ONVIF™

Device IO Device Test Specification

Version 18.06

June 2018

REVISION HISTORY

Vers.	Date	Description
16.06	Mar 16, 2016	Original publication
16.06	Mar 30, 2016	The tests DEVICEIO-3-1-1 - DEVICEIO-3-1-4 have been added.
16.06	Apr 6, 2016	The DEVICEIO-1-2-1 - DEVICEIO-1-2-4 and DEVICEIO-3-1-4 have been updated according to feedback from Sano Hiroyuki.
16.06	Apr 11, 2016	Implemented comments from Sano Hiroyuki about redundant event-testing steps. Implemented comments of Bhetanabottla Sriram from Canon.
16.07	Jun 22, 2016	Minor spellcheck and version number correction
16.07	Jul 13, 2016	Tests sequences updated.
16.07	Jul 27, 2016	Implemented comments from Canon and Sony
16.07	Aug 5, 2016	Bugfixes based on comments from Hiroyuki Sano
16.12	Dec 08, 2016	REALTIME PULLPOINT SUBSCRIPTION – DIGITAL INPUT EVENT was moved from ONVIF Base Test specification to ONVIF Device IO Test specification Test section 'Digital Input Configuration' was renamed to 'Digital Input'
16.12	Dec 15, 2016	The following test cases were updated: IO SETRELAYOUTPUTSTATE – BISTABLE MODE (OPENED IDLE STATE) IO SETRELAYOUTPUTSTATE – BISTABLE MODE (CLOSED IDLE STATE) IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (OPENED IDLE STATE) IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (CLOSED IDLE STATE)
17.01	Jan 16, 2017	Minor changes based on Sony review.
17.06	Jan 31, 2017	DEVICE IO TRIGGER EVENT test was added according to #1279 The following test cases were updated according to #1279: IO SETRELAYOUTPUTSTATE – BISTABLE MODE (OPENED IDLE STATE) IO SETRELAYOUTPUTSTATE – BISTABLE MODE (CLOSED IDLE STATE) IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (OPENED IDLE STATE) IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (CLOSED IDLE STATE)
17.06	Mar 27, 2017	DEVICEIO-4-1-1 IO GET AUDIO SOURCES added according to #1349.

17.06	Mar 27, 2017	Current document name was changed from Device IO Test Specification to Device IO Device Test Specification. The document formating were updated.
17.12	Jul 11, 2017	The following test cases and annexes were added according #1402: DEVICEIO-5-1-1 GET VIDEOSOURCES (DeviceIO) AND GET VIDEOSOURCES (Media) CONSISTENCY
17.12	Jul 11, 2017	The following test cases were added according #1178: DEVICEIO-6-1-1 IO GET SERIAL PORTS DEVICEIO-6-1-2 IO GET SERIAL PORT CONFIGURATION AND GET SERIAL PORT OPTIONS DEVICEIO-6-1-3 IO MODIFY SERIAL PORT CONFIGURATION DEVICEIO-6-1-4 IO RECEIVE SERIAL COMMAND DEVICEIO-6-1-5 IO SEND SERIAL COMMAND DEVICEIO-6-1-6 IO GETSERIALPORTCONFIGURATION COMMAND - INVALID TOKEN DEVICEIO-6-1-7 IO GETSERIALPORTCONFIGURATIONOPTIONS COMMAND - INVALID TOKEN DEVICEIO-6-1-8 IO SETSERIALPORTCONFIGURATION COMMAND - INVALID SETTINGS Annex A.7 Get Device IO Service Capabilities Annex A.8 Get Relay Outputs List
17.12	Jul 12, 2017	The following test cases were changed according to #1403 DEVICEIO-1-2-1 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (OPENED IDLE STATE) DEVICEIO-1-2-2 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (CLOSED IDLE STATE) DEVICEIO-1-2-3 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (OPENED IDLE STATE) DEVICEIO-1-2-4 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (CLOSED IDLE STATE) DEVICEIO-2-1-2 DEVICE IO TRIGGER EVENT
17.12	Jul 13, 2017	The following test cases were changed according to #1406 DEVICEIO-3-1-3 IOGET DIGITAL INPUT CONFIGURATION OPTIONS DEVICEIO-3-1-4 IO DIGITAL INPUT CONFIGURATION
17.12	Oct 30, 2017	The following test case was added according to #1496: DEVICEIO-7-1-1 IO GET VIDEO SOURCES The following test case was added according to #1502:

		DEVICEIO-8-1-1 IO GET AUDIO OUTPUTS
18.06	Jan 11, 2018	The following test case was updated according to #1528: DEVICEIO-1-1-4 IO SETRELAYOUTPUTSETTINGS
18.06	Jan 12, 2018	The following test case was updated according to #1565: DEVICEIO-2-1-1 REALTIME PULLPOINT SUBSCRIPTION – DIGITAL INPUT EVENT
18.06	Jan 24, 2018	The following test case was updated according to #1567: DEVICEIO-1-1-3 IO GETRELAYOUTPUTOPTIONS (pre-requisite updated) DEVICEIO-1-1-4 IO SETRELAYOUTPUTSETTINGS (pre-requisite updated) DEVICEIO-1-2-1 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (OPENED IDLE STATE) (pre-requisite updated) DEVICEIO-1-2-2 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (CLOSED IDLE STATE) (pre-requisite updated) DEVICEIO-1-2-3 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (OPENED IDLE STATE) (pre-requisite updated) DEVICEIO-1-2-4 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (CLOSED IDLE STATE) (pre-requisite updated) Annex A.3 Select Relay Output with supporting of required RelayMode (pre-requisite updated)
18.06	Jun 21, 2018	Reformatting document using new template

Table of Contents

1 Introduction

The goal of the ONVIF test specification set is to make it possible to realize fully interoperable IP physical security implementation from different vendors. The set of ONVIF test specification describes the test cases need to verify the [ONVIF DeviceIO Service Specs] and [ONVIF Conformance] requirements. It also describes the test framework, test setup, pre-requisites, test policies needed for the execution of the described test cases.

This ONVIF Device IO Test Specification acts as a supplementary document to the [ONVIF DeviceIO Service Specs], illustrating test cases need to be executed and passed. And also 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 Device IO 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 Device IO Service which is defined in [ONVIF DeviceIO Service Specs].

The principal intended purposes are:

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

This specification does not address the following:

1. Product use cases and non-functional (performance and regression) testing.
2. SOAP Implementation Interoperability test i.e. Web Service Interoperability Basic Profile version 2.0 (WS-I BP 2.0).
3. Network protocol implementation Conformance test for HTTP, HTTPS, RTP protocol.
4. Wi-Fi Conformance test

The set of ONVIF Test Specification will not cover the complete set of requirements as defined in [ONVIF DeviceIO Service Specs]; instead it would cover subset of it. The scope of this specification is to derive all the normative requirements of [ONVIF DeviceIO Service Specs] which are related to ONVIF Device IO Service and some of the optional requirements.

This ONVIF DeviceIO Test Specification covers Device IO 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.

1.1.1 Relay Outputs

Relay Outputs section covers the test cases needed for the verification of Relay Outputs service features as mentioned in [ONVIF DeviceIO Service Specs]. The DeviceIO service is used to retrieve and configure the settings of physical outputs of a device.

Briefly it covers the following things:

1. Manage Relay Output Configuration
2. Change Relay Output State

1.1.2 Digital Inputs

Digital Inputs section covers the test cases needed for the verification of Digital Inputs service features as mentioned in [ONVIF DeviceIO Service Specs]. The DeviceIO service is used to retrieve and configure the settings of physical inputs of a device.

Briefly it covers the following thing.

1. Configure Digital Input idle state.

1.1.3 Video Source

Video Source section covers the test cases needed for the verification of Get Video Sources command as mentioned in [ONVIF DeviceIO Service Specs]. The DeviceIO service is used to retrieve video source list from a device.

1.1.4 Audio Output

Audio Output section covers the test cases needed for the verification of Get Audio Outputs command as mentioned in [ONVIF DeviceIO Service Specs]. The DeviceIO service is used to retrieve audio output list from a device.

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 Specifications:

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

- [ONVIF DeviceIO Service Specs] ONVIF DEvice IO Specifications:

  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 defines terms that are specific to the ONVIF Device IO Service and tests. For a list of applicable general terms and definitions, please see [ONVIF Base Test].

Relay Output physical outputs of a device.

Digital Input physical inputs of a device

4 Test Overview

This section provides information the test setup procedure and required prerequisites, and the test policies that should be followed for test case execution.

