ONVIF™

Media2 Configuration Device Test Specification

Version 17.06

June 2017

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 Retrive 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 Retrive 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.

Table of Contents

1 Introduction 14
1.1 Scope 14
1.2 Media Configuration 15
2 Normative references 17
3 Terms and Definitions 19
3.1 Conventions 19
3.2 Definitions 19
3.3 Abbreviations 19
4 Test Overview 20
4.1 Test Setup 20
4.1.1 Network Configuration for DUT 20
4.2 Prerequisites 21
4.3 Test Policy 21
4.3.1 Media Configuration 21
5 Media Configuration Test Cases 23
5.1 Media Profile 23
5.1.1 READY TO USE MEDIA PROFILE FOR VIDEO STREAMING 23
5.1.2 CREATE MEDIA PROFILE WITH PRE-DEFINED CONFIGURATION 24
5.1.3 DYNAMIC MEDIA PROFILE CONFIGURATION 27
5.1.4 GET PROFILES 32
5.1.5 CREATE MEDIA PROFILE WITH CONFIGURATIONS 35
5.2 Video Configuration 37
5.2.1 General 37
5.2.1.1 VIDEO ENCODER INSTANCES 37
5.2.2 Video Source Configuration 40
5.2.2.1 GET VIDEO SOURCE CONFIGURATION OPTIONS 40
5.2.2.2 GET VIDEO SOURCE CONFIGURATIONS 42
5.2.2.3 VIDEO SOURCE CONFIGURATIONS AND VIDEO SOURCE CONFIGURATION OPTIONS CONSISTENCY 44

5.2.2.4 PROFILES AND VIDEO SOURCE CONFIGURATIONS CONSISTENCY 46
5.2.2.5 MODIFY ALL SUPPORTED VIDEO SOURCE CONFIGURATIONS 48
5.2.2.6 GET VIDEO SOURCE CONFIGURATIONS – INVALID TOKEN 54
5.2.3 Video Encoder Configuration 55
5.2.3.1 VIDEO ENCODER CONFIGURATION 55
5.2.3.2 VIDEO ENCODER CONFIGURATIONS AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY 57
5.2.3.3 PROFILES AND VIDEO ENCODER CONFIGURATION OPTIONS CONSISTENCY 58
5.2.3.4 VIDEO ENCODER CONFIGURATIONS – ALL SUPPORTED VIDEO ENCODER CONFIGURATIONS 59
5.2.3.5 VIDEO ENCODER CONFIGURATION OPTIONS 64
5.2.4 Video Source 66
5.2.4.1 GET VIDEO SOURCE MODES 66
5.2.4.2 SET VIDEO SOURCE MODES 67
5.3 Audio Configuration 70
5.3.1 Audio Source Configuration 70
5.3.1.1 GET AUDIO SOURCE CONFIGURATION OPTIONS 70
5.3.1.2 GET AUDIO SOURCE CONFIGURATIONS 71
5.3.1.3 AUDIO SOURCE CONFIGURATIONS AND AUDIO SOURCE CONFIGURATION OPTIONS CONSISTENCY 74
5.3.1.4 PROFILES AND AUDIO SOURCE CONFIGURATIONS CONSISTENCY 75
5.3.1.5 MODIFY ALL SUPPORTED AUDIO SOURCE CONFIGURATIONS 76
5.3.1.6 GET AUDIO SOURCE CONFIGURATIONS – INVALID TOKEN 79
5.3.2 Audio Encoder Configuration 80
5.3.2.1 G.711 AUDIO ENCODER CONFIGURATION 80
5.3.2.2 AAC AUDIO ENCODER CONFIGURATION 84

5.3.3 Audio Output Configuration 89
5.3.3.1 GET AUDIO OUTPUT CONFIGURATION OPTIONS 89
5.3.3.2 GET AUDIO OUTPUT CONFIGURATIONS 90
5.3.3.3 AUDIO OUTPUT CONFIGURATIONS AND AUDIO OUTPUT CONFIGURATION OPTIONS CONSISTENCY 93
5.3.3.4 PROFILES AND AUDIO OUTPUT CONFIGURATIONS CONSISTENCY 94
5.3.3.5 MODIFY ALL SUPPORTED AUDIO OUTPUT CONFIGURATIONS 95
5.3.3.6 GET AUDIO OUTPUT CONFIGURATIONS – INVALID TOKEN 99
5.3.4 Audio Decoder Configuration 100
5.3.4.1 GET AUDIO DECODER CONFIGURATION OPTIONS 100
5.3.4.2 GET AUDIO DECODER CONFIGURATIONS 102
5.3.4.3 PROFILES AND AUDIO DECODER CONFIGURATIONS CONSISTENCY 104
5.3.4.4 MODIFY ALL SUPPORTED AUDIO DECODER CONFIGURATIONS 106
5.3.4.5 GET AUDIO DECODER CONFIGURATIONS – INVALID TOKEN ... 107
5.4 PTZ Configuration 108
5.4.1 READY TO USE MEDIA PROFILE FOR PTZ 108
5.4.2 DYNAMIC MEDIA PROFILE CONFIGURATION FOR PTZ 109
5.5 Media Streaming 116
5.5.1 SNAPSHOT URI 116
5.6 OSD Configuration 118
5.6.1 CREATE OSD CONFIGURATION FOR TEXT OVERLAY 118
5.6.2 CREATE OSD CONFIGURATION FOR IMAGE OVERLAY 121
5.6.3 SET OSD CONFIGURATION IMAGE OVERLAY 124
5.6.4 SET OSD CONFIGURATION TEXT OVERLAY 127
5.7 Capabilities 129
5.7.1 MEDIA2 SERVICE CAPABILITIES 129

