ONVIF®

Credential Device Test Specification

Version 20.12

December 2020

REVISION HISTORY

Vers.	Date	Description
15.06	Jul, 2015	First issue of Credentials Test Specification
17.06	Feb 21, 2017	CREATE CREDENTIAL - NOT EMPTY CREDENTIAL TOKEN updated according to #1357
17.06	Mar 15, 2017	Note added into the following tests according to #1366: CREATE CREDENTIAL - VALIDITY VALUES MODIFY CREDENTIAL - VALIDITY VALUES
17.06	Apr 27, 2017	The following test cases were updated according to #1385: CREDENTIAL-3-1-6 CREATE CREDENTIAL (DISABLED) CREDENTIAL-3-1-5 CREATE CREDENTIAL (ENABLED) CREDENTIAL-3-1-7 MODIFY CREDENTIAL CREDENTIAL-5-1-2 SET CREDENTIAL IDENTIFIER – ADDING NEW TYPE CREDENTIAL-5-1-3 SET CREDENTIAL IDENTIFIER – REPLACE OF THE SAME TYPE
17.12	Aug 17, 2017	Current document name was changed from Credential Test Specification to Credential Device Test Specification. The document formatting was updated.
18.06	Jun 21, 2018	Reformatting document using new template
18.12	Oct 12, 2018	The following were updated in the scope of #1672: Scope\Credential (updated with SetCredential) Test Policy\Credential (updated with SetCredential) CREDENTIAL-3-1-16 SET NEW CREDENTIAL (ENABLED) (added) CREDENTIAL-3-1-17 SET NEW CREDENTIAL (DISABLED) (added) CREDENTIAL-3-1-18 SET CREDENTIAL (added) Annex A.21 Create Pull Point Subscription (added) Annex A.22 Delete Subscription (added) Annex A.23 Retrieve Credential Changed Event by PullPoint (added)
18.12	Dec 21, 2018	Switching Hub description in 'Network Configuration for DUT' section was updated according to #1737
19.06	Jan 17, 2019	The following were updated in the scope of #1679:

		CREDENTIAL-1-1-2 GET SERVICES AND GET CREDENTIAL SERVICE CAPABILITIES CONSISTENCY (DefaultCredentialSuspensionDuration was added to the Note)
19.12	Aug 07, 2019	The following were added in the scope of #1880: Scope\Whitelist Management Test Policy\Whitelist Management Whitelist Management section CREDENTIAL-10-1-1 GET WHITELIST - START REFERENCE AND LIMIT CREDENTIAL-10-1-2 GET WHITELIST - FILTERS CREDENTIAL-10-1-3 ADD IDENTIFIER TO WHITELIST CREDENTIAL-10-1-4 REMOVE IDENTIFIER FROM WHITELIST CREDENTIAL-10-1-5 DELETE WHITELIST Annex A.24 Add Number of Credential Identifiers to Whitelist Annex A.25 Get Whitelist Annex A.26 Add Identifiers to Blacklist Annex A.27 Get Blacklist Annex A.28 Add Identifiers to Whitelist
19.12	Aug 08 2019	The following were updated in the scope of #1880 and #1882: CREDENTIAL-1-1-2 GET SERVICES AND GET CREDENTIAL SERVICE CAPABILITIES CONSISTENCY (ClientSuppliedTokenSupported, SupportedExemptionType, MaxWhitelistedItems, and MaxBlacklistedItems were added to the Note)
19.12	Aug 08, 2019	The following were added in the scope of #1882: Scope\Blacklist Management Test Policy\Blacklist Management Blacklist Management section CREDENTIAL-11-1-1 GET BLACKLIST - START REFERENCE AND LIMIT CREDENTIAL-11-1-2 GET BLACKLIST - FILTERS CREDENTIAL-11-1-3 ADD IDENTIFIER TO BLACKLIST CREDENTIAL-11-1-4 REMOVE IDENTIFIER FROM BLACKLIST CREDENTIAL-11-1-5 DELETE BLACKLIST Annex A.29 Add Number of Credential Identifiers to Blacklist
19.12	Oct 15, 2019	The following were added in the scope of #1898: Test Policy\Whitelist Management (updated with check for adding the same credential identifier to the list)

		Test Policy\Blacklist Management (updated with check for adding the same credential identifier to the list) CREDENTIAL-10-1-3 ADD IDENTIFIER TO WHITELIST (updated with check for adding the same credential identifier to the list, steps 17-20) CREDENTIAL-11-1-3 ADD IDENTIFIER TO BLACKLIST (updated with check for adding the same credential identifier to the list, steps 17-20)
19.12	Oct 23, 2019	The following were added in the scope of #1880 and #1882: Annex A.30 Generate Number of Credential Identifiers (was added) Annex A.24 Add Number of Credential Identifiers to Whitelist (was updated to use A.30) Annex A.29 Add Number of Credential Identifiers to Blacklist (was updated to use A.30) Annex A.26 Add Identifiers to Blacklist (was updated to add list of credential identifiers) Annex A.28 Add Identifiers to Whitelist (was updated to add list of credential identifiers) CREDENTIAL-10-1-3 ADD IDENTIFIER TO WHITELIST (was updated to use A.30) CREDENTIAL-10-1-4 REMOVE IDENTIFIER FROM WHITELIST (was updated to use A.30) CREDENTIAL-10-1-5 DELETE WHITELIST (was updated to use A.30) CREDENTIAL-11-1-3 ADD IDENTIFIER TO BLACKLIST (was updated to use A.30) CREDENTIAL-11-1-4 REMOVE IDENTIFIER FROM BLACKLIST (was updated to use A.30) CREDENTIAL-11-1-5 DELETE BLACKLIST (was updated to use A.30)
20.06	May 13, 2020	Pre-Requisite of the following test cases updated with adding of Pull-Point Notification feature according to #1999: CREDENTIAL-3-1-5 CREATE CREDENTIAL (ENABLED) CREDENTIAL-3-1-6 CREATE CREDENTIAL (DISABLED) CREDENTIAL-3-1-7 MODIFY CREDENTIAL CREDENTIAL-3-1-8 DELETE CREDENTIAL CREDENTIAL-3-1-16 SET NEW CREDENTIAL (ENABLED) CREDENTIAL-3-1-17 SET NEW CREDENTIAL (DISABLED) CREDENTIAL-3-1-18 SET CREDENTIAL CREDENTIAL-4-1-2 CHANGE CREDENTIAL STATE CREDENTIAL-5-1-2 SET CREDENTIAL IDENTIFIER – ADDING NEW TYPE

		CREDENTIAL-5-1-3 SET CREDENTIAL IDENTIFIER – REPLACE OF THE SAME TYPE CREDENTIAL-5-1-4 DELETE CREDENTIAL IDENTIFIER CREDENTIAL-6-1-2 SET CREDENTIAL ACCESS PROFILES - ADDING NEW ACCESS PROFILE CREDENTIAL-6-1-3 SET CREDENTIAL ACCESS PROFILES - UPDATING ACCESS PROFILE CREDENTIAL-6-1-4 DELETE CREDENTIAL ACCESS PROFILES CREDENTIAL-7-1-1 RESET ANTIPASSBACK VIOLATIONS
20.06	May 18, 2020	Pre-Requisite of the following test cases updated with adding of Event Service according to #1999: CREDENTIAL-8-1-2 CONFIGURATION CREDENTIAL CHANGED EVENT CREDENTIAL-8-1-3 CONFIGURATION CREDENTIAL REMOVED EVENT CREDENTIAL-8-1-4 CREDENTIAL STATE ENABLED EVENT CREDENTIAL-8-1-5 CREDENTIAL STATE ANTIPASSBACK VIOLATION EVENT (PROPERTY EVENT)
20.12	Oct 28, 2020	Pre-Requisite of the following test cases updated with adding of Credential Entity according to #1866: CREDENTIAL-2-1-1 GET CREDENTIAL INFO CREDENTIAL-2-1-2 GET CREDENTIAL INFO LIST - LIMIT CREDENTIAL-2-1-3 GET CREDENTIAL INFO LIST - START REFERENCE AND LIMIT CREDENTIAL-2-1-4 GET CREDENTIAL INFO LIST - NO LIMIT CREDENTIAL-2-1-5 GET CREDENTIAL INFO WITH INVALID TOKEN CREDENTIAL-2-1-6 GET CREDENTIAL INFO - TOO MANY ITEMS CREDENTIAL-3-1-1 GET CREDENTIALS CREDENTIAL-3-1-2 GET CREDENTIAL LIST - LIMIT CREDENTIAL-3-1-3 GET CREDENTIAL LIST - START REFERENCE AND LIMIT CREDENTIAL-3-1-4 GET CREDENTIAL LIST - NO LIMIT CREDENTIAL-3-1-5 CREATE CREDENTIAL (ENABLED) CREDENTIAL-3-1-6 CREATE CREDENTIAL (DISABLED) CREDENTIAL-3-1-7 MODIFY CREDENTIAL CREDENTIAL-3-1-8 DELETE CREDENTIAL CREDENTIAL-3-1-9 GET CREDENTIALS WITH INVALID TOKEN CREDENTIAL-3-1-10 GET CREDENTIALS - TOO MANY ITEMS

CREDENTIAL-3-1-11 CREATE CREDENTIAL - NOT EMPTY CREDENTIAL TOKEN

CREDENTIAL-3-1-12 MODIFY CREDENTIAL WITH INVALID TOKEN

CREDENTIAL-3-1-13 DELETE CREDENTIAL WITH INVALID TOKEN

CREDENTIAL-3-1-14 CREATE CREDENTIAL - VALIDITY VALUES

CREDENTIAL-3-1-15 MODIFY CREDENTIAL - VALIDITY VALUES

CREDENTIAL-3-1-16 SET NEW CREDENTIAL (ENABLED)

CREDENTIAL-3-1-17 SET NEW CREDENTIAL (DISABLED)

CREDENTIAL-3-1-18 SET CREDENTIAL

CREDENTIAL-4-1-1 GET CREDENTIAL STATE

CREDENTIAL-4-1-2 CHANGE CREDENTIAL STATE

CREDENTIAL-4-1-3 GET CREDENTIAL STATE WITH INVALID TOKEN

CREDENTIAL-4-1-4 ENABLE CREDENTIAL WITH INVALID TOKEN

CREDENTIAL-4-1-5 DISABLE CREDENTIAL WITH INVALID TOKEN

CREDENTIAL-5-1-1 GET CREDENTIAL IDENTIFIER

CREDENTIAL-5-1-2 SET CREDENTIAL IDENTIFIER – ADDING NEW TYPE

CREDENTIAL-5-1-3 SET CREDENTIAL IDENTIFIER – REPLACE OF THE SAME TYPE

CREDENTIAL-5-1-4 DELETE CREDENTIAL IDENTIFIER

CREDENTIAL-5-1-6 GET CREDENTIAL IDENTIFIERS WITH INVALID TOKEN

CREDENTIAL-5-1-7 SET CREDENTIAL IDENTIFIER WITH INVALID TOKEN

CREDENTIAL-5-1-8 DELETE CREDENTIAL IDENTIFIER WITH INVALID CREDENTIAL TOKEN

CREDENTIAL-5-1-9 DELETE CREDENTIAL IDENTIFIER WITH INVALID IDENTIFIER TYPE

CREDENTIAL-5-1-10 DELETE CREDENTIAL IDENTIFIER - MIN IDENTIFIERS PER CREDENTIAL

CREDENTIAL-6-1-1 GET CREDENTIAL ACCESS PROFILES

CREDENTIAL-6-1-2 SET CREDENTIAL ACCESS PROFILES - ADDING NEW ACCESS PROFILE

CREDENTIAL-6-1-3 SET CREDENTIAL ACCESS PROFILES - UPDATING ACCESS PROFILE

		CREDENTIAL-6-1-4 DELETE CREDENTIAL ACCESS PROFILES CREDENTIAL-6-1-5 GET CREDENTIAL ACCESS PROFILES WITH INVALID TOKEN CREDENTIAL-6-1-6 SET CREDENTIAL ACCESS PROFILES WITH INVALID CREDENTIAL TOKEN CREDENTIAL-6-1-7 DELETE CREDENTIAL ACCESS PROFILES WITH INVALID CREDENTIAL TOKEN CREDENTIAL-7-1-1 RESET ANTIPASSBACK VIOLATIONS CREDENTIAL-7-1-2 RESET ANTIPASSBACK VIOLATIONS WITH INVALID TOKEN CREDENTIAL-8-1-2 CONFIGURATION CREDENTIAL CHANGED EVENT CREDENTIAL-8-1-3 CONFIGURATION CREDENTIAL REMOVED EVENT CREDENTIAL-8-1-4 CREDENTIAL STATE ENABLED EVENT CREDENTIAL-8-1-5 CREDENTIAL STATE ANTIPASSBACK VIOLATION EVENT (PROPERTY EVENT) CREDENTIAL-9-1-1 GET CREDENTIAL AND GET ACCESS PROFILE INFO LIST CONSISTENCY

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 Core Specs] and [ONVIF Conformance] requirements. In addition, the test cases are to be basic inputs for some Profile specification requirements. It also describes the test framework, test setup, pre-requisites, test policies needed for the execution of the described test cases.

This ONVIF Credential Test Specification acts as a supplementary document to the [ONVIF Core Specs], illustrating test cases need to be executed and passed. In addition, this specification acts as an input document to the development of test tool that 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 Credential 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 the ONVIF Credential Security Service, which is defined in [ONVIF Credential Security Service].

The principal intended purposes are:

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

This specification does not address the following:

- Product use cases and non-functional (performance and regression) testing.
- SOAP Implementation Interoperability test i.e. Web Service Interoperability Basic Profile version 2.0 (WS-I BP 2.0).
- Full coverage of network protocol implementation test for HTTP, HTTPS, RTP, RTSP, and TLS protocols.

The set of ONVIF Test Specification will not cover the complete set of requirements as defined in [ONVIF Core Specs]; instead, it will cover its subset.

This ONVIF Credential Test Specification covers the ONVIF Credential Service, which is a functional block of [ONVIF Core Specs]. The following section gives a brief overview of each functional block and its scope.

1.1.1 Capabilities

The Capabilities section covers the test cases needed for getting capabilities from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting Credential service address with GetServices command via Device service
- Getting capabilities with GetServiceCapabilities command
- Getting capabilities with GetServices command via Device service

1.1.2 Credential Info

The Credential Info section covers the test cases needed for getting credential list and information from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting credential information with GetCredentialInfo command
- Getting credential information list with GetCredentialInfoList command

1.1.3 Credential

The Credential section covers the test cases needed for getting credential from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting credential with GetCredentials command
- Getting credential list with GetCredentialList command
- Creating credential with CreateCredential command
- Modifying credential with ModifyCredential command
- Deleting credential with DeleteCredential command
- Creating and modifying credential with SetCredential command

1.1.4 Credential States

The Credential States section covers the test cases needed for getting credential states from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting credential states with GetCredentialStates command
- Changing credential states with EnableCredential and DisableCredential commands

1.1.5 Credential Identifiers

The Credential Identifiers section covers the test cases needed for getting credential identifiers for specified credential from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting credential identifiers with GetCredentialIdentifiers command
- Adding credential identifiers with SetCredentialIdentifier command
- Updating credential identifiers with SetCredentialIdentifier command
- Deleting credential identifiers with DeleteCredentialIdentifier command

1.1.6 Credential Access Profiles

The Credential Access Profiles section covers the test cases needed for getting credential access profiles for specified credential from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Getting credential access profiles with GetCredentialAccessProfiles command
- Adding credential access profiles with SetCredentialAccessProfile command
- Updating credential access profiles with SetCredentialAccessProfile command
- Deleting credential access profiles with DeleteCredentialAccessProfile command

1.1.7 Reset Antipassback Violations

The Reset Antipassback Violations section covers the test cases needed for resetting antipassback violations for specified credential from an ONVIF device.

The scope of this specification section is to cover the following functions:

- Resetting antipassback violations with ResetAntipassbackViolation command

1.1.8 Credential Events

The Credential Events section covers the test cases needed for checking specified events format.

The scope of this specification section is to cover the following functions:

- Getting event properties with GetEventProperties command

1.1.9 Consistency

Consistency test cases cover verification of consistency between different entities and commands.

Consistency between the following entities is covered by the following test case:

- Credential and Access Profile Info

1.1.10 Whitelist Management

The Whitelist Management section covers the test cases required for managing whitelisted credential identifiers on ONVIF device.

This section contains test cases for the following functionality:

- Getting whitelisted credential identifiers list with GetWhitelist command
- Adding credential identifiers to whitelist with AddToWhitelist command
- Removing credential identifiers from whitelist with RemoveFromWhitelist command
- Removing all credential identifiers from whitelist with DeleteWhitelist command

1.1.11 Blacklist Management

The Blacklist Management section covers the test cases required for managing blacklisted credential identifiers on ONVIF device.

This section contains test cases for the following functionality:

- Getting blacklisted credential identifiers list with GetBlacklist command
- Adding credential identifiers to blacklist with AddToBlacklist command
- Removing credential identifiers from blacklist with RemoveFromBlacklist command
- Removing all credential identifiers from blacklist with DeleteBlacklist command

2 Terms and Definitions

2.1 Definitions

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

Credential A physical/tangible object, a piece of knowledge, or a facet of a person's physical being, that enables an individual access to a given physical facility or computer-based information system.

Credential Number A sequence of bytes uniquely identifying a credential at an access point.

Credential Identifier

Credential State

Validity Period From a certain point in time, to a later point in time. If a validity period is set on several entities (such as credentials, access profile and the association between them), the resulting validity period is the intersection of the three period.

Duress Forcing a person to provide access to a secure area against that person's wishes.

2.2 Abbreviations

This section describes abbreviations used in this document.

DUT Device Under Test

HTTP Hyper Text Transport Protocol

PACS Physical Access Control System

3 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.

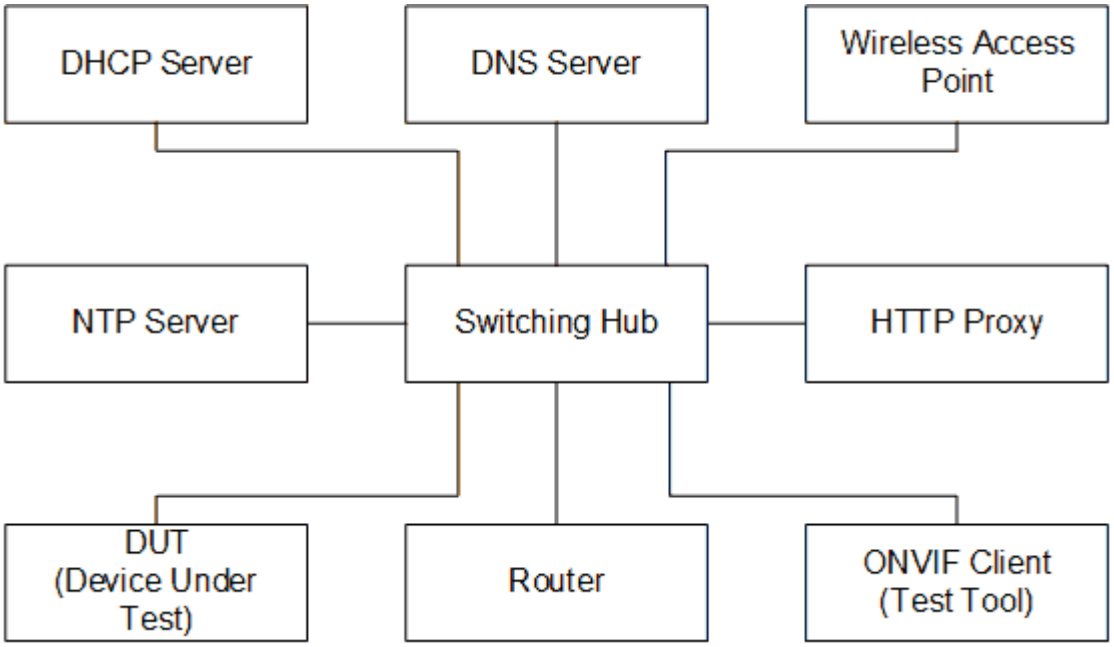
3.1 Test Setup

3.1.1 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 3.1. Test Configuration for DUT



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

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

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

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

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

DHCP Server: provides IPv4 Address to the connected devices.

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

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

Router: provides router advertisements for IPv6 configuration.

3.2 Prerequisites

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

- The DUT shall be configured with an IPv4 address.
- The DUT shall be IP reachable [in the test configuration].
- The DUT shall be able to be discovered by the Test Tool.
- The DUT shall be configured with the time, i.e. manual configuration of UTC time and if NTP is supported by DUT then NTP time shall be synchronized with NTP Server.

3.3 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.

3.3.1 Capabilities

The test policies specific to the test case execution of Capabilities functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetServiceCapabilities
- The following tests are performed

- Getting capabilities with GetServiceCapabilities command
- Getting capabilities with GetServices command

Please, refer to Section 4.1 for Capabilities Test Cases.

3.3.2 Credential Info

The test policies specific to the test case execution of Credential Info functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetServiceCapabilities
  - GetCredentialInfo
  - GetCredentialInfoList
- DUT shall not return more items in GetCredentialInfo and GetCredentialInfoList responses than specified in service capabilities by MaxLimit.
- DUT shall not return more items in GetCredentialInfoList response than specified by Limit parameter in a request.
- DUT shall not return items with the same tokens in GetCredentialInfoList responses for one credential info list receiving.
- DUT shall not return more CredentialInfo items in GetCredentialInfoList responses than specified in service capabilities by MaxCredentials.
- DUT shall not return any fault if GetCredentialInfo was invoked for non-exciting Credential token. Such tokens shall be ignored.
- DUT shall return SOAP 1.2 fault message (InvalidArgs/TooManyItems) if more items than MaxLimit was requested by GetCredentialInfo command.
- The following tests are performed
  - Getting credential info with GetCredentialInfo command
  - Getting credential info list with GetCredentialInfoList command with using different Limit and NextReference values

- Getting credential info with invalid credential token
- Getting credential info with number of requested items is greater than MaxLimit

Please refer to Section 4.2 for Credential Info Test Cases.

3.3.3 Credential

The test policies specific to the test case execution of Credential functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetServiceCapabilities
  - GetCredentials
  - GetCredentialList
  - GetCredentialInfoList
  - CreateCredential
  - ModifyCredential
  - DeleteCredential
- If DUT supports Client Supplied Token as indicated by Capabilities.ClientSuppliedTokenSupported = true, DUT shall support the following commands:
  - SetCredential
- DUT shall return only requested items in GetCredentials response that specified in GetCredentials request.
- DUT shall return all requested items in GetCredentials response that specified in GetCredentials request.
- DUT shall not return more items in GetCredentials responses than specified in service capabilities by MaxLimit.

- DUT shall return the same information in GetCredentials responses and in GetCredentialInfoList responses for the items with the same token.
- DUT shall not return more items in GetCredentialList response than specified by Limit parameter in a request.
- DUT shall not return items with the same tokens in GetCredentialList responses for one credential list receiving.
- DUT shall return the same information in GetCredentials responses and in GetCredentialList responses for the items with the same token.
- DUT shall return the same information in GetCredentialList responses and in GetCredentialInfoList responses for the items with the same token.
- DUT shall return the credentials in GetCredentialList responses and in GetCredentialInfoList responses.
- DUT shall return SOAP 1.2 fault message (InvalidArgs/TooManyItems) if more items than MaxLimit was requested by GetCredentials command.
- The DUT shall support creating of credential.
- The DUT shall support modifying of credential.
- The DUT shall support deleting of credential.
- If DUT supports Client Supplied Token as indicated by Capabilities.ClientSuppliedTokenSupported = true, creating and modifying of schedule using SetCredential command.
- The DUT shall support time value for validity of the credential if ValiditySupportsTimeValue is supported by the DUT.
- DUT shall return SOAP 1.2 fault message (InvalidArgs) if credential token is specified in CreateCredential request.
- DUT should return SOAP 1.2 fault message (InvalidArgVal/NotFound) if ModifyCredential or DeleteCredential command was invoked for non-exciting credential token.
- The following tests are performed
  - Getting credential with GetCredential command and test that it includes the same information with GetCredentialInfoList command
  - Getting credential info list with GetCredentialList command with using different Limit and NextReference values and test that it includes the same information with GetCredentialInfoList command

- Creating credential with CreateCredential command and test that corresponding notification message is received
- Modifying credential with ModifyCredential command and test that corresponding notification message is received
- Deleting credential with DeleteCredential command and test that corresponding notification message is received
- Getting credentials with invalid credential token
- Getting credentials with number of requested items is greater than MaxLimit
- Creating credential with CreateCredential command with specified token
- Modifying credential with ModifyCredential command with invalid token
- Deleting credential with DeleteCredential command with invalid token
- Creating credential with validity values
- Modifying credential with validity values
- If DUT supports Client Supplied Token as indicated by Capabilities.ClientSuppliedTokenSupported = true:
  - Creating schedule with SetCredential command with empty token and test that corresponding notification message is received
  - Modifying schedule with SetCredential command and test that corresponding notification message is received

Please refer to Section 4.3 for Credential Test Cases.