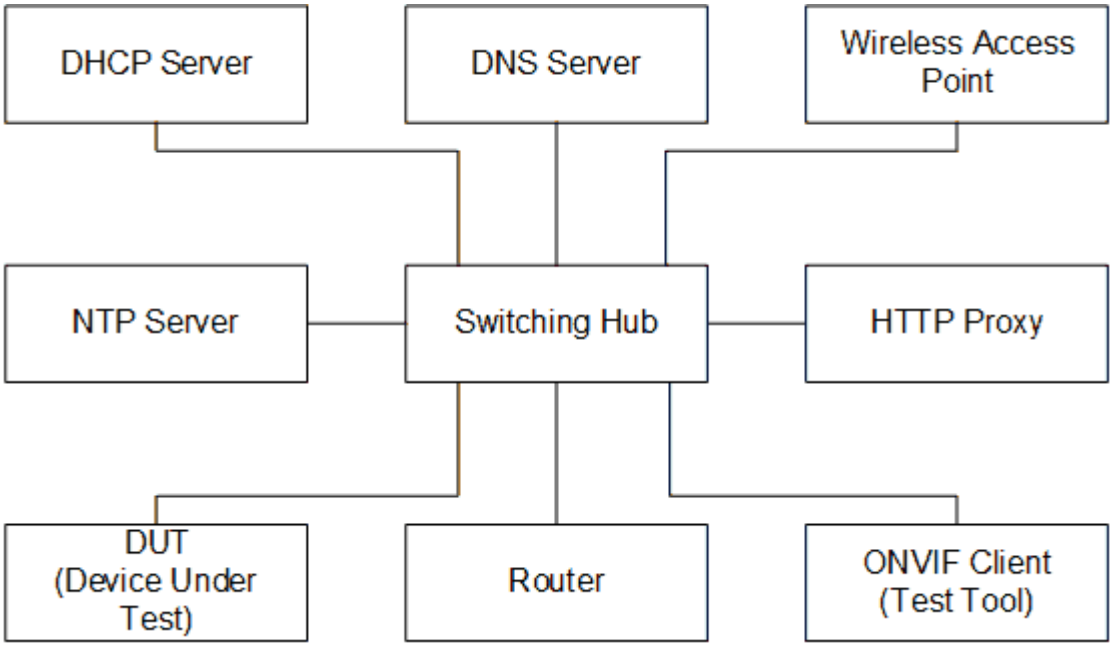
4.1 Test Setup

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 4.1).

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 the 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 a 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 Relay Output

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 5.1 and Section 5.2 for Relay Output Test Cases.

4.3.2 Events

If DUT supports Digital Inputs feature, DUT should support tns1:Device/Trigger/DigitalInput event.

Please refer to Section 5.3 for Digital Input Test Cases.

4.3.3 Digital Input

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 5.4 for Digital Input Test Cases.

4.3.4 Digital Input

DUT should respond with proper response message for all SOAP actions.

Please refer to Section 5.5 for Digital Input Test Cases.

4.3.5 Video Source

DUT shall support the following commands:

- GetVideoSources

Please refer to Section 5.8 for Video Source Test Cases.

4.3.6 Audio Output

If DUT supports Media2 Audio Output, then DUT shall support the following commands:

- GetAudioOutputs

Please refer to Section 5.9 for Audio Output Test Cases.

5 Device IO Test Cases

5.1 Relay Output

5.1.1 IO GETRELAYOUTPUTS

Test Case ID: DEVICEIO-1-1-1

Specification Coverage: None

Feature Under Test: GetRelayOutputs

WSDL Reference: deviceio.wsdl

Test Purpose: To retrieve DUT relay outputs using GetRelayOutputs command.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetRelayOutputs** request to retrieve relay outputs supported by the DUT.
4. Verify the **GetRelayOutputsResponse** message from the DUT.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetRelayOutputsResponse** message.
- The DUT did not send valid **GetRelayOutputsResponse** message.
- The DUT sent at least two RelayOutputs with the same token.

5.1.2 IO GETRELAYOUTPUTS – VERIFY QUANTITY

Test Case ID: DEVICEIO-1-1-2

Specification Coverage: None

Feature Under Test: GetRelayOutputs, GetServiceCapabilities

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the number of Relay outputs from **GetRelayOutputsResponse** message.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServiceCapabilities** request.
4. DUT sends **GetServiceCapabilitiesResponse** message. ONVIF Client verifies the response.
5. ONVIF Client invokes **GetRelayOutputs** request to retrieve relay outputs supported by the DUT.
6. DUT sends **GetRelayOutputsResponse** message with a list of relay outputs supported.
7. Verify the **GetRelayOutputsResponse** message from the DUT.
8. Verify the number of Relay Outputs in **GetRelayOutputsResponse** message. This number should be equal to the Capabilities.RelayOutputs number in **GetServiceCapabilitiesResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetServiceCapabilitiesResponse** message.

- The DUT did not send valid **GetServiceCapabilitiesResponse** message.
- The DUT did not send **GetRelayOutputsResponse** message.
- The DUT did not send valid **GetRelayOutputsResponse** message.
- The number of Relay Outputs in **GetRelayOutputsResponse** message is not equal to Device.IO.RelayOutputs number from **GetServiceCapabilitiesResponse** message.

5.1.3 IO GETRELAYOUTPUTOPTIONS

Test Case ID: DEVICEIO-1-1-3

Specification Coverage: None

Feature Under Test: GetRelayOutputs, GetRelayOutputOptions

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of GetRelayOutputOptions command.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetRelayOutputs** request to retrieve a list of all available relay outputs and their settings
4. The DUT sends the **GetRelayOutputsResponse** message with list of all available relay outputs and their settings.
5. ONVIF Client verifies the **GetRelayOutputsResponse** message from the DUT.
6. ONVIF Client selects first relay output from **GetRelayOutputsResponse** message, saves this relay output in *RelayOutput1* variable. Then it runs the following steps:
   6.1. ONVIF Client invokes **GetRelayOutputOptions** request RelayOutputToken = *RelayOutput1* token as input parameter.
   6.2. The DUT sends **GetRelayOutputOptionsResponse**.

6.3. ONVIF client verifies the **GetRelayOutputOptionsResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetRelayOutputsResponse** message.
- The DUT did not send valid **GetRelayOutputsResponse** message.
- The DUT sent an empty list of RelayOutputs in **GetRelayOutputsResponse** message.
- The DUT did not send **GetRelayOutputOptionsResponse** message.
- The DUT did not send valid **GetRelayOutputOptionsResponse** message.

5.1.4 IO SETRELAYOUTPUTSETTINGS

Test Case ID: DEVICEIO-1-1-4

Specification Coverage: Get relay outputs (Device IO), Get relay output options (Device IO), Set relay output settings (Device IO)

Feature Under Test: GetRelayOutputs, SetRelayOutputSettings, GetRelayOutputOptions

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of SetRelayOutputSettings command.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs is supported by DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetRelayOutputs** request to retrieve a list of all available relay outputs and their settings.

4. The DUT sends the **GetRelayOutputsResponse** message with parameters
   - RelayOutputs list =: *relayOutputsList*
5. If *relayOutputsList* is empty, FAIL the test and skip other steps.
6. For each relay output *relayOutput1* in *relayOutputsList* list repeat the following steps:
   - 6.1. ONVIF Client invokes **GetRelayOutputOptions** request with parameters
     - RelayOutputToken := *relayOutput1*.@token
   - 6.2. The DUT responds with **GetRelayOutputOptionsResponse** message with parameters
     - RelayOutputOptions list := *relayOutputOptionsList*
   - 6.3. If *relayOutputOptionsList* is empty, FAIL the test and skip other steps.
   - 6.4. If *relayOutputOptionsList* contains more then one item, FAIL the test and skip other steps.
   - 6.5. If *relayOutputOptionsList*[0].@token != *relayOutput1*.@token, FAIL the test and skip other steps.
   - 6.6. If *relayOutputOptionsList*[0].Mode list contains item equal to Bistable:
     - 6.6.1. ONVIF Client invokes **SetRelayOutputSettings** request with parameters
       - RelayOutput.@token := *relayOutput1*.@token
       - RelayOutput.Properties.Mode := Bistable
       - RelayOutput.Properties.DelayTime := *relayOutput1*.Properties.DelayTime
       - RelayOutput.Properties.IdleState := open
     - 6.6.2. The DUT responds with **SetRelayOutputSettingsResponse** message.
     - 6.6.3. ONVIF Client invokes **GetRelayOutputs** request to retrieve a list of all available relay outputs and their settings.
     - 6.6.4. The DUT sends the **GetRelayOutputsResponse** message with parameters
       - RelayOutputs list =: *updatedRelayOutputsList*
     - 6.6.5. If *updatedRelayOutputsList* do not contiens item with @token = *relayOutput1*.@token, FAIL the test and skip other steps.

6.6.6. Set *updatedRelayOutput1* := item from *updatedRelayOutputsList* list with @token = *relayOutput1*.@token.

6.6.7. If *updatedRelayOutput1*.Properties.Mode != Bistable, FAIL the test and skip other steps.

6.6.8. If *updatedRelayOutput1*.Properties.IdleState != open, FAIL the test and skip other steps.

6.7. If *relayOutputOptionsList*[0].Mode list contains item equal to Monostable:

6.7.1. If *relayOutputOptionsList*[0].DelayTimes is skipped, FAIL the test and skip other steps.

6.7.2. If *relayOutputOptionsList*[0].Discrete is skipped or equal to false:

6.7.2.1. If *relayOutputOptionsList*[0].DelayTimes list do not contains two items, FAIL the test and skip other steps.

6.7.2.2. If *relayOutputOptionsList*[0].DelayTimes[0] > *relayOutputOptionsList*[0].DelayTimes[1], FAIL the test and skip other steps.

6.7.2.3. Set *delayTime* := nearest to 5 seconds value from the range between *relayOutputOptionsList*[0].DelayTimes[0] and *relayOutputOptionsList*[0].DelayTimes[1].

6.7.3. If *relayOutputOptionsList*[0].Discrete = true:

6.7.3.1. If *relayOutputOptionsList*[0].DelayTimes list do not contains at least one item, FAIL the test and skip other steps.

