Universal Serial Bus Device Class Definition for Audio Devices

USB Command Verifier Compliance Test Specification

Revision 1.10

Date: **August 6, 2019**

Revision: **1.10**

Revision History

Sr. No.	Revision	Author	Date	Change Log
1	0.1	Sandip	Feb 5, 2018	Descriptor Tests are defined for UAC 3.0 CTS spec
2	1.0	Abhijeet	Feb 7, 2018	Descriptor Tests are defined for UAC 3.0 CTS spec
3	1.01	Soren	June 24, 2019	Removed steps 21-24 from TD 4.61 (they duplicate TD 9.21 in USB 2 Chapter 9). Consolidated TD 4.3 and TD 4.28. Now, only the BADD specific requirements are tested in TD 4.28: any requirements that are identical for both BADD and non-BADD AIAs are tested in TD 4.3. For all tests, specified whether they are to be run for BADD AIAs, full Audio 3.0 AIAs or both.

Table of Contents

1. Scope

This document specifies test cases and procedures to verify assertions defined in the documents listed below.

Audio Devices Class 3.0, USB Command Verifier Compliance Test Specification, Revision 0.1

Audio Device Formats 3.0, USB Command Verifier Compliance Test Specification, Revision 0.1

Audio Device Terminal Types 3.0, USB Command Verifier Compliance Test Specification, Revision 0.1

Basic Audio Function Device Class 3.0, USB Command Verifier Compliance Test Specification, Revision 0.1

1.1. Limitations

As mentioned in the specifications, the Test procedures defined in this specification doesn't cover below:

No validation is provided for streaming data (Audio data obtained/sent through isochronous or bulk pipes) So payload headers and the data itself are not validated. The main reasons for this are the lack of support for isochronous pipes in the USBCV framework and the fact that a lot of the expected behavior is subjective.

No support for verification that auto-update controls send control change interrupts. The reason being that it is impossible to really know for sure when a device has changed an auto-update setting. This testing is intended to be in addition to standard USB Compliance testing; assertions covered by the USBCV test document, for instance, are not covered here.

This testing applies to one configuration and all Audio Interface Association (AIA) at a time. This testing covers the validation of the audio class-specific descriptors (including the possible dependencies between them) and of the class-specific control requests (except those explicitly excluded above)

1.2. Terminology

Device Under Test (DUT): Refers to the USB Audio 3.0 Device on which the tests are being run.

USB Implementers Forum (USB-IF): The non-profit organization that owns the trademark for USB and which is responsible for the USB Compliance program.

USBCV Tool: USB Command Verifier (USBCV) is the tool for USB Hub and Device Framework testing. All USB 2.x and USB 3.x peripherals are required to pass the Device Framework tests in order to gain certification.

xHCI: Extensible Host Controller Interface. The USBCV Audio 3.0 Class tests must be run on an xHCI host.

Interface Association Descriptor (IAD): A USB descriptor that groups together one or more USB Interface Descriptors in order to describe a single USB Function.

Audio Interface Association (AIA): A USB Function that implements the USB Audio 3.0 specification. All AIAs on a USB device are associated with an **IAD**.

> *Note: This requirement places an additional restriction to the USB base specification, which permits Functions consisting of a single interface to omit the IAD.*

USB Audio Class 3.0 Function (Audio 3.0 or UAC 3.0 Function): A USB Function that is compliant to the USB Audio Class 3.0 specification. Audio 3.0 Functions available in a given Configuration are specified in the full Configuration Descriptor for that Configuration, with the Class (the IAD field bFunctionClass) is set to AUDIO (0x01) and the Protocol (the IAD field bFunctionProtocol) is set to AF_VERSION_03_00 (0x30).

Legacy USB Audio Class Function (Legacy Audio Function): A USB Function that is compliant to an earlier version of the USB Audio Class specification. Legacy Audio Functions available in a given Configuration are specified in the full Configuration Descriptor for that Configuration, with the Class (the Interface Association Descriptor field bFunctionClass or the Interface Descriptor field bInterfaceClass) set to AUDIO (0x01) and the Protocol (the IAD field bFunctionProtocol or the Interface Descriptor field bInterfaceProtocol) set to either AF_VERSION_01_00 (0x00) or AF_VERSION_02_00 (0x20).

Basic Audio Device Definition (BADD) USB Audio Class 3.0 Function (BADD Function): An **Audio 3.0 Function** wherein *bFunctionSubClass* has a value in the range 0x20 - 0x26. A BADD Function implements a strictly defined subset of the USB Audio Class 3.0 specification.

Full USB Audio Class 3.0 Function (Full Audio 3.0): An Audio 3.0 Function wherein *bFunctionSubClass* = 0x01. A Full Audio 3.0 Function may implement any set of features allowed by the full USB Audio Class 3.0 specification.

Audio Streaming Interface: a USB Interface that supports USB Audio Streaming. An Audio Streaming Interface is identified by a USB Interface Descriptor with the following characteristics:

- **USB Audio 1.0 Class**: bInterfaceClass = 0x01, *bInterfaceSubClass* = 0x02, and bInterfaceProtocol = 0x00.
- **USB Audio 2.0 Class**: bInterfaceClass = 0x01, *bInterfaceSubClass* = 0x02, and bInterfaceProtocol = 0x20
- **USB Audio 3.0 Class**: bInterfaceClass = 0x01, *bInterfaceSubClass* = 0x02, and bInterfaceProtocol = 0x30

2. Test philosophy

2.1. Implementation

The test descriptions specified by this document are integrated into the USBCV tool available from the USB-IF. It is intended that all devices that report an audio class interface will be required to pass this test in order to receive logo certification.

2.2. Multiple speeds.

Unless otherwise specified, tests shall be run at all speeds that on the DUT in which USB Audio Class 3.0 is supported.

2.3. Multiple AIAs

Unless otherwise specified, tests shall be run on all USB Audio Class 3.0 AIAs on the DUT that contain a UAC 3.0 AIA.

Unless otherwise specified, tests will be implemented to run on a single Configuration at a time.

Unless otherwise specified, if a Configuration contains multiple UAC 3.0 AIAs, then tests will be implemented to loop over all of them.

2.4. USB Device state

Unless otherwise specified, tests shall be run with the DUT in the Configured state.

2.5. UAC 3 Compliance Levels

If a test is labeled as ***BADD***, then only run that test on Audio 3.0 Functions that conform to a BADD profile (*bFunctionSubtype* is in the range 0x20-0x26).

If a test is labeled as ***Full***, then only run that test on Audio 3.0 Functions that do not implement a BADD profile (*bFunctionSubtype* is set to 0x01).

If a test is labeled as ***All***, then that test should be run on all Audio 3.0 Functions.

If *bFunctionSubtype* is not valid (is neither ***BADD*** or ***Full***):

If a test is labeled as either ***BADD*** or ***Full*** then fail the test and skip all remaining steps.

3. Test Descriptions for Audio Devices 3.0 Descriptors

3.1. Common Procedures

Many tests perform similar tasks, in particular during test setup. Here we provide common definitions for some of them.

3.1.1. Retrieve the Device Descriptor

Unless otherwise specified, record a failure and skip all remaining steps in the test and record a failure if any step fails for any reason.

1. Send a GetDescriptor request to the DUT with the following values:

- *wValue*: set to 0x01 (Device) in the high byte, 0 in the low byte
- *wIndex*: set to 0
- *wLength*: 18d (The entire Device Descriptor)

3.1.2. Retrieve the Full Configuration Descriptor

Unless otherwise specified, record a failure and skip all remaining steps in the test and record a failure if any step fails for any reason.