3.3.4 Credential State

The test policies specific to the test case execution of Credential functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetCredentialState

- GetCredentialInfoList
- EnableCredential
- DisableCredential

- DUT should return SOAP 1.2 fault message (InvalidArgVal\NotFound) if GetCredentialState or EnableCredential or DisableCredential command was invoked for non-exciting credential token.
- The following tests are performed
  - Getting credential state with GetCredentialState command
  - Changing credential states with EnableCredential and DisableCredential commands
  - Getting credential state with GetCredentialState command with invalid token
  - Changing credential states with EnableCredential and DisableCredential commands with invalid token

Please refer to Section 4.4 for Credential State Test Cases.

3.3.5 Credential Identifiers

The test policies specific to the test case execution of Credential Identifiers functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetCredentialIdentifiers
  - GetCredentialList
  - SetCredentialIdentifier
  - DeleteCredentialIdentifier
- DUT shall return the same credential identifiers information in GetCredentialList responses and in GetCredentialIdentifiers responses for the credentials with the same token.
- DUT shall not return items with the same type name in GetCredentialIdentifiers responses.

- DUT shall not return items in GetCredentials responses with Type.Name other than specified in service capabilities by SupportedIdentifierTypes.
- DUT should return SOAP 1.2 fault message (InvalidArgVal\NotFound) if GetCredentialIdentifiers or SetCredentialIdentifier or DeleteCredentialIdentifier command was invoked for non-exciting credential token.
- DUT should return SOAP 1.2 fault message (InvalidArgVal\NotFound) if DeleteCredentialIdentifiers command was invoked for non-exciting identifier type.
- DUT should return SOAP 1.2 fault message (ConstraintViolated\MinIdentifiersPerCredential) if DeleteCredentialIdentifiers command was invoked when credential contains only one Credential Identifier.
- The following tests are performed
  - Getting credential identifiers with GetCredentialIdentifiers command
  - Adding credential identifiers with SetCredentialIdentifier command
  - Updating credential identifiers with SetCredentialIdentifier command
  - Deleting credential identifiers with DeleteCredentialIdentifier command
  - Getting credential identifiers with GetCredentialIdentifiers command with invalid token
  - Adding credential identifiers with SetCredentialIdentifier command with invalid token
  - Deleting credential identifiers with DeleteCredentialIdentifier command with invalid credential token
  - Deleting credential identifiers with DeleteCredentialIdentifier command with invalid identifier type
  - Deleting the last credential identifier with DeleteCredentialIdentifier command

Please refer to Section 4.5 for Credential Identifiers Test Cases.

3.3.6 Credential Access Profiles

The test policies specific to the test case execution of Credential Access Profiles functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:

  - GetServices
  - GetCredentialAccessProfiles
  - GetCredentialList
  - SetCredentialAccessProfile
  - DeleteCredentialAccessProfile
- DUT shall return the same credential access profiles information in GetCredentialList responses and in GetCredentialAccessProfiles responses for the credentials with the same token.
- DUT shall not return items with the same access profile token in GetCredentialAccessProfiles responses.
- DUT shall not return more items in GetCredentialAccessProfiles responses than specified in service capabilities by MaxAccessProfilesPerCredential.
- The DUT shall support time value for of the validity for the association between the credential and the access profile if ValiditySupportsTimeValue is supported by the DUT.
- DUT should return SOAP 1.2 fault message (InvalidArgVal \NotFound) if GetCredentialAccessProfiles or SetCredentialAccessProfile or DeleteCredentialAccessProfile command was invoked for non-exciting credential token.
- The following tests are performed
  - Getting credential access profiles with GetCredentialAccessProfiles command
  - Adding credential access profiles with SetCredentialAccessProfile command
  - Updating credential access profiles with SetCredentialAccessProfile command
  - Deleting credential access profiles with DeleteCredentialAccessProfile command
  - Getting credential access profiles with GetCredentialAccessProfiles command with invalid token
  - Adding credential access profiles with SetCredentialAccessProfile command with invalid token
  - Deleting credential access profiles with DeleteCredentialAccessProfile command with invalid token

Please refer to Section 4.6 for Credential Access Profiles Test Cases.

3.3.7 Reset Antipassback Violations

The test policies specific to the test case execution of Reset Antipassback Violations functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- If DUT returns Reset Antipassback Violations capability as supported, then DUT shall support ResetAntipassbackViolation command. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - CreateCredential
- DUT should return SOAP 1.2 fault message (InvalidArgVal\NotFound) if ResetAntipassbackViolation command was invoked for non-exciting credential token.
- The following tests are performed:
  - Resetting antipassback violations with ResetAntipassbackViolation command
  - Resetting antipassback violations with ResetAntipassbackViolation command with invalid credential token.

Please refer to Section 4.7 for Reset Antipassback Violations Test Cases.

3.3.8 Credential Events

The test policies specific to the test case execution of Credential Events functional block:

- DUT shall give the Credential Service and Event Service entry points by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetEventProperties
- The following tests are performed
  - Getting event properties with GetEventProperties command

Please refer to Section 4.8 for Credential Events Test Cases.

3.3.9 Consistency

The test policies specific to the test case execution of Consistency functional block:

- DUT shall give the Credential Service and Access Rules Service entry points by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall support the following commands:
  - GetServices
  - GetAccessProfileInfo
  - GetCredentials
- The following tests are performed
  - Credential and Access Profile Info

Please refer to Section 4.9 for Consistency Test Cases.

3.3.10 Whitelist Management

The test policies specific to the test case execution of Whitelist Management functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall have capability MaxWhitelistedItems greater than zero, if DUT supports whitelist functionality. Otherwise, these test cases will be skipped.
- DUT shall support the following commands and notification topics:
  - GetWhitelist
  - AddToWhitelist
  - RemoveFromWhitelist
  - DeleteWhitelist
  - tns1:AccessControl/AccessGranted/Identifier
- Additionally, DUT shall support the following commands which will be used as supplementary during the testing:
  - GetServices

    - GetServiceCapabilities
    - CreatePullPointSubscription
    - PullMessages
    - Unsubscribe
    - GetSupportedFormatTypes
    - AddToBlacklist (if Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability)
- DUT shall return only requested items in GetWhitelist response as specified in GetWhitelist request by IdentifierType, FormatType, and Value fields.
- DUT shall not return more items in GetWhitelist responses than specified in service capabilities by MaxLimit.
- DUT shall not return more items in GetWhitelist response than specified by Limit parameter in a request.
- DUT shall not return same credential identifiers in GetWhitelist responses for one whitelist list receiving.
- The DUT shall support adding of credential identifiers to whitelist.
- If Client sends request with credential identifier with is already in whitelist, the DUT shall return no fault and shall not add provided identifier to the whitelist.
- The DUT shall support removing of specified credential identifiers from whitelist.
- The DUT shall support removing of all credential identifiers from whitelist.
- If DUT supports blacklist functionality as indicated by MaxBlacklistedItems greater than zero capability:
    - The DUT shall remove credential identifiers from blacklist if it was added to whitelist.
- The following tests are performed
    - (TODO: #1899)
    - Getting whitelist with GetWhitelist command using different Limit and NextReference values and test that:
        - the lists contains the same items independently from Limit value;

- the GetWhitelistResponse contains not more items that specified in Limit value;
- the GetWhitelistResponse contains not more items that specified in MaxLimit capability.

- Adding credential identifiers to whitelist with AddToWhitelist command and test that:
  - each credential identifier was added to whitelist;
  - credential identifiers that were in blacklist was removed from it after addition to whitelist.
- Adding credential identifiers which is already in whitelist to whitelist with AddToWhitelist command and test that:
  - no duplicated credential identifier was added to whitelist.
- Removing credential identifiers from whitelist with RemoveFromWhitelist command and test that credential identifiers were removed from whitelist.
- Removing credential identifiers that are not in whitelist from whitelist with RemoveFromWhitelist command and test that no faults were returned.
- Delete all credential identifiers from whitelist with DeleteWhitelist command and test that all credential identifiers were removed from whitelist.
- Delete all credential identifiers from empty whitelist with DeleteWhitelist command and test that no faults were returned.

Please refer to Section 4.10 for Whitelist Management Test Cases.

3.3.11 Blacklist Management

The test policies specific to the test case execution of Blacklist Management functional block:

- DUT shall give the Credential Service entry point by GetServices command, if DUT supports this service. Otherwise, these test cases will be skipped.
- DUT shall have capability MaxBlacklistedItems greater than zero, if DUT supports blacklist functionality. Otherwise, these test cases will be skipped.
- DUT shall support the following commands and notification topics:
  - GetBlacklist
  - AddToBlacklist
  - RemoveFromBlacklist

- DeleteBlacklist

- Additionally, DUT shall support the following commands which will be used as supplementary during the testing:
  - GetServices
  - GetServiceCapabilities
  - GetSupportedFormatTypes
  - AddToBlacklist (if Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability)
- DUT shall return only requested items in GetBlacklist response as specified in GetBlacklist request by IdentifierType, FormatType, and Value fields.
- DUT shall not return more items in GetBlacklist responses than specified in service capabilities by MaxLimit.
- DUT shall not return more items in GetBlacklist response than specified by Limit parameter in a request.
- DUT shall not return same credential identifiers in GetBlacklist responses for one blacklist list receiving.
- The DUT shall support adding of credential identifiers to blacklist.
- If Client sends request with credential identifier with is already in blacklist, the DUT shall return no fault and shall not add provided identifier to the blacklist.
- The DUT shall support removing of specified credential identifiers from blacklist.
- The DUT shall support removing of all credential identifiers from blacklist.
- If DUT supports whitelist functionality as indicated by MaxWhitelistedItems greater than zero capability:
  - The DUT shall remove credential identifiers from whitelist if it was added to blacklist.
- The following tests are performed
  - (TODO: #1899)
  - Getting blacklist with GetBlacklist command using different Limit and NextReference values and test that:

  - the lists contains the same items independently from Limit value;
  - the GetBlacklistResponse contains not more items that specified in Limit value;
  - the GetBlacklistResponse contains not more items that specified in MaxLimit capability.
- Adding credential identifiers to blacklist with AddToBlacklist command and test that:
  - each credential identifier was added to blacklist;
  - credential identifiers that were in blacklist was removed from it after addition to blacklist.
- Adding credential identifiers which is already in blacklist to blacklist with AddToBlacklist command and test that:
  - no duplicated credential identifier was added to blacklist.
- Removing credential identifiers from blacklist with RemoveFromBlacklist command and test that credential identifiers were removed from blacklist.
- Removing credential identifiers that are not in blacklist from blacklist with RemoveFromBlacklist command and test that no faults were returned.
- Delete all credential identifiers from blacklist with DeleteBlacklist command and test that all credential identifiers were removed from blacklist.
- Delete all credential identifiers from empty blacklist with DeleteBlacklist command and test that no faults were returned.

Please refer to Section 4.10 for Blacklist Management Test Cases.

4 Credential Test Cases

4.1 Capabilities

4.1.1 CREDENTIAL SERVICE CAPABILITIES

Test Case ID: CREDENTIAL-1-1-1

Specification Coverage: ServiceCapabilities (ONVIF Credential Service Specification), GetServiceCapabilities command (ONVIF Credential Service Specification)

Feature Under Test: GetServiceCapabilities (for Credential Service)

WSDL Reference: credential.wsdl

Test Purpose: To verify DUT Credential Service Capabilities.

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

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServiceCapabilities**.
4. The DUT responds with a **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.1.2 GET SERVICES AND GET CREDENTIAL SERVICE CAPABILITIES CONSISTENCY

Test Case ID: CREDENTIAL-1-1-2

Specification Coverage: Capability exchange (ONVIF Core Specification), ServiceCapabilities (ONVIF Credential Service Specification), GetServiceCapabilities command (ONVIF Credential Service Specification)

Feature Under Test: GetServices, GetServiceCapabilities (for Credential Service)

WSDL Reference: devicemgmt.wsdl, credential.wsdl

Test Purpose: To verify Get Services and Credential Service Capabilities consistency.

Pre-Requisite: None.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **GetServices** with parameters
   - IncludeCapability := true
4. The DUT responds with a **GetServicesResponse** message with parameters
   - Services list =: *servicesList*
5. ONVIF Client selects Service with Service.Namespace = "http://www.onvif.org/ver10/credential/wsdl":
   - Services list [Namespace = "http://www.onvif.org/ver10/credential/wsdl"] =: *credentialService*
6. ONVIF Client invokes **GetServiceCapabilities**.
7. The DUT responds with a **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*
8. If *cap* differs from *credentialService*.Capabilities.Capabilities, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at step 8:

- SupportedIdentifierType list
- MaxLimit
- CredentialValiditySupported
- CredentialAccessProfileValiditySupported
- MaxCredentials
- MaxAccessProfilesPerCredential
- ResetAntipassbackSupported
- ValiditySupportsTimeValue
- DefaultCredentialSuspensionDuration
- ClientSuppliedTokenSupported
- Extension.SupportedExemptionType
- MaxWhitelistedItems
- MaxBlacklistedItems

4.2 Credential Info

4.2.1 GET CREDENTIAL INFO

Test Case ID: CREDENTIAL-2-1-1

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfo command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfo

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. If *credentialInfoCompleteList* is empty, skip other steps.
5. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
6. Set the following:
   - *tokenList* := [subset of *credentialInfoCompleteList*.token values with items number equal to *cap*.MaxLimit]
7. ONVIF client invokes **GetCredentialInfo** with parameters
   - Token list := *tokenList*
8. The DUT responds with **GetCredentialInfoResponse** message with parameters
   - CredentialInfo list =: *credentialInfoList1*
9. If *credentialInfoList1* does not contain CredentialInfo item for each token from *tokenList*, FAIL the test and skip other steps.
10. If *credentialInfoList1* contains at least two CredentialInfo item with equal token, FAIL the test and skip other steps.
11. If *credentialInfoList1* contains other CredentialInfo items than listed in *tokenList*, FAIL the test and skip other steps.
12. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:

    12.1. ONVIF client invokes **GetCredentialInfo** with parameters

- Token[0] := *token*

12.2. The DUT responds with **GetCredentialInfoResponse** message with parameters

- CredentialInfo list =: *credentialInfoList2*

12.3. If *credentialInfoList2* does not contain only one CredentialInfo item with token equal to *token*, FAIL the test and skip other steps.

12.4. If *credentialInfoList2*[0] item is not equal to *credentialInfoCompleteList*[token = *token*] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: If number of items in *credentialInfoCompleteList* is less than *cap*.MaxLimit, then all *credentialInfoCompleteList*.Token items shall be used for the step 6.

Note: The following fields are compared at step 12.4:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.2.2 GET CREDENTIAL INFO LIST - LIMIT

Test Case ID: CREDENTIAL-2-1-2

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfoList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfoList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info List using Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialInfoList** with parameters
   - Limit := 1
   - StartReference skipped
5. The DUT responds with **GetCredentialInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - CredentialInfo list =: *credentialInfoList1*
6. If *credentialInfoList1* contains more CredentialInfo items than 1, FAIL the test and skip other steps.
7. If *cap*.MaxLimit is equal to 1, skip other steps.
8. ONVIF client invokes **GetCredentialInfoList** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference skipped
9. The DUT responds with **GetCredentialInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - CredentialInfo list =: *credentialInfoList2*
10. If *credentialInfoList2* contains more CredentialInfo items than *cap*.MaxLimit, FAIL the test and skip other steps.

11. If *cap*.MaxLimit is equal to 2, skip other steps.
12. Set the following:
    - *limit* := [number between 1 and *cap*.MaxLimit]
13. ONVIF client invokes **GetCredentialInfoList** with parameters
    - Limit := *limit*
    - StartReference skipped
14. The DUT responds with **GetCredentialInfoListResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - CredentialInfo list =: *credentialInfoList3*
15. If *credentialInfoList3* contains more CredentialInfo items than *limit*, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.2.3 GET CREDENTIAL INFO LIST - START REFERENCE AND LIMIT

Test Case ID: CREDENTIAL-2-1-3

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfoList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfoList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info List using StartReference and Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialInfoList** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference skipped
5. The DUT responds with **GetCredentialInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - CredentialInfo list =: *credentialInfoCompleteList1*
6. If *credentialInfoCompleteList1* contains more CredentialInfo items than *cap*.MaxLimit, FAIL the test and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:
   7.1. ONVIF client invokes **GetCredentialInfoList** with parameters
      - Limit := *cap*.MaxLimit
      - StartReference := *nextStartReference*
   7.2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - CredentialInfo list =: *credentialInfoListPart*
   7.3. If *credentialInfoListPart* contains more CredentialInfo items than *cap*.MaxLimit, FAIL the test and skip other steps.
   7.4. Set the following:
      - *credentialInfoCompleteList1* := *credentialInfoCompleteList1* + *credentialInfoListPart*
8. If *credentialInfoCompleteList1* contains at least two CredentialInfo items with equal token, FAIL the test and skip other steps.

9. If *cap*.MaxLimit is equal to 1, skip other steps.
10. ONVIF client invokes **GetCredentialInfoList** with parameters
    - Limit := 1
    - StartReference skipped
11. The DUT responds with **GetCredentialInfoListResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - CredentialInfo list =: *credentialInfoCompleteList2*
12. If *credentialInfoCompleteList2* contains more CredentialInfo items than 1, FAIL the test and skip other steps.
13. Until *nextStartReference* is not null, repeat the following steps:
    13.1. ONVIF client invokes **GetCredentialInfoList** with parameters
        - Limit := 1
        - StartReference := *nextStartReference*
    13.2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
        - NextStartReference =: *nextStartReference*
        - CredentialInfo list =: *credentialInfoListPart*
    13.3. If *credentialInfoListPart* contains more CredentialInfo items than 1, FAIL the test and skip other steps.
    13.4. Set the following:
        - *credentialInfoCompleteList2* := *credentialInfoCompleteList2* + *credentialInfoListPart*
14. If *credentialInfoCompleteList2* contains at least two CredentialInfo items with equal token, FAIL the test and skip other steps.
15. If *credentialInfoCompleteList2* does not contain all credentials from *credentialInfoCompleteList1*, FAIL the test and skip other steps.
16. If *credentialInfoCompleteList2* contains credentials other than credentials from *credentialInfoCompleteList1*, FAIL the test and skip other steps.
17. If *cap*.MaxLimit is equal to 2, skip other steps.

18. Set the following:
    - *limit* := [number between 1 and *cap*.MaxLimit]
19. ONVIF client invokes **GetCredentialInfoList** with parameters
    - Limit := *limit*
    - StartReference skipped
20. The DUT responds with **GetCredentialInfoListResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - CredentialInfo list =: *credentialInfoCompleteList3*
21. If *credentialInfoCompleteList3* contains more CredentialInfo items than *limit*, FAIL the test and skip other steps.
22. Until *nextStartReference* is not null, repeat the following steps:
    22.1. ONVIF client invokes **GetCredentialInfoList** with parameters
        - Limit := *limit*
        - StartReference := *nextStartReference*

    22.2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
        - NextStartReference =: *nextStartReference*
        - CredentialInfo list =: *credentialInfoListPart*

    22.3. If *credentialInfoListPart* contains more CredentialInfo items than *limit*, FAIL the test and skip other steps.

    22.4. Set the following:
        - *credentialInfoCompleteList3* := *credentialInfoCompleteList3* + *credentialInfoListPart*
23. If *credentialInfoCompleteList3* contains at least two CredentialInfo items with equal token, FAIL the test and skip other steps.
24. If *credentialInfoCompleteList3* does not contain all credentials from *credentialInfoCompleteList1*, FAIL the test and skip other steps.
25. If *credentialInfoCompleteList3* contains credentials other than credentials from *credentialInfoCompleteList1*, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.2.4 GET CREDENTIAL INFO LIST - NO LIMIT

Test Case ID: CREDENTIAL-2-1-4

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfoList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfoList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info List without using Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialInfoList** with parameters
   - Limit skipped
   - StartReference skipped
5. The DUT responds with **GetCredentialInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - CredentialInfo list =: *credentialInfoCompleteList*

6. If *credentialInfoCompleteList* contains more CredentialInfo items than *cap*.MaxLimit, FAIL the test and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:
   7.1. ONVIF client invokes **GetCredentialInfoList** with parameters
      - Limit skipped
      - StartReference := *nextStartReference*
   7.2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - CredentialInfo list =: *credentialInfoListPart*
   7.3. If *credentialInfoListPart* contains more CredentialInfo items than *cap*.MaxLimit, FAIL the test and skip other steps.
   7.4. Set the following:
      - *credentialInfoCompleteList* := *credentialInfoCompleteList* + *credentialInfoListPart*
8. If *credentialInfoCompleteList* contains at least two CredentialInfo items with equal token, FAIL the test.
9. If *credentialInfoCompleteList* contains more CredentialInfo items than *cap*.MaxCredentials, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.2.5 GET CREDENTIAL INFO WITH INVALID TOKEN

Test Case ID: CREDENTIAL-2-1-5

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfo command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfo

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF client invokes **GetCredentialInfo** with parameters
   - Token[0] := *invalidToken*
6. The DUT responds with **GetCredentialInfoResponse** message with parameters
   - CredentialInfo list =: *credentialInfoList*
7. If *credentialInfoList* is not empty, FAIL the test.
8. If *credentialInfoCompleteList* is empty, skip other steps.
9. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
10. If *cap*.MaxLimit is less than 2, skip other steps.
11. ONVIF client invokes **GetCredentialInfo** with parameters
    - Token[0] := *invalidToken*
    - Token[1] := *credentialInfoCompleteList*[0].token
12. The DUT responds with **GetCredentialInfoResponse** message with parameters

- CredentialInfo list =: *credentialInfoList*

13. If *credentialInfoList* is empty, FAIL the test.
14. If *credentialInfoList* contains more than one item, FAIL the test.
15. If *credentialInfoList*[0].token does not equal to *credentialInfoCompleteList*[0].token, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.2.6 GET CREDENTIAL INFO - TOO MANY ITEMS

Test Case ID: CREDENTIAL-2-1-6

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), GetCredentialInfo command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialInfo

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Info in case if there a more items than MaxLimit in request.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

4. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
5. If *credentialInfoCompleteList*.token items number is less than *cap*.MaxLimit or equal to *cap*.MaxLimit, skip other steps.
6. Set the following:
   - *tokenList* := [subset of *credentialInfoCompleteList*.token values with items number equal to *cap*.MaxLimit + 1]
7. ONVIF client invokes **GetCredentialInfo** with parameters
   - Token list := *tokenList*
8. The DUT returns **env:Sender\ter:InvalidArgs\ter:TooManyItems** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgs\ter:TooManyItems** SOAP 1.2 fault.

4.3 Credential

4.3.1 GET CREDENTIALS

Test Case ID: CREDENTIAL-3-1-1

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), GetCredentials command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentials

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
4. If *credentialCompleteList* is empty, skip other steps.
5. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
6. Set the following:
   - *tokenList* := [subset of *credentialCompleteList*.token values with items number equal to *cap*.MaxLimit]
7. ONVIF client invokes **GetCredentials** with parameters
   - Token list := *tokenList*
8. The DUT responds with **GetCredentialsResponse** message with parameters
   - Credential list =: *credentialList1*
9. If *credentialList1* does not contain Credential item for each token from *tokenList*, FAIL the test and skip other steps.
10. If *credentialList1* contains at least two Credential items with equal token, FAIL the test and skip other steps.
11. If *credentialList1* contains other Credential items than listed in *tokenList*, FAIL the test and skip other steps.
12. For each Credential.token *token* from *credentialCompleteList* repeat the following steps:

    12.1. ONVIF client invokes **GetCredentials** with parameters
    - Token[0] := *token*

    12.2. The DUT responds with **GetCredentialsResponse** message with parameters
    - Credential list =: *credentialList2*

12.3. If *credentialList2* does not contain only one Credential item with token equal to *token*, FAIL the test and skip other steps.

12.4. If *credentialList2*[0] item does not have equal field values to *credentialCompleteList*[token = *token*] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: If number of items in *credentialCompleteList* is less than *cap*.MaxLimit, then all *credentialCompleteList*.Token items shall be used for the step 6.

Note: The following fields are compared at step 12.4:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)
    - Type
      - Name
      - FormatType
    - ExemptedFromAuthentication
    - Value
  - CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
    - AccessProfileToken

- ValidFrom
- ValidTo

- Attribute list
  - Name
  - Value

4.3.2 GET CREDENTIAL LIST - LIMIT

Test Case ID: CREDENTIAL-3-1-2

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), GetCredentialList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential List using Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialList** with parameters
   - Limit := 1
   - StartReference skipped
5. The DUT responds with **GetCredentialListResponse** message with parameters
   - NextStartReference =: *nextStartReference*

- Credential list =: *credentialList1*

6. If *credentialList1* contains more Credential items than 1, FAIL the test and skip other steps.
7. If *cap*.MaxLimit is equal to 1, skip other steps.
8. ONVIF client invokes **GetCredentialList** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference skipped
9. The DUT responds with **GetCredentialListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Credential list =: *credentialList2*
10. If *credentialList2* contains more Credential items than *cap*.MaxLimit, FAIL the test and skip other steps.
11. If *cap*.MaxLimit is equal to 2, skip other steps.
12. Set the following:
    - *limit* := [number between 1 and *cap*.MaxLimit]
13. ONVIF client invokes **GetCredentialList** with parameters
    - Limit := *limit*
    - StartReference skipped
14. The DUT responds with **GetCredentialListResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Credential list =: *credentialList3*
15. If *credentialList3* contains more Credential items than *limit*, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.3.3 GET CREDENTIAL LIST - START REFERENCE AND LIMIT

Test Case ID: CREDENTIAL-3-1-3

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), GetCredentialList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential List using StartReference and Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialList** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference skipped
5. The DUT responds with **GetCredentialListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Credential list =: *credentialCompleteList1*
6. If *credentialCompleteList1* contains more Credential items than *cap*.MaxLimit, FAIL the test and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:

   7.1. ONVIF client invokes **GetCredentialList** with parameters

- Limit := *cap*.MaxLimit
- StartReference := *nextStartReference*

7.2. The DUT responds with **GetCredentialListResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Credential list =: *credentialListPart*

7.3. If *credentialListPart* contains more Credential items than *cap*.MaxLimit, FAIL the test and skip other steps.

7.4. Set the following:

- *credentialCompleteList1* := *credentialCompleteList1* + *credentialListPart*

8. If *credentialCompleteList1* contains at least two Credential item with equal token, FAIL the test and skip other steps.