5.7.2 GET SERVICES AND GET MEDIA2 SERVICE CAPABILITIES CONSISTENCY 130

A Helper Procedures and Additional Notes 133

A.1 Create Empty Profile 133

A.2 Name Parameters 134

A.3 VideoEncoderConfigurationOptions and VideoEncoderConfiguration mapping 134

A.4 OSDConfigurationOptions and OSDConfiguration mapping 134

A.5 Configure Media profile with Video Source Configuration and Video Encoder Configuration 138

A.6 Configure Media profile with Video Source Configuration 140

A.7 Get Video Source Configurations List 141

A.8 Get Service Capabilities 142

A.9 Get Audio Source Configurations List 143

A.10 Configure Media profile with Audio Source Configuration 143

A.11 Delete Media Profile if Max Reached 145

A.12 Create Pull Point Subscription 146

A.13 Delete Subscription 147

A.14 Retrive Profile Changed Event by PullPoint 147

A.15 Retrive Configuration Changed Event by PullPoint 149

A.16 Get Video Source Configurations List 150

A.17 Get Audio Encoder Configurations List 150

A.18 Get Audio Output Configurations List 151

A.19 Configure Media profile with Audio Output Configuration 152

A.20 Get Audio Decoder Configurations List 153

A.21 Configure Media profile with Audio Output Configuration and Audio Decoder Configuration 154

A.22 Get Video Sources List 156

A.23 Waiting for Reboot 157

A.24 Find Guaranteed Number of Media Profiles for Video Source Configuration 158

A.25 Configure Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration 159

A.26 Add Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration 162
A.27 Create New Media Profiles to Get Guaranted Number of Media Profiles for Video Source Configuration 165
A.28 Remove all non-fixed Media Profiles and remove all configurations from fixed Media Profiles 168

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

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.

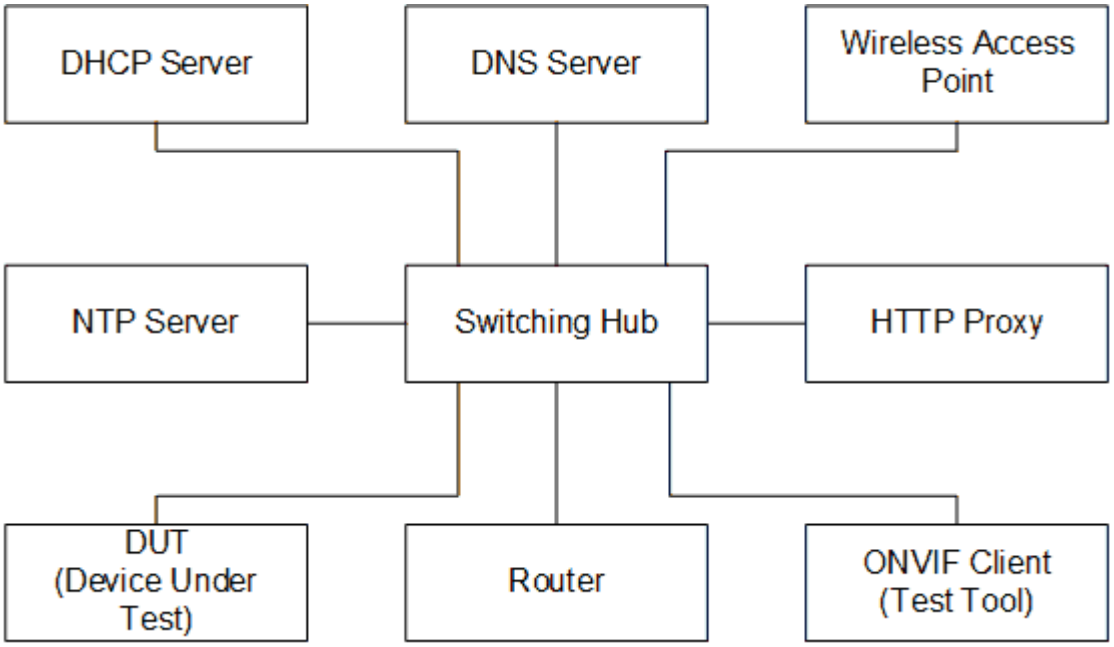
4.1 Test Setup

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.

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.

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.11.
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *videoSourceConfCompleteList* - Video Source Configurations list

5. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *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. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *profileToken* - Media Profile token
18. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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.2 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 retrieves Video Source Configurations list by following the procedure mentioned in Annex A.7 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 creates new Media Profile or removes all configurations from fixed Media Profile by following the procedure mentioned in Annex A.1 with the following input and output parameters
      - out *newProfileFlag* - flag that indicates that new Media Profile was created
      - out *profileToken* - empty Media Profile

4.2. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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

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

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

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

4.5. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *profileToken* - Media Profile token

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

- ConfigurationToken skipped
- ProfileToken = *profileToken*

4.7. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters

- Configurations list =: *videoEncoderConfigurationList*

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

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

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

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

4.11. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *profileToken* - Media Profile token

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

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

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

- Profiles list =: *profileList*

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

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

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

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

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

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

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

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

4.21. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *profileToken* - Media Profile token

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

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

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

- Profiles list =: *profileList*

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

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

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

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

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

- ProfileToken = *profileToken*
- Configuration[0].Type = VideoSource

- Configuration[0].Token = *videoSourceConfiguration*.@token

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

4.30. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile profile 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 *profileToken* - Media Profile token

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

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

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

- Profiles list =: *profileList*

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

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

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

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

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

- Token := *profileToken*

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

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

- Token := *profileToken*
- Type skipped

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

4.37. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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 **DeleteProfileResponse** message.
- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** 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.2 for Name and Token Parameters Length limitations.

