DUT responds with **AddConfigurationResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetProfilesResponse** message.
- DUT did not send **GetVideoSourceConfigurationsResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.

A.25 Device Configuration to Create OSD with Required Type

Name: HelperDeviceConfigurationToCreateOSD

Procedure Purpose: Helper procedure to configure Device to have free space to create OSD with required type.

Pre-requisite: Media2 Service is received from the DUT. OSD is supported by the DUT.

Input: OSD Configurations List (*osdConfList*), OSD Options (*osdOptions*), OSD Type that will be created (*osdType*).

Returns: Updated OSD Configurations List (*osdConfList*).

Procedure:

1. If *osdType* = "Image":
    1.1. Set *osdImageNumber* := number of OSD Configuration items with Type = "Image".
    1.2. If *osdOptions*.MaximumNumberOfOSDs contains "Image" attribute:
        1.2.1. If *osdImageNumber* = *osdOptions*.MaximumNumberOfOSDs.Image:

1.2.1.1. If *osdConfList* does not contain at least one OSD Configuration with Type = "Image", FAIL the test and skip other steps.

1.2.1.2. Set *osdToken* := *osdConfList*.OSDs[0].token, where OSDs[0] is the first OSD Configuration with Type = "Image".

1.2.1.3. ONVIF Client deletes OSD Configuration with image type by following the procedure mentioned in Annex A.26 with the following input parameter

- in *osdToken* - OSD Configuration token to delete

1.2.1.4. Set *osdConfList* := *osdConfList* - *osdConfList*.OSDs[0].token, where OSDs[0] is the OSD Configuration with token = *osdToken*.

1.2.1.5. Skip other steps.

2. If *osdType* = "Text":

2.1. ONVIF Client deletes all OSD Configurations with Type="Text" by following the procedure mentioned in Annex A.27 with the following input and output parameters

- in *osdConfList* - current OSD Configuration list
- out *osdConfList* - updated OSD Configuration list

3. Set *osdNumber* := number of OSD Configuration items in *osdConfList*.

4. If *osdNumber* = *osdOptions*.MaximumNumberOfOSDs.Total:

4.1. If *osdConfList* does not contain at least one OSD Configuration, FAIL the test and skip other steps.

4.2. Set *osdToken* := *osdConfList*.OSDs[0].token

4.3. ONVIF Client deletes OSD Configuration by following the procedure mentioned in Annex A.26 with the following input parameter

- in *osdToken* - OSD Configuration token to delete

4.4. Set *osdConfList* := *osdConfList* - *osdConfList*.OSDs[0].token, where OSDs[0] is the OSD Configuration with token = *osdToken*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- None.

A.26 Delete OSD

Name: HelperDeleteOSD

Procedure Purpose: Helper procedure to delete OSD Configuration.

Pre-requisite: Media2 Service is received from the DUT. OSD is supported by the DUT.

Input: OSD token (*osdToken*).

Returns: None.

Procedure:

1. ONVIF Client invokes **DeleteOSD** request with parameters
   - OSDToken := *osdToken*
2. DUT responds with **DeleteOSDResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **DeleteOSDResponse** message.

A.27 Delete All Text OSDs

Name: HelperDeleteAllTextOSDs

Procedure Purpose: Helper procedure to delete all OSD Configurations with Type="Text".

Pre-requisite: Media2 Service is received from the DUT. OSD is supported by the DUT.

Input: OSD Configuration list (*osdConfList*).

Returns: Updated OSD Configuration list (*updatedOSDConfList*).

Procedure:

1. For each OSD Configuration with Type="Text" *osdConfig* in *osdConfList* repeat the following steps:

   1.1. ONVIF Client invokes **DeleteOSD** request with parameters

   - OSDToken := *osdConfig*.token

   1.2. DUT responds with **DeleteOSDResponse** message.

   1.3. Set *updatedOSDConfList* := *osdConfList* - *osdConfig*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **DeleteOSDResponse** message.