9. If *cap*.MaxLimit is equal to 1, do the following steps:

9.1. ONVIF Client retrieves a complete list of credentials (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

9.2. If *credentialCompleteList1* does not contain all credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.

9.3. If *credentialCompleteList1* contains credentials other than credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.

9.4. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:

9.4.1. If *credentialCompleteList1*[token = *token*] item does not have equal field values to *credentialInfoCompleteList*[token = *token*] item, FAIL the test and skip other steps.

9.5. Skip other steps.

10. ONVIF client invokes **GetCredentialList** with parameters

- Limit := 1
- StartReference skipped

11. The DUT responds with **GetCredentialListResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Credential list =: *credentialCompleteList2*

12. If *credentialCompleteList2* contains more Credential items than 1, FAIL the test and skip other steps.
13. Until *nextStartReference* is not null, repeat the following steps:
    13.1. ONVIF client invokes **GetCredentialList** with parameters
        - Limit := 1
        - StartReference := *nextStartReference*
    13.2. The DUT responds with **GetCredentialListResponse** message with parameters
        - NextStartReference =: *nextStartReference*
        - Credential list =: *credentialListPart*
    13.3. If *credentialListPart* contains more Credential items than 1, FAIL the test and skip other steps.
    13.4. Set the following:
        - *credentialCompleteList2* := *credentialCompleteList2* + *credentialListPart*
14. If *credentialCompleteList2* contains at least two Credential item with equal token, FAIL the test and skip other steps.
15. If *credentialCompleteList2* does not contain all credentials from *credentialCompleteList1*, FAIL the test and skip other steps.
16. If *credentialCompleteList2* contains credentials other than credentials from *credentialCompleteList1*, FAIL the test and skip other steps.
17. If *cap*.MaxLimit is equal to 2 do the following steps:
    17.1. ONVIF Client retrieves a complete list of credentials (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
    17.2. If *credentialCompleteList2* does not contain all credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.
    17.3. If *credentialCompleteList2* contains credentials other than credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.

17.4. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:

17.4.1. If *credentialCompleteList2*[token = *token*] item does not have equal field values to *credentialInfoCompleteList*[token = *token*] item, FAIL the test and skip other steps.

17.5. Skip other steps.

18. Set the following:

- *limit* := [number between 1 and *cap*.MaxLimit]

19. ONVIF client invokes **GetCredentialList** with parameters

- Limit := *limit*
- StartReference skipped

20. The DUT responds with **GetCredentialListResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Credential list =: *credentialCompleteList3*

21. If *credentialCompleteList3* contains more Credential items than *limit*, FAIL the test and skip other steps.

22. Until *nextStartReference* is not null, repeat the following steps:

22.1. ONVIF client invokes **GetCredentialList** with parameters

- Limit := *limit*
- StartReference := *nextStartReference*

22.2. The DUT responds with **GetCredentialListResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Credential list =: *credentialListPart*

22.3. If *credentialListPart* contains more Credential items than *limit*, FAIL the test and skip other steps.

22.4. Set the following:

- *credentialCompleteList3* := *credentialCompleteList3* + *credentialListPart*

23. If *credentialCompleteList3* contains at least two Credential items with equal token, FAIL the test and skip other steps.

24. If *credentialCompleteList3* does not contain all credentials from *credentialCompleteList1*, FAIL the test and skip other steps.

25. If *credentialCompleteList3* contains credentials other than credentials from *credentialCompleteList1*, FAIL the test and skip other steps.

26. ONVIF Client retrieves a complete list of credentials (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

27. If *credentialCompleteList3* does not contain all credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.

28. If *credentialCompleteList3* contains credentials other than credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.

29. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:

    29.1. If *credentialCompleteList3*[token = *token*] item does not have equal field values to *credentialInfoCompleteList*[token = *token*] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at step 29.1:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.3.4 GET CREDENTIAL LIST - NO LIMIT

Test Case ID: CREDENTIAL-3-1-4

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), GetCredentialList command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialList

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential List without using Limit.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF client invokes **GetCredentialList** with parameters
   - Limit skipped
   - StartReference skipped
5. The DUT responds with **GetCredentialListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Credential list =: *credentialCompleteList*
6. If *credentialCompleteList* contains more Credential items than *cap*.MaxLimit, FAIL the test and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:
   7.1. ONVIF client invokes **GetCredentialList** with parameters
      - Limit skipped

- StartReference := *nextStartReference*

7.2. The DUT responds with **GetCredentialListResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Credential list =: *credentialListPart*

7.3. If *credentialListPart* contains more Credential items than *cap*.MaxLimit, FAIL the test and skip other steps.

7.4. Set the following:

- *credentialCompleteList* := *credentialCompleteList* + *credentialListPart*

8. If *credentialCompleteList* contains at least two Credential item with equal token, FAIL the test.
9. ONVIF Client retrieves a complete list of credentials (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
10. If *credentialCompleteList* does not contain all credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.
11. If *credentialCompleteList* contains credentials other than credentials from *credentialInfoCompleteList*, FAIL the test and skip other steps.
12. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:

12.1. If *credentialCompleteList*[token = *token*] item does not have equal field values to *credentialInfoCompleteList*[token = *token*] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at step 12.1:

- CredentialInfo:
  - token

- Description
- CredentialHolderToken
- ValidFrom
- ValidTo

4.3.5 CREATE CREDENTIAL (ENABLED)

Test Case ID: CREDENTIAL-3-1-5

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), CreateCredential command (ONVIF Credential Service Specification)

Feature Under Test: CreateCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify creation of enabled credential and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList1*) by following the procedure mentioned in Annex A.3.
5. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
6. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.

7. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.
8. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
9. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
10. ONVIF client invokes **CreateCredential** with parameters
    - Credential.token := ""
    - Credential.Description := "Test Description"
    - Credential.CredentialHolderReference := "TestUser"
    - Credential.ValidFrom skipped
    - Credential.ValidTo skipped
    - Credential.CredentialIdentifier[0].Type.Name := *typeName*
    - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
    - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
    - Credential.CredentialIdentifier[0].Value := *value*
    - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token or skipped (if *accessProfileCompleteList* is empty)
    - Credential.CredentialAccessProfile.ValidFrom skipped
    - Credential.CredentialAccessProfile.ValidTo skipped
    - Credential.Extension skipped

- State.Enabled := true
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

11. The DUT responds with **CreateCredentialResponse** message with parameters
    - Token =: *credentialToken*
12. Until *oprationDelay* timeout expires, repeat the following steps:
    12.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    12.2. ONVIF Client invokes **PullMessages** to the subscription endpoint s with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    12.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    12.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 24.
    12.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 24.
    12.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 13.
13. If *oprationDelay* timeout expires for step 12 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 24.
14. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.
15. If *credentialList*[0] item does not have equal field values to values from step 10, FAIL the test and go step 24.

16. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.
17. If *credentialInfoList*[0] item does not have equal field values to values from step 9, FAIL the test and go step 24.
18. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
19. If *credentialInfoCompleteList* does not have *credentialInfo*.[token = *credentialToken*] item with equal field values to values from step 9, FAIL the test and go step 24.
20. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList2*) by following the procedure mentioned in Annex A.3.
21. If *credentialCompleteList2* does not have *credential*.[token = *credentialToken*] item with equal field values to values from step 10, FAIL the test and go step 24.
22. ONVIF client retrieves credential state (in *credentialToken*, out *credentialState*) by following the procedure mentioned in Annex A.13.
23. Check the following:
    23.1. If *credentialState*[0].Enabled equal to false, FAIL the test and go step 24.
    23.2. If *credentialState*[0].Reason does not equal to "Test Reason" or missed, FAIL the test and go step 24.
    23.3. If *cap*.ResetAntipassbackSupported value is equal to true check the following:
        23.3.1. If *credentialState*[0] does not contain AntipassbackState element, FAIL the test and go step 24.
        23.3.2. If *credentialState*[0].AntipassbackState.AntipassbackViolated equal to true, FAIL the test and go step 24.
24. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.
25. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.
26. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
27. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialsResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **CreateCredentialResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

Note: The following fields are compared at steps 15, 21:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)
    - Type
      - Name
      - FormatType
    - ExemptedFromAuthentication
    - Value
  - CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
    - AccessProfileToken
    - ValidFrom

    - ValidTo
  - Attribute list
    - Name
    - Value

Note: The following fields are compared at steps 17, 19:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.3.6 CREATE CREDENTIAL (DISABLED)

Test Case ID: CREDENTIAL-3-1-6

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), CreateCredential command (ONVIF Credential Service Specification)

Feature Under Test: CreateCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify creation of disabled credential and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList1*) by following the procedure mentioned in Annex A.3.
5. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
6. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
7. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.
8. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
9. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
10. ONVIF client invokes **CreateCredential** with parameters
    - Credential.token := ""
    - Credential.Description := "Test Description"
    - Credential.CredentialHolderReference := "TestUser"
    - Credential.ValidFrom skipped
    - Credential.ValidTo skipped
    - Credential.CredentialIdentifier[0].Type.Name := *typeName*
    - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*

- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
- Credential.CredentialIdentifier[0].Value := *value*
- Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token or skipped (if *accessProfileCompleteList* is empty)
- Credential.CredentialAccessProfile.ValidFrom skipped
- Credential.CredentialAccessProfile.ValidTo skipped
- Credential.Extension skipped
- State.Enabled := false
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

11. The DUT responds with **CreateCredentialResponse** message with parameters
    - Token =: *credentialToken*
12. Until *oprationDelay* timeout expires, repeat the following steps:
    12.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    12.2. ONVIF Client invokes **PullMessages** to the subscription endpoint s with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    12.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    12.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 24.

12.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 24.

12.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 14.

13. If *oprationDelay* timeout expires for step 12 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 24.

14. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.

15. If *credentialList*[0] item does not have equal field values to values from step 10, FAIL the test and go step 24.

16. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.

17. If *credentialInfoList*[0] item does not have equal field values to values from step 10, FAIL the test and go step 24.

18. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

19. If *credentialInfoCompleteList* does not have *credentialInfo*[token = *credentialToken*] item with equal field values to values from step 10, FAIL the test and go step 24.

20. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList2*) by following the procedure mentioned in Annex A.3.

21. If *credentialCompleteList2* does not have *credential*.[token = *credentialToken*] item with equal field values to values from step 10, FAIL the test and go step 24.

22. ONVIF client retrieves credential state (in *credentialToken*, out *credentialState*) by following the procedure mentioned in Annex A.13.

23. Check the following:

23.1. If *credentialState*[0].Enabled equal to true, FAIL the test and go step 24.

23.2. If *credentialState*[0].Reason does not equal to "Test Reason" or missed, FAIL the test and go step 24.

23.3. If *cap*.ResetAntipassbackSupported value is equal to true check the following:

23.3.1. If *credentialState*[0] does not contain AntipassbackState element, FAIL the test and go step 24.

23.3.2. If *credentialState*[0].AntipassbackState.AntipassbackViolated equal to true, FAIL the test and go step 24.

24. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

25. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

26. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialsResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **CreateCredentialResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

Note: The following fields are compared at steps 15, 21:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)

- Type
  - Name
  - FormatType
- ExemptedFromAuthentication
- Value

- CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
  - AccessProfileToken
  - ValidFrom
  - ValidTo
- Attribute list
  - Name
  - Value

Note: The following fields are compared at steps 17, 19:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.3.7 MODIFY CREDENTIAL

Test Case ID: CREDENTIAL-3-1-7

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), ModifyCredential command (ONVIF Credential Service Specification)

Feature Under Test: ModifyCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify modifying of credential with different states and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList1*) by following the procedure mentioned in Annex A.3.
4. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
5. ONVIF Client retrieves a complete list of access profiles (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
6. ONVIF Client creates credential with ExemptedFromAuthentication equal to false (in false), with Credential identifier item (out *typeName*) with corresponding Credential identifier Format Type (out *formatType*) and corresponding credential identifier value (out *value*) and with credential token (out *credentialToken*) by following the procedure mentioned in Annex A.11.
7. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
8. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
9. ONVIF client invokes **ModifyCredential** with parameters

- Credential.Token := *credentialToken*
- Credential.Description := "Test Description 2"
- Credential.CredentialHolderReference := "TestUser 2"
- Credential.ValidFrom skipped
- Credential.ValidTo skipped
- Credential.CredentialIdentifier[0].Type.Name := *typeName*
- Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
- Credential.CredentialIdentifier[0].Value := *value*
- Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token or skipped (if *accessProfileCompleteList* is empty)
- Credential.CredentialAccessProfile.ValidFrom skipped
- Credential.CredentialAccessProfile.ValidTo skipped
- Credential.Extension skipped

10. The DUT responds with empty **ModifyCredentialResponse** message.
11. Until *oprationDelay* timeout expires, repeat the following steps:
    11.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    11.2. ONVIF Client invokes **PullMessages** to the subscription endpoint s with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    11.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*

11.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 21.

11.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 1.

11.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 13.

12. If *oprationDelay* timeout expires for step 11 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 21.

13. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.

14. If *credentialList*[0] item does not have equal field values to values from step 9, FAIL the test and go step 21.

15. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.

16. If *credentialInfoList*[0] item does not have equal field values to values from step 9, FAIL the test and go step 21.

17. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

18. If *credentialInfoCompleteList* does not have *credentialInfo*.[token:= *credentialToken*] item with equal field values to values from step 9, FAIL the test and go step 21.

19. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList2*) by following the procedure mentioned in Annex A.3.

20. If *credentialCompleteList2* does not have *credential*.[token:= *credentialToken*] item with equal field values to values from step 9, FAIL the test and go step 22.

21. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

22. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

23. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialsResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **ModifyCredentialResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

Note: The following fields are compared at steps 18, 20:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)
    - Type
      - Name
      - FormatType
    - ExemptedFromAuthentication
    - Value
  - CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
    - AccessProfileToken
    - ValidFrom

    - ValidTo
  - Attribute list
    - Name
    - Value

Note: The following fields are compared at steps 14, 16:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.3.8 DELETE CREDENTIAL

Test Case ID: CREDENTIAL-3-1-8

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), DeleteCredential command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredential

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify deleting of credential and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Credential Entity is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

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

3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList1*) by following the procedure mentioned in Annex A.3.
4. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
5. ONVIF Client creates credential (in false, out *credentialToken*) by following the procedure mentioned in Annex A.11.
6. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Configuration/Credential/Removed"
7. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
8. ONVIF Client invokes **DeleteCredential** with parameters
   - Token := *credentialToken*
9. The DUT responds with empty **DeleteCredentialResponse** message.
10. Until *oprationDelay* timeout expires, repeat the following steps:
    10.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    10.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
      - Timeout := PT60S
      - MessageLimit := 1
    10.3. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*
      - NotificationMessage =: *m*
    10.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Removed", FAIL the test and go to the step 21.

10.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 20.

10.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 12.

11. If *oprationDelay* timeout expires for step 10 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 20.
12. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.
13. If *credentialList* is not empty, FAIL the test and go step 20.
14. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.
15. If *credentialInfoList* is not empty, FAIL the test and go step 20.
16. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
17. If *credentialInfoCompleteList* contains *credentialInfo*.[token:= *credentialToken*] item, FAIL the test and go step 20.
18. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList2*) by following the procedure mentioned in Annex A.3.
19. If *credentialCompleteList2* contains *credential*.[token:= *credentialToken*] item, FAIL the test and go step 20.
20. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.
21. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
22. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **DeleteCredentialResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

4.3.9 GET CREDENTIALS WITH INVALID TOKEN

Test Case ID: CREDENTIAL-3-1-9

Specification Coverage: Credential (ONVIF Credential Service Specification), GetCredentials command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentials

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF client invokes **GetCredentials** with parameters
   - Token[0] := *invalidToken*
6. The DUT responds with **GetCredentialsResponse** message with parameters
   - Credential list =: *credentialsList*
7. If *credentialsList* is not empty, FAIL the test.

8. If *credentialInfoCompleteList* is empty, skip other steps.
9. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
10. If *cap*.MaxLimit is less than 2, skip other steps.
11. ONVIF client invokes **GetCredentials** with parameters
    - Token[0] := *invalidToken*
    - Token[1] := *credentialInfoCompleteList*[0].token
12. The DUT responds with **GetCredentialsResponse** message with parameters
    - CredentialInfo list =: *credentialsList*
13. If *credentialsList* is empty, FAIL the test.
14. If *credentialsList* contains more than one item, FAIL the test.
15. If *credentialsList*[0].token does not equal to *credentialInfoCompleteList*[0].token, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.3.10 GET CREDENTIALS - TOO MANY ITEMS

Test Case ID: CREDENTIAL-3-1-10

Specification Coverage: Credential (ONVIF Credential Service Specification), GetCredentials command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentials

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential in case if there a more items than MaxLimit in request.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Test Configuration: ONVIF Client and DUT

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
4. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
5. If *credentialInfoCompleteList*.token items number is less than *cap*.MaxLimit or equal to *cap*.MaxLimit, skip other steps.
6. Set the following:
   - *tokenList* := [subset of *credentialCompleteList*.token values with items number equal to *cap*.MaxLimit + 1]
7. ONVIF client invokes **GetCredentials** with parameters
   - Token list := *tokenList*
8. The DUT returns **env:Sender\ter:InvalidArgs\ter:TooManyItems** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgs\ter:TooManyItems** SOAP 1.2 fault

4.3.11 CREATE CREDENTIAL - NOT EMPTY CREDENTIAL TOKEN

Test Case ID: CREDENTIAL-3-1-11

Specification Coverage: CreateCredential command (ONVIF Credential Service Specification)

Feature Under Test: CreateCredential

WSDL Reference: credential.wsdl

Test Purpose: To verify Create Credential with not Empty Token Verification.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.
5. ONVIF client invokes **CreateCredential** with parameters
   - Credential.token := "CredentialToken"
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped
   - Credential.CredentialIdentifier[0].Type.Name := *typeName*
   - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
   - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
   - Credential.CredentialIdentifier[0].Value := *value*
   - Credential.CredentialAccessProfile.AccessProfileToken skipped
   - State.Enabled := true
   - State.Reason := "Test Reason"

- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

6. The DUT returns **env:Sender\ter:InvalidArgVal** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal** SOAP 1.2 fault.

4.3.12 MODIFY CREDENTIAL WITH INVALID TOKEN

Test Case ID: CREDENTIAL-3-1-12

Specification Coverage: ModifyCredential command (ONVIF Credential Service Specification)

Feature Under Test: ModifyCredential

WSDL Reference: credential.wsdl

Test Purpose: To verify modifying of credential with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
5. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier

Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.

6. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
7. ONVIF client invokes **ModifyCredential** with parameters
   - Credential.Token := *invalidToken*
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped
   - Credential.CredentialIdentifier[0].Type.Name := *typeName*
   - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
   - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
   - Credential.CredentialIdentifier[0].Value := *value*
   - Credential.CredentialAccessProfile.AccessProfileToken skipped
   - Credential.Extension skipped
8. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.3.13 DELETE CREDENTIAL WITH INVALID TOKEN

Test Case ID: CREDENTIAL-3-1-13

Specification Coverage: DeleteCredential command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredential

WSDL Reference: credential.wsdl

Test Purpose: To verify deleting of credential with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF Client invokes **DeleteCredential** with parameters
   - Token := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.3.14 CREATE CREDENTIAL - VALIDITY VALUES

Test Case ID: CREDENTIAL-3-1-14

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), CreateCredential command (ONVIF Credential Service Specification)

Feature Under Test: CreateCredential

WSDL Reference: credential.wsdl and accessrules.wsdl

Test Purpose: To verify creation of credential with credential validity and with credential access profile validity.

Pre-Requisite: Credential Service is received from the DUT. Access Rules Service is received from the DUT. Credential Entity is supported by the DUT. CredentialValiditySupported is supported by the DUT as indicated by the Capabilities.CredentialValiditySupported capability or CredentialAccessProfileValiditySupported is supported by the DUT as indicated by the Capabilities.CredentialAccessProfileValiditySupported capability. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
5. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.
6. Set the following:
   - *credentialValidFrom*:= value of current time
   - *credentialValidTo*:= *credentialValidFrom* + one year
   - *accessProfileValidFrom* := value of current time + 24 h
   - *accessProfileValidTo* := *accessProfileValidFrom* + one year
7. ONVIF client invokes **CreateCredential** with parameters

- Credential.token := ""
- Credential.Description := "Test Description"
- Credential.CredentialHolderReference := "TestUser"
- Credential.ValidFrom := *credentialValidFrom* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
- Credential.ValidTo := *credentialValidTo* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
- Credential.CredentialIdentifier[0].Type.Name := *typeName*
- Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
- Credential.CredentialIdentifier[0].Value := *value*
- Credential.CredentialAccessProfile[0] is skipped if *accessProfileCompleteList* is empty or if *cap*.CredentialAccessProfileValiditySupported is equal to false
- Credential.CredentialAccessProfile[0].AccessProfileToken := *accessProfileCompleteList*[0].token
- Credential.CredentialAccessProfile[0].ValidFrom := *accessProfileValidFrom*
- Credential.CredentialAccessProfile[0].ValidTo := *accessProfileValidTo*
- State.Enabled := false
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

8. The DUT responds with **CreateCredentialResponse** message with parameters
   - Token =: *credentialToken*
9. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.
10. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.

11. If *cap*.ValiditySupportsTimeValue is equal to true, check the following:

    11.1. If *credentialList*[0].Credential.ValidFrom value does not equal to *credentialValidFrom*, FAIL the test and go step 13.

    11.2. If *credentialList*[0].Credential.ValidTo value does not equal to *credentialValidTo*, FAIL the test and go step 13.

    11.3. If *credentialList*[0].CredentialAccessProfile[0].ValidFrom value does not equal to *accessProfileValidFrom*, FAIL the test and go step 13.

    11.4. If *credentialList*[0].CredentialAccessProfile[0].ValidTo value does not equal to *accessProfileValidTo*, FAIL the test and go step 13.

    11.5. If *credentialInfoList*[0].CredentialInfo.ValidFrom value does not equal to *credentialValidFrom*, FAIL the test and go step 13.

    11.6. If *credentialInfoList*[0].CredentialInfo.ValidTo value does not equal to *credentialValidTo*, FAIL the test and go step 13.

12. If *cap*.ValiditySupportsTimeValue is equal to false, check the following:

    12.1. If *credentialList*[0].Credential.ValidFrom value contains data component that does not equal to data component of *credentialValidFrom*, FAIL the test and go step 13.

    12.2. If *credentialList*[0].Credential.ValidTo value contains data component that does not equal to data component of *credentialValidTo*, FAIL the test and go step 13.

    12.3. If *credentialList*[0].CredentialAccessProfile[0].ValidFrom value contains data component that does not equal to data component of *accessProfileValidFrom*, FAIL the test and go step 13.

    12.4. If *credentialList*[0].CredentialAccessProfile[0].ValidTo value contains data component that does not equal to data component of *accessProfileValidTo*, FAIL the test and go step 13.

    12.5. If *credentialInfoList*[0].CredentialInfo.ValidFrom value contains data component that does not equal to data component of *credentialValidFrom*, FAIL the test and go step 13.

    12.6. If *credentialInfoList*[0].CredentialInfo.ValidTo value contains data component that does not equal to data component of *credentialValidTo*, FAIL the test and go step 13.

13. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The ONVIF Client sets and compares values of Credential.ValidFrom, Credential.ValidTo, CredentialAccessProfile.ValidFrom, and CredentialAccessProfile.ValidTo accurate to a second.

4.3.15 MODIFY CREDENTIAL - VALIDITY VALUES

Test Case ID: CREDENTIAL-3-1-15

Specification Coverage: ModifyCredential command (ONVIF Credential Service Specification)

Feature Under Test: ModifyCredential

WSDL Reference: credential.wsdl and accessrules.wsdl

Test Purpose: To verify creation of credential with credential validity and with credential access profile validity.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. CredentialValiditySupported is supported by the DUT as indicated by the Capabilities.CredentialValiditySupported capability or CredentialAccessProfileValiditySupported is supported by the DUT as indicated by the Capabilities.CredentialAccessProfileValiditySupported capability. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
5. ONVIF Client retrieves supported Credential identifier type name (in *cap*.SupportedIdentifierType) (out *typeName*) with corresponding Credential identifier

Format Type (out *formatType*) and credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.