5.1.4 GET PROFILES

Test Case ID: MEDIA2-1-1-4

Specification Coverage: Get media profiles (Profile T 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.

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

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.11.
4. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *videoSourceConfCompleteList* - Video Source Configurations list

5. If Media2 Audio is supported, 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. If Media2 Audio Outputs is supported, ONVIF Client retrieves Audio Source Configurations list by following the procedure mentioned in Annex A.18 with the following input and output parameters
   - out *audioOutputConfCompleteList* - Audio Output Configurations list
7. ONVIF Client invokes **CreateProfile** request with parameters
   - Name := "testMedia2"
   - Configuration[0].Type := VideoSource
   - Configuration[0].Token = *videoSourceConfCompleteList*[0].@token
   - If Media2 Audio is supported:
     7.1. Configuration[1].Type := AudioSource
     7.2. Configuration[1].Token = *audioSourceConfCompleteList*[0].@token
   - If Media2 Audio Outputs is supported:
     7.1. Configuration[2].Type := AudioOutput
     7.2. Configuration[2].Token = *audioOutputConfCompleteList*[0].@token
8. The DUT responds with **CreateProfileResponse** with parameters
   - Token =: *profileToken*
9. ONVIF Client invokes **GetProfiles** request with parameters
   - Token := *profileToken*
   - Type[0] := All
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. If Media2 Audio is supported and *profileList*[0].Configurations.AudioSource.@token != *audioSourceConfCompleteList*[0].@token, FAIL the test and skip other steps.
16. If Media2 Audio Outputs is supported and *profileList*[0].Configurations.AudioOutput.@token != *audioOutputConfCompleteList*[0].@token, FAIL the test and skip other steps.
17. ONVIF Client invokes **DeleteProfile** request with parameters
    - Token := *profileToken*
18. The DUT responds with **DeleteProfileResponse** message.

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.

Note: See Annex A.2 for Name and Token Parameters Length limitations.

5.2 Video Configuration

5.2.1 General

5.2.1.1 VIDEO ENCODER INSTANCES

Test Case ID: MEDIA2-2-1-1

Specification coverage: Get video encoder instance information (Media2), Media Profile Management (Profile T)

Feature under test: GetVideoEncoderInstances

WSDL Reference: media2.wsdl

Test Purpose: To verify that for each video source configuration DUT supports creation of as many Media Profiles as the number of instances, which is returned by GetVideoEncoderInstances for that video source configuration token.

Pre-Requisite: Media2 Service is received from the DUT. Profile T 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. DUT responds with **GetProfilesResponse** message with parameters
   - Profiles list =: *profileList1*
5. ONVIF Client retrieves Media2 Service Capabilities by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *cap1* - Media2 Service Capabilities
6. ONVIF Client retrieves Video Source Configurations list by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *videoSourceConfList1* - Video Source Configurations list
7. For each Video Source Configuration *videoSourceConfig1* from *videoSourceConfList1* repeat the following steps:
   7.1. ONVIF Client invokes **GetVideoEncoderInstances** request with parameters
      - ConfigurationToken := *videoSourceConfig1*.token
   7.2. DUT responds with **GetVideoEncoderInstancesResponse** message with parameters
      - Info = *info1*
   7.3. Set *infoList1*[*videoSourceConfig1*.@token] := *info1*.

8. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
   - ConfigurationToken - skipped
   - ProfileToken - skipped
9. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters
   - Configurations list =: *videoEncoderConfList1*
10. Set *numberOfProfilesToBeCreated1* := sum of all Total values from *infoList1* list.
11. Set *numberOfFixedProfiles1* := number of items at *profileList1* list with @fixed = true.
12. Set *numberOfVEC1* := number of items at *videoEncoderConfList1* list.
13. If *numberOfProfilesToBeCreated1* > *cap1*.ProfileCapabilities.MaximumNumberOfProfiles - *numberOfFixedProfiles1*, FAIL the test and skip other steps.
14. If *numberOfProfilesToBeCreated1* > *numberOfVEC1*, FAIL the test and skip other steps.
15. ONVIF Client removes all non-fixed Media Profiles and removes all configurations from fixed Media Profiles by following the procedure mentioned in Annex A.28 with the following input and output parameters
    - in *profileList1* - Media Profiles List
16. For each Video Source Configuration *videoSourceConfig1* from *videoSourceConfList1* repeat the following steps:
    16.1. ONVIF Client tries to create new Media Profiles to get number of profiles equal to info.Total by following the procedure mentioned in Annex A.27 with the following input and output parameters
    - in *videoSourceConfig1* - Video Source Configuration
    - in *infoList1*[*videoSourceConfig1*.@token] - information about guaranteed Encoder instances for Video Source Configuration
    - out *configuredProfilesList1* - list of configured Media Profiles for Video Source Configuration

    16.2. If number of Media Profiles with @token = *videoSourceConfig1*.@token in *configuredProfilesList1* < *infoList1*[*videoSourceConfig1*.@token].Total, then FAIL the test and go to step 17.
17. ONVIF Client restores Media Profiles list if it was changed at steps 15, 16.1.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.2.2 Video Source Configuration

5.2.2.1 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.

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.8 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.7 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.6 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.

5.2.2.2 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.

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

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

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

   - Configurations list =: *videoSourceConfList*

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

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

   7.5. 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.

5.2.2.3 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.

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.8 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.7 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.

5.2.2.4 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.

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

5.2.2.5 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.