6.7.3.2. Set *delayTime* := nearest to 5 seconds value from *relayOutputOptionsList*[0].DelayTimes list.

6.7.4. ONVIF Client invokes **SetRelayOutputSettings** request with parameters

- RelayOutput.@token := *relayOutput1*.@token
- RelayOutput.Properties.Mode := Monostable
- RelayOutput.Properties.DelayTime := *delayTime*
- RelayOutput.Properties.IdleState := closed

6.7.5. The DUT responds with **SetRelayOutputSettingsResponse** message.

6.7.6. ONVIF Client invokes **GetRelayOutputs** request to retrieve a list of all available relay outputs and their settings.

6.7.7. The DUT sends the **GetRelayOutputsResponse** message with parameters

- RelayOutputs list =: *updatedRelayOutputsList*

6.7.8. If *updatedRelayOutputsList* do not contiens item with @token = *relayOutput1*.@token, FAIL the test and skip other steps.

6.7.9. Set *updatedRelayOutput1* := item from *updatedRelayOutputsList* list with @token = *relayOutput1*.@token.

6.7.10. If *updatedRelayOutput1*.Properties.Mode != Monostable, FAIL the test and skip other steps.

6.7.11. If *updatedRelayOutput1*.Properties.IdleState != closed, FAIL the test and skip other steps.

6.7.12. If *updatedRelayOutput1*.Properties.DelayTime != *delayTime*, FAIL the test and skip other steps.

6.8. ONVIF Client invokes **SetRelayOutputSettings** request with parameters

- RelayOutput := *relayOutput1*

6.9. The DUT responds with **SetRelayOutputSettingsResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetRelayOutputsResponse** message.
- The DUT did not send **GetRelayOutputOptionsResponse** message.
- The DUT did not send **SetRelayOutputSettingsResponse** message.

5.1.5 IO SETRELAYOUTPUTSETTINGS – INVALID TOKEN

Test Case ID: DEVICEIO-1-1-5

Specification Coverage: None

Feature Under Test: SetRelayOutputSettings

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior ofSetRelayOutputSettings command in case of invalid token.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **SetRelayOutputSettings** request with RelayOutput token = "OnvifTest123".
4. The DUT returns **env:Sender/ter:InvalidArgVal/ter:RelayToken** SOAP 1.2 fault.
5. ONVIF Client verifies fault message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send SOAP 1.2 fault message.
- The DUT sent incorrect SOAP 1.2 fault message (fault code, namespace, etc.).

5.2 Relay Output State

5.2.1 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (OPENED IDLE STATE)

Test Case ID: DEVICEIO-1-2-1

Specification Coverage: None

Feature Under Test: GetRelayOutputs, SetRelayOutputSettings, SetRelayOutputState

WSDL Reference: deviceio.wsdl, event.wsdl

Test Purpose: To verify the behavior of SetRelayOutputState command in the case of bistable mode and opened idle state as well as appropriate event messaging.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT. Profile T is supported by the DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Relay Outputs list by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *relayOutputsList* - Relay Outputs list
4. If *relayOutputsList* is empty, FAIL the test and skip other steps.
5. ONVIF Client selects the first relay output which supports required Relay Output mode by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - in *relayOutputsList* - Relay Outputs list
   - in Bistable – required Relay Output Mode
   - out *relayOutput* - Relay Output (optional)
6. If *relayOutput* is null, PASS the test and skip other steps.
7. Set the following:
   - *initialRelayOutput* := *relayOutput*
8. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Device/Trigger/Relay"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time

9. Until *timeout1* timeout expires, repeat the following steps:

   9.1. ONVIF Client invokes **PullMessages** request with parameters

   - Timeout := PT20S
   - MessageLimit := 1

   9.2. The DUT responds with **PullMessagesResponse** message with parameters

   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
   - NotificationMessage list =: *notificationMessageList*

   9.3. If *notificationMessageList* contains notification (notification) with the following parameters:

   - Topic = "tns1:Device/Trigger/Relay",
   - PropertyOperation = Initialized,
   - Source.SimpleItem.Name = "RelayToken",
   - Source simple item Token value equals to *relayOutput*.@token,
   - Data.SimpleItem.Name = "LogicalState"

   9.3.1. Set the following:

   - *initialLogicalState* := value of LogicalState data simple item in *notification*

   9.3.2. Go to step 10.

   9.4. If *timeout1* timeout expires for step 9 without Notification described at step 9.3, FAIL the test and skip other steps.

10. ONVIF Client invokes **SetRelayOutputSettings** request with parameters

- RelayOutput.@token := *relayOutput*.@token
- RelayOutput.Properties.Mode := Bistable
- RelayOutput.Properties.DelayTime := *relayOutput*.Properties.DelayTime
- RelayOutput.Properties.IdleState := open

11. The DUT responds with **SetRelayOutputSettingsResponse** message.

12. ONVIF Client invokes **SetRelayOutputState** request with parameters
    - RelayOutputToken := *relayOutput*.@token
    - LogicalState := active if *initialLogicalState* = inactive, otherwise inactive
13. The DUT responds with **SetRelayOutputStateResponse** message.
14. Until *timeout1* timeout expires, repeat the following steps:
    14.1. ONVIF Client invokes **PullMessages** request with parameters
    - Timeout := PT20S
    - MessageLimit := 1

    14.2. The DUT responds with **PullMessagesResponse** message with parameters
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
    - NotificationMessage list =: *notificationMessageList*

    14.3. If *notificationMessageList* contains notification (*notification*) with the following parameters:
    - Topic = "tns1:Device/Trigger/Relay",
    - PropertyOperation = Changed,
    - Source.SimpleItem.Name = "RelayToken",
    - Source simple item Token value equals to *relayOutput*.@token

    14.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

    14.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 12, FAIL the test and skip other steps.

    14.3.3. Go to step 15.

    14.4. If *timeout1* timeout expires for step 14 without Notification described at step 14.3, FAIL the test and skip other steps.
15. ONVIF Client invokes **SetRelayOutputState** request with parameters

- RelayOutputToken := *relayOutput*.@token
- LogicalState := *initialLogicalState*

16. The DUT responds with **SetRelayOutputStateResponse** message.
17. Until *timeout1* timeout expires, repeat the following steps:
    17.1. ONVIF Client invokes **PullMessages** request with parameters
    - Timeout := PT20S
    - MessageLimit := 1

    17.2. The DUT responds with **PullMessagesResponse** message with parameters
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
    - NotificationMessage list =: *notificationMessageList*

    17.3. If *notificationMessageList* contains the following notification (*notification*):
    - Topic = "tns1:Device/Trigger/Relay",
    - PropertyOperation = Changed,
    - Source.SimpleItem.Name = "RelayToken",
    - Source simple item Token value equals to *relayOutput*.@token,

    17.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

    17.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 15, FAIL the test and skip other steps.

    17.3.3. Go to step 18.

    17.4. If *timeout1* timeout expires for step 17 without Notification described at step 17.3, FAIL the test and skip other steps.
18. ONVIF Client restores initial Relay Output settings by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *initialRelayOutput* – Relay Output to restore

19. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.5 with the following input and output parameters
    - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.
- The **GetRelayOutputOptionsResponse** message did not contain option with Mode = Bistable

FAIL –

- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **SetRelayOutputSettingsResponse** message.
- The DUT did not send **SetRelayOutputStateResponse** message.

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

5.2.2 IO SETRELAYOUTPUTSTATE – BISTABLE MODE (CLOSED IDLE STATE)

Test Case ID: DEVICEIO-1-2-2

Specification Coverage: None

Feature Under Test: GetRelayOutputs, SetRelayOutputSettings, SetRelayOutputState

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of SetRelayOutputState command in the case of bistable mode and closed idle state.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT. Profile T is supported by the DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client retrieves Relay Outputs list by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *relayOutputsList* - Relay Outputs list
4. If *relayOutputsList* is empty, FAIL the test and skip other steps.
5. ONVIF Client selects the first relay output which supports required Relay Output mode by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - in *relayOutputsList* - Relay Outputs list
   - in Bistable – required Relay Output Mode
   - out *relayOutput* - Relay Output (optional)
6. If *relayOutput* is null, PASS the test and skip other steps.
7. Set the following:
   - *initialRelayOutput* := *relayOutput*
8. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Device/Trigger/Relay"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
9. Until *timeout1* timeout expires, repeat the following steps:
   9.1. ONVIF Client invokes **PullMessages** request with parameters
      - Timeout := PT20S
      - MessageLimit := 1
   9.2. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*

- NotificationMessage list =: *notificationMessageList*

9.3. If *notificationMessageList* contains notification (notification) with the following parameters:

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Initialized,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token,
- Data.SimpleItem.Name = "LogicalState"

9.3.1. Set the following:

- *initialLogicalState* := value of LogicalState data simple item in *notification*

9.3.2. Go to step 10.

9.4. If *timeout1* timeout expires for step 9 without Notification described at step 9.3, FAIL the test and skip other steps.

10. ONVIF Client invokes **SetRelayOutputSettings** request with parameters
    - RelayOutput.@token := *relayOutput*.@token
    - RelayOutput.Properties.Mode := Bistable
    - RelayOutput.Properties.DelayTime := *relayOutput*.Properties.DelayTime
    - RelayOutput.Properties.IdleState := closed
11. The DUT responds with **SetRelayOutputSettingsResponse** message.
12. ONVIF Client invokes **SetRelayOutputState** request with parameters
    - RelayOutputToken := *relayOutput*.@token
    - LogicalState := active if *initialLogicalState* = inactive, otherwise inactive
13. The DUT responds with **SetRelayOutputStateResponse** message.
14. Until *timeout1* timeout expires, repeat the following steps:

14.1. ONVIF Client invokes **PullMessages** request with parameters

- Timeout := PT20S
- MessageLimit := 1

14.2. The DUT responds with **PullMessagesResponse** message with parameters

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

14.3. If *notificationMessageList* contains notification (*notification*) with the following parameters:

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token

14.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

14.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 12, FAIL the test and skip other steps.