6. Set the following:
   - *credentialValidFrom1*:= value of current time
   - *credentialValidTo1*:= *credentialValidFrom1* + one year
   - *accessProfileValidFrom1* := value of current time + 24 h
   - *accessProfileValidTo1* := *accessProfileValidFrom1* + one year
7. ONVIF client invokes **CreateCredential** with parameters
   - Credential.token := ""
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom := *credentialValidFrom1* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
   - Credential.ValidTo := *credentialValidTo1* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
   - Credential.CredentialIdentifier[0].Type.Name := *typeName*
   - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
   - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
   - Credential.CredentialIdentifier[0].Value := *value*
   - Credential.CredentialAccessProfile[0] is skipped if *accessProfileCompleteList* is empty or if *cap*.CredentialAccessProfileValiditySupported is equal to false
   - Credential.CredentialAccessProfile[0].AccessProfileToken := *accessProfileCompleteList*[0].token
   - Credential.CredentialAccessProfile[0].ValidFrom := *accessProfileValidFrom1*
   - Credential.CredentialAccessProfile[0].ValidTo := *accessProfileValidTo1*
   - State.Enabled := false
   - State.Reason := "Test Reason"

- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

8. The DUT responds with **CreateCredentialResponse** message with parameters
   - Token =: *credentialToken*
9. Set the following:
   - *credentialValidFrom2*:= *credentialValidFrom2* + one day + one hour
   - *credentialValidTo2*:= *credentialValidTo2* + one day + one hour
   - *accessProfileValidFrom2* := *accessProfileValidFrom1* + one day + one hour
   - *accessProfileValidTo2* := *accessProfileValidFrom1* + one day + one hour
10. ONVIF client invokes **ModifyCredential** with parameters
    - Credential.Token := *credentialToken*
    - Credential.Description := "Test Description"
    - Credential.CredentialHolderReference := "TestUser"
    - Credential.ValidFrom := *credentialValidFrom2* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
    - Credential.ValidTo := *credentialValidTo2* if *cap*.CredentialValiditySupported value is equal to true or skipped if *cap*.CredentialValiditySupported value is equal to false
    - Credential.CredentialIdentifier[0].Type.Name := *typeName*
    - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
    - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
    - Credential.CredentialIdentifier[0].Value := *value*
    - Credential.CredentialAccessProfile[0] is skipped if accessProfileCompleteList is empty or if *cap*.CredentialAccessProfileValiditySupported is equal to false
    - Credential.CredentialAccessProfile[0].AccessProfileToken := *accessProfileCompleteList*[0].token
    - Credential.CredentialAccessProfile[0].ValidFrom := *accessProfileValidFrom2*

- Credential.CredentialAccessProfile[0].ValidTo := *accessProfileValidTo2*

11. The DUT responds with empty **ModifyCredentialResponse** message.
12. ONVIF Client retrieves a credential (in *credentialToken*, out *credentialList*) by following the procedure mentioned in Annex A.8.
13. ONVIF Client retrieves a credential info (in *credentialToken*, out *credentialInfoList*) by following the procedure mentioned in Annex A.9.
14. If *cap*.ValiditySupportsTimeValue is equal to true, check the following:
    14.1. If *credentialList*[0].Credential.ValidFrom value does not equal to *credentialValidFrom2*, FAIL the test and go step 16.
    14.2. If *credentialList*[0].Credential.ValidTo value does not equal to *credentialValidTo2*, FAIL the test and go step 16.
    14.3. If *credentialList*[0].CredentialAccessProfile[0].ValidFrom value does not equal to *accessProfileValidFrom2*, FAIL the test and go step 16.
    14.4. If *credentialList*[0].CredentialAccessProfile[0].ValidTo value does not equal to *accessProfileValidTo2*, FAIL the test and go step 16.
    14.5. If *credentialInfoList*[0].CredentialInfo.ValidFrom value does not equal to *credentialValidFrom2*, FAIL the test and go step 16.
    14.6. If *credentialInfoList*[0].CredentialInfo.ValidTo value does not equal to *credentialValidTo2*, FAIL the test and go step 16.
15. If *cap*.ValiditySupportsTimeValue is equal to false, check the following:
    15.1. If *credentialList*[0].Credential.ValidFrom value contains data component that does not equal to data component of *credentialValidFrom2*, FAIL the test and go step 16.
    15.2. If *credentialList*[0].Credential.ValidTo value contains data component that does not equal to data component of *credentialValidTo2*, FAIL the test and go step 16.
    15.3. If *credentialList*[0].CredentialAccessProfile[0].ValidFrom value contains data component that does not equal to data component of *accessProfileValidFrom2*, FAIL the test and go step 16.
    15.4. If *credentialList*[0].CredentialAccessProfile[0].ValidTo value contains data component that does not equal to data component of *accessProfileValidTo2*, FAIL the test and go step 16.

15.5. If *credentialInfoList*[0].CredentialInfo.ValidFrom value contains data component that does not equal to data component of *credentialValidFrom2*, FAIL the test and go step 16.

15.6. If *credentialInfoList*[0].CredentialInfo.ValidTo value contains data component that does not equal to data component of *credentialValidTo2*, FAIL the test and go step 16.

16. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **CreateCredentialResponse** message.
- The DUT did not send **ModifyCredentialResponse** message.

Note: The ONVIF Client sets and compares values of Credential.ValidFrom, Credential.ValidTo, CredentialAccessProfile.ValidFrom, and CredentialAccessProfile.ValidTo accurate to a second.

4.3.16 SET NEW CREDENTIAL (ENABLED)

Test Case ID: CREDENTIAL-3-1-16

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), SetCredential command (ONVIF Credential Service Specification)

Feature Under Test: SetCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify creation of enabled credential using SetCredential command and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Access Rules Service is received from the DUT. Client Supplied Token is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Credential Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
4. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *initialCredentialCompleteList* - credential complete list
5. ONVIF Client checks free storage for additional credential by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - in *initialCredentialCompleteList* - credential complete list
   - out *credentialToRestore* - removed credential
   - out *stateToRestore* - state of removed credential
6. ONVIF Client retrieves complete list of access profiles by following the procedure mentioned in Annex A.5 with the following input and output parameters
   - out *accessProfileCompleteList* - access profiles complete list
7. ONVIF Client retrieves supported credential identifier type name with corresponding credential identifier format type and credential identifier value by following the procedure mentioned in Annex A.15 with the following input and output parameters
   - in *cap*.SupportedIdentifierType - list of supported identifier types
   - out *typeName* - selected identifier type name
   - out *formatType* - selected identifier format type
   - out *value* - credential identifier value for selected identifier format type
8. Set *credentialToken* := token that differs from tokens listed in *initialCredentialCompleteList*.
9. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.21 with the following input and output parameters

- in **tns1:Configuration/Credential/Changed** - Notification Topic
- out *s* - Subscription reference
- out *currentTime* - current time for the DUT
- out *terminationTime* - Subscription termination time

10. Set *credential* :=
    - Credential.token := *credentialToken*
    - Credential.Description := "Test Description"
    - Credential.CredentialHolderReference := "TestUser"
    - Credential.ValidFrom skipped
    - Credential.ValidTo skipped
    - Credential.CredentialIdentifier[0].Type.Name := *typeName*
    - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
    - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
    - Credential.CredentialIdentifier[0].Value := *value*
    - If *accessProfileCompleteList* contains at least one item:
        - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token
        - Credential.CredentialAccessProfile.ValidFrom skipped
        - Credential.CredentialAccessProfile.ValidTo skipped

      otherwise:
        - Credential.CredentialAccessProfile is skipped
    - Credential.Extension skipped
11. ONVIF client invokes **SetCredential** request with parameters
    - Credential := *credential*

- State.Enabled := true
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

12. The DUT responds with **SetCredentialResponse** message.
13. ONVIF Client retrieves and checks **tns1:Configuration/Credential/Changed** event for the specified Credential token by following the procedure mentioned in Annex A.23 with the following input and output parameters
    - in *s* - Subscription reference
    - in *currentTime* - current time for the DUT
    - in *terminationTime* - subscription termination time
    - in *credentialToken* - Credential token
14. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.22 with the following input and output parameters
    - in *s* - Subscription reference
15. ONVIF Client retrieves a credential by following the procedure mentioned in Annex A.8 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialList* - the list of credentials
16. If *credentialList*[0] item does not have equal to *credential*, FAIL the test, restore the DUT state, and skip other steps.
17. ONVIF Client retrieves a credential info by following the procedure mentioned in Annex A.9 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialInfoList* - the list of credentials info
18. If *credentialInfoList*[0] item does not have equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.

19. ONVIF Client retrieves complete list of credentials info by following the procedure mentioned in Annex A.1 with the following input and output parameters
    - out *credentialInfoCompleteList* - credential info complete list
20. If *credentialInfoCompleteList* does not have CredentialInfo[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
21. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
    - out *credentialCompleteList* - credential complete list
22. If *credentialCompleteList* does not have Credential[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
23. ONVIF Client retrieves credential state by following the procedure mentioned in Annex A.13 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialState* - credential state
24. If *credentialState*[0].Enabled equal to false, FAIL the test, restore the DUT state, and skip other steps.
25. If *credentialState*[0].Reason does not equal to "Test Reason" or missed, FAIL the test, restore the DUT state, and skip other steps.
26. If *cap*.ResetAntipassbackSupported value is equal to true check the following:
    26.1. If *credentialState*[0] does not contain AntipassbackState element, FAIL the test, restore the DUT state, and skip other steps.
    26.2. If *credentialState*[0].AntipassbackState.AntipassbackViolated equal to true, FAIL the test, restore the DUT state, and skip other steps.
27. ONVIF Client deletes the Credential by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *credentialToken* - credential token
28. If there was credential deleted at step 5:
    28.1. ONVIF Client restores credential deletes special day group by following the procedure mentioned in Annex A.10 with the following input and output parameters
        - in *credentialToRestore* - removed credential

- in *stateToRestore* - state of removed credential

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at steps 16 and 22:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)
    - Type
      - Name
      - FormatType
    - ExemptedFromAuthentication
    - Value
  - CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
    - AccessProfileToken
    - ValidFrom
    - ValidTo
  - Attribute list
    - Name

- Value

Note: The following fields are compared at steps 18 and 20:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.3.17 SET NEW CREDENTIAL (DISABLED)

Test Case ID: CREDENTIAL-3-1-17

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), SetCredential command (ONVIF Credential Service Specification)

Feature Under Test: SetCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify creation of disabled credential using SetCredential command and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Access Rules Service is received from the DUT. Client Supplied Token is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Credential Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters

- out *cap* - Credential Service capabilities

4. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *initialCredentialCompleteList* - credential complete list
5. ONVIF Client checks free storage for additional credential by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - in *initialCredentialCompleteList* - credential complete list
   - out *credentialToRestore* - removed credential
   - out *stateToRestore* - state of removed credential
6. ONVIF Client retrieves complete list of access profiles by following the procedure mentioned in Annex A.5 with the following input and output parameters
   - out *accessProfileCompleteList* - access profiles complete list
7. ONVIF Client retrieves supported credential identifier type name with corresponding credential identifier format type and credential identifier value by following the procedure mentioned in Annex A.15 with the following input and output parameters
   - in *cap*.SupportedIdentifierType - list of supported identifier types
   - out *typeName* - selected identifier type name
   - out *formatType* - selected identifier format type
   - out *value* - credential identifier value for selected identifier format type
8. Set *credentialToken* := token that differs from tokens listed in *initialCredentialCompleteList*.
9. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.21 with the following input and output parameters
   - in **tns1:Configuration/Credential/Changed** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
10. Set *credential* :=

- Credential.token := *credentialToken*
- Credential.Description := "Test Description"
- Credential.CredentialHolderReference := "TestUser"
- Credential.ValidFrom skipped
- Credential.ValidTo skipped
- Credential.CredentialIdentifier[0].Type.Name := *typeName*
- Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
- Credential.CredentialIdentifier[0].Value := *value*
- If *accessProfileCompleteList* contains at least one item:
  - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token
  - Credential.CredentialAccessProfile.ValidFrom skipped
  - Credential.CredentialAccessProfile.ValidTo skipped

  otherwise:

  - Credential.CredentialAccessProfile is skipped
- Credential.Extension skipped

11. ONVIF client invokes **SetCredential** request with parameters
    - Credential := *credential*
    - State.Enabled := false
    - State.Reason := "Test Reason"
    - State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

12. The DUT responds with **SetCredentialResponse** message.
13. ONVIF Client retrieves and checks **tns1:Configuration/Credential/Changed** event for the specified Credential token by following the procedure mentioned in Annex A.23 with the following input and output parameters
    - in *s* - Subscription reference
    - in *currentTime* - current time for the DUT
    - in *terminationTime* - subscription termination time
    - in *credentialToken* - Credential token
14. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.22 with the following input and output parameters
    - in *s* - Subscription reference
15. ONVIF Client retrieves a credential by following the procedure mentioned in Annex A.8 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialList* - the list of credentials
16. If *credentialList*[0] item does not have equal to *credential*, FAIL the test, restore the DUT state, and skip other steps.
17. ONVIF Client retrieves a credential info by following the procedure mentioned in Annex A.9 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialInfoList* - the list of credentials info
18. If *credentialInfoList*[0] item does not have equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
19. ONVIF Client retrieves complete list of credentials info by following the procedure mentioned in Annex A.1 with the following input and output parameters
    - out *credentialInfoCompleteList* - credential info complete list
20. If *credentialInfoCompleteList* does not have CredentialInfo[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.

21. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
    - out *credentialCompleteList* - credential complete list
22. If *credentialCompleteList* does not have Credential[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
23. ONVIF Client retrieves credential state by following the procedure mentioned in Annex A.13 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialState* - credential state
24. If *credentialState*[0].Enabled equal to true, FAIL the test, restore the DUT state, and skip other steps.
25. If *credentialState*[0].Reason does not equal to "Test Reason" or missed, FAIL the test, restore the DUT state, and skip other steps.
26. If *cap*.ResetAntipassbackSupported value is equal to true check the following:
    26.1. If *credentialState*[0] does not contain AntipassbackState element, FAIL the test, restore the DUT state, and skip other steps.
    26.2. If *credentialState*[0].AntipassbackState.AntipassbackViolated equal to true, FAIL the test, restore the DUT state, and skip other steps.
27. ONVIF Client deletes the Credential by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *credentialToken* - credential token
28. If there was credential deleted at step 5:
    28.1. ONVIF Client restores credential deletes special day group by following the procedure mentioned in Annex A.10 with the following input and output parameters
        - in *credentialToRestore* - removed credential
        - in *stateToRestore* - state of removed credential

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at steps 16 and 22:

- Credential:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo
  - CredentialIdentifier list (Type.Name is used as unique key for comparing)
    - Type
      - Name
      - FormatType
    - ExemptedFromAuthentication
    - Value
  - CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
    - AccessProfileToken
    - ValidFrom
    - ValidTo
  - Attribute list
    - Name
    - Value

Note: The following fields are compared at steps 18 and 20:

- CredentialInfo:
  - token

- Description
- CredentialHolderToken
- ValidFrom
- ValidTo

4.3.18 SET CREDENTIAL

Test Case ID: CREDENTIAL-3-1-18

Specification Coverage: CredentialInfo (ONVIF Credential Service Specification), Credential (ONVIF Credential Service Specification), SetCredential command (ONVIF Credential Service Specification)

Feature Under Test: ModifyCredential

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify modifying of credential with different states using SetCredential command and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Access Rules Service is received from the DUT. Client Supplied Token is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves Credential Service Capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
4. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
   - out *initialCredentialCompleteList* - credential complete list

5. ONVIF Client checks free storage for additional credential by following the procedure mentioned in Annex A.7 with the following input and output parameters
   - in *initialCredentialCompleteList* - credential complete list
   - out *credentialToRestore* - removed credential
   - out *stateToRestore* - state of removed credential
6. ONVIF Client retrieves complete list of access profiles by following the procedure mentioned in Annex A.5 with the following input and output parameters
   - out *accessProfileCompleteList* - access profiles complete list
7. ONVIF Client creates credential by following the procedure mentioned in Annex A.11 with the following input and output parameters
   - in false - ExemptedFromAuthentication value
   - out *typeName* - selected identifier type name
   - out *formatType* - selected identifier format type
   - out *value* - credential identifier value for selected identifier format type
   - out *credentialToken* - Credential token
8. ONVIF Client creates PullPoint subscription for the specified topic by following the procedure mentioned in Annex A.21 with the following input and output parameters
   - in **tns1:Configuration/Credential/Changed** - Notification Topic
   - out *s* - Subscription reference
   - out *currentTime* - current time for the DUT
   - out *terminationTime* - Subscription termination time
9. Set *credential* :=
   - Credential.token := *credentialToken*
   - Credential.Description := "Test Description2"
   - Credential.CredentialHolderReference := "TestUser2"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped

- Credential.CredentialIdentifier[0].Type.Name := *typeName*
- Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
- Credential.CredentialIdentifier[0].Value := *value*
- If *accessProfileCompleteList* contains at least one item:
  - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token
  - Credential.CredentialAccessProfile.ValidFrom skipped
  - Credential.CredentialAccessProfile.ValidTo skipped

  otherwise:
  - Credential.CredentialAccessProfile is skipped
- Credential.Extension skipped

10. ONVIF client invokes **SetCredential** request with parameters
    - Credential := *credential*
    - State.Enabled := false
    - State.Reason := "Test Reason2"
    - State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped
11. The DUT responds with **SetCredentialResponse** message.
12. ONVIF Client retrieves and checks **tns1:Configuration/Credential/Changed** event for the specified Credential token by following the procedure mentioned in Annex A.23 with the following input and output parameters
    - in *s* - Subscription reference
    - in *currentTime* - current time for the DUT

- in *terminationTime* - subscription termination time
- in *credentialToken* - Credential token

13. ONVIF Client deletes PullPoint subscription by following the procedure mentioned in Annex A.22 with the following input and output parameters
    - in *s* - Subscription reference
14. ONVIF Client retrieves a credential by following the procedure mentioned in Annex A.8 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialList* - the list of credentials
15. If *credentialList*[0] item does not have equal to *credential*, FAIL the test, restore the DUT state, and skip other steps.
16. ONVIF Client retrieves a credential info by following the procedure mentioned in Annex A.9 with the following input and output parameters
    - in *credentialToken* - credential token
    - out *credentialInfoList* - the list of credentials info
17. If *credentialInfoList*[0] item does not have equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
18. ONVIF Client retrieves complete list of credentials info by following the procedure mentioned in Annex A.1 with the following input and output parameters
    - out *credentialInfoCompleteList* - credential info complete list
19. If *credentialInfoCompleteList* does not have CredentialInfo[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
20. ONVIF Client retrieves complete list of credentials by following the procedure mentioned in Annex A.3 with the following input and output parameters
    - out *credentialCompleteList* - credential complete list
21. If *credentialCompleteList* does not have Credential[token = *credentialToken*] item with equal fields with *credential*, FAIL the test, restore the DUT state, and skip other steps.
22. ONVIF Client retrieves credential state by following the procedure mentioned in Annex A.13 with the following input and output parameters

- in *credentialToken* - credential token
- out *credentialState* - credential state

23. If *credentialState*[0].Enabled equal to true, FAIL the test, restore the DUT state, and skip other steps.
24. If *credentialState*[0].Reason does not equal to "Test Reason2" or missed, FAIL the test, restore the DUT state, and skip other steps.
25. If *cap*.ResetAntipassbackSupported value is equal to true check the following:
    25.1. If *credentialState*[0] does not contain AntipassbackState element, FAIL the test, restore the DUT state, and skip other steps.
    25.2. If *credentialState*[0].AntipassbackState.AntipassbackViolated equal to true, FAIL the test, restore the DUT state, and skip other steps.
26. ONVIF Client deletes the Credential by following the procedure mentioned in Annex A.6 with the following input and output parameters
    - in *credentialToken* - credential token
27. If there was credential deleted at step 5:
    27.1. ONVIF Client restores credential deletes special day group by following the procedure mentioned in Annex A.10 with the following input and output parameters
        - in *credentialToRestore* - removed credential
        - in *stateToRestore* - state of removed credential

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: The following fields are compared at steps 15 and 21:

- Credential:
  - token

- Description
- CredentialHolderToken
- ValidFrom
- ValidTo
- CredentialIdentifier list (Type.Name is used as unique key for comparing)
  - Type
    - Name
    - FormatType
  - ExemptedFromAuthentication
  - Value
- CredentialAccessProfile list (AccessProfileToken is used as unique key for comparing)
  - AccessProfileToken
  - ValidFrom
  - ValidTo
- Attribute list
  - Name
  - Value

Note: The following fields are compared at steps 17 and 19:

- CredentialInfo:
  - token
  - Description
  - CredentialHolderToken
  - ValidFrom
  - ValidTo

4.4 Credential State

4.4.1 GET CREDENTIAL STATE

Test Case ID: CREDENTIAL-4-1-1

Specification Coverage: CredentialState (ONVIF Credential Service Specification), GetCredentialState command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialState

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential State.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
5. If *credentialInfoCompleteList* is empty, skip other steps.
6. For each CredentialInfo.token *token* from *credentialInfoCompleteList* repeat the following steps:
   6.1. ONVIF client invokes **GetCredentialState** with parameters
      - Token := *token*
   6.2. The DUT responds with **GetCredentialStateResponse** message with parameters
      - State =: *credentialState*
   6.3. If *cap*.ResetAntipassbackSupported is equal to true and *credentialState* does not contain AntipassbackState element, FAIL the test.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.4.2 CHANGE CREDENTIAL STATE

Test Case ID: CREDENTIAL-4-1-2

Specification Coverage: CredentialState (ONVIF Credential Service Specification), GetCredentialState command (ONVIF Credential Service Specification), EnableCredential command (ONVIF Credential Service Specification), DisableCredential command (ONVIF Credential Service Specification).

Feature Under Test: GetCredentialState, EnableCredential, DisableCredential.

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify enabling and disabling of Credential and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList1*) by following the procedure mentioned in Annex A.3.
4. ONVIF Client checks free storage (in *credentialCompleteList1*) for additional Credential and removes one if needed (out *credentialToRestore*, out *stateToRestore*) by following the procedure mentioned in Annex A.7.
5. ONVIF Client creates credential (out *credentialToken*) with antipass back state equal to false (in false) by following the procedure mentioned in Annex A.11.

6. ONVIF client invokes **GetCredentialState** with parameters
   - Token := *credentialToken*
7. The DUT responds with **GetCredentialStateResponse** message with parameters
   - State =: *credentialState1*
8. If *credentialState1*.Enabled is not equal to true, FAIL the test and go to step 23.
9. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Credential/State/Enabled"
10. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
11. ONVIF Client changes credential state (in *credentialState1*) for created credential (in *credentialToken*) by following the procedure mentioned in Annex A.4.
12. Until *oprationDelay* timeout expires, repeat the following steps:
    12.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    12.2. ONVIF Client invokes **PullMessages** to the subscription endpoint s with parameters
    - Timeout := PT60S
    - MessageLimit := 1

    12.3. The DUT responds with **PullMessagesResponse** message with parameters
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
    - NotificationMessage =: *m*

    12.4. If *m* is not null:
    12.4.1. If TopicExpression item in *m* is not equal to "tns1:Credential/State/Enabled", FAIL the test and go to the step 23.

12.4.2. If *m* does not contain Source.SimpleItem item with Name = "CredentialToken" and with Value = *credentialToken*, FAIL the test and go to the step 23.

12.4.3. If *m* does not contain Data.SimpleItem item with Name = "State" and with Value = false, FAIL the test and go to the step 23.

12.4.4. If *m* does not contain Data.SimpleItem item with Name = "Reason" and with Value = "Test Reason", FAIL the test and go to the step 23.

12.4.5. If *m* does not contain Data.SimpleItem item with Name = "ClientUpdated", FAIL the test and go to the step 23.

12.4.6. If *m*.Message.Message.Data.SimpleItem.ClientUpdated has value type different from xs:boolean type, FAIL the test and go to the step 23.

12.4.7. Go to step 14.

13. If *oprationDelay* timeout expires for step 12 without any Notification, FAIL the test and go to the step 23.

14. ONVIF client invokes **GetCredentialState** with parameters

- Token := *credentialToken*

15. The DUT responds with **GetCredentialStateResponse** message with parameters

- State =: *credentialState2*

16. If *credentialState1*.Enabled equal to *credentialState2*.Enabled, FAIL the test and go to step 23.

17. ONVIF Client changes credential state (in *credentialState2*) for created credential (in *credentialToken*) by following the procedure mentioned in Annex A.4.

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

18.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.

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

- Timeout := PT60S
- MessageLimit := 1

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

- CurrentTime =: *ct*

- TerminationTime =: *tt*
- NotificationMessage =: *m*

18.4. If *m* is not null:

18.4.1. If TopicExpression item in *m* is not equal to "tns1:Credential/State/Enabled", FAIL the test and go to the step 23.

18.4.2. If *m* does not contain Source.SimpleItem item with Name = "CredentialToken" and with Value = *credentialToken*, FAIL the test and go to the step 23.