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.7 with the following input and output parameters
   - out *videoSourceConfList* - Video Source Configurations list
4. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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:
  - Configuration.Extension.Rotate.Mode := first value from *options*.Extension.Rotate.Mode list
- 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
  - If *options*.Extension.Rotate.DegreeList is not specified:
    - Configuration.Extension.Rotate.Degree := -180

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

5.5. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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:
  - Configuration.Extension.Rotate.Mode := last value from *options*.Extension.Rotate.Mode list
- If Configuration.Extension.Rotate.Mode = ON:
  - 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 with the following input and output parameters
   - in *s* - Subscription reference

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

5.2.2.6 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.

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.7 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
- DUT did not send **GetVideoEncoderInstancesResponse** message.

5.2.3 Video Encoder Configuration

5.2.3.1 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 and Video Encoder Configurations compatible with specific 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 **GetVideoEncoderConfigurations** request with parameters
   - ConfigurationToken - skipped
   - ProfileToken - skipped
4. The DUT responds with **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 **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.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.2.3.2 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.

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.3 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.3.

5.2.3.3 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.

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.3 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.3

5.2.3.4 VIDEO ENCODER CONFIGURATIONS – 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.

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.16 with the following input and output parameters
   - out *videoEncoderConfList* - Video Encoder Configurations list
4. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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 := *videoEncoderConfiguration*.Name
         - 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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 **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 := *videoEncoderConfiguration*.Name
- 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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 **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.

5.3.13. ONVIF Client restores settings of Video Encoder Configuration with @token = *videoEncoderConfiguration*.@token.

6. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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:

- Encoding
- Quality
- Resolution.Height
- Resolution.Width

- GovLength
- RateControl.FrameRateLimit
- RateControl.BitrateLimit

5.2.3.5 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.

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. DUT responds with **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.

5.2.4 Video Source

5.2.4.1 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.22 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.

5.2.4.2 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.22 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.23.

   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.23.

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.10 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. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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.17 with the following input and output parameters

- out *audioEncoderConfList* - Audio Encoder Configurations list

4. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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.17 with the following input and output parameters
   - out *audioEncoderConfList* - Audio Encoder Configurations list
4. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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, do the following steps:
      5.3.1. Set *options* := first item from *optionsList* with Encoding = MP4A-LATM.

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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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

5.3.3 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.18 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.19 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.18 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.18 with the following input and output parameters
   - out *audioOutputConfList* - Audio Output Configurations list
4. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client retrieves and checks **tns1:Media/ConfigurationChanged** event for the specified Configuration by following the procedure mentioned in Annex A.15 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. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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.18 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.20 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 configure Media Profile containing Audio Output Configuration and Audio Decoder Configuration by following the procedure mentioned in Annex A.21 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.20 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.20 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, RemoveConfiguration, GetVideoSourceConfigurations, GetConfigurations, GetCompatibleConfigurations

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. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.12 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. For each Media Profile *profile* in *profileList*, which contains Configurations.VideoSource repeat the following steps:
   6.1. ONVIF Client invokes **GetCompatibleConfigurations** request with parameters
      - ProfileToken := *profileToken*
   6.2. The DUT responds with **GetCompatibleConfigurationsResponse** request with parameters
      - PTZConfiguration list =: *ptzConfigurationList*
   6.3. For each PTZ Configuration *ptzConfiguration* in *ptzConfigurationList* repeat the following steps:
      6.3.1. ONVIF Client invokes **AddConfiguration** request with parameters
         - ProfileToken := *profileToken*

- Name skipped
- Configuration[0].Type := PTZ
- Configuration[0].Token := *ptzConfiguration*.@token

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

6.3.3. If *ptzConfiguration*.@token is differ from *profile*.Configurations.PTZ.@token, ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile 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 *profileToken* - Media Profile token

6.3.4. ONVIF Client invokes **GetConfigurationOptions** request with parameters

- ConfigurationToken := *ptzConfiguration*.@token

6.3.5. DUT responds with **GetConfigurationOptionsResponse** message with parameters

- PTZConfigurationOptions =: *ptzOptions*

6.3.6. ONVIF Client invokes **SetConfiguration** request with parameters

- PTZConfiguration.@token := *ptzConfiguration*.@token
- PTZConfiguration.Name := "TestName1"
- 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

6.3.7. DUT responds with **SetConfigurationResponse** message.

6.3.8. ONVIF Client invokes **GetConfiguration** request with parameters

- PTZConfigurationToken := *ptzConfiguration*.@token

6.3.9. The DUT responds with **GetConfigurationResponse** with parameters

- PTZConfiguration =: *ptzConfiguration*

6.3.10. If *ptzConfiguration* is not equal to PTZConfiguration from step 6.3.6, FAIL the test and skip other steps.

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

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

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

- Profiles list =: *profileList*

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

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

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

6.3.16. If *profileList*[0].Configurations.PTZ.@token != *ptzConfiguration*.@token, FAIL the test and skip other steps.

6.3.17. If *profileList*[0].Configurations.PTZ is not equal to PTZConfiguration from step 6.3.6, FAIL the test and skip other steps.

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

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

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

6.3.20. ONVIF Client retrieves and checks **tns1:Media/ProfileChanged** event for the specified Media Profile 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 *profileToken* - Media Profile token

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

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

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

- Profiles list =: *profileList*

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

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

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

6.3.26. ONVIF Client restores Media Profiles with @token = *profileToken*.

7. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.13 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.

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

Note: The following fields are compared at steps 6.3.17 and 6.3.10:

- 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. ONVIF Client configure Media Profile containing Video Source Configuration and Video Encoder Configuration by following the procedure mentioned in Annex A.5 with the following input and output parameters
   - out *profile* - Media Profile containing Video Source Configuration and Video Encoder Configuration
4. ONVIF Client invokes **GetSnapshotUri** request with parameters
   - ProfileToken := *profile*.@token
5. DUT responds with **GetSnapshotUriResponse** message with parameters
   - Uri =: *snapshotUri*
6. ONVIF Client invokes **HTTP GET** request on the *snapshotUri*.
7. DUT responds with **HTTP 200 OK** message and the single shot JPEG image data.
8. ONVIF Client verifies the JPEG image sent by the DUT.
9. 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.