14.3.3. Go to step 15.

14.4. If *timeout1* timeout expires for step 14 without Notification described at step 14.3, FAIL the test and skip other steps.

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

- RelayOutputToken := *relayOutput*.@token
- LogicalState := *initialLogicalState*

16. The DUT responds with **SetRelayOutputStateResponse** message.

17. Until *timeout1* timeout expires, repeat the following steps:

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

- Timeout := PT20S

- MessageLimit := 1

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

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

17.3. If *notificationMessageList* contains the following notification (*notification*):

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token,

17.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

17.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 15, FAIL the test and skip other steps.

17.3.3. Go to step 18.

17.4. If *timeout1* timeout expires for step 17 without Notification described at step 17.3, FAIL the test and skip other steps.

18. ONVIF Client restores initial Relay Output settings by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *initialRelayOutput* – Relay Output to restore
19. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.5 with the following input and output parameters
    - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.

- The **GetRelayOutputOptionsResponse** message did not contain option with Mode = Bistable

FAIL –

- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **SetRelayOutputSettingsResponse** message.
- The DUT did not send **SetRelayOutputStateResponse** message.

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

5.2.3 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (OPENED IDLE STATE)

Test Case ID: DEVICEIO-1-2-3

Specification Coverage: None

Feature Under Test: GetRelayOutputs, SetRelayOutputSettings, SetRelayOutputState

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of SetRelayOutputState command in the case of Monostable mode and opened idle state as well as appropriate event messaging.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT. Profile T is supported by the DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Relay Outputs list by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *relayOutputsList* - Relay Outputs list
4. If *relayOutputsList* is empty, FAIL the test and skip other steps.
5. ONVIF Client selects the first relay output which supports required Relay Output mode by following the procedure mentioned in Annex A.3 with the following input and output parameters

- in *relayOutputsList* - Relay Outputs list
- in Monostable – required Relay Output Mode
- out *relayOutput* - Relay Output (optional)
- out *relayOutputOptions* - Realy Output Options (optional)

6. If *relayOutput* is null, PASS the test and skip other steps.
7. If *relayOutputOptions*.Discrete value is equal to true and relayOutputOptions does not contain DelayTimes element, FAIL the test and skip other steps.
8. Set the following:
   - *initialRelayOutput* := *relayOutput*
9. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Device/Trigger/Relay"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
10. Until *timeout1* timeout expires, repeat the following steps:
    10.1. ONVIF Client invokes **PullMessages** request with parameters
      - Timeout := PT20S
      - MessageLimit := 1

    10.2. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*
      - NotificationMessage list =: *notificationMessageList*

    10.3. If *notificationMessageList* contains notification (notification) with the following parameters:
      - Topic = "tns1:Device/Trigger/Relay",

- PropertyOperation = Initialized,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token,
- Data.SimpleItem.Name = "LogicalState"

10.3.1. Set the following:

- *initialLogicalState* := value of LogicalState data simple item in *notification*

10.3.2. Go to step 11.

10.4. If *timeout1* timeout expires for step 10 without Notification described at step 10.3, FAIL the test and skip other steps.

11. ONVIF Client set the following:

- *delayTime* := value from *relayOutputOptions*.DelayTimes list closest to 5 seconds if OutputOptions.Discrete = true, otherwise PT5S

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

- RelayOutput.@token := *relayOutput*.@token
- RelayOutput.Properties.Mode := Monostable
- RelayOutput.Properties.DelayTime := *delayTime*
- RelayOutput.Properties.IdleState := open

13. The DUT responds with **SetRelayOutputSettingsResponse** message.

14. ONVIF Client invokes **SetRelayOutputState** request with parameters

- RelayOutputToken := *relayOutput*.@token
- LogicalState := active if *initialLogicalState* = inactive, otherwise inactive

15. The DUT responds with **SetRelayOutputStateResponse** message.

16. Until *timeout1* timeout expires, repeat the following steps:

16.1. ONVIF Client invokes **PullMessages** request with parameters

- Timeout := PT20S
- MessageLimit := 1

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

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

16.3. If *notificationMessageList* contains notification (*notification*) with the following parameters:

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token

16.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

16.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 14, FAIL the test and skip other steps.

16.3.3. Go to step 17.

16.4. If *timeout1* timeout expires for step 16 without Notification described at step 16.3, FAIL the test and skip other steps.

17. ONVIF Client waits until *delayTime* timeout is expired.

18. Until *timeout1* timeout expires, repeat the following steps:

18.1. ONVIF Client invokes **PullMessages** request with parameters

- Timeout := PT20S
- MessageLimit := 1

18.2. The DUT responds with **PullMessagesResponse** message with parameters

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

18.3. If *notificationMessageList* contains the following notification (*notification*):

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token,

18.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

18.3.2. If LogicalState data SimpleItem value is not equal to *initialLogicalState*, FAIL the test and skip other steps.

18.3.3. Go to step 19.

18.4. If *timeout1* timeout expires for step 18 without Notification described at step 18.3, FAIL the test and skip other steps.

19. ONVIF Client restores initial Relay Output settings by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *initialRelayOutput* – Relay Output to restore
20. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.5 with the following input and output parameters
    - in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.
- The **GetRelayOutputOptionsResponse** message did not contain option with Mode = Monostable

FAIL –

- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **SetRelayOutputSettingsResponse** message.
- The DUT did not send **SetRelayOutputStateResponse** message.

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

5.2.4 IO SETRELAYOUTPUTSTATE – MONOSTABLE MODE (CLOSED IDLE STATE)

Test Case ID: DEVICEIO-1-2-4

Specification Coverage: None

Feature Under Test: GetRelayOutputs, SetRelayOutputSettings, SetRelayOutputState

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of SetRelayOutputState command in the case of monostable mode and closed idle state.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by DUT. Profile T is supported by the DUT. Relay Output Options supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Relay Outputs list by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *relayOutputsList* - Relay Outputs list
4. If *relayOutputsList* is empty, FAIL the test and skip other steps.
5. ONVIF Client selects the first relay output which supports required Relay Output mode by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - in *relayOutputsList* - Relay Outputs list
   - in Monostable – required Relay Output Mode
   - out *relayOutput* - Relay Output (optional)
   - out *relayOutputOptions* - Realy Output Options (optional)

6. If *relayOutput* is null, PASS the test and skip other steps.
7. If *relayOutputOptions*.Discrete value is equal to true and relayOutputOptions does not contain DelayTimes element, FAIL the test and skip other steps.
8. Set the following:
   - *initialRelayOutput* := *relayOutput*
9. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.4 with the following input and output parameters
   - in **"tns1:Device/Trigger/Relay"** - Notification Topic
   - out *s* - Subscription Reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription Termination time
10. Until *timeout1* timeout expires, repeat the following steps:
    10.1. ONVIF Client invokes **PullMessages** request with parameters
      - Timeout := PT20S
      - MessageLimit := 1

    10.2. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*
      - NotificationMessage list =: *notificationMessageList*

    10.3. If *notificationMessageList* contains notification (notification) with the following parameters:
      - Topic = "tns1:Device/Trigger/Relay",
      - PropertyOperation = Initialized,
      - Source.SimpleItem.Name = "RelayToken",
      - Source simple item Token value equals to *relayOutput*.@token,
      - Data.SimpleItem.Name = "LogicalState"

10.3.1. Set the following:

- *initialLogicalState* := value of LogicalState data simple item in *notification*

10.3.2. Go to step 11.

10.4. If *timeout1* timeout expires for step 10 without Notification described at step 10.3, FAIL the test and skip other steps.

11. ONVIF Client set the following:

- *delayTime* := value from *relayOutputOptions*.DelayTimes list closest to 5 seconds if OutputOptions.Discrete = true, otherwise PT5S

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

- RelayOutput.@token := *relayOutput*.@token
- RelayOutput.Properties.Mode := Monostable
- RelayOutput.Properties.DelayTime := *delayTime*
- RelayOutput.Properties.IdleState := closed

13. The DUT responds with **SetRelayOutputSettingsResponse** message.

14. ONVIF Client invokes **SetRelayOutputState** request with parameters

- RelayOutputToken := *relayOutput*.@token
- LogicalState := active if *initialLogicalState* = inactive, otherwise inactive

15. The DUT responds with **SetRelayOutputStateResponse** message.

16. Until *timeout1* timeout expires, repeat the following steps:

16.1. ONVIF Client invokes **PullMessages** request with parameters

- Timeout := PT20S
- MessageLimit := 1

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

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

16.3. If *notificationMessageList* contains notification (*notification*) with the following parameters:

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",
- Source simple item Token value equals to *relayOutput*.@token

16.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

16.3.2. If LogicalState data SimpleItem value is not equal to LogicalState value from step 14, FAIL the test and skip other steps.

16.3.3. Go to step 17.

16.4. If *timeout1* timeout expires for step 16 without Notification described at step 16.3, FAIL the test and skip other steps.