1. Send a GetDescriptor (*wValue*, *wIndex*, *wLength*, Data) request with the following values:
    -
    - ***wValue*:** set to 0x02 (Configuration) in the high byte, the index of the current configuration in the low byte
    - ***wIndex*:** set to 0
    - ***wLength*:** 9d
2. Send a GetDescriptor (*wValue*, *wIndex*, *wLength*, Data) request with the following values:
    -
    - ***wValue*:** set to 0x02 (Configuration) in the high byte, the index of the current configuration in the low byte
    - ***wIndex*:** set to 0
    - ***wLength*:** the *wTotalLength* field of the Configuration Descriptor obtained in the previous step

3.1.3. Retrieve the Full BOS Descriptor

Unless otherwise specified, record a failure and skip all remaining steps in the test and record a failure if any step fails for any reason.

1. Send a GetDescriptor (*wValue*, *wIndex*, *wLength*, Data) request with the following values:

-
- ***wValue*:** set to 0x0F (BOS) in the high byte, 0 in the low byte
- ***wIndex*:** set to 0
- ***wLength*:** 5d

2. Send a GetDescriptor (*wValue*, *wIndex*, *wLength*, Data) request with the following values:

-
- ***wValue*:** set to 0x0F (BOS) in the high byte, 0 in the low byte
- ***wIndex*:** set to 0
- ***wLength*:** the *wTotalLength* field of the BOS Descriptor obtained in the previous step.

3.1.4. Find all USB Audio Class 3.0 Functions

Unless otherwise specified, record a failure and skip all remaining steps in the test and record a failure if any step fails for any reason.

1. Retrieve the full Configuration Descriptor for the configuration being tested.

Parse the Configuration Descriptor into USB Functions, saving Audio 3.0 Functions for future use.

3.1.5. Initialize the Test

Unless otherwise specified, record a failure and skip all remaining steps in the test and record a failure if any step fails for any reason.

1. **Retrieve the device descriptor**.
2. **Find all USB Audio Class 3.0 Functions** for the configuration being tested.
3. Unless otherwise specified, if the configuration being tested does not contain a UAC 3.0 Function, then exit the test.

3.1.6. Determining BADD Generic I/O Profile Subtype:

A **BADD Function** that supports the Generic I/O Profile (*bFunctionSubtype* == 0x20) may take one of three subtypes:

- Generic Out
- Generic In
- Generic I/O

To determine which it is, follow the following steps.

1. If the function has 2 streaming interfaces, then it is *Generic I/O*.

2. If the Function has 1 streaming interface, find the first Isochronous endpoint in that interface.
    - If *bmAttributes* bit D7 is set to 1, then it is *Generic Out*.
    - If *bmAttributes* bit D7 is set to 0, then it is *Generic In*.
3. If the Function does not have a streaming interface, or if it has more than 2, then fail the test, and skip the remaining steps.
4. If the Function does not have an isochronous Endpoint, then fail the test and skip the remaining steps.

3.2. Compliance Tests

3.2.1. Device Descriptor Test (All)

Assertions Verified in this Test

4.1#1, 4.1#4

Test Procedure

This test encompasses all Audio 3.0 Functions in all Configurations. Therefore, it does not need to be repeated for each Configuration.

1. **Initialize the test**.
2. Retrieve the full Configuration Descriptors for all Configurations the DUT supports and save them for later use.

    Note: this includes Configurations that do not have an Audio 3.0 Function.
3. Determine whether the DUT has a Configuration with at least one **Audio Streaming Interface**. Save this result for further use.
4. Verify that bDeviceClass == 0xEF, bDeviceSubClass == 0x02 and bDeviceProtocol == 0x01.
5. Verify that *bcdUSB* is greater than 0x201.
6. Verify that if the DUT has an Audio Streaming Interface, then bNumConfigs is greater than 0.

3.2.2. Standard Interface Association Descriptor Test (All)

Assertions verified in this test

3.4.1#1, 3.4.2#1, 3.4.3#2, 3.14#1

Test Procedure

1. **Initialize the test**.
2. For each Audio 3.0 Function, verify the following:
    a. The first descriptor in the Function is an IAD.
    b. *bFunctionSubClass* is set to one of the values {0x00, 0x01, 0x20-0x26}

3.2.3. Standard Interface Descriptor Test (All)

Assertions Verified in this Test

3.4#1, 3.4.3#1, 3.5#1, 3.6#1, 3.14.1#1, 3.14.1#6, 3.14.2#1, 3.14.2#3, 4.1#5, 4.1#6, 4.1#7, 4.1#8, 4.1#9, 4.5.1#1 to 4.5.1#6, 4.6.1.1#1, 4.7.1#1, 4.7.1#2

Test Procedure

1. **Initialize the test**.
2. Parse each Audio 3.0 Function, saving the IAD and all Interface Descriptors for further use.
3. For each Interface Descriptor, verify the following:
    a. bInterfaceClass = AUDIO (0x01)
    b. bInterfaceProtocol = bFunctionProtocol of the IAD.
    c. The first Interface Descriptor of the Function is AUDIOCONTROL (*bInterfaceSubClass* = AUDIOCONTROL (0x01)).
    d. No other Interface Descriptor in the Function is AUDIOCONTROL.
    e. The AUDIOCONTROL Descriptor has the following values:
        - *bNumEndPoints* shall be either 0x00 or 0x01.
        - bInterfaceProtocol shall be 0x30.
    f. AUDIOSTREAMING Interface Descriptors (*bInterfaceSubClass* = AUDIOSTREAMING (0x02)) (if any) come directly after the AUDIOCONTROL Interface Descriptor and before any other Audio 3.0 Interface Descriptors.
    g. AUDIOSTREAMING Interface Descriptors have more than one Alternate Setting (bAlternateSetting).
    h. AUDIOSTREAMING Interface Descriptors in which bAlternateSetting = 0 have no Endpoints (*bNumEndpoints* == 0).
    i. AUDIOSTREAMING Interface Descriptors shall have the following values:
        - bInterfaceProtocol shall be 0x30.
        - *bNumEndpoints* shall be one of 0x00, 0x01, or 0x02.
        - bInterfaceProtocol shall be 0x30.

j. MIDISTREAMING Interface Descriptors (bInterfaceSubclass = MIDISTREAMING (0x03)) (if any) shall come directly after the AUDIOSTREAMING Interface Descriptor (if any). If there are no AUDIOSTREAMING Interface Descriptors, then all MIDISTREAMING Interface Descriptors shall come directly after the AUDIOCONTROL Interface Descriptor.

3.2.4. Standard Endpoint Descriptor Test (Full)

Assertions Verified in this Test

3.11#1, 3.14.2#2, 4.6.2.1#2, 4.6.2.1#3, 4.6.2.1#4, 4.6.2.1#5, 4.8.1#1, 4.8.1#2, 4.8.1#3, 4.8.2#1, 4.8.2.1#2, 4.8.2.1#3

Test Procedure

1. **Initialize the test**.
2. Examine the AUDIOCONTROL Interface Descriptor (*bInterfaceSubClass* == AUDIOCONTROL(0x01)). If *bNumEndpoints* == 0, then skip step 3.
3. Parse the Endpoint Descriptor for the AUDIOCONTROL Interface.
4. Verify that D7 of *bEndpointAddress* is set to 1 (IN)
5. Verify that D1..0 of *bmAttributes* is set to 11b (Interrupt)
6. Verify that D7..2 of *bmAttributes* is set to 0.
7. If the Function does not contain an AUDIOSTREAMING Interface Descriptor (*bInterfaceSubClass* == AUDIOCONTROL(0x02)), then skip the remaining steps.
8. Examine each AUDIOSTREAMING Interface Descriptor. If *bNumEndpoints* = 0, then skip the remaining steps and continue to the next AUDIOSTREAMING Interface (if any).
9. For the first Endpoint Descriptor in an AUDIOSTREAMING Interface, verify the following:
10. D1..0 of *bmAttributes* are set to 01b (Isochronous).
11. D3..2 of *bmAttributes* is not set to 00b (No Synchronization
12. D5..4 of *bmAttributes* is set to one of 00b (Data endpoint) or 10b (Implicit feedback Data endpoint).
13. If the AUDIOSTREAMING Interface has a second Endpoint, verify the following:
14. D1..0 of *bmAttributes* are set to 01b (Isochronous).
15. D3..2 of *bmAttributes* is set to 00b (No Synchronization
16. D5..4 of *bmAttributes* is set to 01b (Feedback endpoint).