A.28 OSDConfigurationOptions and OSDConfiguration mapping

The following rules should be used for **CreateOSD** message if *osdOptions* is the OSD Options, which were received in **GetOSDOptionsResponse**:

- OSD.@token := token for new OSD
- OSD.VideoSourceConfigurationToken := Video Source Configurtion token
- OSD.Type := one of the values that are listed in *osdOptions*.Type, for test propose only Text or Image shall be used.
- OSD.Position.Type := one of the values that are listed in *osdOptions*.PositionOption
- If OSD.Position.Type = Custom:
  - OSD.Position.Pos.@x := any value from the range [-1,1]
  - OSD.Position.Pos.@y := any value from the range [-1,1]

otherwise, OSD.Position.Pos skipped

- OSD.Position.Extension skippedd
- If OSD.Type = Text:
  - OSD.TextString.Type := one of the values that are listed in *osdOptions*.TextOption.Type, for test propose only Plain, Date, Time, or DateAndTime shall be used.
  - If OSD.TextString.Type = Date or DateAndTime:
    - OSD.TextString.DateFormat := one of the values that are listed in *osdOptions*.TextOption.DateFormat

    otherwise, OSD.TextString.DateFormat skipped
  - If OSD.TextString.Type = Time or DateAndTime:
    - OSD.TextString.TimeFormat := one of the values that are listed in *osdOptions*.TextOption.TimeFormat

    otherwise, OSD.TextString.TimeFormat skipped
  - If *osdOptions*.TextOption.FontSizeRange is specified:
    - OSD.TextString.FontSize := any value from the range [*osdOptions*.TextOption.FontSizeRange.Min,*osdOptions*.TextOption.FontSizeRange.Max]

    otherwise, OSD.TextString.FontSize skipped
  - If *osdOptions*.TextOption.FontColor and *osdOptions*.TextOption.FontColor.Color is specified:
    - If *osdOptions*.TextOption.FontColor.Color.ColorList is specified:
      - OSD.TextString.FontColor.Color := one of the values that are listed in *osdOptions*.TextOption.FontColor.Color.ColorList
    - If *osdOptions*.TextOption.FontColor.Color.ColorspaceRange is specified:
      - OSD.TextString.FontColor.Color.@Colorspace := Colorspace value of one of the items that are listed in *osdOptions*.TextOption.FontColor.Color.ColorList.ColorspaceRange
      - OSD.TextString.FontColor.Color.@X := value from the range [X.Min,X.Max] of the same item in *osdOptions*.TextOption.FontColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.FontColor.Color.@Colorspace

- OSD.TextString.FontColor.Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *osdOptions*.TextOption.FontColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.FontColor.Color.@Colorspace
- OSD.TextString.FontColor.Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *osdOptions*.TextOption.FontColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.FontColor.Color.@Colorspace

- If *osdOptions*.TextOption.FontColor.Transparent is specified:
  - OSD.TextString.FontColor.@Transparent := any value from the range [*osdOptions*.TextOption.FontColor.Transparent.Min,*osdOptions*.TextOption.FontColor.Transparent.Max]

  otherwise, OSD.TextString.FontColor.@Transparent skipped

otherwise, OSD.TextString.FontColor skipped

- If *osdOptions*.TextOption.BackgroundColor and *osdOptions*.TextOption.BackgroundColor.Color is specified:
  - If *osdOptions*.TextOption.BackgroundColor.Color.ColorList is specified:
    - OSD.TextString.BackgroundColor.Color := one of the values that are listed in *osdOptions*.TextOption.BackgroundColor.Color.ColorList
  - If *osdOptions*.TextOption.BackgroundColor.Color.ColorspaceRange is specified:
    - OSD.TextString.BackgroundColor.Color.@Colorspace := Colorspace value of one of the items that are listed in *osdOptions*.TextOption.BackgroundColor.Color.ColorList.ColorspaceRange
    - OSD.TextString.BackgroundColor.Color.@X := value from the range [X.Min,X.Max] of the same item in *osdOptions*.TextOption.BackgroundColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.BackgroundColor.Color.@Colorspace
    - OSD.TextString.BackgroundColor.Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *osdOptions*.TextOption.BackgroundColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.BackgroundColor.Color.@Colorspace
    - OSD.TextString.BackgroundColor.Color.@Z := value from the range [Z.Min,Z.Max] of the same item in