5.6 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. The maximum number of OSD configurations with Type = Text is not reached for at least one Video Source Configuration with Text OSD support.

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** with configuration token = *VideoSourceConfigurationToken1*. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client saves this list in *OSDConfigurationsList1* variable.

   4.2. ONVIF Client invokes **GetOSDOptions** message with Configuration Token = *VideoSourceConfigurationToken1* to retrieve the OSD configuration options for the specified video source configuration.

4.3. DUT sends the range of configurable values for the given video source configuration in the **GetOSDOptionsResponse** message. ONVIF Client saves options from **GetOSDOptionsResponse** in *OSDOptions1* variable.

4.4. ONVIF Client checks if current Video Source Configuration supports OSD configurations of the Type = Text. Otherwise, skip steps 4.5-4.23 and go to the next Video Source Configuration.

4.5. 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.4.

4.6. 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.

4.7. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken1*.

4.8. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.9. 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 4.5.

4.10. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken1* as input parameter.

4.11. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.12. ONVIF Client invokes **GetOSDs** with OSD Token = *OSDConfigurationToken1* as input parameter to verify that OSD Configuration has been deleted by DUT.

4.13. DUT send the SOAP 1.2 fault message (InvalidArgVal/NoConfig).

4.14. ONVIF Client verifies fault message.

4.15. 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.4.

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 = *OSDConfigurationsList1* 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 the fields of this configuration equal to the fields set at step 4.15.

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*.

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 DUT did not send SOAP 1.2 fault message (InvalidArgVal/NoConfig) for **GetOSDs** with OSD Token of deleted configuration.
- 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. The maximum number of OSD configurations with Type = Image is not reached for at least one Video Source Configuration with Image OSD support. DUT returned at least one image overlay in GetOSDOptions.

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** with configuration token = *VideoSourceConfigurationToken1*. DUT sends **GetOSDsResponse** with a list of OSD Configurations. ONVIF Client saves this list in *OSDConfigurationsList1* variable.
   - 4.2. ONVIF Client invokes **GetOSDOptions** message with Configuration Token = *VideoSourceConfigurationToken1* to retrieve the OSD configuration options for the specified video source configuration.
   - 4.3. DUT sends the range of configurable values for the given video source configuration in the **GetOSDOptionsResponse** message.
   - 4.4. If **GetOSDOptionsResponse** contains at least one item in Image Option list (image overlay), then ONVIF Client saves the options from **GetOSDOptionsResponse** in *OSDOptions1* variable and goes to step 4.6.
   - 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 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.
   - 4.7. 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.
   - 4.8. ONVIF Client verifies that *OSDConfigurationsList1* does not contain configuration with token = *OSDConfigurationToken1*.

4.9. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.10. 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.

4.11. ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurationToken1* as input parameter.

4.12. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

4.13. ONVIF Client invokes **GetOSDs** with OSD Token = *OSDConfigurationToken1* as input parameter to verify that OSD Configuration has been deleted by DUT.

4.14. DUT sends the SOAP 1.2 fault message (InvalidArgVal/NoConfig).

4.15. ONVIF Client verifies fault message.

4.16. 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.

4.17. 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 = *OSDConfigurationToken2* 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 configuration fields equal to the fields set at step in 4.16.

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*.

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 DUT did not send SOAP 1.2 fault message (InvalidArgVal/NoConfig) for **GetOSDs** with OSD Token of deleted configuration.
- 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, DUT returned at least one image overlay in GetOSDOptions. The maximum number of OSD configurations with Type = Image is not reached for at least one Video Source Configuration with Image OSD support.

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** message with Configuration Token = *VideoSourceConfigurationToken1* to retrieve the OSD configuration options for the specified video source configuration.
   4.2. DUT sends the range of configurable values for the given video source configuration in the **GetOSDOptionsResponse** message.
   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** with configuration token = *VideoSourceConfigurationToken1*. DUT sends **GetOSDsResponse** with a list of OSD Configurations.
   4.6. 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.

4.7. 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.

4.8. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.9. 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.

4.10. ONVIF Client changes position parameter in OSDConfigurations1 variable. The Position.Type should be changed to the last item from *OSDOptions1*.PositionOption list.

4.11. ONVIF Client invokes **SetOSD** with OSD = OSDConfigurations1 as input parameter. DUT applies the changes and sends **SetOSDResponse**.

4.12. ONVIF Client invokes **GetOSDs** message with OSD Token = OSDConfigurations1 token as input parameter.

4.13. 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.

4.14. If new OSD Configuration has been created at step 4.5, then ONVIF Client invokes **DeleteOSD** with OSD Token = OSDConfigurations1 token as input parameter.

4.15. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

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. The maximum number of OSD configurations with Type = Text is not reached for at least one Video Source Configuration with Text OSD support.

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 - 4.13:

4.1. ONVIF Client invokes **GetOSDOptions** message with Configuration Token = *VideoSourceConfigurationToken1* to retrieve the OSD configuration options for the specified video source configuration.

4.2. DUT sends the range of configurable values for the given video source configuration in the **GetOSDOptionsResponse** message. ONVIF Client saves options from **GetOSDOptionsResponse** in *OSDOptions1* variable.

4.3. ONVIF Client invokes **GetOSDs** with configuration token = *VideoSourceConfigurationToken1*. DUT sends **GetOSDsResponse** with a list of OSD Configurations.