18.4.3. If *m* does not contain Data.SimpleItem item with Name = "State" and with Value = true, FAIL the test and go to the step 23.

18.4.4. If *m* does not contain Data.SimpleItem item with Name = "Reason" and with Value = "Test Reason", FAIL the test and go to the step 23.

18.4.5. If *m* does not contain Data.SimpleItem item with Name = "ClientUpdated", FAIL the test and go to the step 23.

18.4.6. If *m*.Message.Message.Data.SimpleItem.ClientUpdated has value type different from xs:boolean type, FAIL the test and go to the step 23. Go to step 20.

19. If *oprationDelay* timeout expires for step 11 without any Notification, FAIL the test and go to the step 22.

20. ONVIF client invokes **GetCredentialState** with parameters

- Token := *credentialToken*

21. The DUT responds with **GetCredentialStateResponse** message with parameters

- State =: *credentialState3*

22. If *credentialState3*.Enabled equal to *credentialState2*.Enabled, FAIL the test and go to step 23.

23. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

24. If there was credential deleted at step 4, restore it (in *credentialToRestore*) with initial state (in *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

25. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialStateResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

4.4.3 GET CREDENTIAL STATE WITH INVALID TOKEN

Test Case ID: CREDENTIAL-4-1-3

Specification Coverage: GetCredentialState command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialState

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential State with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:

- *invalidToken* := value not equal to any *credentialInfoCompleteList*.token

5. ONVIF client invokes **GetCredentialState** with parameters
   - Token := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.4.4 ENABLE CREDENTIAL WITH INVALID TOKEN

Test Case ID: CREDENTIAL-4-1-4

Specification Coverage: EnableCredential command (ONVIF Credential Service Specification)

Feature Under Test: EnableCredential

WSDL Reference: credential.wsdl

Test Purpose: To verify Enable Credential with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF client invokes **EnableCredential** with parameters
   - Token := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.4.5 DISABLE CREDENTIAL WITH INVALID TOKEN

Test Case ID: CREDENTIAL-4-1-5

Specification Coverage: DisableCredential command (ONVIF Credential Service Specification)

Feature Under Test: DisableCredential

WSDL Reference: credential.wsdl

Test Purpose: To verify Disable Credential with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.

4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF client invokes **DisableCredential** with parameters
   - Token := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.5 Credential Identifiers

4.5.1 GET CREDENTIAL IDENTIFIERS

Test Case ID: CREDENTIAL-5-1-1

Specification Coverage: CredentialIdentifier (ONVIF Credential Service Specification), CredentialIdentiferValue (ONVIF Credential Service Specification), GetCredentialIdentifiers command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialIdentifiers

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Identifiers.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
5. If *credentialCompleteList* is empty, skip other steps.
6. For each Credential.token *token* from *credentialCompleteList* repeat the following steps:
   6.1. ONVIF client invokes **GetCredentialIdentifiers** with parameters
      - CredentialToken := *token*
   6.2. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters
      - CredentialIdentifier list =: *credentialIdentifierList*
   6.3. If *credentialIdentifierList* contains at least two credential identifier items with equal Type.Name, FAIL the test and skip other steps.
   6.4. If *credentialIdentifierList* contains at least one credential identifier item with Type.Name other than listed in *cap*.SupportedIdentifierTypes, FAIL the test and skip other steps.
   6.5. If *credentialIdentifierList* does not contain all credential identifiers from *credentialCompleteList*[token = *token*].CredentialIdentifierList, FAIL the test and skip other steps.
   6.6. If *credentialIdentifierList* contains credential identifiers other than credential identifiers from *credentialCompleteList*[token = *token*].CredentialIdentifierList, FAIL the test and skip other steps.
   6.7. For each credential identifier *credentialIdentifier* from *credentialIdentifierList* repeat the following steps:
      6.7.1. If *credentialIdentifier* item does not have equal field values to *credentialCompleteList*[token = *token*].CredentialIdentifierList [Type.Name = *credentialIdentifier*.Type.Name] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: To compare identifiers list as steps 6.5 and 6.6 Type.Name is used as unique value for identifier.

Note: The following fields are compared at step 6.7.1:

- Type
  - Name
  - FormatType
- ExemptedFromAuthentication
- Value

4.5.2 SET CREDENTIAL IDENTIFIER – ADDING NEW TYPE

Test Case ID: CREDENTIAL-5-1-2

Specification Coverage: CredentialIdentifier (ONVIF Credential Service Specification), CredentialIdentiferValue (ONVIF Credential Service Specification), GetCredentialIdentifiers command (ONVIF Credential Service Specification), SetCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialIdentifier

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify creation of credential identifier and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. If *cap*.SupportedIdentifierType does not contain more than one IdentifierType item, skip other steps.
5. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
6. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
7. ONVIF Client creates credential (in *cap*) with ExemptedFromAuthentication equal to false (in false), with Credential identifier item (out *typeName1*) with corresponding Credential identifier Format Type (out *formatType1*) and corresponding credential identifier value (out *value1*) and with credential token (out *credentialToken*) by following the procedure mentioned in Annex A.11.
8. Set the following:
   - *identifierTypeNameList* := *cap*.SupportedIdentifierType - *typeName1*
9. ONVIF Client retrieves a Credential identifier type name (out *typeName2*) different from typeName1 (in *identifierTypeNameList*) with corresponding Credential identifier Format Type (out *formatType2*) and corresponding credential identifier value (out *value2*) by following the procedure mentioned in Annex A.15.
10. ONVIF Client invokes **CreatePullPointSubscription** with parameters
    - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
11. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
12. ONVIF client invokes **SetCredentialIdentifier** with parameters
    - CredentialToken := *credentialToken*
    - CredentialIdentifier.Type.Name := *typeName2*

- CredentialIdentifier.Type.FormatType := *formatType2*
- CredentialIdentifier.ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
- CredentialIdentifier.Value := *value2*

13. The DUT responds with empty **SetCredentialIdentifierResponse** message.
14. Until *oprationDelay* timeout expires, repeat the following steps:
    14.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    14.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    14.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    14.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 28.
    14.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 28.
    14.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 14.
15. If *oprationDelay* timeout expires for step 14 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 28.
16. ONVIF client invokes **GetCredentialIdentifiers** with parameters
    - CredentialToken := *credentialToken*
17. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters
    - CredentialIdentifier list =: *credentialIdentifierList*

18. If *credentialIdentifierList* contains more or less than two items, FAIL the test and go to step 28.

19. If *credentialIdentifierList* does not have item with CredentialIdentifier.Type.Name value equal to *typeName1*, FAIL the test and go to step 28.

20. If *credentialIdentifierList* does not have item with CredentialIdentifier.Type.Name value equal to *typeName2*, FAIL the test and go to step 28.

21. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName1* has FormatType value different from *formatType1*, FAIL the test and go to step 28.

22. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName1* has Value value different from *value1*, FAIL the test and go to step 28.

23. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName1* has ExemptedFromAuthentication value different from false, FAIL the test and go to step 28.

24. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName2* has FormatType value different from *formatType2*, FAIL the test and go to step 28.

25. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName2* has Value value different from *value2*, FAIL the test and go to step 28.

26. If *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication:

    26.1. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName2* has ExemptedFromAuthentication value different from true, FAIL the test and go to step 28.

27. If *cap*.Extension does not contain SupportedExemptionType element with value = pt:ExemptFromAuthentication:

    27.1. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName2* has ExemptedFromAuthentication value different from false, FAIL the test and go to step 28.

28. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

29. If there was credential deleted at step 6, restore it (in *credentialToRestore*, in *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

30. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialIdentifiersResponse** message.
- The DUT did not send **SetCredentialIdentifierResponse** message.
- The DUT did not send **CreatePullpointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

Note: If the DUT supports only one credential identifier type, type warning message at step 4.

4.5.3 SET CREDENTIAL IDENTIFIER – REPLACE OF THE SAME TYPE

Test Case ID: CREDENTIAL-5-1-3

Specification Coverage: CredentialIdentifier (ONVIF Credential Service Specification), CredentialIdentiferValue (ONVIF Credential Service Specification), GetCredentialIdentifiers command (ONVIF Credential Service Specification), SetCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialIdentifier

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify replacing of credential identifier and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
5. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList*, out *credentialToRestore*, out *stateToRestore*) by following the procedure mentioned in Annex A.7.
6. ONVIF Client retrieves a list of Credential identifier Type Name items, which support at least two Format Types (out *CredentialIdentifierTypeNameList*) by following the procedure mentioned in Annex A.16.
7. If *CredentialIdentifierTypeNameList* is empty, skip other steps.
8. ONVIF Client retrieves Credential identifier Type Name (out *typeName*) with corresponding Credential identifier Format Type (out *formatType1*) and Credential identifier value (out *value1*), and with corresponding Credential identifier Format Type (out *formatType2*) and Credential identifier value (out *value2*) based on Credential identifier Type Name items (in *CredentialIdentifierTypeNameList*) by following the procedure mentioned in Annex A.17.
9. ONVIF client invokes **CreateCredential** with parameters
   - Credential.token := ""
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped
   - Credential.CredentialIdentifier[0].Type.Name := *typeName*
   - Credential.CredentialIdentifier[0].Type.FormatType := *formatType1*
   - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
   - Credential.CredentialIdentifier[0].Value := *value1*
   - Credential.CredentialAccessProfile skipped

- Credential.Extension skipped
- State.Enabled := true
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := false if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

10. The DUT responds with **CreateCredentialResponse** message with parameters
    - Token =: *credentialToken*
11. ONVIF Client invokes **CreatePullPointSubscription** with parameters
    - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
12. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
13. ONVIF client invokes **SetCredentialIdentifier** with parameters
    - CredentialToken := *credentialToken*
    - CredentialIdentifier.Type.Name := *typeName*
    - CredentialIdentifier.Type.FormatType := formatType2
    - CredentialIdentifier.ExemptedFromAuthentication := true if *cap*.Extension contains SupportedExemptionType element with value = pt:ExemptFromAuthentication, otherwise false
    - CredentialIdentifier.Value := *value2*
    - CredentialIdentifier.Extension skipped
14. The DUT responds with empty **SetCredentialIdentifierResponse** message.
15. Until *oprationDelay* timeout expires, repeat the following steps:

    15.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.

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

- Timeout := PT60S
- MessageLimit := 1

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

- CurrentTime =: *ct*
- TerminationTime =: *tt*
- NotificationMessage =: *m*

15.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 21.

15.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 21.

15.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 16.

16. If oprationDelay timeout expires for step 14 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 21.

17. ONVIF client invokes **GetCredentialIdentifiers** with parameters

- CredentialToken := *credentialToken*

18. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters

- CredentialIdentifier list =: *credentialIdentifierList*

19. If *credentialIdentifierList* contains more or less than one item, FAIL the test and go to step 21.

20. If *credentialIdentifierList*[0].CredentialIdentifier.Type.Name item is not equal to *typeName*, FAIL the test and go to step 21.

21. If *credentialIdentifierList* item with CredentialIdentifier.Type.Name value equal to *typeName* has different field values to values from step 13, FAIL the test and go to step 21.

22. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

23. If there was credential deleted at step 5, restore it (in *credentialToRestore*, in *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

24. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialIdentifiersResponse** message.
- The DUT did not send **SetCredentialIdentifierResponse** message.
- The DUT did not send **CreatePullpointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

- CredentialIdentifier list (Type.Name is used as unique key for comparing)
  - Type
    - Name
    - FormatType
  - ExemptedFromAuthentication
  - Value

4.5.4 DELETE CREDENTIAL IDENTIFIER

Test Case ID: CREDENTIAL-5-1-4

Specification Coverage: CredentialIdentifier (ONVIF Credential Service Specification), CredentialIdentiferValue (ONVIF Credential Service Specification), GetCredentialIdentifiers command (ONVIF Credential Service Specification), SetCredentialIdentifiers command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialIdentifier

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify replacing of credential identifier and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. If *cap*.SupportedIdentifierType contains less than two SupportedIdentifierType items, skip other steps.
5. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
6. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
7. ONVIF Client creates credential with two Credential identifier items (out *typeName1*), (out *typeName2*), with corresponding Format Types (out *formatType1*), (out *formatType2*) and with corresponding values (out *value1*), (out *value2*), with antipass back state equal to false (in false), and with credential token (out *credentialToken*) by following the procedure mentioned in Annex A.18.
8. ONVIF Client invokes **CreatePullPointSubscription** with parameters
   - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
9. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
   - SubscriptionReference =: *s*
   - CurrentTime =: *ct*
   - TerminationTime =: *tt*
10. ONVIF client invokes **DeleteCredentialIdentifier** with parameters
    - CredentialToken := *credentialToken*

- CredentialIdentifierTypeName := *typeName1*

11. The DUT responds with empty **DeleteCredentialIdentifierResponse** message.
12. Until *oprationDelay* timeout expires, repeat the following steps:
    12.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    12.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    12.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    12.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 22.
    12.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 22.
    12.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 14.
13. If oprationDelay timeout expires for step 12 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 22.
14. ONVIF client invokes **GetCredentialIdentifiers** with parameters
    - CredentialToken := *credentialToken*
15. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters
    - CredentialIdentifier list =: *credentialIdentifierList*
16. If *credentialIdentifierList* contains less than one item, FAIL the test and go to step 22.
17. If *credentialIdentifierList* contains more than one item, FAIL the test and go to step 22.
18. If *credentialIdentifierList*[0].CredentialIdentifier.Type.Name value does not equal to *typeName2*, FAIL the test and go to step 22.

19. If *credentialIdentifierList*[0].CredentialIdentifier.Type.FormatType value does not equal to *formatType2*, FAIL the test and go to step 22.
20. If *credentialIdentifierList*[0].CredentialIdentifier.ExemptedFromAuthentication value does not equal to false, FAIL the test and go to step 22.
21. If *credentialIdentifierList*[0].CredentialIdentifier.Value value does not equal to *value2*, FAIL the test and go to step 22.
22. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.
23. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.
24. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
25. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialIdentifiersResponse** message.
- The DUT did not send **DeleteCredentialIdentifierResponse** message.
- The DUT did not send **CreatePullpointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

4.5.5 GET SUPPORTED FORMAT TYPES

Test Case ID: CREDENTIAL-5-1-5

Specification Coverage: CredentialIdentifier (ONVIF Credential Service Specification), GetSupportedFormatTypes command (ONVIF Credential Service Specification)

Feature Under Test: GetSupportedFormatTypes

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Supported Format Types.

Pre-Requisite: Credential 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. For each *cap*.SupportedIdentifierType *identifierType* repeat the following steps:
   4.1. ONVIF client invokes **GetSupportedFormatTypes** with parameters
      - CredentialIdentifierTypeName := *identifierType*
   4.2. The DUT responds with **GetSupportedFormatTypesResponse** message with parameters
      - FormatTypeInfo list =: *receivedformatTypeInfoList*
   4.3. Set *formatTypeInfoList*[*identifierType*] := *receivedFormatTypeInfoList*.
   4.4. If *formatTypeInfoList*[*identifierType*] is empty, FAIL the test and skip other steps.
   4.5. If *formatTypeInfoList*[*identifierType*] contains at least two format type info items with equal FormatType, FAIL the test and skip other steps.
5. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
6. If *credentialCompleteList* is empty, skip other steps.
7. For each Credential.token *token* from *credentialCompleteList* repeat the following steps:
   7.1. ONVIF client invokes **GetCredentialIdentifiers** with parameters
      - CredentialToken := *token*
   7.2. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters

- CredentialIdentifier list =: *credentialIdentifierList*

7.3. For each CredentialIdentifier.Type.Name *typeName* from *credentialIdentifierList* repeat the following steps:

7.3.1. If credential identifier type with *typeName* contains other format type (CredentialIdentifier.Type.FormatType) than listed in *formatTypeInfoList*[*typeName*], FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialIdentifiersResponse** message.
- The DUT did not send **GetSupportedFormatTypesResponse** message.

4.5.6 GET CREDENTIAL IDENTIFIERS WITH INVALID TOKEN

Test Case ID: CREDENTIAL-5-1-6

Specification Coverage: GetCredentialIdentifiers command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialIdentifiers

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Identifiers with Invalid Token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

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

3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF client invokes **GetCredentialIdentifiers** with parameters
   - CredentialToken := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.5.7 SET CREDENTIAL IDENTIFIER WITH INVALID TOKEN

Test Case ID: CREDENTIAL-5-1-7

Specification Coverage: SetCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialIdentifier with invalid token

WSDL Reference: credential.wsdl

Test Purpose: To verify Set Credential Identifier with Invalid Token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

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

3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
6. ONVIF Client retrieves (in *cap*.SupportedIdentifierType) a supported Credential identifier type name (out *typeName*) with Credential identifier Format Type (out *formatType*) and with credential identifier value (out *value*) by the following procedure mentioned in Annex A.15.
7. ONVIF client invokes **SetCredentialIdentifier** with parameters
   - CredentialToken := *invalidToken*
   - CredentialIdentifier.Type.Name := *typeName*
   - CredentialIdentifier.Type.FormatType := *formatType*
   - CredentialIdentifier.ExemptedFromAuthentication := false
   - CredentialIdentifier.Value := *value*
   - CredentialIdentifier.Extension skipped
8. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.5.8 DELETE CREDENTIAL IDENTIFIER WITH INVALID CREDENTIAL TOKEN

Test Case ID: CREDENTIAL-5-1-8

Specification Coverage: DeleteCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialIdentifier

WSDL Reference: credential.wsdl

Test Purpose: To verify Delete Credential Identifier with invalid credential token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
5. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
6. ONVIF client invokes **DeleteCredentialIdentifier** with parameters
   - CredentialToken := *invalidToken*
   - CredentialIdentifierTypeName := *cap*.SupportedIdentifierType[0]
7. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.5.9 DELETE CREDENTIAL IDENTIFIER WITH INVALID IDENTIFIER TYPE

Test Case ID: CREDENTIAL-5-1-9

Specification Coverage: DeleteCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialIdentifier

WSDL Reference: credential.wsdl

Test Purpose: To verify Delete Credential Identifier with invalid identifier type.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. If *cap*.SupportedIdentifierType contains less than two SupportedIdentifierType items, skip other steps.
5. ONVIF Client creates credential with two Credential identifier items (out *typeName1*), (out *typeName2*), with corresponding Format Types (out *formatType1*), (out *formatType2*) and with corresponding values (out *value1*), (out *value2*), with antipass back state equal to false (in false), and with credential token (out credentialToken) by following the procedure mentioned in Annex A.18.
6. Set the following:
   - *invalidIdentifierType* := value not equal to any *cap*.SupportedIdentifierType.
7. ONVIF client invokes **DeleteCredentialIdentifier** with parameters
   - CredentialToken := *credentialToken*

- CredentialIdentifierTypeName := *invalidIdentifierType*

8. The DUT returns **DeleteCredentialIdentifierResponse**.
9. ONVIF client invokes **GetCredentialIdentifiers** with parameters
   - CredentialToken := *credentialToken*
10. The DUT responds with **GetCredentialIdentifiersResponse** message with parameters
    - CredentialIdentifier list =: *credentialIdentifierList*
11. If *credentialIdentifierList* contains less than two items, FAIL the test and go to step 15.
12. If *credentialIdentifierList* contains more than two items, FAIL the test and go to step 15.
13. If *credentialIdentifierList* does not contain *typeName1*, FAIL the test and go to step 15.
14. If *credentialIdentifierList* does not contain *typeName2*, FAIL the test and go to step 15.
15. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.5.10 DELETE CREDENTIAL IDENTIFIER - MIN IDENTIFIERS PER CREDENTIAL

Test Case ID: CREDENTIAL-5-1-10

Specification Coverage: DeleteCredentialIdentifier command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialIdentifier

WSDL Reference: credential.wsdl

Test Purpose: To verify Delete Credential Identifier from a credential when credential contains only one Credential Identifier.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client creates credential with ExemptedFromAuthentication equal to false (in false), with Credential identifier item (out *typeName*) and with credential token (out *credentialToken*) by following the procedure mentioned in Annex A.11.
4. ONVIF client invokes **DeleteCredentialIdentifier** with parameters
   - CredentialToken := *credentialToken*
   - CredentialIdentifierTypeName := *typeName*
5. The DUT returns **env:Receiver\ter:ConstraintViolated\ter:MinIdentifiersPerCredential** SOAP 1.2 fault.
6. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Receiver\ter:ConstraintViolated \ter:MinIdentifiersPerCredential** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.6 Credential Access Profiles

4.6.1 GET CREDENTIAL ACCESS PROFILES

Test Case ID: CREDENTIAL-6-1-1

Specification Coverage: CredentialAccessProfile (ONVIF Credential Service Specification), GetCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialAccessProfiles

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Access Profiles.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity 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 gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
5. If *credentialCompleteList* is empty, skip other steps.
6. For each Credential.token *token* from *credentialCompleteList* repeat the following steps:
   6.1. ONVIF client invokes **GetCredentialAccessProfiles** with parameters
      - CredentialToken := *token*
   6.2. The DUT responds with **GetCredentialAccessProfilesResponse** message with parameters
      - CredentialAccessProfile list =: *credentialAccessProfileList*
   6.3. If *credentialAccessProfileList* contains at least two credential access profile items with equal AccessProfileToken, FAIL the test and skip other steps.
   6.4. If *credentialAccessProfileList* contains more AccessProfileInfo items than *cap*.MaxAccessProfilesPerCredential, FAIL the test and skip other steps.
   6.5. If *credentialAccessProfileList* does not contain all credential access profiles from *credentialCompleteList*[token = *token*].CredentialAccessProfileList, FAIL the test and skip other steps.

6.6. If *credentialAccessProfileList* contains credential access profiles other than credential access profiles from *credentialCompleteList*[token = *token*].CredentialAccessProfileList, FAIL the test and skip other steps.

6.7. For each credential access profile *accessProfile* from *credentialAccessProfileList* repeat the following steps:

6.7.1. If accessProfile item does not have equal field values to *credentialCompleteList*[token = *token*].CredentialAccessProfileList[AccessProfileToken = *accessProfile*.AccessProfileToken] item, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: To compare credential access profile list at steps 6.5 and 6.6, AccessProfileToken is used as unique value for credential access profile.

Note: The following fields are compared at step 6.7.1:

- AccessProfileToken
- ValidFrom
- ValidTo

4.6.2 SET CREDENTIAL ACCESS PROFILES - ADDING NEW ACCESS PROFILE

Test Case ID: CREDENTIAL-6-1-2

Specification Coverage: CredentialAccessProfile (ONVIF Credential Service Specification), SetCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialAccessProfiles

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify Set Credential Access Profiles (adding new Access Profile).

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
5. Set the following:
   - *accessProfileToken* := *accessProfileCompleteList*[0].token
6. If *accessProfileCompleteList* is empty, ONVIF Client creates access profile (out *accessProfileToken*) by following the procedure mentioned in Annex A.19.
7. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
8. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
9. ONVIF Client creates credential (in *cap*, out *credentialToken*) with Antipassback Violation State value equal to false (in false) by following the procedure mentioned in Annex A.11.
10. ONVIF Client invokes **CreatePullPointSubscription** with parameters
    - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
11. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*

- TerminationTime =: *tt*

12. ONVIF client invokes **SetCredentialAccessProfiles** with parameters
    - CredentialToken := *credentialToken*
    - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileToken*
    - Credential.CredentialAccessProfile.ValidFrom := *validFromDateTime* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidFrom skipped
    - Credential.CredentialAccessProfile.ValidTo := *validToDateTime* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidTo skipped
13. The DUT responds with empty **SetCredentialAccessProfilesResponse** message.
14. Until *oprationDelay* timeout expires, repeat the following steps:
    14.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    14.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1

    14.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*

    14.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 21.
    14.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 21.
    14.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 16.
15. If oprationDelay timeout expires for step 14 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 21.

16. ONVIF client invokes **GetCredentialAccessProfiles** with parameters
    - CredentialToken := *credentialToken*
17. The DUT responds with **GetCredentialAccessProfilesResponse** message with parameters
    - CredentialAccessProfile list =: *credentialAccessProfileList*
18. If *credentialAccessProfileList* contains more or less than one CredentialAccessProfile item, FAIL the test and go to step 21.
19. If *cap*.CredentialAccessProfileValiditySupported is equal to true and *cap*.ValiditySupportsTimeValue is equal to false:
    19.1. If *credentialAccessProfileList*[0].AccessProfileToken value is not equal to *accessProfileToken*, FAIL the test and go to step 21.
    19.2. If *credentialAccessProfileList*[0].ValidFrom value contains data component is not equal to data component of *validFromDateTime*, FAIL the test and go to step 21.
    19.3. If *credentialAccessProfileList*[0].ValidTo value contains data component that is not equal to data component of *validToDateTime*, FAIL the test and go to step 21.
    19.4. Go to the step 21.
20. If *credentialAccessProfileList*[0] item does not have equal field values to values from step 12, FAIL the test and go to step 21.
21. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.
22. If there was credential deleted at step 8, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.
23. If there was access profile created at step 6, ONVIF Client deletes it (in *accessProfileToken*) by following the procedure mentioned in Annex A.20 to restore DUT configuration.
24. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
25. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialAccessProfilesResponse** message.
- The DUT did not send **SetCredentialAccessProfilesResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

Note: *validFromDateTime* is set as current time, *validToDateTime* is set as current time + 1 year.