osdOptions.TextOption.BackgroundColor.Color.ColorList.ColorspaceRange that was used for the OSD.TextString.BackgroundColor.Color.@Colorspace

- If *osdOptions*.TextOption.BackgroundColor.Transparent is specified:
  - OSD.TextString.BackgroundColor.@Transparent := any value from the range [*osdOptions*.TextOption.BackgroundColor.Transparent.Min, *osdOptions*.TextOption.BackgroundColor.Transparent.Max]

  otherwise, SD.TextString.BackgroundColor.@Transparent skipped

otherwise, OSD.BackgroundColor.FontColor skipped

- If OSD.TextString.Type = Plain:
  - OSD.TextString.PlainText := any string value

  otherwise, OSD.TextString.PlainText skipped
- OSD.TextString.Extension skipped

otherwise, OSD.TextString skipped

- If OSD.Type = Image:
  - OSD.Image.ImgPath := one of the values that are listed in *osdOptions*.ImageOption.ImagePath
  - OSD.Image.Extension skipped

  otherwise, OSD.Image skipped
- OSD.Extension skippedd

Note: If OSD.Type = Text, but *osdOptions*.TextOption is skipped, the test will be FAILED.

Note: If OSD.Type = Image, but *osdOptions*.ImageOption is skipped, the test will be FAILED.

Note: If OSD.TextString.Type = Date or DateAndTime, but *osdOptions*.TextOption.DateFormat is skipped, the test will be FAILED.

Note: If OSD.TextString.Type = Time or DateAndTime, but *osdOptions*.TextOption.TimeFormat is skipped, the test will be FAILED.

A.29 OSD Picture File Parameters

ONVIF Device Test Tool uses the picture file to upload on the device with the following patameters:

- File format: PNG
- Size: no more than 1024 bytes
- Width: 16 pixels
- Height: 16 pixels

A.30 Get Metadata Configurations List

Name: HelperGetMetadataConfigurationsList

Procedure Purpose: Helper procedure to retrieve Metadata Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Metadata feature under Media2 Service is supported by the DUT.

Input: None.

Returns: Metadata Configurations List (*metadataConfList*).

Procedure:

1. ONVIF Client invokes **GetMetadataConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetMetadataConfigurationsResponse** with parameters
   - Configurations list =: *metadataConfList*
3. If *metadataConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetMetadataConfigurationsResponse** message.

A.31 Get Analytics Configurations List

Name: HelperGetAnalyticsConfigurationsList

Procedure Purpose: Helper procedure to retrieve Analytics Configurations List.

Pre-requisite: Media2 Service is received from the DUT. Analytics feature under Media2 Service is supported by the DUT.

Input: None.

Returns: Analytics Configurations list (*analyticsConfList*).

Procedure:

1. ONVIF Client invokes **GetAnalyticsConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken skipped
2. The DUT responds with **GetAnalyticsConfigurationsResponse** with parameters
   - Configurations list =: *analyticsConfList*
3. If *analyticsConfList* is empty, FAIL the test.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetAnalyticsConfigurationsResponse** message.

A.32 Device Configuration For Create Mask

Name: HelperDeviceConfigurationForCreateMask

Procedure Purpose: Helper procedure for configuring DUT in order to add a mask.

Pre-requisite: Media2 Service is received from the DUT. Mask feature is supported by the DUT.