3.2.5. Configuration Summary Descriptor Test (All)

Assertions Verified in this Test

4.1#2, 4.1#3

Test Procedure

This test encompasses all Audio 3.0 Functions in all Configurations. Therefore, it does not need to be repeated for each Configuration.

1. **Initialize the test**, with the following modification:
2. **Retrieve the full Configuration Descriptors** for all Configurations that have an Audio 3.0 function.
3. If Device Descriptor field *bcdUSB* is less than 0x201, then skip the remaining steps.
4. **Retrieve the full BOS Descriptor**.
5. Parse the full BOS descriptor, saving all Configuration Summary Descriptors (*bDescriptorType* = 0x10, *bDevCapabilityType* = 0x10) for future use.
6. If the DUT has only one Configuration and there are no Configuration Summary Descriptors then skip all remaining steps.
7. For each Audio 3.0 Function in each Configuration, verify that there is a Configuration Summary Descriptor wherein:
   - *bClass* == *bFunctionClass*
   - *bSubClass* == *bFunctionSubClass*
   - *bProtocol* == *bFunctionProtocol*

3.2.6. Class Specific Audio Control Interface Descriptor Test (Full)

Assertions Verified in this Test

3.8#1, 4.2.1#2, 4.2.1#3, 4.5.2#2, 4.5.2#3, 4.5.2#4, 4.5.2#5, 4.5.2#6, 4.5.2#7, 4.2.1.1.1#1 to 4.2.1.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* == CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found fail the test.
6. For the class-specific audio control interface descriptor verify:

a. *bLength* is 10

b. bCategory is in the range of 0x01-0x10

c. Bit D7 of the *bDescriptorType* field of a traditional class-specific descriptor is Reserved and shall be set to 0

d. Bits D6..5 of the *bDescriptorType* field of a traditional class-specific descriptor must be 0b01 to indicate class specific descriptor

e. bTotalLength field of the class-specific AC interface descriptor contains the total length in bytes of the entire descriptor, including header, CDs, UDs, TDs and PDD

f. Bits 2-31 of the *bmControls* field of the class-specific AC Interface Header Descriptor are reserved and shall be set to 0

g. Bits 1..0 of the *bmControls* should be either 01b(Read only) or 00b(control is not present)

7. further parse the class specific audio control interface and check that there exists a descriptor with *bDescriptorType* = CS_INTERFACE and *bDescriptorSubType* = INPUT_TERMINAL/OUTPUT_TERMINAL/EXTENDED_TERMINAL which are present in audio function
8. further parse the class specific audio control interface and check that there exists a descriptor with *bDescriptorType* = CS_INTERFACE and *bDescriptorSubType* = MIXER_UNIT/SELECTOR_UNIT/FEATURE_UNIT/EFFECT_UNIT/PROCESSING_UNIT/EXTENSION_UNIT which are present in audio function
9. The ID value 0x00 is reserved and must not be used in the ID (bClockID, *bUnitId*, bTermanlID, bPowerDomainID) of an entity included in the class-specific AC Interface Descriptor.
10. The IDs (bClockID, *bUnitId*, *bTerminalID*, bPowerDomainID) of all Entities included in the class-specific AC Interface Descriptor must be unique
11. Test fails if any of the values are not as specified.

3.2.7. High Capability Descriptor Test (Full)

Assertions Verified in this Test

4.2#1, 4.2.2#1, 4.2.2#2, 4.2.2#4, 4.2.2#5, 4.2.2#6, 4.2#2

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with:
3. *bDescriptorType* =CS_INTERFACE and

4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. For the class-specific audio control interface descriptor check that
7. Further parse the class specific audio control interface and verify that there is no descriptor with *bDescriptorType*=CS_INTERFACE and *bDescriptorSubType*=EXTENDED_TERMINAL/CONNECTORS and also verify that there is no descriptor with *bDescriptorType* = CS_CLUSTER.
8. further parse the class specific audio control interface and check that there exists a descriptor with *bDescriptorType* = CS_INTERFACE and *bDescriptorSubType* = INPUT_TERMINAL/MIXER_UNIT/PROCESSING_UNIT etc. which are present in audio function and find the *wClusterDescrID*/ *wExTerminalDescrID*/ *wConnectorsDescrID*
9. Test exits if any of the values are not as specified.
10. Now issue a HIGH CAPABILITY REQUEST with following data & *wLength* = 2.
11. Fail if there is no ACK response or data retrieved is less than 2 Bytes
12. Parse the response and find the total length of the descriptor
13. Now issue a HIGH CAPABILITY REQUEST with following data & *wLength* =total length of the descriptor found in step 4.
14. Parse the response and verify the following
15. *wLength* should be less than 64K bytes
16. Bit 7 of *wDescriptorType* should be 0b0
17. Bit 6,5 of *wDescriptorType* should be 0b01
18. *wDescriptorId* of high capability descriptor is unique
19. Test fails if any of the values are not as specified.
20. Now issue a HIGH CAPABILITY REQUEST with following wrong interface/*wDescriptorID*
21. Fail if there is no STALL response.

3.2.8. Cluster Descriptor Test (Full)

Assertions Verified in this Test

4.2.2#3, 4.3#2, 4.3#4, 4.3#5, 4.3.2#1, 4.3.2#2, 4.3.2#3, 4.3.2#4, 4.3.2#5, 4.3.2#6, 4.3.2.1#1, 4.3.2.1.1#1, 4.3.2.1.1#2, 4.3.2.1.2#1, 4.3.2.1.2#3, 4.3.2.1.3.1#1, 4.3.2.1.3.1#2, 4.3.2.1.3.1#3, 4.3.2.1.3.1#5, 4.3.2.1.3.2#1, 4.3.2.1.3.2#2, 4.3.2.1.3.2#3, 4.3.2.1.3.2#4, 4.3.2.1.3.2#5, 4.3.2.1.3.2#6, 4.3.2.1.3.2#8, 4.3.2.1.3.3#1, 4.3.2.1.3.4#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found skip 10th step.
6. For the class-specific audio control interface descriptor
7. Further parse the class specific audio control interface and check that there exists a descriptor with
8. *bDescriptorType* = CS_INTERFACE and
9. *bDescriptorSubType* = INPUT_TERMINAL/MIXER_UNIT/PROCESSING_UNIT/EXTENSION_UNIT for logical cluster and/or
10. Find the *wClusterDescrID*
11. Parse & Read audio Streaming Descriptor with *bDescriptorType* == INTERFACE & *bInterfaceSubClass* == AUDIO_STREAMING.
12. Parse Descriptor *bDescriptorType* = CS_INTERFACE and *bDescriptorSubType* = AS_GENERAL for physical cluster which are present in audio function.
13. Find the *wClusterDescrID*.
14. Now issue a request with:
    - ***bmRequestType*:** 10100001B
    - ***bRequest*:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*:** *wClusterDescrID*
    - ***wIndex*:** 0 in higher byte & Interface number in lower byte
    - ***wLength*:** 2
15. Fail if there is no ACK response or retrieved data is less than 2 bytes.
16. Parse the response and find the total length of the descriptor
17. Now issue a request with:
    - ***bmRequestType*:** 10100001B
    - ***bRequest*:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*:** *wClusterDescrID*
    - ***wIndex*:** 0 in higher byte & Interface number in lower byte
    - ***wLength*:** total length of the descriptor found in step 16.