4.4. Onvif client checks if DUT supports OSD configurations of the type=text. Otherwise, skip all steps and pass the test.

4.5. 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.4.

4.6. 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.

4.7. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurationToken1* as input parameter to retrieve newly created OSD Configuration.

4.8. 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.

4.9. 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. See details of fields mapping in Annex A.4.

4.10. ONVIF Client invokes **SetOSD** with OSD = *OSDConfigurations1* as input parameter. DUT applies the changes and sends **SetOSDResponse**.

4.11. ONVIF Client invokes **GetOSDs** message with OSD Token = *OSDConfigurations1* token as input parameter.

4.12. 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.

4.13. If new OSD Configuration has been created at step 4.5, then ONVIF Client invokes **DeleteOSD** with OSD Token = *OSDConfigurations1* token as input parameter.

4.14. DUT sends **DeleteOSDResponse**, which indicates that DUT has deleted OSD Configuration. ONVIF Client verifies the response.

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.

5.7 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.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetServiceCapabilitiesResponse** 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

Annex A Helper Procedures and Additional Notes

A.1 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 (*profile*). Flag that indicates that new Media Profile was created (*videoSourceConfList*).

Procedure:

1. ONVIF Client invokes **CreateProfile**. If ONVIF Client receives **CreateProfileResponse** response, it gets profile token from **CreateProfileResponse** and returns the profile token to the test, which executed this Annex and skip other tests. If DUT responses with Action/ MaxNVTProfiles, then go to step 2.

2. ONVIF Client invokes **GetServiceCapabilities** message to retrieve maximum number of profiles (MaximumNumberOfProfiles) supported by DUT.

3. ONVIF Client will invoke **GetProfiles** message to retrieve complete profiles list. Verify the **GetProfilesResponse** message from the DUT.

4. If number of media profiles in **GetProfilesResponse** is greater or equal to MaximumNumberOfProfiles then go to step 5. Otherwise, if number of media profiles in **GetProfilesResponse** is less than MaximumNumberOfProfiles, then the test should FAIL.

5. If **GetProfilesResponse** contains profiles with fixed attribute equal to false, then ONVIF Client will invoke **DeleteProfile** message (Profile Token = Token1, where Token1 is the first Profile Token from the **GetProfilesResponse** with fixed = false) to delete Profile. Verify the **DeleteProfileResponse** message from the DUT. Go to step 1.

6. If there are no profiles with fixed = "false", remove all configurations from one fixed profile and use this profile for test. If there are no profiles, skip other steps and fail test.

Note: See Annex A.2 for Name and Token Parameters Length limitations.

A.2 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.3 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.4 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 shell 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 shell 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, SD.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

  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.5 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 **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.6 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.7 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.8 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.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 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.11 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.8 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.12 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.13 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.14 Retrive 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.15 Retrive 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.16 Get Video Source 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 **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.17 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.18 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.19 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.20 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.21 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.22 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.23 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. Until *timeout1* timeout expires, repeat the following steps:

   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

1.2. If *xAddrsList* contains URI address with IPv4 address other than LinkLocal from ONVIF Client subnet, go to step 3.

2. If *timeout1* timeout expires for the step 1 without **Hello** with URI address with not a LinkLocal IPv4 address from ONVIF Client subnet, FAIL the test and skip other steps.

3. ONVIF client waits for 5 seconds after **Hello** was received.

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.24 Find Guaranteed Number of Media Profiles for Video Source Configuration

Name: HelperFindGuaranteedProfiles

Procedure Purpose: Helper procedure, which tries to find already existing Media Profiles to reach number of guaranteed encoder instances without change of already configured profiles.

Pre-requisite: Media2 Service is supported by the DUT.

Input: Information about guaranteed encoder instances (*info*). List of Media Profiles that should not be changed during the procedure (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*). Video Source Configuration (*videoSourceConfig*).

Returns: List of configured media profiles (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*).

Procedure:

1. Set *configuredProfilesListForVSC* := empty.

2. For each Media Profile *profile* from *profileList* repeat the following steps:

   2.1. If number of items in *configuredProfilesListForVSC* is equal to *info*.Total, skip other steps of procedure.

   2.2. If *profile* contains Configurations.VideoSource with @token = *videoSourceConfig*.token and Configurations.VideoEncoder that was not used in Media Profiles from *configuredProfilesList* list:

      2.2.1. Set *encoding* := *profile*.Configurations.VideoEncoder.Encoding.

      2.2.2. If *info*.Codec list contains no items with Encoding = *encoding* or if *info*.Codec list contains an item with Encoding = *encoding* and number of Media Profiles with Configurations.VideoEncoder.Encoding = *encoding* in *configuredProfilesListForVSC* is less then *info*.Codec.Number for this *encoding*:

         2.2.2.1. Add *profile* to *configuredProfilesListForVSC* list.

         2.2.2.2. Add *profile* to *configuredProfilesList* list.

         2.2.2.3. Remove *profile* from *profilesList* list.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- None.

A.25 Configure Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration

Name: HelperFindGuaranteedProfiles2

Procedure Purpose: Helper procedure, which tries to configure Video Encoder Configurations into already existing Media Profiles to reach number of guaranteed encoder instances without change of already configured profiles.

Pre-requisite: Media2 Service is supported by the DUT.

Input: Information about guaranteed encoder instances (*info*). List of Media Profiles that should not be changed during the procedure (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*). Video Source Configuration (*videoSourceConfig*).

Returns: List of configured media profiles (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*).