Input: None.

Returns: Mask Options (*maskOptions*). Video Source configuration token (*vscToken*). Mask to restore (optional) (*maskToRestore*)

Procedure:

1. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters

- out *videoSourceConfList1* - Video Source Configurations list

2. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:

2.1. ONVIF Client invokes **GetMaskOptions** request with parameters

- ConfigurationToken := *videoSourceConf1*.@Token

2.2. DUT responds with **GetMaskOptionsResponse** message with parameters

- MaskOptions =: *maskOptions*

2.3. If *maskOptions* does not contain MaxMasks attribute, FAIL the test and skip other steps.

2.4. If *maskOptions*.MaxMasks > 0:

2.4.1. Set *vscToken* := *videoSourceConf1*.@Token

2.4.2. ONVIF Client invokes **GetMasks** request with parameters

- Token skipped
- ConfigurationToken := *vscToken*

2.4.3. DUT responds with **GetMasksResponse** message with parameters

- Masks =: *maskConfList1*

2.4.4. Set *amountOfExistingMasks* := amount of Mask items in *maskConfList1*

2.4.5. If *amountOfExistingMasks* = *maskOptions*.MaxMasks

2.4.5.1. Set *maskToDelete* := *maskConfList1*.Masks[0].token

2.4.5.2. ONVIF Client deletes Mask Configuration by following the procedure mentioned in Annex A.33 with the following input parameter

- in *maskToDelete* - Mask Configuration token to delete

2.4.6. Skip other steps of procedure.

3. FAIL the test and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetMaskOptionsResponse** message.
- DUT did not send **GetMasksResponse** message.

A.33 Delete Mask

Name: HelperDeleteMask

Procedure Purpose: Helper procedure to delete specified Mask.

Pre-requisite: Media2 Service is received from the DUT. Mask feature is supported by the DUT.

Input: Mask token (*maskToDelete*).

Returns: None.

Procedure:

1. ONVIF Client invokes **DeleteMask** request with parameters
   - Token := *maskToDelete*
2. DUT responds with **DeleteMaskResponse** message

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **DeleteMaskResponse** message.

A.34 Device Configuration For Get Masks Test Case

Name: HelperDeviceConfigurationForGetMasks

Procedure Purpose: Helper procedure to create mask if no mask exists at Device.

Pre-requisite: Media2 Service is received from the DUT. Mask feature is supported by the DUT.

Input: None.

Returns: Mask to remove (optional) (*maskToRemove*)

Procedure:

1. ONVIF Client invokes **GetMasks** request with parameters
   - Token skipped
   - ConfigurationToken skipped
2. DUT responds with **GetMasksResponse** message with parameters
   - Masks =: *maskConfCompleteList1*
3. If *maskConfCompleteList1* contains at least one item, skip other steps and return to the test.
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.3 with the following input and output parameters:
   - out *videoSourceConfList1* - Video Source Configurations list
5. For each Video Source Configuration *videoSourceConf1* in *videoSourceConfList1* list repeat the following steps:
   5.1. ONVIF Client invokes **GetMaskOptions** request with parameters
      - ConfigurationToken := *videoSourceConf1*.@Token
   5.2. DUT responds with **GetMaskOptionsResponse** message with parameters
      - MaskOptions =: *maskOptions*
   5.3. If *maskOptions* does not contain MaxMasks attribute, FAIL the test and skip other steps.
   5.4. If *maskOptions*.MaxMasks > 0:
      5.4.1. Set *vscToken* := *videoSourceConf1*.@Token
      5.4.2. ONVIF Client invokes **CreateMask** request with parameters
         - token := "TestMask" (note: this token can be ignored by DUT)
         - ConfigurationToken := *vscToken*
         - Polygon :=
            - Point[0].@x := -0.5
            - Point[0].@y := -0.5