17. ONVIF Client waits until *delayTime* timeout is expired.

18. Until *timeout1* timeout expires, repeat the following steps:

18.1. ONVIF Client invokes **PullMessages** request with parameters

- Timeout := PT20S
- MessageLimit := 1

18.2. The DUT responds with **PullMessagesResponse** message with parameters

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage list =: *notificationMessageList*

18.3. If *notificationMessageList* contains the following notification (*notification*):

- Topic = "tns1:Device/Trigger/Relay",
- PropertyOperation = Changed,
- Source.SimpleItem.Name = "RelayToken",

- Source simple item Token value equals to *relayOutput*.@token,

18.3.1. If *notification* does not contain Data.SimpleItem with LogicalState name, FAIL the test and skip other steps.

18.3.2. If LogicalState data SimpleItem value is not equal to *initialLogicalState*, FAIL the test and skip other steps.

18.3.3. Go to step 19.

18.4. If *timeout1* timeout expires for step 18 without Notification described at step 18.3, FAIL the test and skip other steps.

19. ONVIF Client restores initial Relay Output settings by following the procedure mentioned in Annex A.6 with the following input and output parameters

- in *initialRelayOutput* – Relay Output to restore

20. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.5 with the following input and output parameters

- in *s* - Subscription reference

Test Result:

PASS –

- DUT passes all assertions.
- The **GetRelayOutputOptionsResponse** message did not contain option with Mode = Monostable

FAIL –

- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **SetRelayOutputSettingsResponse** message.
- The DUT did not send **SetRelayOutputStateResponse** message.

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

5.3 Events

5.3.1 REALTIME PULLPOINT SUBSCRIPTION – DIGITAL INPUT EVENT

Test Case ID: DEVICEIO-2-1-1

Specification Coverage: DigitalInput State Change (Device IO)

Feature Under Test: tns1:Device/Trigger/DigitalInput

WSDL Reference: event.wsdl

Test Purpose: To verify tns1:Device/Trigger/DigitalInput event generation after subscription and to verify tns1:Device/Trigger/DigitalInput event format.

Pre-Requisite: Device supports Digital Inputs feature.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetEventProperties**.
4. The DUT responds with a **GetEventPropertiesResponse** message with parameters
   - TopicNamespaceLocation list
   - FixedTopicSet
   - TopicSet =: *topicSet*
   - TopicExpressionDialect list
   - MessageContentFilterDialect list
   - MessageContentSchemaLocation list
5. If *topicSet* does not contain **tns1:Device/Trigger/DigitalInput** topic, FAIL the test and skip other steps.
6. ONVIF Client verifies **tns1:Device/Trigger/DigitalInput** topic (*triggerDigitalInput*) from *topicSet*:

6.1. If *triggerDigitalInput*.MessageDescription.IsProperty is skipped or equals false, FAIL the test and skip other steps.

6.2. If *triggerDigitalInput* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "InputToken", FAIL the test and skip other steps.

6.3. If *triggerDigitalInput*.MessageDescription.Source.SimpleItemDescription with Name = "InputToken" does not have Type = "tt:ReferenceToken", FAIL the test and skip other steps.

6.4. If *triggerDigitalInput* does not contain MessageDescription.Data.SimpleItemDescription item with Name = "LogicalState", FAIL the test and skip other steps.

6.5. If *triggerDigitalInput*.MessageDescription.Data.SimpleItemDescription item with Name = "LogicalState" does not have Type = "xs:boolean", FAIL the test and skip other steps.

7. ONVIF Client invokes **CreatePullPointSubscription** with parameters

   - Filter.TopicExpression := "tns1:Device/Trigger/DigitalInput"
   - Filter.TopicExpression.@Dialect := "http://www.onvif.org/ver10/tev/topicExpression/ConcreteSet"

8. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters

   - SubscriptionReference =: *s*
   - CurrentTime
   - TerminationTime

9. Until *timeout1* timeout expires, repeat the following steps:

   9.1. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters

      - Timeout := PT60S
      - MessageLimit := 1

   9.2. The DUT responds with **PullMessagesResponse** message with parameters

      - CurrentTime

- TerminationTime
- NotificationMessage =: *m*

9.3. If *m* is not null and *m* Message.Message.PropertyOperation = Initialized ONVIF Client verifies *m*:

9.3.1. If *m*.Topic does not equal to **tns1:Device/Trigger/DigitalInput**, FAIL the test and go to the step 10.

9.3.2. If *m* does not contain Message.Message.Source.SimpleItem.InputToken, FAIL the test and go to the step 10.

9.3.3. If *m*.Message.Message.Source.SimpleItem.InputToken has value type different from tt:ReferenceToken type, FAIL the test and go to the step 10.

9.3.4. If *m* does not contain Message.Message.Data.SimpleItem.LogicalState, FAIL the test and go to the step 10.

9.3.5. If *m*.Message.Message.Data.SimpleItem.LogicalState has value type different from xs:boolean type, FAIL the test and go to the step 10.

9.3.6. Go to the step 10.

9.4. If *timeout1* timeout expires for step 9 without Notification with PropertyOperation = Initialized, FAIL the test and go to the step 10.

10. ONVIF Client invokes **Unsubscribe** to the subscription endpoint *s*.

11. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetEventPropertiesResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message(s).
- The DUT did not send **UnsubscribeResponse** message.

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

5.3.2 DEVICE IO TRIGGER EVENT

Test Case ID: DEVICEIO-2-1-2

Specification Coverage: Relay Output Trigger (Device IO Service)

Feature Under Test: GetEventProperties

WSDL Reference: event.wsdl

Test Purpose: To verify tns1:Device/Trigger/Relay event format in TopicSet.

Pre-Requisite: Device IO service is supported by DUT. Relay Outputs supported by 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 **GetEventProperties** request.
4. The DUT responds with a **GetEventPropertiesResponse** message with parameters
   - TopicNamespaceLocation list
   - FixedTopicSet
   - TopicSet =: *topicSet*
   - TopicExpressionDialect list
   - MessageContentFilterDialect list
   - MessageContentSchemaLocation list
5. If *topicSet* does not contain tns1:Device/Trigger/Relay topic, FAIL the test and skip other steps.
6. ONVIF Client verifies tns1:Device/Trigger/Relay topic (*triggerTopic*) from *topicSet*:
   - If *triggerTopic*.MessageDescription.IsProperty does not equal to true, FAIL the test and skip other steps.

- If *triggerTopic* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "RelayToken", FAIL the test and skip other steps.
- If *triggerTopic*.MessageDescription.Source.SimpleItemDescription with Name = "RelayToken" does not have Type = "tt:ReferenceToken", FAIL the test and skip other steps.
- If *triggerTopic* does not contain MessageDescription.Data.SimpleItemDescription item with Name = "LogicalState", FAIL the test and skip other steps.
- If *triggerTopic*.MessageDescription.Data.SimpleItemDescription with Name = "LogicalState" does not have Type = "tt:RelayLogicalState", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetEventPropertiesResponse** message.

5.4 Digital Input

5.4.1 IO GETDIGITALINPUTS

Test Case ID: DEVICEIO-3-1-1

Specification Coverage: None

Feature Under Test: GetDigitalInputs

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the DUT returns proper message for GetDigitalInputs request.

Pre-Requisite: Device IO service is supported by DUT. Digital Inputs is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client sends **GetDigitalInputs** request to DUT to retrieve the list of supported digital input configurations.
4. The DUT sends **GetDigitalInputsResponse** message with the list of supported DigitalInputs.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetDigitalInputsResponse** message.
- The **GetDigitalInputsResponse** message did not contain Digital Inputs.

5.4.2 IO GETDIGITALINPUTS – VERIFY QUANTITY

Test Case ID: DEVICEIO-3-1-2

Specification Coverage: None

Feature Under Test: GetDigitalInputs, GetServiceCapabilities

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the DUT returns proper message for GetDigitalInputs request.

Pre-Requisite: Device IO service is supported by DUT. Digital Inputs is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServiceCapabilities** request.
4. The DUT sends **GetServiceCapabilitiesResponse** with the capabilities of the device IO service.

5. ONVIF Client sends **GetDigitalInputs** request to DUT to retrieve the list of supported digital input configurations.
6. The DUT sends **GetDigitalInputsResponse** message with the list of supported DigitalInputs.
7. ONVIF Client verifies the number of digital inputs in **GetDigitalInputsResponse** message. This number should be equal to the Capabilities.DigitalInputsnumber in **GetServiceCapabilitiesResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetDigitalInputsResponse** message.
- The DUT sent incorrect **GetDigitalInputsResponse** message.
- The DUT did not send **GetServiceCapabilitiesResponse** message.
- The DUT sent empty list of DigitalInputsin **GetDigitalInputsResponse** message.
- The number of Digital Inputs in **GetDigitalInputsResponse** message is not equal to Capabilities.DigitalInputs number from **GetServiceCapabilitiesResponse** message.

5.4.3 IOGET DIGITAL INPUT CONFIGURATION OPTIONS

Test Case ID: DEVICEIO-3-1-3

Specification Coverage: None

Feature Under Test: GetDigitalInputs, GetDigitalInputConfigurationOptions

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of GetDigitalInputConfigurationOptions command.