Procedure:

1. Set *configuredProfilesListForVSC* := items from *configuredProfilesList* that contains Configurations.VideoSource with @token = *videoSourceConfig*.@token.
2. For each Media Profile *profile* from *profileList* repeat the following steps:
   2.1. If number of items in *configuredProfilesListForVSC* is equal to *info*.Total, skip other steps of procedure.
   2.2. If *profile* contains Configurations.VideoSource with @token = *videoSourceConfig*.@token and Configurations.VideoEncoder that was not used in Media Profiles and from *configuredProfilesList* list:
      2.2.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters
         - ConfigurationToken := *profile*.Configurations.VideoEncoder.@token
         - ProfileToken skipped
      2.2.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters
         - Options list =: *optionsList*
      2.2.3. For each Video Encoder Options *vecOptions* in *optionsList* repeat the following steps:
         2.2.3.1. If *info*.Codec list contains no items with Encoding = *vecOptions*.Encoding or if *info*.Codec list contains an item with Encoding = *vecOptions*.Encoding and number of Media Profiles with Configurations.VideoEncoder.Encoding = *vecOptions*.Encoding in *configuredProfilesListForVSC* is less then *info*.Codec.Number for this *vecOptions*.Encoding:
            2.2.3.1.1. Set *options* := *vecOptions*
            2.2.3.1.2. Go to step 2.2.7.

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

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

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

2.2.6. Go to the next Media Profile for the step 2.

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

- Configuration.@token := **profile**.Configurations.VideoEncoder.@token
- Configuration.Name := **profile**.Configurations.VideoEncoder.Name
- Configuration.@GovLength skipped
- Configuration.@Profile skipped
- Configuration.Encoding := **options**.Encoding
- Configuration.Resolution := **options**.ResolutionsAvailable[0]
- Configuration.RateControl skipped
- Configuration.Multicast := **profile**.Configurations.VideoEncoder.Multicast
- Configuration.Quality := **options**.QualityRange.Min

2.2.8. DUT responds with **SetVideoEncoderConfigurationResponse** message.

2.2.9. Add **profile** to **configuredProfilesListForVSC** list.

2.2.10. Add **profile** to **configuredProfilesList** list.

2.2.11. Remove **profile** from **profilesList** list.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.
- DUT did not send **SetVideoEncoderConfigurationResponse** message.

A.26 Add Video Encoder Configuration to Get Guaranteed Number of Media Profiles for Video Source Configuration

Name: HelperFindGuaranteedProfiles3

Procedure Purpose: Helper procedure, which tries to add Video Encoder Configurations into already existing Media Profiles to reach number of guaranteed encoder instances without change of already configured profiles.

Pre-requisite: Media2 Service is supported by the DUT.

Input: Information about guaranteed encoder instances (*info*). List of Media Profiles that should not be changed during the procedure (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*). Video Source Configuration (*videoSourceConfig*).

Returns: List of configured media profiles (*configuredProfilesList*). List of Media Profiles that could be changed (*profilesList*).

Procedure:

1. Set *configuredProfilesListForVSC* := items from *configuredProfilesList* that contains Configurations.VideoSource with @token = *videoSourceConfig*.@token.
2. For each Media Profile *profile* from *profileList* repeat the following steps:
   2.1. If number of items in *configuredProfilesListForVSC* is equal to *info*.Total, skip other steps of procedure.
   2.2. If *profile* contains Configurations.VideoSource with @token = *videoSourceConfig*.@token:
      2.2.1. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
         - ConfigurationToken skipped
         - ProfileToken := *profile*.@token
      2.2.2. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters

- Configurations list =: *videoEncoderConfList*

2.2.3. If *videoEncoderConfList* is empty, FAIL the test and skip other steps.

2.2.4. If *videoEncoderConfList* contais only items that were used in Media Profiles from *configuredProfilesList* list, FAIL the test and skip other steps.

2.2.5. For each Video Encoder Configuration *videoEncoderConf* from *videoEncoderConfList*, which was not used in Media Profiles from *configuredProfilesList* list repeat the following steps:

2.2.5.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters

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

2.2.5.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters

- Options list =: *optionsList*

2.2.5.3. For each Video Encoder Options *vecOptions* in *optionsList* repeat the following steps:

2.2.5.3.1. If *info*.Codec list contains no items with Encoding = *vecOptions*.Encoding or if *info*.Codec list contains an item with Encoding = *vecOptions*.Encoding and number of Media Profiles with Configurations.VideoEncoder.Encoding = *vecOptions*.Encoding in *configuredProfilesListForVSC* is less then *info*.Codec.Number for this *vecOptions*.Encoding:

2.2.5.3.1.1. Set *options* := *vecOptions*

2.2.5.3.1.2. Go to step 2.2.5.5.

2.2.5.4. Go to the next Video Encoder Configuration for the step 2.2.5

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

- Configuration.@token := *videoEncoderConf*.@token
- Configuration.Name := *videoEncoderConf*.Name
- Configuration.@GovLength skipped
- Configuration.@Profile skipped
- Configuration.Encoding := *options*.Encoding
- Configuration.Resolution := *options*.ResolutionsAvailable[0]
- Configuration.RateControl skipped
- Configuration.Multicast := *videoEncoderConf*.Multicast
- Configuration.Quality := *options*.QualityRange.Min

2.2.5.6. DUT responds with **SetVideoEncoderConfigurationResponse** message.

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

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

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

2.2.5.9. Add *profile* to *configuredProfilesListForVSC* list.

2.2.5.10. Add *profile* to *configuredProfilesList* list.

2.2.5.11. Remove *profile* from *profilesList* list.

2.2.5.12. Go to the next Media Profile for the step 2.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.
- DUT did not send **SetVideoEncoderConfigurationResponse** message.