18. Fail if there is no ACK response.
19. Parse the response and verify the following
20. *wLength* should be less than 64K bytes
21. *bDescriptorSubType* of the Cluster descriptor must be SUBTYPE_UNDEFINED (0x00)
22. bDescriptorID of the Cluster descriptor is the Unique ID
23. bNrChannels field is 0 then *wLength* value must be equal to 7 and skip the following
24. first *bSegmentType* is equal to either CLUSTER_DESCRIPTION(0x01) or CLUSTER_VENDOR_DEFINED(0x1F), then
25. Following *bSegmentType* in descriptor after this will never be CLUSTER_DESCRIPTION(0x01) or CLUSTER_VENDOR_DEFINED(0x1F)
26. Next *bSegmentType* must be equal to CHANNEL_INFORMATION(0x20) or CHANNEL_AMBISONIC(0x21)
27. First *bSegmentType* is not equal to CLUSTER_DESCRIPTION(0x01) or CLUSTER_VENDOR_DEFINED(0x1F) then first *bSegmentType* must be equal to CHANNEL_INFORMATION (0x20), CHANNEL_AMBISONIC (0x21), CHANNEL_DESCRIPTION (0x22) or CHANNEL_VENDOR_DEFINED (0xfe)
28. In each block there should be at least a segment with *bSegmentType* CHANNEL_INFORMATION (0x20) or CHANNEL_AMBISONIC (0x21), but not both.
29. Count number of END_SEGMENTS, it should be equal to bNrchannels field value or value + 1
30. *bSegmentType* field value is equal to CLUSTER_DESCRIPTION(0x01), then *wLength* of that segment must be equal to 5
31. *bSegmentType* field value is equal to CLUSTER_VENDOR_DEFINED(0x1F) then *wLength* of that segment must be greater than 3.
32. there should be bNrChannels segments with *bSegmentType* = END_SEGMENT(0xFF)
33. *bSegmentType* field value is equal to END_SEGMENT(0xFF) then *wLength* of that segment must be equal to 3
34. Verify that the same segment order (segment layout) is used for each channel.
35. If *bSegmentType* field value is equal to CHANNEL_INFORMATION (0x20) then
36. bchPurpose field value is other than specified values fail it.
37. wChRelationship is equal to one of Channel Relationships (Table 4-8)
38. *bchGroupID* is equal among all the related channels.
39. *wLength* value is 6.

40. For every *bSegmentType* field value which is equal to CHANNEL_AMBISONIC(0x21) check the following
41. bCompOrdering value (which should be one of the values specified in Appendix A.13, “Ambisonic Component Ordering Convention Types.”) and make sure in every segment it has same value
42. *bACN* contains Ambisonic Channel Number and should be less than bNrchannels field value.
43. *bAmbNorm* value (which should be one of the values in {NORM_TYPE_UNDEFINED (0x00), maxN (0x01), SN3D (0x02), N3D (0x03), SN2D (0x04), N2D (0x05)}) and make sure in every segment it has same value
44. *bchGroupID* is equal among all the related channels
45. *wLength* value is 7
46. Test fails if any of the values are not as specified.

3.2.9. Input Terminal Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.1#1, 4.5.2.1#2, 4.5.2.1#3, 4.5.2.1#5, 4.5.2.1#6, 4.5.2.1#7, 4.5.2.1#8, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Input Terminal Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == INPUT_TERMINAL
9. If INPUT_TERMINAL not found, skip the remaining steps and throw the related assertion
10. For all Input terminal descriptors check that
11. *bLength* is 20.
12. *bTerminalID* field of an Input Terminal Descriptor must be unique.

13. *bAssocTerminal* field of an Input Terminal Descriptor shall be set to 0 if there is no association with a terminal pair.
14. *wClusterDescrID* field of an Input Terminal Descriptor contains the unique ID of the Cluster descriptor
15. *wExTerminalDescrID* field of an Input Terminal Descriptor contains the unique ID of the Extended Terminal descriptor
16. *wConectorsDescrID* field of an Input Terminal Descriptor contains the unique ID of the Connectors descriptor
17. Bits 8-31 of the *bmControls* field of an Input Terminal are reserved and shall be set to 0
18. In *bmControls*
    a. Bits 1..0 should be 01b (Read Only) or 00b (Control Not Present).
    b. Bits 3..2 should be 01b (Read Only) or 00b (Control Not Present).
    c. Bits 5..4 should be 01b (Read Only) or 00b (Control Not Present).
    d. Bits 7..6 should be 01b (Read Only) or 00b (Control Not Present).
19. Test fails if any of the values are not as specified.

3.2.10. Output Terminal Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.2#1 to 4.5.2.2#5, 4.5.2.2#5, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and *bDescriptorSubType* == HEADER. If HEADER not found, fail the test.
4. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all output Terminal Descriptors (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == OUTPUT_TERMINAL. If OUTPUT_TERMINAL not found, skip the remaining steps and throw the related assertion.
5. For all Output terminal descriptors check that
6. *bLength* is 19.
7. *bTerminalID* field of an Output Terminal Descriptor must be unique.

8. *bAssocTerminal* field of an Output Terminal Descriptor shall be set to 0 if there is no association with a terminal pair.
9. *wExTerminalDescrID* field of an Input Terminal Descriptor contains the unique ID of the Extended Terminal descriptor
10. *wConectorsDescrID* field of an Input Terminal Descriptor contains the unique ID of the Connectors descriptor
11. Bits 8-31 of the *bmControls* field of an Input Terminal are reserved and shall be set to 0
12. In *bmControls*:
    a. Bits 1..0 should be 01b (Read Only) or 00b (Control Not Present).
    b. Bits 3..2 should be 01b (Read Only) or 00b (Control Not Present).
    c. Bits 5..4 should be 01b (Read Only) or 00b (Control Not Present).
    d. Bits 7..6 should be 01b (Read Only) or 00b (Control Not Present).
13. Test fails if any of the values are not as specified.

3.2.11. Extended Terminal Descriptor Test (Full)

Assertions Verified in this Test

4.2.2#3, 4.5.2.3#1 to 4.5.2.3#4, 4.5.2.3.2, 4.5.2.3.3.1#1, 4.5.2.3.3.1#2, 4.5.2.3.3.2.1#1, 4.5.2.3.3.3.1#1, 4.5.2.3.3.3.2#1, 4.5.2.3.3.3.3#1, 4.5.2.3.3.3.4#1, 4.5.2.3.3.3.5#1, 4.5.2.3.3.3.6#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
    a. *bDescriptorType* == CS_INTERFACE
    b. *bDescriptorSubType* == HEADER
3. If HEADER not found, fail the test.
4. For the class-specific audio control interface descriptor
5. Further parse the class specific audio control interface and check that there exist a descriptor with
6. *bDescriptorType* = CS_INTERFACE and
7. *bDescriptorSubType* == INPUT_TERMINAL/MIXER_UNIT/PROCESSING_UNIT/EXTENSION_UNIT for Extended Terminal