Pre-Requisite: Device IO service is supported by DUT. Digital Inputs is supported by DUT. Digital Input Options is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetDigitalInputs** request to DUT to retrieve the list of supported digital input configurations.
4. The DUT sends **GetDigitalInputsResponse** message with the list of supported digital inputs.
5. ONVIF Client invokes **GetDigitalInputConfigurationOptions** message to DUT to retrieve the generic input configuration options.
6. The DUT sends **GetDigitalInputConfigurationOptionsResponse** message with generic digital input configuration options.
7. For each digital input in **GetDigitalInputsResponse** message, ONVIF Client saves this digital input in *DigitalInput1* variable and runs the following steps:
   7.1. ONVIF Client invokes **GetDigitalInputConfigurationOptions** request with Token = *DigitalInput1* token as input argument.
   7.2. The DUT sends **GetDigitalInputConfigurationOptionsResponse** message with configuration options for the given token.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetDigitalInputsResponse** message.
- The **GetDigitalInputsResponse** message did not contain Digital Inputs.
- The DUT Did not send **GetDigitalInputConfigurationOptionsResponse** message.
- The **GetDigitalInputConfigurationOptionsResponse** message did not contain Digital Input Options.

5.4.4 IO DIGITAL INPUT CONFIGURATION

Test Case ID: DEVICEIO-3-1-4

Specification Coverage: None

Feature Under Test: GetDigitalInputs, GetDigitalInputConfigurationOptions, SetDigitalInputConfigurations

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of GetDigitalInputs, GetDigitalInputConfigurationOptions, SetDigitalInputConfigurations commands.

Pre-Requisite: Device IO service is supported by DUT. Digital Inputs is supported by DUT. Digital Input Options is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetDigitalInputs** request to DUT to retrieve the list of supported digital input configurations.
4. The DUT sends **GetDigitalInputsResponse** message with the list of supported DigitalInputs.
5. For each digital input in **GetDigitalInputsResponse** message, ONVIF Client saves this digital input in *DigitalInput1* variable and runs the following steps:
    5.1. ONVIF Client invokes **GetDigitalInputConfigurationOptions** request with Token = *DigitalInput1* token as input argument.
    5.2. The DUT sends **GetDigitalInputConfigurationOptionsResponse** message with configuration options for the given token.
    5.3. If **GetDigitalInputConfigurationOptionsResponse** message contains DigitalInputOptionsIdleState = closed then run the following steps:
        5.3.1. ONVIF Client changes *DigitalInput1* IdleState property to closed.
        5.3.2. ONVIF Client invokes **SetDigitalInputConfigurations** request with *DigitalInput1* as input argument.
        5.3.3. The DUT sends **SetDigitalInputConfigurationsResponse** message. ONVIF Client verifies the response.

5.3.4. ONVIF Client invokes **GetDigitalInputs** request.

5.3.5. The DUT sends **GetDigitalInputsResponse** message with the list of Digital Inputs.

5.3.6. ONVIF Client verifies that the **GetDigitalInputsResponse** message contains digital input with token = *DigitalInput1* token, also it verifies that IdleState value equals to the value set up in the step 5.3.1.

5.4. If **GetDigitalInputConfigurationOptionsResponse** message contains DigitalInputOptions.IdleState = open then run the following steps:

5.4.1. ONVIF Client changes *DigitalInput1*.IdleState property to open.

5.4.2. ONVIF Client invokes **SetDigitalInputConfigurations** request with *DigitalInput1* as input argument.

5.4.3. The DUT sends **SetDigitalInputConfigurationsResponse** message. ONVIF Client verifies the response.

5.4.4. ONVIF Client invokes **GetDigitalInputs** request.

5.4.5. The DUT sends **GetDigitalInputsResponse** message with the list of Digital Inputs.

5.4.6. ONVIF Client verifies that the **GetDigitalInputsResponse** message contains digital input with token = *DigitalInput1* token, also it verifies that IdleState value equals to the value set up in the step 5.4.1.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetDigitalInputsResponse** message.
- The **GetDigitalInputsResponse** message did not contain Digital Inputs.
- The DUT Did not send **GetDigitalInputConfigurationOptionsResponse** message.
- The **GetDigitalInputConfigurationOptionsResponse** message did not contain Digital Input Options.
- The DUT did not send **SetDigitalInputConfigurationsResponse** message.

- The DUT did not change IdleState.

5.5 Audio Source

5.5.1 IO GET AUDIO SOURCES

Test Case ID: DEVICEIO-4-1-1

Specification Coverage: GetAudioSources (ONVIF Device IO Service Specification).

Feature Under Test: GetAudioSources

WSDL Reference: deviceio.wsdl

Test Purpose: To verify retrieving supported Audio Sources.

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

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetAudioSources** request.
4. The DUT responds with **GetAudioSourcesResponse** with parameters
   - Token list =: *audioSourcesList*
5. If *audioSourcesList* is empty, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.6 Consistency

5.6.1 GET VIDEOSOURCES (DeviceIO) AND GET VIDEOSOURCES (Media) CONSISTENCY

Test Case ID: DEVICEIO-5-1-1

Specification coverage: GetVideoSources (Media Service Specification), GetVideoSources (Device IO Service Specification)

Feature under test: GetVideoSources (DeviceIO), GetVideoSources (Media)

WSDL Reference: media.wsdl, deviceio.wsdl

Test Purpose: To verify GetVideoSources (DeviceIO) and GetVideoSources (Media) consistency.

Pre-Requisite: Media 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** (DeviceIO) request.
4. The DUT responds with **GetVideoSourcesResponse** message with parameters
   - Token list =: *tokenList*
5. ONVIF Client invokes **GetVideoSources** (Media) request.
6. The DUT responds with **GetVideoSourcesResponse** message with parameters
   - VideoSources list =: *videoSourcesList*
7. If *tokenList* is not equal to list of @token items from *videoSourcesList* list, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- DUT did not send **GetVideoSources** (DeviceIO) message.

  DUT did not send **GetVideoSources** (Media) message.

5.7 Serial Port

5.7.1 IO GET SERIAL PORTS

Test Case ID: DEVICEIO-6-1-1

Specification Coverage: GetSerialPorts (Device IO Service Specification)

Feature Under Test: GetSerialPorts

WSDL Reference: deviceio.wsdl

Test Purpose: To verify retrieving of DUT Serial Ports using GetSerialPorts command.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Device IO Service capabilities by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *cap* - Device IO Service capabilities
4. ONVIF Client invokes **GetSerialPorts** request.
5. The DUT responds with **GetSerialPortsResponse** message with parameters:
   - SerialPort list =: *serialPortList*
6. If number of items in *serialPortList* is not equal to *cap*.SerialPorts, FAIL the test and skip other steps.
7. If *serialPortList* contains at least two items with the same @token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetSerialPortsResponse** message.

5.7.2 IO GET SERIAL PORT CONFIGURATION AND GET SERIAL PORT OPTIONS

Test Case ID: DEVICEIO-6-1-2

Specification Coverage: GetSerialPortConfiguration (Device IO Service Specification), GetSerialPortConfigurationOptions (Device IO Service Specification)

Feature Under Test: GetSerialPortConfiguration, GetSerialPortConfigurationOptions

WSDL Reference: deviceio.wsdl

Test Purpose: To verify retrieving of DUT Serial Port Configuration using GetSerialPortConfiguration command. To verify retrieving of DUT Serial Port Configuration Options using GetSerialPortConfigurationOptions command. To verify that all Serial Port Configurations are consistent with Serial Port Configuration Options.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Serial Ports list by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *serialPortList* - Serial Ports list
4. For each SerialPort *serialPort* in *serialPortList* list repeat the following steps:
   4.1. ONVIF Client invokes **GetSerialPortConfiguration** request with parameters
      - SerialPortToken := *serialPort*.@token
   4.2. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:
      - SerialPortConfiguration =: *serialPortConfiguration*

4.3. If *serialPortConfiguration*.@token not equal to *serialPort*.@token, FAIL the test and skip other steps.

4.4. ONVIF Client invokes **GetSerialPortConfigurationOptions** request with parameters

- SerialPortToken := *serialPortConfiguration*.@token

4.5. The DUT responds with **GetSerialPortConfigurationOptionsResponse** message with parameters:

- SerialPortOptions =: *serialPortOptions*

4.6. If *serialPortConfiguration*.@token is not equal to *serialPortOptions*.@token, FAIL the test and skip other steps.

4.7. If *serialPortOptions*.BaudRateList.Items list is empty, FAIL the test and skip other steps.

4.8. If *serialPortOptions*.ParityBitList.Items list is empty, FAIL the test and skip other steps.

4.9. If *serialPortOptions*.CharacterLengthList.Items list is empty, FAIL the test and skip other steps.

4.10. If *serialPortOptions*.StopBitList.Items list is empty, FAIL the test and skip other steps.

4.11. If *serialPortConfiguration*.BaudRate is not in *serialPortOptions*.BaudRateList.Items list, FAIL the test and skip other steps.

4.12. If *serialPortConfiguration*.ParityBit is not in *serialPortOptions*.ParityBitList.Items list, FAIL the test and skip other steps.

4.13. If *serialPortConfiguration*.CharacterLength is not in *serialPortOptions*.CharacterLengthList.Items list, FAIL the test and skip other steps.

4.14. If *serialPortConfiguration*.StopBit is not in *serialPortOptions*.StopBitList.Items list, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetSerialPortConfigurationResponse** message.

- The DUT did not send **GetSerialPortConfigurationOptionsResponse** message.