4.6.3 SET CREDENTIAL ACCESS PROFILES - UPDATING ACCESS PROFILE

Test Case ID: CREDENTIAL-6-1-3

Specification Coverage: CredentialAccessProfile (ONVIF Credential Service Specification), SetCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialAccessProfiles

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify Set Credential Access Profiles (updating Access Profile).

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.

4. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
5. Set the following:
   - *accessProfileToken* := *accessProfileCompleteList*[0].token
6. If *accessProfileCompleteList* is empty, ONVIF Client creates access profile (out *accessProfileToken*) by following the procedure mentioned in Annex A.19.
7. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
8. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
9. ONVIF Client creates credential (in *cap*, out *credentialToken*) with Antipassback Violation State equal to false (in false) by following the procedure mentioned in Annex A.11.
10. Set the following:
    - *validFromDateTime1* := value of current time
    - *validToDateTime1* := *validToDateTime1* + one year
11. ONVIF client invokes **SetCredentialAccessProfiles** with parameters
    - CredentialToken := *credentialToken*
    - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileToken*
    - Credential.CredentialAccessProfile.ValidFrom := *validFromDateTime1* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidFrom skipped.
    - Credential.CredentialAccessProfile.ValidTo := *validToDateTime1* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidTo skipped.
12. The DUT responds with empty **SetCredentialAccessProfilesResponse** message.
13. Set the following:
    - *validFromDateTime2* := *validFromDateTime1* + one day + one hour
    - *validToDateTime2* := *validToDateTime1* + one day + one hour
14. ONVIF Client invokes **CreatePullPointSubscription** with parameters

- Filter.TopicExpression := "tns1:Configuration/Credential/Changed"

15. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
16. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
17. ONVIF client invokes **SetCredentialAccessProfiles** with parameters
    - CredentialToken := *credentialToken*
    - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileCompleteList*[0].token
    - Credential.CredentialAccessProfile.ValidFrom := *validFromDateTime2* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidFrom skipped.
    - Credential.CredentialAccessProfile.ValidTo := *ValidToDateTime2* if *cap*.CredentialAccessProfileValiditySupported is equal to true. Otherwise Credential.CredentialAccessProfile.ValidTo skipped.
18. The DUT responds with empty **SetCredentialAccessProfilesResponse** message.
19. Until *oprationDelay* timeout expires, repeat the following steps:
    19.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    19.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    19.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*

19.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 26.

19.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 26.

19.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 21.

20. If oprationDelay timeout expires for step 19 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 26.

21. ONVIF client invokes **GetCredentialAccessProfiles** with parameters

- CredentialToken := *credentialToken*

22. The DUT responds with **GetCredentialAccessProfilesResponse** message with parameters

- CredentialAccessProfile list =: *credentialAccessProfileList*

23. If *credentialAccessProfileList* contains more or less than one CredentialAccessProfile item, FAIL the test and go to step 26.

24. If *cap*. CredentialAccessProfileValiditySupported is equal to true and *cap*.ValiditySupportsTimeValue is equal to false:

24.1. If *credentialAccessProfileList*[0].AccessProfileToken value is not equal to *accessProfileToken*, FAIL the test and go to step 26.

24.2. If *credentialAccessProfileList*[0].ValidFrom value contains data component that does not equal to data component of *validFromDateTime2*, FAIL the test and go to step 26.

24.3. If *credentialAccessProfileList*[0].ValidTo value contains data component that does not equal to data component of *validToDateTime2*, FAIL the test and go to step 26.

24.4. Go to the step 26.

25. If *credentialAccessProfileList*[0] item does not have equal field values to values from step 15, FAIL the test and go step 26.

26. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

27. If there was credential deleted at step 8, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

28. If there was access profile created at step 6, ONVIF Client deletes it (in *accessProfileToken*) by following the procedure mentioned in Annex A.20 to restore DUT configuration.
29. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
30. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialAccessProfilesResponse** message.
- The DUT did not send **SetCredentialAccessProfilesResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

4.6.4 DELETE CREDENTIAL ACCESS PROFILES

Test Case ID: CREDENTIAL-6-1-4

Specification Coverage: CredentialAccessProfile (ONVIF Credential Service Specification), DeleteCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialAccessProfiles

WSDL Reference: credential.wsdl, accessrules.wsdl, and event.wsdl

Test Purpose: To verify Delete Credential Access Profiles.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of access profile (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
4. Set the following:
   - *accessProfileToken* := *accessProfileCompleteList*[0].token
5. If *accessProfileCompleteList* is empty, ONVIF Client creates access profile (out *accessProfileToken*) by following the procedure mentioned in Annex A.19.
6. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
7. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
8. ONVIF Client creates credential (out *credentialToken*) with Antipassback Violation State equal to false (in false) by following the procedure mentioned in Annex A.11.
9. ONVIF client invokes **SetCredentialAccessProfiles** with parameters
   - CredentialToken := *credentialToken*
   - Credential.CredentialAccessProfile.AccessProfileToken := *accessProfileToken*
   - Credential.CredentialAccessProfile.ValidFrom skipped
   - Credential.CredentialAccessProfile.ValidTo skipped
10. The DUT responds with empty **SetCredentialAccessProfilesResponse** message.
11. ONVIF Client invokes **CreatePullPointSubscription** with parameters
    - Filter.TopicExpression := "tns1:Configuration/Credential/Changed"
12. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
13. ONVIF client invokes **DeleteCredentialAccessProfiles** with parameters

- CredentialToken := *credentialToken*
- AccessProfileToken := *accessProfileToken*

14. The DUT responds with empty **DeleteCredentialAccessProfilesResponse** message.
15. Until *oprationDelay* timeout expires, repeat the following steps:
    15.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    15.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    15.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    15.4. If *m* is not null and the TopicExpression item in *m* is not equal to "tns1:Configuration/Credential/Changed", FAIL the test and go to the step 20.
    15.5. If *m* is not null and does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 20.
    15.6. If *m* is not null and contains Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, go to the step 17.
16. If *oprationDelay* timeout expires for step 15 without Notification with CredentialToken source simple item equal to *credentialToken*, FAIL the test and go to the step 20.
17. ONVIF client invokes **GetCredentialAccessProfiles** with parameters
    - CredentialToken := *credentialToken*
18. The DUT responds with **GetCredentialAccessProfilesResponse** message with parameters
    - CredentialAccessProfile list =: *credentialAccessProfileList*
19. If *credentialAccessProfileList* contains at least one credential access profile item, FAIL the test and go to step 20.

20. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.
21. If there was credential deleted at step 7, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.
22. If there was access profile created at step 5, ONVIF Client deletes it (in *accessProfileToken*) by following the procedure mentioned in Annex A.20 to restore DUT configuration.
23. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.
24. The DUT responds with **UnsubscribeResponse** message.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **GetCredentialAccessProfilesResponse** message.
- The DUT did not send **SetCredentialAccessProfilesResponse** message.
- The DUT did not send **GetCredentialAccessProfilesResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

4.6.5 GET CREDENTIAL ACCESS PROFILES WITH INVALID TOKEN

Test Case ID: CREDENTIAL-6-1-5

Specification Coverage: GetCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: GetCredentialAccessProfiles

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Credential Access Profiles with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
5. ONVIF Client invokes *GetCredentialAccessProfiles* with parameters
   - Token := *invalidToken*
6. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.6.6 SET CREDENTIAL ACCESS PROFILES WITH INVALID CREDENTIAL TOKEN

Test Case ID: CREDENTIAL-6-1-6

Specification Coverage: SetCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: SetCredentialAccessProfiles

WSDL Reference: credential.wsdl

Test Purpose: To verify Set Credential Access Profiles with invalid credential token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of access profiles (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
4. If *accessProfileCompleteList* is not empty:
   4.1. Set the following:
      - *accessProfileToken* := *accessProfileCompleteList*[0].token
   4.2. Go to the step 6.
5. ONVIF Client creates access profile (out *accessProfileToken*) by following the procedure mentioned in Annex A.19.
6. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
7. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
8. ONVIF client invokes **SetCredentialAccessProfiles** with parameters
   - CredentialToken := *invalidToken*
   - Credential.CredentialAccessProfile.*AccessProfileToken* := *accessProfileToken*
   - Credential.CredentialAccessProfile.ValidFrom skipped.
   - Credential.CredentialAccessProfile.ValidTo skipped.
9. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.
10. If there was access profile created at step 5, ONVIF Client deletes it (in *accessProfileToken*) by following the procedure mentioned in Annex A.20 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.6.7 DELETE CREDENTIAL ACCESS PROFILES WITH INVALID CREDENTIAL TOKEN

Test Case ID: CREDENTIAL-6-1-7

Specification Coverage: DeleteCredentialAccessProfiles command (ONVIF Credential Service Specification)

Feature Under Test: DeleteCredentialAccessProfiles

WSDL Reference: credential.wsdl

Test Purpose: To verify Delete Credential Access Profiles with invalid credential token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of access profiles (out *accessProfileCompleteList*) by following the procedure mentioned in Annex A.5.
4. If *accessProfileCompleteList* is not empty:
   4.1. Set the following:
   - *accessProfileToken* := *accessProfileCompleteList*[0].token

4.2. Go to the step 6.

5. ONVIF Client creates access profile (out *accessProfileToken*) by following the procedure mentioned in Annex A.19.
6. ONVIF Client retrieves a complete list of credential info (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
7. Set the following:
   - *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
8. ONVIF client invokes **DeleteCredentialAccessProfiles** with parameters
   - CredentialToken := *invalidToken*
   - AccessProfileToken[0] := *accessProfileToken*
9. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.
10. If there was access profile created at step 5, ONVIF Client deletes it (in *accessProfileToken*) by following the procedure mentioned in Annex A.20 to restore DUT configuration.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.7 Reset Antipassback Violations

4.7.1 RESET ANTIPASSBACK VIOLATIONS

Test Case ID: CREDENTIAL-7-1-1

Specification Coverage: CredentialAccessProfile (ONVIF Credential Service Specification), ResetAntipassbackViolation command (ONVIF Credential Service Specification).

Feature Under Test: ResetAntipassbackViolation command (ONVIF Credential Service Specification)

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify resetting of antipassback violations and generating of appropriate notifications.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Event Service is received from the DUT. Device supports Pull-Point Notification feature. Reset Antipassback Violations is supported by the DUT as indicated by the Capabilities.ResetAntipassbackSupported capability. The DUT shall have enough free storage capacity for one additional Credential.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
4. If *cap*.ResetAntipassbackSupported is equal to false, FAIL the test and skip other steps.
5. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
6. ONVIF Client checks free storage for additional Credential (in *credentialCompleteList1*, out *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.7.
7. ONVIF Client creates credential (out *credentialToken*) with Antipassback Violation State equal to true (in true) by following the procedure mentioned in Annex A.11.
8. ONVIF client retrieves credential state (in *credentialToken*, out *credentialState*) by following the procedure mentioned in Annex A.13.
9. If *credentialState* does not contain AntipassbackState element, FAIL the test and go to step 20.
10. If *credentialState*.AntipassbackState.AntipassbackViolated equal to false, FAIL the test and go to step 20.
11. ONVIF Client invokes **CreatePullPointSubscription** with parameters

- Filter.TopicExpression := "tns1:Credential/State/ApbViolation"

12. The DUT responds with a **CreatePullPointSubscriptionResponse** message with parameters
    - SubscriptionReference =: *s*
    - CurrentTime =: *ct*
    - TerminationTime =: *tt*
13. ONVIF client invokes **ResetAntipassbackViolation** with parameter
    - Token := *credentialToken*
14. The DUT responds with empty **ResetAntipassbackViolationResponse** message.
15. Until *oprationDelay* timeout expires, repeat the following steps:
    15.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
    15.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* with parameters
        - Timeout := PT60S
        - MessageLimit := 1
    15.3. The DUT responds with **PullMessagesResponse** message with parameters
        - CurrentTime =: *ct*
        - TerminationTime =: *tt*
        - NotificationMessage =: *m*
    15.4. If *m* is not null:
        15.4.1. If the TopicExpression item in *m* is not equal to "tns1:Credential/State/ApbViolation", FAIL the test and go to the step 20.
        15.4.2. If *m* does not contain Source.SimpleItem item with Name = "CredentialToken" and Value = *credentialToken*, FAIL the test and go to the step 20.
            15.4.2.1. If *m* does not contain Data.SimpleItem item with Name = "ApbViolated" and Value = false, FAIL the test and go to the step 20.
        15.4.3. If *m* does not contain Data.SimpleItem item with Name = "ClientUpdated", FAIL the test and go to the step 20.

15.4.4. If *m*.Message.Message.Data.SimpleItem.ClientUpdated has value type different from xs:boolean type, FAIL the test and go to the step 20.

15.4.5. Go to step 17.

16. If oprationDelay timeout expires for step 15 without Notification with Source.SimpleItem item with Name = "CredentialToken" and Value = credentialToken and Data.SimpleItem item with Name = "ApbViolated" and Value = false, FAIL the test and go to the step 20.

17. ONVIF client retrieves credential state (in *credentialToken*, out *credentialState*) by following the procedure mentioned in Annex A.13.

18. If *credentialState* does not contain AntipassbackState element, FAIL the test and go to step 20.

19. If *credentialState*.AntipassbackState.AntipassbackViolated equal to true, FAIL the test and go to step 20.

20. ONVIF Client deletes the Credential (in *credentialToken*) by following the procedure mentioned in Annex A.6 to restore DUT configuration.

21. If there was credential deleted at step 4, restore it (in *credentialToRestore*, *stateToRestore*) by following the procedure mentioned in Annex A.10 to restore DUT configuration.

22. ONVIF Client sends an **Unsubscribe** to the subscription endpoint *s*.

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

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **ResetAntipassbackViolationResponse** message.
- The DUT did not send **CreatePullPointSubscriptionResponse** message.
- The DUT did not send **PullMessagesResponse** message.
- The DUT did not send **UnsubscribeResponse** message.

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

4.7.2 RESET ANTIPASSBACK VIOLATIONS WITH INVALID TOKEN

Test Case ID: CREDENTIAL-7-1-2

Specification Coverage: ResetAntipassbackViolation command (ONVIF Credential Service Specification).

Feature Under Test: ResetAntipassbackViolation command (ONVIF Credential Service Specification)

WSDL Reference: credential.wsdl

Test Purpose: To verify Reset Antipassback Violations with invalid token.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Reset Antipassback Violations is supported by the DUT as indicated by the Capabilities.ResetAntipassbackSupported capability.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete credential information list (out *credentialInfoCompleteList*) by following the procedure mentioned in Annex A.1.
4. Set the following:
5. *invalidToken* := value not equal to any *credentialInfoCompleteList*.token
6. ONVIF client invokes **ResetAntipassbackViolation** with parameters
   - Token := *invalidToken*
7. The DUT returns **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **env:Sender\ter:InvalidArgVal\ter:NotFound** SOAP 1.2 fault

Note: If the DUT sends other SOAP 1.2 fault message than specified, log WARNING message, and PASS the test.

4.8 Credential Events

4.8.1 CONFIGURATION CREDENTIAL CHANGED EVENT

Test Case ID: CREDENTIAL-8-1-2

Specification Coverage: Notification topics (ONVIF Credential Service Specification), Get event properties (ONVIF Core specification).

Feature Under Test: GetEventProperties

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify tns1:Configuration/Credential/Changed event format.

Pre-Requisite: Credential Service is supported by the DUT. Credential Entity is supported by the DUT. Event Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **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:Configuration/Credential/Changed topic, FAIL the test and skip other steps.
6. ONVIF Client verifies tns1:Configuration/Credential/Changed topic (*changedTopic*) from *topicSet*:
   6.1. If *changedTopic*.MessageDescription.IsProperty equals to true, FAIL the test and skip other steps.
   6.2. If *changedTopic* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "CredentialToken", FAIL the test and skip other steps.
   6.3. If *changedTopic*.MessageDescription.Source.SimpleItemDescription with Name = "CredentialToken" does not have Type = "pt:ReferenceToken", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.8.2 CONFIGURATION CREDENTIAL REMOVED EVENT

Test Case ID: CREDENTIAL-8-1-3

Specification Coverage: Notification topics (ONVIF Credential Service Specification), Get event properties (ONVIF Core specification).

Feature Under Test: GetEventProperties

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify tns1:Configuration/Credential/Removed event format.

Pre-Requisite: Credential Service is supported by the DUT. Credential Entity is supported by the DUT. Event Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.

2. Start the DUT.
3. ONVIF Client invokes **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:Configuration/Credential/Removed topic, FAIL the test and skip other steps.
6. ONVIF Client verifies tns1:Configuration/Credential/Removed topic (*removedTopic*) from *topicSet*:
   6.1. If *removedTopic*.MessageDescription.IsProperty equals to true, FAIL the test and skip other steps.
   6.2. If *removedTopic* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "CredentialToken", FAIL the test and skip other steps.
   6.3. If *removedTopic*.MessageDescription.Source.SimpleItemDescription with Name = "CredentialToken" does not have Type = "pt:ReferenceToken", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.8.3 CREDENTIAL STATE ENABLED EVENT

Test Case ID: CREDENTIAL-8-1-4

Specification Coverage: Notification topics (ONVIF Credential Service Specification), Get event properties (ONVIF Core specification).

Feature Under Test: GetEventProperties

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify tns1:Credential/State/Enabled event format.

Pre-Requisite: Credential Service is supported by the DUT. Credential Entity is supported by the DUT. Event Service is supported by the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client invokes **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:Credential/State/Enabled topic, FAIL the test and skip other steps.
6. ONVIF Client verifies tns1:Credential/State/Enabled topic (*StateEnabledTopic*) from *topicSet*:
   6.1. If *StateEnabledTopic*.MessageDescription.IsProperty equals true, FAIL the test and skip other steps.
   6.2. If *StateEnabledTopic* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "CredentialToken", FAIL the test and skip other steps.

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

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

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

6.6. If *StateEnabledTopic* does not contain MessageDescription.Data.SimpleItemDescription item with Name = "Reason", FAIL the test and skip other steps.

6.7. If *StateEnabledTopic*.MessageDescription.Data.SimpleItemDescription with Name = "Reason" does not have Type = "xs:string", FAIL the test and skip other steps.

6.8. If *StateEnabledTopic* does not contain Data.SimpleItemDescription with Name = "ClientUpdated", FAIL the test and skip other steps.

6.9. If *StateEnabledTopic*.Message.Message.Data.SimpleItemDescription with Name = "ClientUpdated" does not have Type = "xs:boolean", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.8.4 CREDENTIAL STATE ANTIPASSBACK VIOLATION EVENT

Test Case ID: CREDENTIAL-8-1-5

Specification Coverage: Notification topics (ONVIF Credential Service Specification), Get event properties (ONVIF Core specification).

Feature Under Test: GetEventProperties

WSDL Reference: credential.wsdl and event.wsdl

Test Purpose: To verify tns1:Credential/State/ApbViolation event format.

Pre-Requisite: Credential Service is supported by the DUT. Credential Entity is supported by the DUT. Event Service is supported by the DUT. ResetAntipassbackViolation capability 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**.
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:Credential/State/ApbViolation topic, FAIL the test and skip other steps.
6. ONVIF Client verifies tns1:Credential/State/ApbViolation topic (*ApbViolationTopic*) from *topicSet*:
   6.1. If *ApbViolationTopic*.MessageDescription.IsProperty equal to true, FAIL the test and skip other steps.
   6.2. If *ApbViolationTopic* does not contain MessageDescription.Source.SimpleItemDescription item with Name = "CredentialToken", FAIL the test and skip other steps.
   6.3. If *ApbViolationTopic*.MessageDescription.Source.SimpleItemDescription with Name = "CredentialToken" does not have Type = "pt:ReferenceToken", FAIL the test and skip other steps.

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

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

6.6. If *ApbViolationTopic* does not contain Data.SimpleItemDescription with Name = "ClientUpdated", FAIL the test and skip other steps.

6.7. If *ApbViolationTopic*.Message.Message.Data.SimpleItemDescription with Name = "ClientUpdated" does not have Type = "xs:boolean", FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

FAIL –

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

4.9 Consistency

4.9.1 GET CREDENTIAL AND GET ACCESS PROFILE INFO LIST CONSISTENCY

Test Case ID: CREDENTIAL-9-1-1

Specification Coverage: Credential (ONVIF Credential Service Specification), AccessProfileInfo (ONVIF Access Rules Service Specification)

Feature Under Test: GetCredentials, GetAccessProfileInfo

WSDL Reference: credential.wsdl and accessrules.wsdl

Test Purpose: To verify that all Access Profile Tokens from GetCredentialResponses could be listed through GetAccessProfileInfoList command.

Pre-Requisite: Credential Service is received from the DUT. Credential Entity is supported by the DUT. Access Rules Service is received from the DUT.

Test Configuration: ONVIF Client and DUT

Test Procedure:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete list of credentials (out *credentialCompleteList*) by following the procedure mentioned in Annex A.3.
4. ONVIF Client retrieves a complete list of access profile info (out *accessProfileInfoCompleteList*) by following the procedure mentioned in Annex A.12.
5. For each Credential.CredentialAccessProfile.AccessProfileToken (*credentialAccessProfileToken*) from *credentialCompleteList* repeat the following steps:
   5.1. If *credentialAccessProfileToken* is not listed in *accessProfileInfoCompleteList*, FAIL the test and skip other steps.

Test Result:

PASS –

- DUT passes all assertions.

4.10 Whitelist Management

4.10.1 GET WHITELIST - START REFERENCE AND LIMIT

Test Case ID: CREDENTIAL-10-1-1

Specification Coverage: GetWhitelist command (ONVIF Credential Service Specification)

Feature Under Test: GetWhitelist

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Whitelist using StartReference and Limit.

Pre-Requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client adds number of credential identifiers to whitelist by following the procedure mentioned in Annex A.24 with the following input and output parameters
   - out *credentialIdentifiersList* - list of added credential identifiers
   - out *cap* - Credential Service capabilities
   - out *completeWhitelist* - complete whitelist
4. ONVIF client invokes **GetWhitelist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped
5. The DUT responds with **GetWhitelistResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Identifier list =: *identifierCompleteList0*
6. If *identifierCompleteList0* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:
   7.1. ONVIF client invokes **GetWhitelist** with parameters
      - Limit is skipped
      - StartReference := *nextStartReference*
      - IdentifierType is skipped
      - FormatType is skipped
      - Value is skipped
   7.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

7.3. If *identifierPart* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.

7.4. Set *identifierCompleteList0* := *identifierCompleteList0* + *identifierPart*.

8. If *identifierCompleteList0* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
9. ONVIF client invokes **GetWhitelist** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped
10. The DUT responds with **GetWhitelistResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList1*
11. If *identifierCompleteList1* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.
12. Until *nextStartReference* is not null, repeat the following steps:

    12.1. ONVIF client invokes **GetWhitelist** with parameters
    - Limit := *cap*.MaxLimit
    - StartReference := *nextStartReference*
    - IdentifierType is skipped
    - FormatType is skipped
    - Value is skipped

    12.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

12.3. If *identifierPart* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.

12.4. Set *identifierCompleteList1* := *identifierCompleteList1* + *identifierPart*.

13. If *identifierCompleteList1* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
14. If *identifierCompleteList1* does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
15. If *identifierCompleteList1* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
16. If *cap*.MaxLimit is equal to 1, restore the DUT state, and skip other steps.
17. ONVIF client invokes **GetWhitelist** with parameters
    - Limit := 1
    - StartReference skipped
    - IdentifierType is skipped
    - FormatType is skipped
    - Value is skipped
18. The DUT responds with **GetWhitelistResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList2*
19. If *identifierCompleteList2* contains more Identifier items than 1, FAIL the test, restore the DUT state, and skip other steps.
20. Until *nextStartReference* is not null, repeat the following steps:

    20.1. ONVIF client invokes **GetWhitelist** with parameters
    - Limit := 1
    - StartReference := *nextStartReference*

- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

20.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierListPart*

20.3. If *identifierListPart* contains more Identifier items than 1, FAIL the test, restore the DUT state, and skip other steps.

20.4. Set *identifierCompleteList2* := *identifierCompleteList2* + *identifierListPart*

21. If *identifierCompleteList2* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

22. If *identifierCompleteList2* does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

23. If *identifierCompleteList2* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

24. If *cap*.MaxLimit is equal to 2, restore the DUT state, and skip other steps.

25. Set *limit* := [number between 1 and *cap*.MaxLimit].

26. ONVIF client invokes **GetWhitelist** with parameters

- Limit := *limit*
- StartReference skipped
- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

27. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList3*

28. If *identifierCompleteList3* contains more Identifier items than *limit*, FAIL the test, restore the DUT state, and skip other steps.
29. Until *nextStartReference* is not null, repeat the following steps:
    29.1. ONVIF client invokes **GetWhitelist** with parameters
        - Limit := *limit*
        - StartReference := *nextStartReference*
        - IdentifierType is skipped
        - FormatType is skipped
        - Value is skipped
    29.2. The DUT responds with **GetWhitelistResponse** message with parameters
        - NextStartReference =: *nextStartReference*
        - Identifier list =: *identifierListPart*
    29.3. If *identifierListPart* contains more Identifier items than *limit*, FAIL the test, restore the DUT state, and skip other steps.
    29.4. Set *identifierCompleteList3* := *identifierCompleteList3* + *identifierListPart*
30. If *identifierCompleteList3* contains at least two equal Identifiers, FAIL the test, restore the DUT state, and skip other steps.
31. If *identifierCompleteList3*does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
32. If *identifierCompleteList3* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
33. Remove all credential identifiers from *credentialIdentifiersList* from whitelist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.10.2 GET WHITELIST - FILTERS

Test Case ID: CREDENTIAL-10-1-2

Specification Coverage: GetWhitelist command (ONVIF Credential Service Specification)

Feature Under Test: GetWhitelist

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Whitelist using filters.