8. Find the *wExTerminalDescrID*
9. Now issue a request with:
    - ***bmRequestType*****:** 10100001B
    - ***bRequest*****:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*****:** *wExTerminalDescrID* and
    - ***wIndex*****:** 0 in higher byte & Interface number in lower byte
    - ***wLength*****:** 2
10. Fail if there is no ACK response or retrieved data is less than 2 bytes.
11. Parse the response and find the total length of the descriptor
12. Now issue a request with:
    - ***bmRequestType*****:** 10100001B
    - ***bRequest*****:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*****:** *wExTerminalDescrID*
    - ***wIndex*****:** 0 in higher byte & Interface number in lower byte
    - ***wLength*****:** total length of the descriptor found in step 13.
13. Fail if there is no ACK response.
14. Check that *wLength* is total length of the extended terminal Descriptor
15. For all Extended terminal descriptors check that
16. *wDescriptorID* field of an Extended Terminal Descriptor Header shall have contained the Unique ID
17. If the first *bSegmentType* is equal to TERMINAL_VENDOR_DEFINED (1F), then this value must not come till the end of this descriptor.
18. Verify that in each channel "*bSegmentType*" of each segment is following same order. If not notify the user about mismatch of segments ordering in channel block
19. If the segment with *bSegmentType* is equal to TERMINAL_VENDOR_DEFINED (1F), then its *wLength* must be greater than or equal to '3'.
20. Check that *bSegmentType* must be END_SEGMENT (0xFF) before ending of a block.
21. *wLength* field of an End Segment shall be set to 3
22. If the segment with *bSegmentType* CHANNEL_BANDWIDTH(0x20), then its *wLength* must be equal to 11
23. If the segment with *bSegmentType* CHANNEL_MAGNITUDE_RESPONSE (0x21), then its *wLength* must be equal to 3+n*6. Where n is the bNrChannels field value.
24. If the segment with *bSegmentType* CHANNEL_MAGNITUDE/PHASE_RESPONSE (0x22), then its *wLength* must be equal to 3+n*8. Where n is the bNrChannels field value.

25. If the segment with *bSegmentType* is CHANNEL_POSITION_XYZ(0x23) or CHANNEL_POSITION_RΘΦ(0x24), then its *wLength* must be equal to 15.
26. If the segment with *bSegmentType* is CHANNEL_VENDOR_DEFINED(0XFE), then its *wLength* must be equal to greater than '3'
27. Test fails if any of the values are not as specified.

3.2.12. Connector Descriptor Test (Full)

Assertions Verified in this Test

4.2.2#3, 4.5.2.4#2 to 4.5.2.4#9, 4.5.2.4#10, 4.5.2.4#11, 4.5.2.4#12, 4.5.2.4#13, 4.5.2.4#14

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. For the class-specific audio control interface descriptor
7. Further parse the class specific audio control interface and check that there exist a descriptor with
8. *bDescriptorType* = CS_INTERFACE and
9. *bDescriptorSubType* = INPUT_TERMINAL/MIXER_UNIT/PROCESSING_UNIT/EXTENSION_UNIT for

   Connector Descriptor ID
10. Find the *wConnectorsDescrID*
11. If all elements of an array of *wConnectorsDescrID*'s are 0, skip the test.
12. Now issue a request with:
    - ***bmRequestType*:** 10100001B
    - ***bRequest*:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*:** *wConnectorsDescrID* and
    - ***wIndex*:** 0 in higher byte & Interface number in lower byte
    - ***wLength*:** 2
13. Fail if there is no ACK response or retrieved data is less than 2 bytes.
14. Parse the response and find the total length of the descriptor

15. Now issue a request with:
    - ***bmRequestType*:** 10100001B
    - ***bRequest*:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*:** *wConnectorsDescrID*
    - ***wIndex*:** 0 in higher byte & Interface number in lower byte
    - ***wLength*:** total length of the descriptor found in step 11.
16. Fail if there is no ACK response
17. For all Connectors descriptors check that
18. *wDescriptorID* field of a Connectors descriptor is unique and shall not be set to 0.
19. The *wLength* should be equal to (7+*bNrConnectors**11).
20. for *bNrConnectors* check that the following fields exists.
21. The field *baConID(i)* of the Connectors descriptor contains a unique identifier for Connector(i)
22. The field *waClusterDescrID(i)* of the Connectors descriptor contains the unique ID of a physical cluster descriptor
23. The field *baConType(i)* of the Connectors descriptor contains must be in the range 0x000x23 or 0xFF.
24. The value of 11b is reserved and shall not be used for bits 0-1 of the *bmaConAttributes(i)* field in the Connectors Descriptor
25. If the value is 00b for bits 0-1 of the *bmaConAttributes(i)* field in the Connectors Descriptor, then inform user that the connecter is gender neutral
26. If the value is 01b for bits 0-1 of the *bmaConAttributes(i)* field in the Connectors Descriptor, then inform user that the connecter is male.
27. If the value is 10b for bits 0-1 of the *bmaConAttributes(i)* field in the Connectors Descriptor, then inform user that the connecter is female.
28. The field *daConColor(i)* in the Connectors Descriptor shall contain either 0x00 or 0x01 in the upper byte
29. Test fails if any of the values are not as specified.

3.2.13. Mixer Unit Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.5#1, 4.5.2.5#2, 4.5.2.5#3, 4.5.2.5#5, 4.5.2.5#6, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Mixer Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == MIXER_UNIT.
9. If MIXER_UNIT not found, skip the remaining steps and throw the related assertion
10. For all Mixer Unit descriptors check that
11. *bUnitId* has unique value
12. Verify that *bLength* of this descriptor is equal to 13 + p + n. where p is the bNrInPins field and if ((((n *m)) % 8) < 0 || (((n *m)) % 8) > 0),then N=((n*m) DIV 8) + 1,else N=((n*m) DIV 8).
13. Verify each field after “bNrInPins” that “baSourceID(i) “represents the unit or terminal ID
14. *wClusterDescrID* field contains the unique ID of cluster descriptor
15. In *bmControls*
16. Bits 1..0 should be 01b Read Only or 00b control not present.
17. Bits 3..2 should be 01b Read only or 00b control not present.
18. Bits 4-31 of the *bmControls* field are reserved and shall be set to 0
19. Test fails if any of the values are not as specified.

3.2.14. Selector Unit Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.6#1, 4.5.2.6#2, 4.5.2.6#4, 4.5.2.6#5, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with

3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Selector Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == SELECTOR_UNIT.
9. If SELECTOR_UNIT not found, skip the remaining steps and throw the related assertion
10. For all Feature Unit descriptors check that
11. bUnidID has unique value
12. Verify each field after “bNrInPins” that “baSourceID(i) “represents the unit or terminal ID
13. Verify the *bLength* is equal to 11+p where p is bNrInpins
14. In *bmControls*
15. Bits 1..0 should be either 01b Read Only or 11b Read-Write.
16. Bits 2-31 of the *bmControls* field are reserved and shall be set to 0
17. Test fails if any of the values are not as specified.

3.2.15. Feature Unit Descriptor Test (Full)

Assertions Verified in this Test

3.13.6#2, 4.5.2#1, 4.5.2.7#1, 4.5.2.7#2, 4.5.2.7#3, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Feature Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == FEATURE_UNIT.

9. If FEATURE_UNIT not found, skip the remaining steps and throw the related assertion
10. For all Feature Unit descriptors check that
11. The *bLength* shall be set to 7+(ch+1) *4.
12. The *bUnitId* should be unique.
13. In *bmControls*:
    a. bits 1..0 should be 11b (Read-Write) or 00b control not present.
    b. Bits 3..2 should be 11b (Read-Write) or 00b control not present.
    c. Bits 5..4 should be 11b (Read-Write) or 00b control not present.
    d. Bits 7..6 should be 11b (Read-Write) or 00b control not present.
    e. Bits 9..8 should be 11b (Read-Write) or 00b control not present.
    f. Bits 11..10 should be 11b (Read-Write) or 00b control not present.
    g. Bits 13..12 should be 11b (Read-Write) or 00b control not present.
    h. Bits 15..14 should be 11b (Read-Write) or 00b control not present.
    i. Bits 17..16 should be 11b (Read-Write) or 00b control not present.
    j. Bits 19..18 should be 11b (Read-Write) or 00b control not present.
    k. Bits 21..20 should be 11b (Read-Write) or 00b control not present.
    l. Bits 23..22 should be 11b (Read-Write) or 00b control not present.
    m. Bits 25..24 should be 11b (Read-Write) or 00b control not present.
    n. Bits 27..26 should be 01b (Read Only) or 00b control not present.
    o. Bits 29..28 should be 01b (Read Only) or 00b control not present.
    p. Bits 30-31 of bmControl fields in the Feature Unit descriptor are reserved and must be set to 0.
14. Test fails if any of the values are not as specified.