5.7.3 IO MODIFY SERIAL PORT CONFIGURATION

Test Case ID: DEVICEIO-6-1-3

Specification Coverage: SetSerialPortConfiguration (Device IO Service Specification)

Feature Under Test: SetSerialPortConfiguration

WSDL Reference: deviceio.wsdl

Test Purpose: To verify modification of Serial Port Configuration using SetSerialPortConfiguration command.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Serial Ports list by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *serialPortList* - Serial Ports list
4. For each SerialPort *serialPort* in *serialPortList* repeat the following steps:
   4.1. ONVIF Client invokes **GetSerialPortConfiguration** request with parameters
      - SerialPortToken := *serialPort*.@token
   4.2. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:
      - SerialPortConfiguration =: *initialSerialPortConfig*
   4.3. ONVIF Client invokes **GetSerialPortConfigurationOptions** request with parameters
      - SerialPortToken := *serialPort*.@token
   4.4. The DUT responds with **GetSerialPortConfigurationOptionsResponse** message with parameters:
      - SerialPortOptions =: *serialPortOptions*

4.5. ONVIF Client invokes **SetSerialPortConfiguration** request with parameters

- SerialPortConfiguration.token := *serialPort*.@token
- SerialPortConfiguration.type := *initialSerialPortConfig*.@type
- SerialPortConfiguration.BaudRate := *serialPortOptions*.BaudRateList.Items[0] (if this value is equal to current, the next item in the list should be used)
- SerialPortConfiguration.ParityBit := *serialPortOptions*.ParityBitList.Items[0] (if this value is equal to current, the next item in the list should be used)
- SerialPortConfiguration.CharacterLength := *serialPortOptions*.CharacterLengthList.Items[0] (if this value is equal to current, the next item in the list should be used)
- SerialPortConfiguration.StopBit := *serialPortOptions*.StopBitList.Items[0] (if this value is equal to current, the next item in the list should be used)
- ForcePersistance := false

4.6. The DUT responds with **SetSerialPortConfigurationResponse** message.

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

- SerialPortToken := *serialPort*.@token

4.8. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:

- SerialPortConfiguration =: *serialPortConfiguration*

4.9. If *serialPortConfiguration* is not equal to Serial Port Configuration from step 4.5, FAIL the test and skip other steps.

4.10. ONVIF Client invokes **SetSerialPortConfiguration** request with parameters

- SerialPortConfiguration.token := *serialPort*.@token
- SerialPortConfiguration.type := *serialPortConfiguration*.type
- SerialPortConfiguration.BaudRate := *serialPortOptions*.BaudRateList.Items[last]
- SerialPortConfiguration.ParityBit := *serialPortOptions*.ParityBitList.Items[last]

- SerialPortConfiguration.CharacterLength := *serialPortOptions*.CharacterLengthList.Items[last]
- SerialPortConfiguration.StopBit := *serialPortOptions*.StopBitList.Items[last]
- ForcePersistance := true

4.11. The DUT responds with **SetSerialPortConfigurationResponse** message.

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

- SerialPortToken := *serialPort*.@token

4.13. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:

- SerialPortConfiguration =: *serialPortConfiguration*

4.14. If *serialPortConfiguration* is not equal to SerialPortConfiguration from step 4.10, FAIL the test and skip other steps.

4.15. ONVIF Client restores settings of Serial Port Configuration with @token = *serialPort*.@token.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetSerialPortConfigurationResponse** message(s).
- The DUT did not send **GetSerialPortConfigurationOptionsResponse** message(s).
- The DUT did not send **SetSerialPortConfigurationResponse** message(s).

Note: The following fields are compared at step 4.5 and 4.10:

- token
- type
- BaudRate
- ParityBit
- CharacterLength

- StopBit

5.7.4 IO RECEIVE SERIAL COMMAND

Test Case ID: DEVICEIO-6-1-4

Specification Coverage: Send ReceiveSerialCommand (Device IO Service Specification)

Feature Under Test: SendReceiveSerialCommand

WSDL Reference: deviceio.wsdl

Test Purpose: To verify receiving of generic controlling data from Serial Port using SendReceiveSerialCommand command.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Serial Ports list by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *serialPortList* - Serial Ports list
4. For each SerialPort *serialPort* in *serialPortList* repeat the following steps:
   4.1. ONVIF Client invokes **SendReceiveSerialCommand** request with parameters
      - Token := *serialPort*.@token
      - SerialData - skipped
      - TimeOut - skipped
      - DataLength - skipped
      - Delimiter - skipped
   4.2. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:
      - SerialData

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

- Token := *serialPort*.@token
- SerialData - skipped
- TimeOut := PT1M1S
- DataLength - skipped
- Delimiter - skipped

4.4. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:

- SerialData

4.5. ONVIF Client invokes **SendReceiveSerialCommand** request with parameters

- Token := *serialPort*.@token
- SerialData - skipped
- TimeOut := PT0S
- DataLength - skipped
- Delimiter - skipped

4.6. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:

- SerialData

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

- Token := *serialPort*.@token
- SerialData - skipped
- TimeOut := -PT1S
- DataLength - skipped
- Delimiter - skipped

4.8. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:

- SerialData

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **SendReceiveSerialCommandResponse** message(s).

Note: The following message timeouts should be used:

- for step 4.4: 61s + Message Timeout field of ONVIF Device Test Tool
- for steps 4.2 and 4.8: Operation Delay field of ONVIF Device Test Tool

5.7.5 IO SEND SERIAL COMMAND

Test Case ID: DEVICEIO-6-1-5

Specification Coverage: Send ReceiveSerialCommand (Device IO Service Specification)

Feature Under Test: SendReceiveSerialCommand

WSDL Reference: deviceio.wsdl

Test Purpose: To verify sending of generic controlling data from Serial Port using SendReceiveSerialCommand command.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Serial Ports list by following the procedure mentioned in Annex A.8 with the following input and output parameters

- out *serialPortList* - Serial Ports list

4. For each SerialPort *serialPort* in *serialPortList* repeat the following steps:

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

   - Token := *serialPort*.@token
   - SerialData.Binary := <binaryData>
   - TimeOut := PT0S
   - DataLength - skipped
   - Delimiter - skipped

   4.2. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:

   - SerialData

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

   - Token := *serialPort*.@token
   - SerialData.String := <stringData>
   - TimeOut := PT0S
   - DataLength - skipped
   - Delimiter - skipped

   4.4. The DUT responds with **SendReceiveSerialCommandResponse** message with parameters:

   - SerialData

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **SendReceiveSerialCommandResponse** message(s).

5.7.6 IO GETSERIALPORTCONFIGURATION COMMAND - INVALID TOKEN

Test Case ID: DEVICEIO-6-1-6

Specification Coverage: GetSerialPortConfiguration (Device IO Service Specification)

Feature Under Test: GetSerialPortConfiguration

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of GetSerialPortConfiguration command in case of invalid token.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetSerialPortConfiguration** request with parameters
   - SerialPortToken := "OnvifTest123"
4. The DUT returns **env:Sender/ter:InvalidArgVal/ter:InvalidSerialPort** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.7.7 IO GETSERIALPORTCONFIGURATIONOPTIONS COMMAND - INVALID TOKEN

Test Case ID: DEVICEIO-6-1-7

Specification Coverage: GetSerialPortConfigurationOptions (Device IO Service Specification)

Feature Under Test: GetSerialPortConfigurationOptions

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of GetSerialPortConfigurationOptions command in case of invalid token.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port is supported by DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetSerialPortConfigurationOptions** request with parameters
   - SerialPortToken := "OnvifTest123"
4. The DUT returns **env:Sender/ter:InvalidArgVal/ter:InvalidSerialPort** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.7.8 IO SETSERIALPORTCONFIGURATION COMMAND - INVALID SETTINGS

Test Case ID: DEVICEIO-6-1-8

Specification Coverage: SetSerialPortConfiguration (Device IO Service Specification)

Feature Under Test: SetSerialPortConfiguration

WSDL Reference: deviceio.wsdl

Test Purpose: To verify the behavior of SetSerialPortConfiguration command in case of invalid settings.

Pre-Requisite: Device IO Service is received from the DUT. Serial Port 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 Serial Ports list by following the procedure mentioned in Annex A.8 with the following input and output parameters
   - out *serialPortList* - Serial Ports list
4. For each SerialPort *serialPort* in *serialPortList* repeat the following steps:
   4.1. ONVIF Client invokes **GetSerialPortConfiguration** request with parameters
      - SerialPortToken := *serialPort*.@token
   4.2. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:
      - SerialPortConfiguration =: *initialSerialPortConfiguration*
   4.3. ONVIF Client invokes **GetSerialPortConfigurationOptions** request with parameters
      - SerialPortToken := *serialPort*.@token
   4.4. The DUT responds with **GetSerialPortConfigurationOptionsResponse** message with parameters:
      - SerialPortOptions =: *serialPortOptions*
   4.5. ONVIF Client invokes **SetSerialPortConfiguration** request with parameters
      - SerialPortConfiguration.token := *serialPort*.@token
      - SerialPortConfiguration.type := *serialPortConfiguration*.type
      - SerialPortConfiguration.BaudRate := value not from *serialPortOptions*.BaudRateList.Items list
      - SerialPortConfiguration.ParityBit := value not from *serialPortOptions*.ParityBitList.Itemslist (if possible)

- SerialPortConfiguration.CharacterLength := value not from *serialPortOptions*.CharacterLengthList.Items list
- SerialPortConfiguration.StopBit := value not from *serialPortOptions*.StopBitList.Items list
- ForcePersistance := false

4.6. The DUT returns **env:Sender/ter:InvalidArgVal/ter:ConfigModify** SOAP 1.2 fault.

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

- SerialPortToken := *serialPort*.@token

4.8. The DUT responds with **GetSerialPortConfigurationResponse** message with parameters:

- SerialPortConfiguration =: *serialPortConfiguration*

4.9. If *initialSerialPortConfiguration* is not equal to *serialPortConfiguration*, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetSerialPortConfigurationResponse** message(s).
- The DUT did not send valid **GetSerialPortConfigurationOptionsResponse** message.
- The DUT did not send the **env:Sender/ter:InvalidArgVal/ter:ConfigModify** SOAP 1.2 fault message.