A.27 Create New Media Profiles to Get Guaranted Number of Media Profiles for Video Source Configuration

Name: HelperFindGuaranteedProfiles4

Procedure Purpose: Helper procedure, which tries to create new Media Profiles to reach number of guaranteed encoder instances without changing already configured profiles.

Pre-requisite: Media2 Service is supported by the DUT. Profile T is supported by the DUT.

Input: Information about guaranteed encoder instances (*info*). Video Source Configuration (*videoSourceConfig*). List of configured media profiles (*configuredProfilesList*).

Returns: List of configured Media Profiles (*configuredProfilesList*).

Procedure:

1. Set *configuredProfilesListForVSC1* := empty.
2. If number of items in *configuredProfilesListForVSC1* is equal to *info*.Total, skip other steps of procedure.
3. ONVIF Client invokes **CreateProfile** request with parameters
   - Name := "testMedia"
   - Configuration list - skipped
4. DUT responds with **CreateProfileResponse** message with parameters
   - Token =: *clearProfileToken1*
5. ONVIF Client invokes **GetVideoSourceConfigurations** request with parameters
   - ConfigurationToken skipped
   - ProfileToken := *clearProfileToken1*
6. The DUT responds with **GetVideoSourceConfigurationsResponse** with parameters

- Configurations list =: *videoSourceConfigurationList1*

7. If *videoSourceConfigurationList1* does not contain item with @token = *videoSourceConfig*.@token, FAIL the test and skip other steps.
8. ONVIF Client invokes **AddConfiguration** request with parameters
   - ProfileToken := *clearProfileToken1*
   - Name skipped
   - Configuration[0].Type := VideoSource
   - Configuration[0].Token := *videoSourceConfig*.@token
9. The DUT responds with **AddConfigurationResponse** message.
10. ONVIF Client invokes **GetVideoEncoderConfigurations** request with parameters
    - ConfigurationToken skipped
    - ProfileToken := *clearProfileToken1*
11. The DUT responds with **GetVideoEncoderConfigurationsResponse** with parameters
    - Configurations list =: *videoEncoderConfList1*
12. If *videoEncoderConfList1* is empty, FAIL the test and skip other steps.
13. If *videoEncoderConfList1* contais only items that were used in Media Profiles from *configuredProfilesList* list, FAIL the test and skip other steps.
14. For each Video Encoder Configuration *videoEncoderConf1* from *videoEncoderConfList1*, which was not used in Media Profiles from *configuredProfilesList* list repeat the following steps:
    14.1. ONVIF Client invokes **GetVideoEncoderConfigurationOptions** request with parameters
    - ConfigurationToken := *videoEncoderConf1*.@token
    - ProfileToken skipped

    14.2. DUT responds with **GetVideoEncoderConfigurationOptionsResponse** message with parameters
    - Options list =: *optionsList1*

14.3. For each Video Encoder Options *vecOptions1* in *optionsList1* repeat the following steps:

14.3.1. If *info*.Codec list contains no items with Encoding = *vecOptions1*.Encoding or if *info*.Codec list contains an item with Encoding = *vecOptions1*.Encoding and number of Media Profiles with Configurations.VideoEncoder.Encoding = *vecOptions1*.Encoding in *configuredProfilesListForVSC1* is less then *info*.Codec.Number for this *vecOptions1*.Encoding:

14.3.1.1. Set *options1* := *vecOptions1*

14.3.1.2. Set *videoEncoderConfToAdd1* := *videoEncoderConf1*

14.3.1.3. Go to step 16.

14.4. Go to the next Video Encoder Configuration for the step 14

15. FAIL the test and skip other steps.

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

- Configuration.@token := *videoEncoderConfToAdd1*.@token
- Configuration.Name := *videoEncoderConfToAdd1*.Name
- Configuration.@GovLength skipped
- Configuration.@Profile skipped
- Configuration.Encoding := *options1*.Encoding
- Configuration.Resolution := *options1*.ResolutionsAvailable[0]
- Configuration.RateControl skipped
- Configuration.Multicast := *videoEncoderConfToAdd1*.Multicast
- Configuration.Quality := *options1*.QualityRange.Min

17. DUT responds with **SetVideoEncoderConfigurationResponse** message.

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

- ProfileToken := *clearProfileToken1*
- Name skipped
- Configuration[0].Type := VideoEncoder

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

19. The DUT responds with **AddConfigurationResponse** message.
20. Add Media Profile with @token = *clearProfileToken1* to *configuredProfilesListForVSC* list.
21. Add Media Profile with @token = *clearProfileToken1* to *configuredProfilesList* list.
22. Go to step 2.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **AddConfigurationResponse** message.
- DUT did not send **SetVideoEncoderConfigurationResponse** message.
- DUT did not send **GetVideoEncoderConfigurationOptionsResponse** message.
- DUT did not send **GetVideoEncoderConfigurationsResponse** message.
- DUT did not send **GetVideoSourceConfigurationsResponse** message.
- DUT did not send **CreateProfileResponse** message.

A.28 Remove all non-fixed Media Profiles and remove all configurations from fixed Media Profiles

Name: HelperMediaProfilesCleanUp

Procedure Purpose: Helper procedure, which removes all non-fixed Media Profiles and removes all configurations from fixed Media Profiles.

Pre-requisite: Media2 Service is supported by the DUT.

Input: Media Profiles List (*profileList*).

Returns: None.

Procedure:

1. For each Media Profile *profile1* in *profileList* repeat the following steps:

1.1. If *profile1*.@fixed = true:

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

- ProfileToken := *profile1*.@token
- Configuration[0].Type := All
- Configuration[0].Token skipped

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

1.2. If *profile1*.@fixed = false or skipped:

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

- Token := *profile1*.@token

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

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **DeleteProfileResponse** message.
- DUT did not send **RemoveConfigurationResponse** message.