3.2.16. Sampling Rate Converter Unit Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.8#1, 4.5.2.8#2, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with

3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Sampling Rate Converter Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == SAMPLE_RATE_CONVERTER_UNIT
9. If SAMPLE_RATE_CONVERTER_UNIT not found, skip the remaining steps and throw the related assertion
10. For all Sampling Rate Converter Unit descriptors check that
11. *bUnitId* field value is unique ID.
12. blength must be is 9
13. Test fails if any of the values are not as specified.

3.2.17. Effect Unit Descriptor Test (Full)

Assertions Verified in this Test

3.13.8#1, 4.5.2#1, 4.5.2.9#1, 4.5.2.9#2, 4.5.2.9#3, 4.5.2.9.1#2, 4.5.2.9.1#3, 4.5.2.9.2#1, 4.5.2.9.2#2, 4.5.2.9.3#1, 4.5.2.9.3#2, 4.5.2.9.4#1, 4.5.2.9.4#2, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER.
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Effect Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == EFFECT_UNIT
9. If EFFECT_UNIT not found, skip the remaining steps and throw the related assertion

10. For all Effect Unit descriptors check that
11. *bUnitId* field value is unique ID.
12. bEffectType field of the Effect Unit descriptor must contain a value in the range of 0x0001 to 0x0004
13. If bEffectType is PARAM_EQ_SECTION_EFFECT (0x0001) then
14. *bLength* field shall be set to 17+(ch*4)
15. In *bmControls* (for all channels present in PARAM_EQ_SECTION_EFFECT)
16. Bits 1..0 should be 11b (Read-write) or 00b control not present.
17. Bits 3..2 should be 11b (Read-Write) or 00b control not present.
18. Bits 5..4 should be 11b (Read-Write) or 00b control not present.
19. Bits 7..6 should be 01b (Read Only) or 00b control not present.
20. Bits 9..8 should be 01b (Read Only) or 00b control not present.
21. Bits 10-31 of the bmaControls are reserved and shall be set to 0.
22. If bEffectType is REVERBERATION_EFFECT (0x0002) then
23. *bLength* field shall be set to 17+(ch*4)
24. In *bmControls* (for all channels present in REVERBERATION_EFFECT)
25. Bits 1..0 should be 11b (Read-Write) or 00b control not present.
26. Bits 3..2 should be 11b (Read-Write) or 00b control not present.
27. Bits 5..4 should be 11b (Read-Write) or 00b control not present.
28. Bits 7..6 should be 11b (Read-Write) or 00b control not present.
29. Bits 9..8 should be 11b (Read-Write) or 00b control not present.
30. Bits 11..10 should be 11b (Read-Write) or 00b control not present.
31. Bits 13..12 should be 11b (Read-Write) or 00b control not present.
32. Bits 15..14 should be 01b (Read Only) or 00b (Control not present)
33. Bits 17..16 should be 01b (Read Only) or 00b (Control not present)
34. Bits 10-31 of the bmaControls are reserved and shall be set to 0.
35. If bEffectType is MOD_DELAY_EFFECT(0x0003) then
36. *wLength* field shall be set to 17+(ch*4)
37. In *bmControls* (for all channels present in MOD_DELAY_EFFECT)
38. Bits 1..0 should be 11b (Read-Write) or 00b control not present.

39. Bits 3..2 should be 11b (Read-Write) or 00b control not present.
40. Bits 5..4 should be 11b (Read-Write) or 00b control not present.
41. Bits 7..6 should be 11b (Read-Write) or 00b control not present.
42. Bits 9..8 should be 11b (Read-Write) or 00b control not present.
43. Bits 11..10 should be 11b (Read-Write) or 00b control not present.
44. Bits 13..12 should be 01b (Read Only) or 00b control not present.
45. Bits 15..14 should be 01b (Read Only) or 00b control not present.
46. Bits 16-31 of the bmaControls fields must be 0.
47. If bEffectType DYN_RANGE_COMP_EFFECT(0x004) then
48. *bLength* field shall be set to 17+(ch*4)
49. In *bmControls* (for all channels present in MOD_DELAY_EFFECT)
    a. Bits 1..0 should be 11b (Read-Write) or 00b control not present.
    b. Bits 3..2 should be 11b (Read-Write) or 00b control not present.
    c. Bits 5..4 should be 11b (Read-Write) or 00b control not present.
    d. Bits 7..6 should be 11b (Read-Write) or 00b control not present.
    e. Bits 9..8 should be 11b (Read-Write) or 00b control not present.
    f. Bits 11..10 should be 11b (Read-Write) or 00b control not present.
    g. Bits 13..12 should be 01b (Read Only) or 00b control not present.
    h. Bits 15..14 should be 01b (Read Only) or 00b control not present.
    i. Bits 16-31 of the *bmControls* fields must be 0.
50. Test fails if any of the values are not as specified.

3.2.18. Processing Unit Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.10#1, 4.5.2.10#2, 4.5.2.10#3, 4.5.2.10#5, 4.5.2.10.1#1, 4.5.2.10.1#2, 4.5.2.10.1#3, 4.5.2.10.1#4, 4.5.2.10.1#5, 4.5.2.10.2#1, 4.5.2.10.2#3, 4.5.2.10.2#4, 4.5.2.10.3#1 to 4.5.2.10.3#6, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with

3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER.
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Processing Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == PROCESSING_UNIT(0x09)
9. If PROCESSING_UNIT not found, skip the test and throw the related assertion
10. For all Processing Unit descriptors check that
11. *bUnitId* field value is unique ID.
12. wProcessType field shall be within the range of x0000-0x004.Note: range is from 0x0000 0x0003
13. check that a unique baSourceID (I) exist for bNrInPins
14. If wProcessType is UP/DOWNMIX_PROCESS (0x001) then
15. bNrLnPins must be 1
16. bNrModes value must be equal to last index of array waClusterDescrId ()
17. *bLength* must be 15+2*m, where m is bNrModes.
18. In *bmControls* Bits 1..0 should be 11b (Read-Write) or 00b control not present.
19. Bits 3..2 should be 01b (Read Only) or 00b control not present.
20. Bits 5..4 should be 01b (Read Only) or 00b control not present.
21. Bits 6-31 of the *bmControls* field must be 0.
22. If wProcessType is STEREO_EXTENDER_POCESS (0x0002) then
23. bNrLnPins must be 1
24. *bLength* field shall be set to 14
25. In *bmControls*
26. Bits 1..0 should be 11b (Read-Write)
27. Bits 3..2 should be 01b (Read Only) or 00b (Control not present)
28. Bits 5..4 should be 01b (Read Only) or 00b (Control not present)
29. Bits 6-31 of the bmaControls are reserved and shall be set to 0.
30. If wProcessType MULTI_FUNCTION_PROCESS (0x003) then

31. *bLength* must be 19 + p, where is p is bNrInPins
32. *wClusterDescrID* field value must not be equal to 0
33. bNrinPins value must be equal to last index of array baSourceID ()
34. *wLength* field shall be set to 17+(ch*4)
35. In *bmControls*
36. Bits 1..0 should be 01b (Read Only) or 00b (Control not present)
37. Bits 3..2 should be 01b (Read Only) or 00b (Control not present)
38. Bits 4-31 of the bmaControls fields must be 0.
39. 6-31 bits of the bmAlgorithms field must be 0.
40. Test fails if any of the values are not as specified.