5.8 Video Source

5.8.1 IO GET VIDEO SOURCES

Test Case ID: DEVICEIO-7-1-1

Specification Coverage: GetVideoSources (ONVIF Device IO Service Specification).

Feature Under Test: GetVideoSources

WSDL Reference: deviceio.wsdl

Test Purpose: To verify retrieving supported Video Sources.

Pre-Requisite: Device IO Service is received from the DUT. Media2 Service is supported by Device. Media2 Video is supported by Device as indicated by the ProfileCapabilities.ConfigurationsSupported = VideoSource capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Device IO Service capabilities by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *cap* - Device IO Service capabilities
4. ONVIF Client invokes **GetVideoSources** request.
5. The DUT responds with **GetVideoSourcesResponse** with parameters
   - Token list =: *videoSourcesList*
6. If *videoSourcesList* is empty, FAIL the test.
7. If number of items in *videoSourcesList* is not equal to *cap*.VideoSources, FAIL the test and skip other steps.
8. If *videoSourcesList* contains at least two items with the same @token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

5.9 Audio Output

5.9.1 IO GET AUDIO OUTPUTS

Test Case ID: DEVICEIO-8-1-1

Specification Coverage: GetAudioOutputs (ONVIF Device IO Service Specification).

Feature Under Test: GetAudioOutputs

WSDL Reference: deviceio.wsdl

Test Purpose: To verify retrieving supported Audio Outputs.

Pre-Requisite: Device IO Service is received from the DUT. Media2 Service is supported by Device. Media2 Audio Output 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 Device IO Service capabilities by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - out *cap* - Device IO Service capabilities
4. ONVIF Client invokes **GetAudioOutputs** request.
5. The DUT responds with **GetAudioOutputsResponse** with parameters
   - Token list =: *audioOutputsList*
6. If *audioOutputsList* is empty, FAIL the test.
7. If number of items in *audioOutputsList* is not equal to *cap*.AudioOutputs, FAIL the test and skip other steps.
8. If *audioOutputsList* contains at least two items with the same @token, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Annex A Helper Procedures and Additional Notes

A.1 Action URI's for Event Service Messages

The following Action URI's shall be used for Event Service:

Table A.1. Action URI's for Event Service Messages

Message	Action URI of WS-Addressing
Notify	http://docs.oasis-open.org/wsn/bw-2/NotificationConsumer/Notify
SubscribeRequest	http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeRequest
SubscribeResponse	http://docs.oasis-open.org/wsn/bw-2/NotificationProducer/SubscribeResponse
RenewRequest	http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/RenewRequest
RenewResponse	http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/RenewResponse
UnsubscribeRequest	http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/UnsubscribeRequest
UnsubscribeResponse	http://docs.oasis-open.org/wsn/bw-2/SubscriptionManager/UnsubscribeResponse
GetEventPropertiesRequest	http://www.onvif.org/ver10/events/wsdl/EventPortType/GetEventPropertiesRequest
GetEventPropertiesResponse	http://www.onvif.org/ver10/events/wsdl/EventPortType/GetEventPropertiesResponse
CreatePullPointSubscriptionRequest	http://www.onvif.org/ver10/events/wsdl/EventPortType/CreatePullPointSubscriptionRequest
CreatePullPointSubscriptionResponse	http://www.onvif.org/ver10/events/wsdl/EventPortType/CreatePullPointSubscriptionResponse
PullMessagesRequest	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/PullMessagesRequest
PullMessagesResponse	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/PullMessagesResponse

Message	Action URI of WS-Addressing
SetSynchronizationPointRequest	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/SetSynchronizationPointRequest
SetSynchronizationPointResponse	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/SetSynchronizationPointResponse
GetServiceCapabilitiesResponse	http://www.onvif.org/ver10/events/wsdl/EventPortType/GetServiceCapabilities
GetServiceCapabilitiesRequest	http://www.onvif.org/ver10/events/wsdl/EventPortType/GetServiceCapabilitiesRequest
SeekRequest	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/SeekRequest
SeekResponse	http://www.onvif.org/ver10/events/wsdl/PullPointSubscription/SeekResponse
All faults	http://www.w3.org/2005/08/addressing/soap/fault

A.2 Get Relay Outputs List

Name: HelperGetRelayOutputsList

Procedure Purpose: Helper procedure to retrieve Relay Outputs List.

Pre-requisite: DeviceIO Service is received from the DUT. DUT supports Relay Outputs.

Input: None.

Returns: Relay Outputs list (*relayOutputsList*).

Procedure:

1. ONVIF Client invokes **GetRelayOutputs** request
2. The DUT sends the **GetRelayOutputsResponse** message with parameters
   - RelayOutputs list =: *relayOutputsList*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetRelayOutputsResponse** message

A.3 Select Relay Output with supporting of required RelayMode

Name: HelperSelectRelayOutputWithRequiredMode

Procedure Purpose: Helper procedure to find RelayOutput which supports required Relay Mode if it exists.

Pre-requisite: DevuceIO Service is received from the DUT. DUT supports Relay Outputs. Relay Output Options supported by DUT.

Input: List of RelayOutputs (*relayOutputsList*), Relay Mode (*relayMode*).

Returns: Relay Output (optional *relayOutput*) with supporting of required Relay Mode if it exists in Relay Outputs list, Realy Output Options (optional *relayOutputOptions*).

Procedure:

1. For each RelayOutput (*relayOutput1*) in *relayOutputsList* repeat the following steps:

   1.1. ONVIF Client invokes **GetRelayOutputOptions** request with parameters

   - RelayOutputToken := *relayOutput1*.@token

   1.2. The DUT responds with **GetRelayOutputOptionsResponse** message with parameters

   - RelayOutputOptions list := *options*

   1.3. If *options*[0] contains Mode with value equals to *relayMode*:

   1.3.1. Set *relayOutputOptions* := *options*[0]

   1.3.2. Set *relayOutput* := *relayOutput1*

   1.3.3. Skip other steps in procedure.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetRelayOutputOptionsResponse** message

A.4 Create Pull Point Subscription

Name: HelperCreatePullPointSubscription

Procedure Purpose: Helper procedure to create PullPoint Subscription with specified Topic.

Pre-requisite: Event Service is received from the DUT.

Input: Notification Topic (*topic*).

Returns: Subscription reference (*s*), current time for the DUT (*ct*), subscription termination time (*tt*).

Procedure:

1. ONVIF Client invokes **CreatePullPointSubscription** request with parameters
   - Filter.TopicExpression := *topic*
   - Filter.TopicExpression.@Dialect := "http://www.onvif.org/ver10/tev/topicExpression/ConcreteSet"
2. The DUT responds with **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **CreatePullPointSubscriptionResponse** message

A.5 Delete Subscription

Name: HelperDeleteSubscription

Procedure Purpose: Helper procedure to delete subscription.

Pre-requisite: Event Service is received from the DUT.

Input: Subscription reference (*s*).

Returns: None.

Procedure:

1. ONVIF Client invokes **Unsubscribe** request to the subscription endpoint *s*.
2. The DUT responds with **UnsubscribeResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **UnsubscribeResponse** message.

A.6 Restore Relay Output settings

Name: HelperRestoreRelayOutput

Procedure Purpose: Helper procedure to restore Relay Output settings.

Pre-requisite: DeviceIO Service is received from the DUT. DUT supports Relay Outputs.

Input: Relay Output (*initialRelayOutput*).

Returns: None.

Procedure:

1. ONVIF Client invokes **SetRelayOutputSettings** request with parameters
   - RelayOutput.@token := *initialRelayOutput*.@token
   - RelayOutput.Properties.Mode := *initialRelayOutput*.Properties.Mode
   - RelayOutput.Properties.DelayTime := *initialRelayOutput*.Properties.DelayTime
   - RelayOutput.Properties.IdleState := *initialRelayOutput*.Properties.IdleState
2. The DUT responds with **SetRelayOutputSettingsResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **SetRelayOutputSettingsResponse** message

A.7 Get Device IO Service Capabilities

Name: HelperGetServiceCapabilities

Procedure Purpose: Helper procedure to get Device IO Service Capabilities from the DUT.

Pre-requisite: Device IO Service is received from the DUT.

Input: None

Returns: The 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.8 Get Serial Ports List

Name: HelperGetSerialPortsList

Procedure Purpose: Helper procedure to retrieve Serial Ports List.

Pre-requisite: DeviceIO Service is received from the DUT. Serial Port is supported by DUT.

Input: None.

Returns: Serial Ports list (*serialPortList*).

Procedure:

1. ONVIF Client invokes **GetSerialPorts** request.
2. The DUT responds with **GetSerialPortsResponse** message with parameters:
   - SerialPort list =: *serialPortList*
3. If *serialPortList* is empty, FAIL the test and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetSerialPortsResponse** message.