- Point[1].@x := -0.5
- Point[1].@y := 0.5
- Point[2].@x := 0.5
- Point[2].@y := 0.5
- Point[3].@x := 0.5
- Point[3].@y := -0.5

- Type := first value from *maskOptions*.Types list
- If Type = "Color":
  - If *maskOptions*.Color.ColorList is specified:
    - Color := first value that are listed in *maskOptions*.Color.ColorList
  - If *maskOptions*.Color.ColorspaceRange is specified:
    - Color.@Colorspace := Colorspace value of the first item that is listed in *maskOptions*.Color.ColorspaceRange
    - Color.@X := value from the range [X.Min,X.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
    - Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
    - Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace
  - Enabled := false

5.4.3. DUT responds with **CreateMaskResponse** message with parameters

- Token =: *maskToRemove*

5.4.4. Skip other steps of procedure.

6. FAIL the test and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetMaskOptionsResponse** message.
- DUT did not send **GetMasksResponse** message.
- DUT did not send **CreateMaskResponse** message.

A.35 Create Mask

Name: HelperCreateMask

Procedure Purpose: Helper procedure for mask creation for specified Video Source Configuration with valid options.

Pre-requisite: Media2 Service is received from the DUT. Mask feature is supported by the DUT.

Input: Video Source configuration token (*vscToken*). Mask Options (*maskOptions*). Type of Mask (*maskType*).

Returns: Token of the created mask (*maskToken*).

Procedure:

1. ONVIF Client invokes **CreateMask** request with parameters
   - token := "TestMask" (note: this token can be ignored by DUT)
   - ConfigurationToken := *vscToken*
   - Polygon := Point[0].@x="-0.4", Point[0].@y="-0.2", Point[1].@x="-0.2", Point[1].@y="0.3", Point[2].@x="0.1", Point[2].@y="0.4", Point[3].@x="0.3", Point[3].@y="-0.3" if *maskOptions*.RectangleOnly is false or skipped. Else Polygon := Point[0].@x="-0.5", Point[0].@y="-0.5", Point[1].@x="-0.5", Point[1].@y="0.5", Point[2].@x="0.5", Point[2].@y="0.5", Point[3].@x="0.5", Point[3].@y="-0.5"
   - Type := *maskType*
   - If Type = “Color”:
     - If *maskOptions*.Color.ColorList is specified:

- Color := first value that are listed in *maskOptions*.Color.ColorList

- If *maskOptions*.Color.ColorspaceRange is specified:

  - Color.@Colorspace := Colorspace value of the first item that is listed in *maskOptions*.Color.ColorspaceRange

  - Color.@X := value from the range [X.Min,X.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace

  - Color.@Y := value from the range [Y.Min,Y.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace

  - Color.@Z := value from the range [Z.Min,Z.Max] of the same item in *maskOptions*.Color.ColorspaceRange that was used for the Color.@Colorspace

- Enabled := false

2. DUT responds with **CreateMaskResponse** message with parameters

   - Token of the created mask =: *maskToken*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **CreateMaskResponse** message.

A.36 Remove all Masks from Video Source Configuration

Name: HelperMasksCleanUp

Procedure Purpose: Helper procedure, which removes all masks from Video Source Configuration.

Pre-requisite: Media2 Service is received from the DUT. Mask feature is supported by the DUT.

Input: Video Source configuration token (*vscToken*).

Returns: None.

Procedure:

1. ONVIF Client invokes **GetMasks** request with parameters
   - Token skipped
   - ConfigurationToken := *vscToken*
2. DUT responds with **GetMasksResponse** message with parameters
   - Masks =: *maskConfList1*
3. For each Mask *mask* in *maskConfList1* list repeat the following steps:
   3.1. ONVIF Client invokes **DeleteMask** request with parameters
      - Token := *mask*.@Token
   3.2. DUT responds with **DeleteMaskResponse** message

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **DeleteMaskResponse** message.
- DUT did not send **GetMasksResponse** message.