3.2.19. Extension Unit Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.11#1, 4.5.2.11#3, 4.5.2.11#4, 4.5.2.11#5, 4.5.2.11#6, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Extension Unit Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == EXTENSION_UNIT(0x0A)
9. If EXTENSION_UNIT not found, skip the remaining steps and throw the related assertion
10. For all Extension Unit descriptors check that
11. *bUnitId* field has unique value
12. *bLength* field of the Extension Unit descriptor shall be set to 15+p where p is bNrInPns
13. bnrinPins value must be equal to last index of array baSourceID ()

14. *wClusterDescrID* field value must not be equal to 0
15. In *bmControls*
16. Bits 1..0 should be 01b (Read Only) or 00b (Control not present)
17. Bits 3..2 should be 01b (Read Only) or 00b (Control not present)
18. Bits 4-31 of bmControl fields in the Feature Unit descriptor are reserved and must be set to 0.
19. Test fails if any of the values are not as specified.

3.2.20. Clock Source Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.12#1, 4.5.2.12#3, 4.5.2.12#4, 4.5.2.12#5, 4.5.2.12#6, 4.5.2.12#7, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER.
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Clock Source Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == CLOCK_SOURCE(0x0B)
9. If CLOCK_SOURCE not found, skip the remaining steps and throw the related assertion
10. For all Clock Source descriptors check that
11. *bUnitId* field has unique value
12. If D0 bit is 0 then D1 bit must be 0 in *bmAttributes* field
13. the *bLength* value must be equal to 12.
14. Bits 2-7 of the *bmAttributes* field must be 0
15. Bits 4-31 of bmControl fields in the Feature Unit descriptor are reserved and must be set to 0.
16. In *bmControls*

17. Bits 1..0 should be 01b (Read Only) or 11b (Read-Write) or 00b control not present.
18. Bits 3..2 should be 01b (Read Only) or 00b control not present.
19. in *bmAttributes*, if bit D0 is 0 or D1 is 1, then bReferenceTerminal should be 0
20. Test fails if any of the values are not as specified.

3.2.21. Clock Selector Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.13#1, 4.5.2.13#2, 4.5.2.13#3, 4.5.2.13#4, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER. If HEADER not found, fail the test.
5. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Clock Selector Descriptors
6. *bDescriptorType* == CS_INTERFACE and
7. *bDescriptorSubType* == CLOCK_SELECTOR(0x0C)
8. If CLOCK_SELECTOR not found, skip the remaining steps and throw the related assertion
9. For all Clock Selector descriptors check that
10. *bUnitId* field has unique value
11. the bnrinPins value must be equal to last index of array baSourceID ()
12. *bLength* must be 11+p
13. In *bmControls*
14. Bits 1..0 should be 01b (Read Only) or 00b (Control not present) or 00b control not present.
15. Bits 2-31 of the *bmControls* field must be 0
16. Test fails if any of the values are not as specified.

3.2.22. Clock Multiplier Descriptor Test (Full)

Assertions Verified in this Test

4.5.2#1, 4.5.2.14#1, 4.5.2.15#2, 4.2.1.1.1#1 to 4.2.1.1.1#4, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Clock Multiplier Descriptors
7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == CLOCK_MULTIPLIER(0x0D)
9. If CLOCK_MULTIPLIER not found, skip the remaining steps and throw the related assertion
10. For all Clock multiplier descriptors check that
11. *bUnitId* field has unique value
12. In *bmControls*
13. Bits 1..0 should be 01b (Read Only) or 00b (Control not present)
14. Bits 3..2 should be 01b (Read Only) or 00b (Control not present)
15. Bits 4-31 of the *bmControls* field must be 0
16. Test fails if any of the values are not as specified.

3.2.23. Power Domain Descriptor Test (Full)

Assertions Verified in this Test

3.10#1, 4.5.2#1, 4.5.2.15#1 to 4.5.2.15#5, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio control descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType* == HEADER.
5. If HEADER not found, fail the test.
6. Parse the Descriptors following class-specific audio control descriptor (starting *bLength* after), and stopping *wTotalLength* after. Look for all Power Domain Descriptors

7. *bDescriptorType* == CS_INTERFACE and
8. *bDescriptorSubType* == POWER_DOMAIN(0x10)
9. If POWER_DOMAIN not found, skip the remaining steps and throw the related assertion
10. For all Power Domain descriptors check that
11. *bUnitId* field has unique value
12. Only ID's of Input Terminals, Output Terminals, Effect Units, Processing Units, and Extension Units shall be present in baEntityID ()
13. No ID's should be overlap with other power domain descriptor
14. bNrEntities field value must be equal to last index of array baEntityID ()
15. *bLength* must be 11+p, where p is bNrInPins
16. Test fails if any of the values are not as specified.

3.2.24. Class Specific Audio Streaming Interface Descriptor Test (Full)

Assertions Verified in this Test

3.13.2#1, 3.13.3#1, 3.14.2#4, 3.14.2#5, 3.14.2.1#1, 4.4#4, 4.7.2#1, 4.7.2#2, 4.7.3#1

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio streaming descriptor with
3. *bDescriptorType* =CS_INTERFACE and
4. *bDescriptorSubType*=AS_GENERAL(0x01)
5. If AS_GENERAL not found skip 10th step.
6. For the class-specific audio streaming interface descriptor check that
7. *bLength* is 23
8. Bits 6-31 of the *bmControls* of the Class-Specific AS Interface descriptor are Reserved and shall be set to 0.
9. The *wClusterDescrID* should be unique or 0.
10. Test fails if any of the values are not as specified.
11. Parse the configuration descriptor for class-specific audio streaming descriptor with
12. *bDescriptorType* == CS_INTERFACE and
13. *bDescriptorSubType* == AS_VALID_FREQ_RANGE(0x02)

14. If AS_VALID_FREQ_RANGE not found skip step 7 the test.
15. For the all class-specific AS valid frequency range audio streaming interface descriptor check that *bLength* is 23
16. Test fails if any of the values are not as specified.
17. Parse the endpoint descriptor which is present after this, read the *bEndpointAddress* field in the endpoint descriptor.
18. From the bTerminalLink read the respective Terminal descriptor, then if the wTerminal value is other than 0x101 or 0x100 or 0x1FF then *bNumEndPoints* in audio streaming descriptor should be equal to 0x00.
19. If the step 9 fails or skips, then do steps 11 & 12.
20. If bit D7 is specifies OUT, then bTerminalLink in class specific audio streaming descriptor should be the unique terminal ID of INPUT TERMINAL.
21. If bit D7 is specifies IN, then bTerminalLink in class specific audio streaming descriptor should be the unique terminal ID of OUTPUT TERMINAL.

3.2.25. Class-Specific AS Isochronous Audio Data Endpoint Descriptor Test (Full)

Assertions Verified in this Test

4.8.1.2#2, 4.8.1.2#3, 4.8.1.2#4

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for class-specific audio streaming isochronous data endpoint descriptor with
3. *bDescriptorType* = CS_ENDPOINT(0x25) and
4. *bDescriptorSubType*=EP_GENERAL(0x01)
5. If EP_GENERAL not found skip the test.
6. For the class-specific audio streaming interface descriptor check that
7. *bLength* is 10
8. Bits 6-31 of the *bmControls* are reserved and shall be set to 0.
9. Values 3-255 of the bLockDelayUnits field must not be present.
10. Test fails if any of the values are not as specified.