Pre-Requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client adds number of credential identifiers to whitelist by following the procedure mentioned in Annex A.24 with the following input and output parameters
   - out *credentialIdentifiersList* - list of added credential identifiers
   - out *cap* - Credential Service capabilities
   - out *completeWhitelist* - complete whitelist
4. Set *identifier* := *completeWhitelist*[0].
5. ONVIF client invokes **GetWhitelist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType := *identifier*.Type.Name
   - FormatType is skipped
   - Value is skipped
6. The DUT responds with **GetWhitelistResponse** message with parameters
   - NextStartReference =: *nextStartReference*

- Identifier list =: *identifierCompleteList0*

7. Until *nextStartReference* is not null, repeat the following steps:

   7.1. ONVIF client invokes **GetWhitelist** with parameters

   - Limit is skipped
   - StartReference := *nextStartReference*
   - IdentifierType := *identifier*.Type.Name
   - FormatType is skipped
   - Value is skipped

   7.2. The DUT responds with **GetWhitelistResponse** message with parameters

   - NextStartReference =: *nextStartReference*
   - Identifier list =: *identifierPart*

   7.3. Set *identifierCompleteList0* := *identifierCompleteList0* + *identifierPart*.

8. If *identifierCompleteList0* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
9. If *identifierCompleteList0* contains at least one item with Type.Name other than *identifier*.Type.Name, FAIL the test, restore the DUT state, and skip other steps.
10. ONVIF client invokes **GetWhitelist** with parameters

    - Limit is skipped
    - StartReference is skipped
    - IdentifierType is skipped
    - FormatType := *identifier*.Type.FormatType
    - Value is skipped

11. The DUT responds with **GetWhitelistResponse** message with parameters

    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList1*

12. Until *nextStartReference* is not null, repeat the following steps:

12.1. ONVIF client invokes **GetWhitelist** with parameters

- Limit is skipped
- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType := *identifier*.Type.FormatType
- Value is skipped

12.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

12.3. Set *identifierCompleteList1* := *identifierCompleteList1* + *identifierPart*.

13. If *identifierCompleteList1* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

14. If *identifierCompleteList1* contains at least one item with Type.FormatType other than *identifier*.Type.FormatType, FAIL the test, restore the DUT state, and skip other steps.

15. ONVIF client invokes **GetWhitelist** with parameters

- Limit is skipped
- StartReference is skipped
- IdentifierType is skipped
- FormatType is skipped
- Value := *identifier*.Value

16. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList2*

17. Until *nextStartReference* is not null, repeat the following steps:

17.1. ONVIF client invokes **GetWhitelist** with parameters

- Limit is skipped
- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped
- Value := *identifier*.Value

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

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

17.3. Set *identifierCompleteList2* := *identifierCompleteList2* + *identifierPart*.

18. If *identifierCompleteList2* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

19. If *identifierCompleteList2* contains at least one item with Value other than *identifier*.Value, FAIL the test, restore the DUT state, and skip other steps.

20. ONVIF client invokes **GetWhitelist** with parameters

- Limit is skipped
- StartReference is skipped
- IdentifierType := *identifier*.Type.Name
- FormatType := *identifier*.Type.FormatType
- Value := *identifier*.Value

21. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList3*

22. Until *nextStartReference* is not null, repeat the following steps:

22.1. ONVIF client invokes **GetWhitelist** with parameters

- Limit is skipped

- StartReference := *nextStartReference*
- IdentifierType := *identifier*.Type.Name
- FormatType := *identifier*.Type.FormatType
- Value := *identifier*.Value

22.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

22.3. Set *identifierCompleteList3* := *identifierCompleteList3* + *identifierPart*.

23. If *identifierCompleteList3* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
24. If *identifierCompleteList3* contains at least one item with Type.Name other than *identifier*.Type.Name, FAIL the test, restore the DUT state, and skip other steps.
25. If *identifierCompleteList3* contains at least one item with Type.FormatType other than *identifier*.Type.FormatType, FAIL the test, restore the DUT state, and skip other steps.
26. If *identifierCompleteList3* contains at least one item with Value other than *identifier*.Value, FAIL the test, restore the DUT state, and skip other steps.
27. Remove all credential identifiers from *credentialIdentifiersList* from whitelist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.10.3 ADD IDENTIFIER TO WHITELIST

Test Case ID: CREDENTIAL-10-1-3

Specification Coverage: AddToWhitelist command (ONVIF Credential Service Specification)

Feature Under Test: AddToWhitelist

WSDL Reference: credential.wsdl

Test Purpose: To verify adding identifier to whitelist.

Pre-Requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in whitelist. The DUT shall have enough free storage capacity for two additional items in blacklist, if blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *initialWhiteList* - complete whitelist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxWhitelistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
6. If *cap*.MaxBlacklistedItems > 0:
   6.1. ONVIF Client adds credential identifiers to blacklist by following the procedure mentioned in Annex A.26 with the following input and output parameters
      - in *credentialIdentifiersList* - credential identifier list to be added to blacklist
7. ONVIF client invokes **AddToWhitelist** with parameters

- Identifier list := *credentialIdentifiersList*

8. The DUT responds with **AddToWhitelistResponse** message.
9. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *updatedWhiteList* - complete whitelist
10. If *updatedWhiteList* does not contain all items from *initialWhiteList*, FAIL the test, restore the DUT state, and skip other steps.
11. If *updatedWhiteList* does not contain all items from *credentialIdentifiersList* item, FAIL the test, restore the DUT state, and skip other steps.
12. If *cap*.MaxBlacklistedItems > 0:
    12.1. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
    - out *updatedBlackList* - complete blacklist

    12.2. If *updatedBlackList* contains any item from *credentialIdentifiersList* item, FAIL the test, restore the DUT state, and skip other steps.
13. ONVIF client invokes **AddToWhitelist** with parameters
    - Identifier list := *credentialIdentifiersList*
14. The DUT responds with **AddToWhitelistResponse** message.
15. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
    - out *updatedWhiteList2* - complete whitelist
16. If *updatedWhiteList2* contains more items that *updatedWhiteList*, FAIL the test, restore the DUT state, and skip other steps.
17. Remove all items from *credentialIdentifiersList* from whitelist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.10.4 REMOVE IDENTIFIER FROM WHITELIST

Test Case ID: CREDENTIAL-10-1-4

Specification Coverage: RemoveFromWhitelist command (ONVIF Credential Service Specification)

Feature Under Test: RemoveFromWhitelist

WSDL Reference: credential.wsdl

Test Purpose: To verify removing identifier to whitelist.

Pre-Requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in whitelist.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *initialWhiteList* - complete whitelist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxWhitelistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list

6. ONVIF Client adds credential identifiers to whitelist by following the procedure mentioned in Annex A.28 with the following input and output parameters
   - in *credentialIdentifiersList* - credential identifiers to be added to whitelist
7. ONVIF client invokes **RemoveFromWhitelist** with parameters
   - Identifier list := *credentialIdentifiersList*
8. The DUT responds with **RemoveFromWhitelistResponse** message.
9. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *updatedWhiteList* - complete whitelist
10. If *updatedWhiteList* does not contain all items from *initialWhiteList*, FAIL the test, restore the DUT state, and skip other steps.
11. If *updatedWhiteList* contains any item from *credentialIdentifiersList*, FAIL the test, restore the DUT state, and skip other steps.
12. ONVIF client invokes **RemoveFromWhitelist** with parameters
    - Identifier list := *credentialIdentifiersList*
13. The DUT responds with **RemoveFromWhitelistResponse** message.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.10.5 DELETE WHITELIST

Test Case ID: CREDENTIAL-10-1-5

Specification Coverage: DeleteWhitelist command (ONVIF Credential Service Specification)

Feature Under Test: DeleteWhitelist

WSDL Reference: credential.wsdl

Test Purpose: To verify deleting whitelist.

Pre-Requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in whitelist.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *initialWhiteList* - complete whitelist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxWhitelistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
6. ONVIF Client adds credential identifiers to whitelist by following the procedure mentioned in Annex A.28 with the following input and output parameters
   - in *credentialIdentifiersList* - credential identifiers to be added to whitelist
7. ONVIF client invokes **DeleteWhitelist**.
8. The DUT responds with **DeleteWhitelistResponse** message.
9. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters

- out *updatedWhiteList* - complete whitelist

10. If *updatedWhiteList* contains any items, FAIL the test, restore the DUT state, and skip other steps.
11. ONVIF client invokes **DeleteWhitelist**.
12. The DUT responds with **DeleteWhitelistResponse** message.
13. Client restores all items from *initialWhiteList*.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.11 Blacklist Management

4.11.1 GET BLACKLIST - START REFERENCE AND LIMIT

Test Case ID: CREDENTIAL-11-1-1

Specification Coverage: GetBlacklist command (ONVIF Credential Service Specification)

Feature Under Test: GetBlacklist

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Blacklist using StartReference and Limit.

Pre-Requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

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

3. ONVIF Client adds number of credential identifiers to blacklist by following the procedure mentioned in Annex A.29 with the following input and output parameters
   - out *credentialIdentifiersList* - list of added credential identifiers
   - out *cap* - Credential Service capabilities
   - out *completeBlacklist* - complete blacklist
4. ONVIF client invokes **GetBlacklist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped
5. The DUT responds with **GetBlacklistResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Identifier list =: *identifierCompleteList0*
6. If *identifierCompleteList0* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.
7. Until *nextStartReference* is not null, repeat the following steps:
   7.1. ONVIF client invokes **GetBlacklist** with parameters
      - Limit is skipped
      - StartReference := *nextStartReference*
      - IdentifierType is skipped
      - FormatType is skipped
      - Value is skipped
   7.2. The DUT responds with **GetBlacklistResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - Identifier list =: *identifierPart*

7.3. If *identifierPart* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.

7.4. Set *identifierCompleteList0* := *identifierCompleteList0* + *identifierPart*.

8. If *identifierCompleteList0* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

9. ONVIF client invokes **GetBlacklist** with parameters
   - Limit := *cap*.MaxLimit
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped

10. The DUT responds with **GetBlacklistResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList1*

11. If *identifierCompleteList1* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.

12. Until *nextStartReference* is not null, repeat the following steps:

    12.1. ONVIF client invokes **GetBlacklist** with parameters
    - Limit := *cap*.MaxLimit
    - StartReference := *nextStartReference*
    - IdentifierType is skipped
    - FormatType is skipped
    - Value is skipped

    12.2. The DUT responds with **GetBlacklistResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierPart*

12.3. If *identifierPart* contains more Identifier items than *cap*.MaxLimit, FAIL the test, restore the DUT state, and skip other steps.

12.4. Set *identifierCompleteList1* := *identifierCompleteList1* + *identifierPart*.

13. If *identifierCompleteList1* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

14. If *identifierCompleteList1* does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

15. If *identifierCompleteList1* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

16. If *cap*.MaxLimit is equal to 1, restore the DUT state, and skip other steps.

17. ONVIF client invokes **GetBlacklist** with parameters

- Limit := 1
- StartReference skipped
- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

18. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList2*

19. If *identifierCompleteList2* contains more Identifier items than 1, FAIL the test, restore the DUT state, and skip other steps.

20. Until *nextStartReference* is not null, repeat the following steps:

20.1. ONVIF client invokes **GetBlacklist** with parameters

- Limit := 1
- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped

- Value is skipped

20.2. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierListPart*

20.3. If *identifierListPart* contains more Identifier items than 1, FAIL the test, restore the DUT state, and skip other steps.

20.4. Set *identifierCompleteList2* := *identifierCompleteList2* + *identifierListPart*

21. If *identifierCompleteList2* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
22. If *identifierCompleteList2* does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
23. If *identifierCompleteList2* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.
24. If *cap*.MaxLimit is equal to 2, restore the DUT state, and skip other steps.
25. Set *limit* := [number between 1 and *cap*.MaxLimit].
26. ONVIF client invokes **GetBlacklist** with parameters
    - Limit := *limit*
    - StartReference skipped
    - IdentifierType is skipped
    - FormatType is skipped
    - Value is skipped
27. The DUT responds with **GetBlacklistResponse** message with parameters
    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList3*
28. If *identifierCompleteList3* contains more Identifier items than *limit*, FAIL the test, restore the DUT state, and skip other steps.
29. Until *nextStartReference* is not null, repeat the following steps:

29.1. ONVIF client invokes **GetBlacklist** with parameters

- Limit := *limit*
- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

29.2. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierListPart*

29.3. If *identifierListPart* contains more Identifier items than *limit*, FAIL the test, restore the DUT state, and skip other steps.

29.4. Set *identifierCompleteList3* := *identifierCompleteList3* + *identifierListPart*

30. If *identifierCompleteList3* contains at least two equal Identifiers, FAIL the test, restore the DUT state, and skip other steps.

31. If *identifierCompleteList3*does not contain all Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

32. If *identifierCompleteList3* contains Identifiers other than Identifiers from *identifierCompleteList0*, FAIL the test, restore the DUT state, and skip other steps.

33. Remove all credential identifiers from *credentialIdentifiersList* from blacklist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.11.2 GET BLACKLIST - FILTERS

Test Case ID: CREDENTIAL-11-1-2

Specification Coverage: GetBlacklist command (ONVIF Credential Service Specification)

Feature Under Test: GetBlacklist

WSDL Reference: credential.wsdl

Test Purpose: To verify Get Blacklist using filters.

Pre-Requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client adds number of credential identifiers to blacklist by following the procedure mentioned in Annex A.29 with the following input and output parameters
   - out *credentialIdentifiersList* - list of added credential identifiers
   - out *cap* - Credential Service capabilities
   - out *completeBlacklist* - complete blacklist
4. Set *identifier* := *completeBlacklist*[0].
5. ONVIF client invokes **GetBlacklist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType := *identifier*.Type.Name
   - FormatType is skipped
   - Value is skipped
6. The DUT responds with **GetBlacklistResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Identifier list =: *identifierCompleteList0*

7. Until *nextStartReference* is not null, repeat the following steps:

   7.1. ONVIF client invokes **GetBlacklist** with parameters

   - Limit is skipped
   - StartReference := *nextStartReference*
   - IdentifierType := *identifier*.Type.Name
   - FormatType is skipped
   - Value is skipped

   7.2. The DUT responds with **GetBlacklistResponse** message with parameters

   - NextStartReference =: *nextStartReference*
   - Identifier list =: *identifierPart*

   7.3. Set *identifierCompleteList0* := *identifierCompleteList0* + *identifierPart*.

8. If *identifierCompleteList0* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

9. If *identifierCompleteList0* contains at least one item with Type.Name other than *identifier*.Type.Name, FAIL the test, restore the DUT state, and skip other steps.

10. ONVIF client invokes **GetBlacklist** with parameters

    - Limit is skipped
    - StartReference is skipped
    - IdentifierType is skipped
    - FormatType := *identifier*.Type.FormatType
    - Value is skipped

11. The DUT responds with **GetBlacklistResponse** message with parameters

    - NextStartReference =: *nextStartReference*
    - Identifier list =: *identifierCompleteList1*

12. Until *nextStartReference* is not null, repeat the following steps:

    12.1. ONVIF client invokes **GetBlacklist** with parameters

- Limit is skipped
- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType := *identifier*.Type.FormatType
- Value is skipped

12.2. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

12.3. Set *identifierCompleteList1* := *identifierCompleteList1* + *identifierPart*.

13. If *identifierCompleteList1* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

14. If *identifierCompleteList1* contains at least one item with Type.FormatType other than *identifier*.Type.FormatType, FAIL the test, restore the DUT state, and skip other steps.

15. ONVIF client invokes **GetBlacklist** with parameters

- Limit is skipped
- StartReference is skipped
- IdentifierType is skipped
- FormatType is skipped
- Value := *identifier*.Value

16. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList2*

17. Until *nextStartReference* is not null, repeat the following steps:

17.1. ONVIF client invokes **GetBlacklist** with parameters

- Limit is skipped

- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped
- Value := *identifier*.Value

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

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

17.3. Set *identifierCompleteList2* := *identifierCompleteList2* + *identifierPart*.

18. If *identifierCompleteList2* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.

19. If *identifierCompleteList2* contains at least one item with Value other than *identifier*.Value, FAIL the test, restore the DUT state, and skip other steps.

20. ONVIF client invokes **GetBlacklist** with parameters

- Limit is skipped
- StartReference is skipped
- IdentifierType := *identifier*.Type.Name
- FormatType := *identifier*.Type.FormatType
- Value := *identifier*.Value

21. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierCompleteList3*

22. Until *nextStartReference* is not null, repeat the following steps:

22.1. ONVIF client invokes **GetBlacklist** with parameters

- Limit is skipped
- StartReference := *nextStartReference*

- IdentifierType := *identifier*.Type.Name
- FormatType := *identifier*.Type.FormatType
- Value := *identifier*.Value

22.2. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

22.3. Set *identifierCompleteList3* := *identifierCompleteList3* + *identifierPart*.

23. If *identifierCompleteList3* contains at least two equal Identifier items, FAIL the test, restore the DUT state, and skip other steps.
24. If *identifierCompleteList3* contains at least one item with Type.Name other than *identifier*.Type.Name, FAIL the test, restore the DUT state, and skip other steps.
25. If *identifierCompleteList3* contains at least one item with Type.FormatType other than *identifier*.Type.FormatType, FAIL the test, restore the DUT state, and skip other steps.
26. If *identifierCompleteList3* contains at least one item with Value other than *identifier*.Value, FAIL the test, restore the DUT state, and skip other steps.
27. Remove all credential identifiers from *credentialIdentifiersList* from blacklist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.11.3 ADD IDENTIFIER TO BLACKLIST

Test Case ID: CREDENTIAL-11-1-3

Specification Coverage: AddToBlacklist command (ONVIF Credential Service Specification)

Feature Under Test: AddToBlacklist

WSDL Reference: credential.wsdl

Test Purpose: To verify adding identifier to blacklist.

Pre-Requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in blacklist. The DUT shall have enough free storage capacity for two additional items in whitelist, if whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *initialBlackList* - complete blacklist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxBlacklistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
6. If *cap*.MaxWhitelistedItems > 0:
   6.1. ONVIF Client adds credential identifiers to whitelist by following the procedure mentioned in Annex A.28 with the following input and output parameters
      - in *credentialIdentifiersList* - credential identifiers to be added to whitelist
7. ONVIF client invokes **AddToBlacklist** with parameters
   - Identifier list := *credentialIdentifiersList*

8. The DUT responds with **AddToBlacklistResponse** message.
9. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *updatedBlackList* - complete blacklist
10. If *updatedBlackList* does not contain all items from *initialBlackList*, FAIL the test, restore the DUT state, and skip other steps.
11. If *updatedBlackList* does not contain all items from *credentialIdentifiersList*, FAIL the test, restore the DUT state, and skip other steps.
12. If *cap*.MaxWhitelistedItems > 0:
    12.1. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
    - out *updatedWhiteList* - complete whitelist

    12.2. If *updatedWhiteList* contains any item from *credentialIdentifiersList* item, FAIL the test, restore the DUT state, and skip other steps.
13. ONVIF client invokes **AddToBlacklist** with parameters
    - Identifier list := *credentialIdentifiersList*
14. The DUT responds with **AddToBlacklistResponse** message.
15. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
    - out *updatedBlackList2* - complete blacklist
16. If *updatedBlackList2* contains more items that *updatedBlackList*, FAIL the test, restore the DUT state, and skip other steps.
17. Remove all items from *credentialIdentifiersList* from blacklist.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.11.4 REMOVE IDENTIFIER FROM BLACKLIST

Test Case ID: CREDENTIAL-11-1-4

Specification Coverage: RemoveFromBlacklist command (ONVIF Credential Service Specification)

Feature Under Test: RemoveFromBlacklist

WSDL Reference: credential.wsdl

Test Purpose: To verify removing identifier to blacklist.

Pre-Requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in blacklist.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *initialBlackList* - complete blacklist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxBlacklistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list

6. ONVIF Client adds credential identifiers to blacklist by following the procedure mentioned in Annex A.26 with the following input and output parameters
   - in *credentialIdentifiersList* - credential identifiers to be added to blacklist
7. ONVIF client invokes **RemoveFromBlacklist** with parameters
   - Identifier list := *credentialIdentifiersList*
8. The DUT responds with **RemoveFromBlacklistResponse** message.
9. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *updatedBlackList* - complete blacklist
10. If *updatedBlackList* does not contain all items from *initialBlackList*, FAIL the test, restore the DUT state, and skip other steps.
11. If *updatedBlackList* contains all items from *credentialIdentifiersList*, FAIL the test, restore the DUT state, and skip other steps.
12. ONVIF client invokes **RemoveFromBlacklist** with parameters
    - Identifier list := *credentialIdentifiersList*
13. The DUT responds with **RemoveFromBlacklistResponse** message.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

4.11.5 DELETE BLACKLIST

Test Case ID: CREDENTIAL-11-1-5

Specification Coverage: DeleteBlacklist command (ONVIF Credential Service Specification)

Feature Under Test: DeleteBlacklist

WSDL Reference: credential.wsdl

Test Purpose: To verify deleting blacklist.

Pre-Requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional items in blacklist.

Test Configuration: ONVIF Client and DUT

Test Sequence:

1. Start an ONVIF Client.
2. Start the DUT.
3. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *initialBlackList* - complete blacklist
4. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
5. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in min{2; *cap*.MaxBlacklistedItems; *cap*.MaxLimit} - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
6. ONVIF Client adds credential identifiers to blacklist by following the procedure mentioned in Annex A.26 with the following input and output parameters
   - in *credentialIdentifiersList* - credential identifiers to be added to blacklist
7. ONVIF client invokes **DeleteBlacklist**.
8. The DUT responds with **DeleteBlacklistResponse** message.
9. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters

- out *updatedBlackList* - complete blacklist

10. If *updatedBlackList* contains any items, FAIL the test, restore the DUT state, and skip other steps.
11. ONVIF client invokes **DeleteBlacklist**.
12. The DUT responds with **DeleteBlacklistResponse** message.
13. Client restores all items from *initialBlackList*.

Test Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

Annex A Helper Procedures and Additional Notes

A.1 Get credentials information list

Name: HelperGetCredentialInfoList

Procedure Purpose: Helper procedure to get complete credentials information list.

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

Input: None.

Returns: The complete list of credentials information (*credentialInfoCompleteList*).

Procedure:

1. ONVIF client invokes **GetCredentialInfoList** with parameters
   - Limit skipped
   - StartReference skipped
2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - CredentialInfo list =: *credentialInfoCompleteList*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetCredentialInfoList** with parameters
      - Limit skipped
      - StartReference := *nextStartReference*
   3.2. The DUT responds with **GetCredentialInfoListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - CredentialInfo list =: *credentialInfoListPart*
   3.3. Set the following:
      - *credentialInfoCompleteList* := *credentialInfoCompleteList* + *credentialInfoListPart*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.2 Get service capabilities

Name: HelperGetServiceCapabilities

Procedure Purpose: Helper procedure to get service capabilities.

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

Input: None

Returns: The service capabilities (*cap*).

Procedure:

1. ONVIF Client invokes **GetServiceCapabilities**.
2. The DUT responds with a **GetServiceCapabilitiesResponse** message with parameters
   - Capabilities =: *cap*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.3 Get credentials list

Name: HelperGetCredentialList

Procedure Purpose: Helper procedure to get complete credentials list with.

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

Input: None.

Returns: The complete list of credentials (*credentialCompleteList*).

Procedure:

1. ONVIF client invokes **GetCredentialList** with parameters
   - Limit skipped
   - StartReference skipped
2. The DUT responds with **GetCredentialListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Credential list =: *credentialCompleteList*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetCredentialList** with parameters
      - Limit skipped
      - StartReference := *nextStartReference*
   3.2. The DUT responds with **GetCredentialListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - Credential list =: *credentialListPart*
   3.3. Set the following:
      - *credentialCompleteList* := *credentialCompleteList* + *credentialListPart*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.4 Change credential state

Name: HelperChangeCredentialState

Procedure Purpose: Helper procedure to change credential state.

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

Input: Credential token (*credentialToken*), credential state (*credentialState*).

Returns: None.

Procedure:

1. If *credentialState*.Enabled equal to *true*, do the following steps:
   1.1. ONVIF client invokes **DisableCredential** with parameters
      - Token := *credentialToken*
      - Reason := "Test Reason"
   1.2. The DUT responds with empty **DisableCredentialResponse** message.
2. If *credentialState*.Enabled is equal to *false*, perform the following steps:
   2.1. ONVIF client invokes **EnableCredential** with parameters
      - Token := *credentialToken*
      - Reason := "Test Reason"
   2.2. The DUT responds with empty **EnableCredentialResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The DUT did not send **EnableCredentialResponse** message.
- The DUT did not send **DisableCredentialResponse** message.

A.5 Get access profiles list

Name: HelperGetAccessProfilesList

Procedure Purpose: Helper procedure to get complete access profiles list with.

Pre-requisite: Access Rules Service is received from the DUT.

Input: None.

Returns: The complete list of access profiles (*accessProfileCompleteList*).

Procedure:

1. ONVIF client invokes **GetAccessProfileList** with parameters
   - Limit skipped
   - StartReference skipped
2. The DUT responds with **GetAccessProfileListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - AccessProfile list =: *accessProfileCompleteList*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetAccessProfileList** with parameters
      - Limit skipped
      - StartReference := *nextStartReference*
   3.2. The DUT responds with **GetAccessProfileListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - AccessProfile list =: *accessProfileListPart*
   3.3. Set the following:
      - *accessProfileCompleteList* := *accessProfileCompleteList* + *accessProfileListPart*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.6 Delete credential

Name: HelperDeleteCredential

Procedure Purpose: Helper procedure to delete credential.

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

Input: Credential Token (*credentialToken*).

Returns: None.

Procedure:

1. ONVIF Client invokes **DeleteCredential** request with parameters
   - Token =: *credentialToken*
2. The DUT responds with **DeleteCredentialResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.7 Free storage for additional credential

Name: HelperCheckFreeStorageForCredential

Procedure Purpose: Helper procedure to provide possibility to add a credential.

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

Input: The complete list of credentials (*credentialCompleteList*).

Returns: Removed credential (*credentialToRestore*) and its state (*stateToRestore*) if any.

Procedure:

1. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
2. ONVIF client compares *cap*.MaxCredentials with number of items at *credentialCompleteList*.
3. If number of items of *credentialCompleteList* less than cap.MaxCredential, skip other steps.
4. If number of items at *credentialCompleteList* equal to *cap*.MaxCredentials, execute the following steps:

   4.1. ONVIF client invokes **GetCredentials** with parameters
   - Token list := *credentialCompleteList*[0].token

4.2. The DUT responds with **GetCredentialsResponse** message with parameters

- Credential list =: *credentialToRestore*

4.3. ONVIF client invokes **GetCredentialState** with parameters

- Token[0] := *credentialCompleteList*[0].token

4.4. The DUT responds with **GetCredentialStateResponse** message with parameters

- State =: *stateToRestore*

4.5. ONVIF Client deletes the Credential (in *credentialCompleteList*[0].token) by following the procedure mentioned in Annex A.6.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

- The number of items at *accessProfileCompleteList* more than *cap*.MaxAccessProfiles.
- The DUT did not send **GetCredentialsResponse** message.
- The DUT did not send **GetCredentialStateResponse** message.

A.8 Get credential

Name: HelperGetCredential

Procedure Purpose: Helper procedure to get credential.

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

Input: Credential Token (*credentialToken*).

Returns: Credential List (*credentialList*).

Procedure:

1. ONVIF Client invokes **GetCredentials** with parameters
   - Token[0] := *credentialToken*
2. The DUT responds with **GetCredentialsResponse** message with parameters:

- Credential list =: *credentialList*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.9 Get credential info

Name: HelperGetCredentialInfo

Procedure Purpose: Helper procedure to get credential info.

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

Input: Credential Token (*credentialToken*).

Returns: Credential Info List (*credentialInfoList*).

Procedure:

1. ONVIF Client invokes **GetCredentialInfo** with parameters
   - Token[0] := *credentialToken*
2. The DUT sends the **GetCredentialInfoResponse** message with parameters
   - CredentialInfo =: *credentialInfoList*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.10 Restore credential

Name: HelperRestoreCredential

Procedure Purpose: Helper procedure to restore credential.

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

Input: Credential (*credentialToRestore*) and its state (*stateToRestore*).

Returns: None.

Procedure:

1. Set:
   - *credentialToRestore*.token := "";
2. ONVIF client invokes **CreateCredential** with parameters
   - Credential := *credentialToRestore*
   - State := *stateToRestore*
3. The DUT responds with **CreateCredentialResponse** message with parameters
   - Token =: *credentialToken*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.11 Create credential

Name: HelperCreateCredential

Procedure Purpose: Helper procedure to create credential.

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

Input: Credential Service capabilities (*cap*) (optional input parameter, could be skipped), Antipassback Violation State (*AntipassbackViolated*).

Returns: Credential Token (*credentialToken*), Credential identifier Type Name (*typeName*), Credential identifier Format Type (*formatType*), Credential identifier value (*value*).

Procedure:

1. If cap is skipped, ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
2. ONVIF Client retrieves (in *cap*.SupportedIdentifierType) a supported Credential identifier type name (out *typeName*) with Credential identifier Format Type (out *formatType*) and with credential identifier value (out *value*) by following the procedure mentioned in Annex A.15.
3. ONVIF client invokes **CreateCredential** with parameters
   - Credential.token := ""
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped
   - Credential.CredentialIdentifier[0].Type.Name := *typeName*
   - Credential.CredentialIdentifier[0].Type.FormatType := *formatType*
   - Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
   - Credential.CredentialIdentifier[0].Value := *value*
   - Credential.CredentialAccessProfile skipped
   - Credential.Extension skipped
   - State.Enabled := true
   - State.Reason := "Test Reason"
   - State.AntipassbackState.AntipassbackViolated := *AntipassbackViolated* if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped
4. The DUT responds with **CreateCredentialResponse** message with parameters
   - Token =: *credentialToken*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.12 Get access profiles information list

Name: HelperGetAccessProfileInfoList

Procedure Purpose: Helper procedure to get complete access profiles information list.

Pre-requisite: Access Rules Service is received from the DUT.

Input: None.

Returns: The complete list of access profiles information (*accessProfileInfoCompleteList*).

Procedure:

1. ONVIF client invokes **GetAccessProfileInfoList** with parameters
   - Limit skipped
   - StartReference skipped
2. The DUT responds with **GetAccessProfileInfoListResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - AccessProfileInfo list =: *accessProfileInfoCompleteList*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetAccessProfileInfoList** with parameters
      - Limit skipped
      - StartReference := *nextStartReference*
   3.2. The DUT responds with **GetAccessProfileInfoListResponse** message with parameters
      - NextStartReference =: *nextStartReference*
      - AccessProfileInfo list =: *accessProfileInfoListPart*
   3.3. Set the following:

- *accessProfileInfoCompleteList* := *accessProfileInfoCompleteList* + *accessProfileInfoListPart*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.13 Get credential state

Name: HelperGetCredentialState

Procedure Purpose: Helper procedure to get credential state.

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

Input: Credential Token (*credentialToken*).

Returns: Credential State (*credentialState*).

Procedure:

1. ONVIF Client invokes **GetCredentialState** with parameters
   - Token[0] := *credentialToken*
2. The DUT responds with **GetCredentialStateResponse** message with parameters
   - State =: *credentialState*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.14 Supported credential identifier format types

The list of supported credential identifier format types is the following:

- WIEGAND26
- WIEGAND37
- WIEGAND37_FACILITY
- FACILITY16_CARD32
- FACILITY32_CARD32
- FASC_N
- FASC_N_BCD
- FASC_N_LARGE
- FASC_N_LARGE_BCD
- GSA75
- GUID
- CHUID
- USER_PASSWORD
- SIMPLE_NUMBER16
- SIMPLE_NUMBER32
- SIMPLE_NUMBER56
- SIMPLE_ALPHA_NUMERIC
- ABA_TRACK2

A.15 Get Credential Identifier type and value

Name: HelperGetCredentialIdentifierTypeAndValue

Procedure Purpose: Helper procedure to get one Credential identifier Type Name with one corresponding FormatType and corresponding Value.

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

Input: List of supported Identifies types (*identifierTypeList*).

Returns: Credential identifier Type Name (*typeName*) with corresponding Credential identifier Format Type (*formatType*) and Credential identifier value (*value*).

Procedure:

1. For each IdentifierType (*typeName*) contained in *identifierTypeList* repeat the following steps:
   1.1. ONVIF client invokes **GetSupportedFormatTypes** with parameters
      - CredentialIdentifierTypeName := *typeName*
   1.2. The DUT responds with **GetSupportedFormatTypesResponse** message with parameters
      - FormatTypeInfo list =: *formatTypeInfoList*
   1.3. If *formatTypeInfoList* is empty, FAIL the test and skip other steps.
   1.4. For each FormatType (*formatType*) from *formatTypeInfoList* repeat the following steps:
      1.4.1. If *formatType* is listed in Annex A.14 go to step 2.
2. ONVIF Client generate appropriate value (*value*) for *formatType* and skip other steps.
3. ONVIF Client gets values from the Management tab for *typeName*, *formatType* and *value*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: If values for *typeName*, *formatType* or *value* were empty on the Management tab at step 3, FAIL the test.

A.16 Get Credential Identifier type name list with at least two Format Type

Name: HelperGetCredentialIdentifierName

Procedure Purpose: Helper procedure to get list of Credential identifier Type Name items which support at least two Format Types.

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

Input: None.

Returns: *CredentialIdentifierTypeNameList*

Procedure:

1. ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
2. For each SupportedIdentifierType (*typeName*) contained in *cap*.SupportedIdentifierType repeat the following steps:
   2.1. ONVIF client invokes **GetSupportedFormatTypes** with parameters
      - CredentialIdentifierTypeName := *typeName*
   2.2. The DUT responds with **GetSupportedFormatTypesResponse** message with parameters
      - FormatTypeInfo list =: *formatTypeInfoList*
   2.3. If *formatTypeInfoList* is empty, FAIL the test and skip other steps.
   2.4. If *formatTypeInfoList* contains at least two FormatType items do the following:
      2.4.1. *CredentialIdentifierTypeNameList* := *CredentialIdentifierTypeNameList* + *typeName*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.17 Get Credential Identifier type and value for Type Name with at least two Format Type

Name: HelperGetCredentialIdentifierTypeAndValue2

Procedure Purpose: Helper procedure to get one Credential identifier Type Name with two corresponding FormatType and corresponding Values.

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

Input: *CredentialIdentifierTypeNameList*.

Returns: Credential identifier Type Name (*typeName*) with corresponding Credential identifier Format Type (*formatType1*) and Credential identifier value (*value1*), and with corresponding Credential identifier Format Type (*formatType2*) and Credential identifier value (*value2*).

Procedure:

1. For each IdentifierTypeName (*typeName*) contained in *CredentialIdentifierTypeNameList* repeat the following steps:

   1.1. ONVIF client invokes **GetSupportedFormatTypes** with parameters

   - CredentialIdentifierTypeName := *typeName*

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

   - FormatTypeInfo list =: *formatTypeInfoList*

   1.3. If *formatTypeInfoList* is empty, FAIL the test and skip other steps.

   1.4. If at least two FormatTypes (*formatType1*, *formatType2*) from *formatTypeInfoList* are listed in Annex A.14 go to step 2.

2. ONVIF Client generates an appropriate value (*value1*) for *formatType1* and appropriate value (*value2*) for *formatType2* and skip other steps.
3. ONVIF Client gets *typeName* with at least two Format Type items from the Management tab.
4. ONVIF Client gets appropriate values (*formatType1*, *value1*) and (*formatType2*, *value2*) from the Management tab for *typeName*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

Note: If there was no *typeName* with at least two Format Type items and corresponding values for FormatType on the Management tab, FAIL the test.

A.18 Create credential with two Credential identifier items

Name: HelperCreateCredential2

Procedure Purpose: Helper procedure to create credential with two Credential identifier items.

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

Input: Credential Service capabilities (*cap*) (optional input parameter, could be skipped), Antipassback Violation State (*AntipassbackViolated*).

Returns: Credential Token (*credentialToken*), the first Credential identifier Type Name (*typeName1*) with corresponding Credential identifier Format Type (*formatType1*) and corresponding Credential identifier value (*value1*), the second Credential identifier Type Name (*typeName2*) with corresponding Credential identifier Format Type (*formatType2*) and corresponding Credential identifier value (*value2*).

Procedure:

1. If *cap* is skipped, ONVIF Client gets the service capabilities (out *cap*) by following the procedure mentioned in Annex A.2.
2. ONVIF Client retrieves (in *cap*.SupportedIdentifierType) a supported Credential identifier type name (out *typeName1*) with Credential identifier Format Type (out *formatType1*) and with credential identifier value (out *value1*) by following the procedure mentioned in Annex A.15.
3. Set the following:
   - *identifierTypeList* := *cap*.SupportedIdentifierType - *typeName1*
4. ONVIF Client retrieves (in *identifierTypeList*) other supported Credential identifier type name (out *typeName2*) with Credential identifier Format Type (out *formatType2*) and with credential identifier value (out *value2*) by following the procedure mentioned in Annex A.15.
5. ONVIF client invokes **CreateCredential** with parameters
   - Credential.token := ""
   - Credential.Description := "Test Description"
   - Credential.CredentialHolderReference := "TestUser"
   - Credential.ValidFrom skipped
   - Credential.ValidTo skipped

- Credential.CredentialIdentifier[0].Type.Name := *typeName1*
- Credential.CredentialIdentifier[0].Type.FormatType := *formatType1*
- Credential.CredentialIdentifier[0].ExemptedFromAuthentication := false
- Credential.CredentialIdentifier[0].Value := *value1*
- Credential.CredentialIdentifier[1].Type.Name := *typeName2*
- Credential.CredentialIdentifier[1].Type.FormatType := *formatType2*
- Credential.CredentialIdentifier[1].ExemptedFromAuthentication := false
- Credential.CredentialIdentifier[1].Value := *value2*
- Credential.CredentialAccessProfile skipped
- Credential.Extension skipped
- State.Enabled := true
- State.Reason := "Test Reason"
- State.AntipassbackState.AntipassbackViolated := *AntipassbackViolated* if *cap*.ResetAntipassbackSupported value is equal to true, otherwise State.AntipassbackState is skipped

6. The DUT responds with **CreateCredentialResponse** message with parameters
   - Token =: *credentialToken*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.19 Create access profile

Name: HelperCreateAccessProfile

Procedure Purpose: Helper procedure to Create access profile.

Pre-requisite: Access Rules Service is received from the DUT.

Input: None.

Returns: Access Profile Token (*accessProfileToken*).

Procedure:

1. ONVIF Client invokes **CreateAccessProfile** with parameters
   - AccessProfile.token := ""
   - AccessProfile.Name := "Test Access Profile"
   - AccessProfile.Description := "Test Description"
   - AccessProfile.AccessPolicy skipped
   - AccessProfile.Extension skipped
2. The DUT responds with **CreateAccessProfileResponse** message with parameters
   - Token =: *accessProfileToken*

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.20 Delete access profile

Name: HelperDeleteAccessProfile

Procedure Purpose: Helper procedure to delete access profile.

Pre-requisite: Access Rules Service is received from the DUT.

Input: Access Profile Token (*accessProfileToken*).

Returns: None.

Procedure:

1. ONVIF Client invokes **DeleteAccessProfile** with parameters

- Token =: *accessProfileToken*

2. The DUT sends the **DeleteAccessProfileResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.21 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. Device supports Pull-Point Notification feature.

Input: Notification Topic (*topic*).

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

Procedure:

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

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.22 Delete Subscription

Name: HelperDeleteSubscription

Procedure Purpose: Helper procedure to delete supscribtion.

Pre-requisite: Event Service is received from the DUT. Device supports Pull-Point Notification feature.

Input: Subscription reference (*s*)

Returns: None

Procedure:

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

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.23 Retrieve Credential Changed Event by PullPoint

Name: HelperPullCredentialChanged

Procedure Purpose: Helper procedure to retrieve and check tns1:Configuration/Credential/Changed event with PullMessages.

Pre-requisite: Event Service is received from the DUT. Device supports Pull-Point Notification feature.

Input: Subscription reference (*s*), current time for the DUT (*ct*), Subscription termination time (*tt*) and Credential token (*credentialToken*).

Returns: None

Procedure:

1. Until *operationDelay* timeout expires, repeat the following steps:
   1.1. ONVIF Client waits for time *t* := min{(*tt*-*ct*)/2, 1 second}.
   1.2. ONVIF Client invokes **PullMessages** to the subscription endpoint *s* request with parameters
      - Timeout := PT60S
      - MessageLimit := 1
   1.3. The DUT responds with **PullMessagesResponse** message with parameters
      - CurrentTime =: *ct*
      - TerminationTime =: *tt*
      - NotificationMessage list =: *notificationMessageList*
   1.4. If *notificationMessageList* is not empty and the CredentialToken source simple item in *notificationMessageList* is equal to *credentialToken*, skip other steps and finish the procedure.
2. If *operationDelay* timeout expires for step 1 without Notification with Token source simple item equal to *credentialToken*, FAIL the test, restore the DUT state, and skip other steps.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

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

A.24 Add Number of Credential Identifiers to Whitelist

Name: HelperAddCredentialIdentifiersToWhitelist

Procedure Purpose: Helper procedure to add number of credential identifiers required for test cases to whitelist.

Pre-requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability.

Input: None.

Returns: List of added credential identifiers (*credentialIdentifiersList*). The service capabilities (*cap*). The complete whitelist (*completeWhitelist*).

Procedure:

1. ONVIF Client retrieves a complete whitelist by following the procedure mentioned in Annex A.25 with the following input and output parameters
   - out *initialWhiteList* - complete whitelist
2. Set *completeWhitelist* := *initialWhiteList*.
3. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
4. Set *requiredNumberOfItems* := min {50; *cap*.MaxLimit; *cap*.MaxWhitelistedItems}.
5. If *requiredNumberOfItems* <= number Identifier items in *initialWhiteList*, skip other steps of the procedure.
6. Set *numberOfIdentefiersToBeAdded* := *requiredNumberOfItems* - number of Identifier items in *initialWhiteList*.
7. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialWhiteList* - initial list of credential identifiers (to prevent creation of duplications)
   - in *numberOfIdentefiersToBeAdded* - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
8. ONVIF client invokes **AddToWhitelist** with parameters
   - Identifier list := *credentialIdentifiersList*
9. The DUT responds with **AddToWhitelistResponse** message.
10. Set *completeWhitelist* := *completeWhitelist* + *credentialIdentifiersList*.

Procedure Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

A.25 Get Whitelist

Name: HelperGetWhitelist

Procedure Purpose: Helper procedure to get complete whitelist.

Pre-requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability.

Input: None.

Returns: The complete whitelist (*completeWhitelist*).

Procedure:

1. ONVIF client invokes **GetWhitelist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped
2. The DUT responds with **GetWhitelistResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Identifier list =: *completeWhitelist*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetWhitelist** with parameters
      - Limit is skipped

- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

3.2. The DUT responds with **GetWhitelistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

3.3. Set *completeWhitelist* := *completeWhitelist* + *identifierPart*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.26 Add Identifiers to Blacklist

Name: HelperAddToBlacklist

Procedure Purpose: Helper procedure to add identifier to blacklist.

Pre-requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional item in blacklist.

Input: Identifier list to be added (*identifierList*).

Returns: None.

Procedure:

1. ONVIF client invokes **AddToBlacklist** with parameters
   - Identifier list := *identifierList*
2. The DUT responds with **AddToBlacklistResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.27 Get Blacklist

Name: HelperGetBlacklist

Procedure Purpose: Helper procedure to get complete blacklist.

Pre-requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability.

Input: None.

Returns: The complete blacklist (*completeBlacklist*).

Procedure:

1. ONVIF client invokes **GetBlacklist** with parameters
   - Limit is skipped
   - StartReference is skipped
   - IdentifierType is skipped
   - FormatType is skipped
   - Value is skipped
2. The DUT responds with **GetBlacklistResponse** message with parameters
   - NextStartReference =: *nextStartReference*
   - Identifier list =: *completeBlacklist*
3. Until *nextStartReference* is not null, repeat the following steps:
   3.1. ONVIF client invokes **GetBlacklist** with parameters
   - Limit is skipped

- StartReference := *nextStartReference*
- IdentifierType is skipped
- FormatType is skipped
- Value is skipped

3.2. The DUT responds with **GetBlacklistResponse** message with parameters

- NextStartReference =: *nextStartReference*
- Identifier list =: *identifierPart*

3.3. Set *completeBlacklist* := *completeBlacklist* + *identifierPart*.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.28 Add Identifiers to Whitelist

Name: HelperAddToWhitelist

Procedure Purpose: Helper procedure to add identifier to whitelist.

Pre-requisite: Credential Service is received from the DUT. Whitelist is supported by the DUT as indicated by MaxWhitelistedItems greater than zero capability. The DUT shall have enough free storage capacity for two additional item in whitelist.

Input: Identifier list to be added (*identifierList*).

Returns: None.

Procedure:

1. ONVIF client invokes **AddToWhitelist** with parameters
   - Identifier list := *identifierList*
2. The DUT responds with **AddToWhitelistResponse** message.

Procedure Result:

PASS –

- DUT passes all assertions.

FAIL –

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

A.29 Add Number of Credential Identifiers to Blacklist

Name: HelperAddCredentialIdentifiersToBlacklist

Procedure Purpose: Helper procedure to add number of credential identifiers required for test cases to blacklist.

Pre-requisite: Credential Service is received from the DUT. Blacklist is supported by the DUT as indicated by MaxBlacklistedItems greater than zero capability.

Input: None.

Returns: List of added credential identifiers (*credentialIdentifiersList*). The service capabilities (*cap*). The complete blacklist (*completeBlacklist*).

Procedure:

1. ONVIF Client retrieves a complete blacklist by following the procedure mentioned in Annex A.27 with the following input and output parameters
   - out *initialBlackList* - complete blacklist
2. Set *completeBlacklist* := *initialBlackList*.
3. ONVIF Client gets the service capabilities by following the procedure mentioned in Annex A.2 with the following input and output parameters
   - out *cap* - Credential Service capabilities
4. Set *requiredNumberOfItems* := min {50; *cap*.MaxLimit; *cap*.MaxBlacklistedItems}.
5. If *requiredNumberOfItems* <= number Identifier items in *initialBlackList*, skip other steps of the procedure.
6. Set *numberOfIdentefiersToBeAdded* := *requiredNumberOfItems* - number of Identifier items in *initialBlackList*.

7. ONVIF Client generates list of credential identifiers by following the procedure mentioned in Annex A.30 with the following input and output parameters
   - in *cap* - Credential Service capabilities
   - in *initialBlackList* - initial list of credential identifiers (to prevent creation of duplications)
   - in *numberOfIdentefiersToBeAdded* - requiered number of credential identifiers
   - out *credentialIdentifiersList* - credential identifiers list
8. ONVIF client invokes **AddToBlacklist** with parameters
   - Identifier list := *credentialIdentifiersList*
9. The DUT responds with **AddToBlacklistResponse** message.
10. Set *completeBlacklist* := *completeBlacklist* + *credentialIdentifiersList*.

Procedure Result:

PASS –

- The DUT passed all assertions.

FAIL –

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

A.30 Generate Number of Credential Identifiers

Name: HelperGenerateNumberOfCredentialIdentifiers

Procedure Purpose: Helper procedure to generate number of credential identifiers.

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

Input: Initial list of credential identifiers (*credentialIdentifiersInitialList*). The service capabilities (*cap*). Requiered number of credential identifiers (*requiredNumberOfNewItems*).

Returns: List of credential identifiers (*credentialIdentifiersList*).

Procedure:

1. For each IdentifierType (*typeName*) contained in *cap*.SupportedIdentifierType repeat the following steps:
   1.1. ONVIF client invokes **GetSupportedFormatTypes** with parameters

- CredentialIdentifierTypeName := *typeName*

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

- FormatTypeInfo list =: *formatTypeInfoList*

1.3. If *formatTypeInfoList* is empty, FAIL the test and skip other steps.

1.4. For each FormatType (*formatType*) from *formatTypeInfoList* repeat the following steps:

1.4.1. If *formatType* is listed in Annex A.14 go to step 6.

2. Set *identifier1* :=

- Type.Name := type name from the Management tab for first value
- Type.FormatType := format type from the Management tab for first value
- Value := first value from management tab

3. Set *identifier2* :=

- Type.Name := type name from the Management tab for second value
- Type.FormatType := format type from the Management tab for second value
- Value := second value from management tab

4. Set *credentialIdentifiersList* := list with *identifier1* (if it is not listed in *credentialIdentifiersInitialList*) and *identifier2* (if it is not listed in *credentialIdentifiersInitialList*).

5. Skip other steps of the procedure and return to the test.

6. Set *identifier* :=

- Type.Name := *typeName*
- Type.FormatType := *formatType*
- Value := appropriate random value for *formatType* which is not contains in *credentialIdentifiersList*

7. Set *credentialIdentifiersList* := *credentialIdentifiersList* with added *identifier*.

8. Set *requiredNumberOfNewItems* := *requiredNumberOfNewItems* - 1.

9. If *requiredNumberOfNewItems* = 0, then skip other steps of the procedure and return to the test else go to step 6.

Procedure Result:

PASS –

- The DUT passed all assertions.

FAIL –

- None.

Note: If values are not specified at Management tab and DUT does not support any format type listed in Annex A.14, fail the test.