3.2.26. Class Specific String Descriptors Test (Full)

Assertions Verified in this Test

4.9#4, 4.9#5, 4.9#6, 4.9#7

Test Procedure

1. **Initialize the test**.
2. Parse the configuration descriptor for specific descriptor to get its corresponding wStrDescId
3. Now issue a CONTROL REQUEST with following values
4. Parse the response for class specific string descriptor in Data field. If class specific string descriptor not found, fail the test.
5. Check for following in class specific string descriptor
6. *wLength* field of the Class-Specific String descriptor must be less than 65,529 bytes in length.
7. *bDescriptorType* field of a Class-Specific String descriptor shall be set to CS_STRING (0x23)
8. bDescriptorSubtype field of a Class-Specific String descriptor shall be set to SUBTYPE_UNDEFINED (0x00)
9. bLangID field of the Class-Specific String descriptor shall be in the range 0-125
10. Test fails if any of the values are not as specified.

3.2.27. Backward Compatibility Test (All)

Assertions Verified in this Test

3.3#1, 3.3#2, 3.3#3, 3.3#4

Test Procedure

This test encompasses all Audio 3.0 Functions in all Configurations. Therefore it does not need to be repeated for each Configuration.

1. **Initialize the test**, with the following modifications:
   a. Retrieve the full Configuration Descriptors for all Configurations the DUT supports and save them for later use.

      Note: this includes those that do not have an Audio 3.0 Function.
2. Determine whether the DUT has a Configuration with at least one **Audio Streaming Interface**. Save this result for further use.

3. Test fails if there is no **Legacy Audio Function** defined in the full Configuration Descriptor for Configuration Index 0.
4. Retrieve all **Audio 3.0 Functions** from all Configurations.
5. If the DUT has an **Audio Streaming Interface**, verify the following:
    - The DUT has more than one Configuration.
    - A Configuration other than Configuration index 0 has a **BADD Function**.
6. Verify that no Configuration has more than one **BADD Function**.
7. Verify that if the Device has more than one **BADD Function**, then all of them are the same.
    a. Two **BADD Functions** are the same if the following Descriptors are byte-wise identical:
        - The IAD (other than the bFirstInterface field)
        - All Interface Descriptors (other than the bInterfaceNumber field):
        - Endpoint Descriptors (other than bits 3:0 of the *bEndpointAddress* field)
        - Endpoint Companion Descriptors (if applicable)

3.2.28. BADD Standard Interface Descriptors Test (BADD)

Assertions Verified in this Test

3#1, 3#2, 4.2#3, 4.2.2#1, 6.1#1, 6.1#2, 6.2.2.1#2, 6.2.2.1#3, 6.2.2.1#4, 6.2.2.1#5, 6.2.4#1, 6.2.4.1.1#1, 6.2.4.1.1#2, 6.2.4.1.1#3, 6.2.4.1.1#4, 6.2.4.1.1#5, 6.2.4.1.1#6, 6.2.4.1.1#7, 6.2.4.1.2.1#1, .2.4.1.2.1#2, 6.2.4.1.2.1#3, 6.2.4.1.2.1#4, 6.2.4.1.2.1#5, 6.2.4.1.2.1#6, 4.2.1#1

Test Procedure

1. **Initialize the test**.
2. Test fails if the Device is Low Speed.
3. Test fails unless there is at least one AUDIOSTREAMING Interface Descriptor in the data returned.
4. Test fails if there are any audio-specific descriptors in the Function.
5. For each of the returned interface descriptors verify the following:
    - If *bFunctionSubClass* is 0x25 (HEADSET_ADAPTER)/*bInterfaceSubClass* is AUDIOCONTROL (0x01), then *bNumEndPoints* shall be set to 1
    - If bInterfaceSubclass is not MIDISTREAMING (0x03)

3.2.29. BADD Standard Endpoint Descriptor Test (BADD)

Assertions Verified in this Test

4.2#2, 4.2.3#1, 4.2.3#2, 6.2.3#1, 6.2.3#2, 6.2.3#3, 6.2.3.1#1, 6.2.3.1#2, 6.2.3.1#3, 6.2.3.1#4, 6.2.4.1.2.3#1, 6.2.4.1.2.3#2, 4.2.3#3

Test Procedure

1. **Initialize the test**.
2. For each Function, parse the BADD Function descriptors, saving all Endpoint Descriptors for further use.
3. If *bFunctionSubClass* is 0x25 (HEADSET_ADAPTER), then *bmAttributes* should be equal to 0x03.
4. There should be minimum 2 endpoint descriptor with bit '7' in *bEndpointAddress* one with value 1 and other with value 0.
5. For all standard endpoint descriptors verify the following
6. Bits 3-2 of the *bmAttributes* are same for all data endpoint descriptors.
7. The allowed *bmAttributes* for the data endpoint descriptors are 0x05 or 0x0D.
8. If bit 7 of the *bEndpointAddress* field is 0 and *bmAttributes* is 0x0D, then there should be an endpoint descriptor with *bmAttributes* equal to 0x11.
9. *bmAttributes* is as follows
10. D1..0: Transfer type
11. 01b = Isochronous
12. 11b = Interrupt
13. If D1..0 is 11b then
14. *bmAttributes* should be 0x03.
15. maxPacketSize should be 0x06.
16. (all other bits are reserved) - this is used to notify the host for changes in audio function.
17. If (D1..0: Transfer type: (01b == Isochronous)) then
18. check that D3..2: Synchronization Type: 01b = Asynchronous.
19. 10b = Adaptive.
20. 11b = Synchronous.
21. D5..4: Usage Type:
22. 00b = Data endpoint or
23. 10b = Implicit feedback Data endpoint
24. Test fails if any of the values are not as specified

3.2.30. BADD Terminal Control Test (BADD)

Assertions Verified in this Test

7.1.2.1#1

Test Procedure

1. **Initialize the test**.
2. For each BADD Function, parse the descriptors, saving all Interface Descriptors for further use.
3. If *bFunctionSubClass* is not 0x25 skip the remaining steps and go on to the next BADD Function.
4. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
5. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = TE_INSERTION_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = 0x04/Interface = Interface number
   - ***wLength*:** Desired
6. If the request does not succeed, then fail the test and throw the related assertion
7. Read the parameter data block for get request, and verify
8. Now for each *bNrConnectors*, verify that if D2 of *bmaConAttributes(i)* is 0 then D[i-1] in bmaConInserted is also 0
9. Verify that if D2 of *bmaConAttributes(i)* is one then D[i-1] in bmaConInserted is either 0/one
10. Verify that *bSize* of response is equal to D[n..o]/8 of bmaConInserted.
11. Verify that the bits (*bSize* * 8 – *bNrConnectors*) on MSB of bmConINserted is always 0
12. If anyone fails then fail the test, and throw the related assertion.
13. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = TE_INSERTION_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0x04/Interface = Interface number
    - ***wLength*:** Desired
14. If the control does not answer with STALL, then fail the test and throw the related assertion.

15. Repeat steps 6 - 10 for Output terminal descriptors with *bUnitId* = 0x03 & *wConnectorsDescrID* = 0x004 in audio control interface descriptor.

3.2.31. BADD Mixer Control Test (BADD)

Assertions Verified in this Test

7.1.2.2#1

Test Procedure

1. **Initialize the test**.
2. For each BADD Function, if *bFunctionSubClass* is not equal to 0x25 or 0x24 skip the remaining steps and go on to the next Function.
3. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.

   Mixer Control Test

   Note: The Mixer Unit descriptor reports which Controls are programmable in the bmMixerControls *bitmap field. This bitmap shall be interpreted as a two-dimensional bit array that has a row for each logical input channel and a column for each logical output channel. If a bit at position [u, v] is set to one, this means that the Mixer Unit contains a programmable Mixer Control that connects input channel u to output channel v. If bit [u, v] is set to 0, this indicates that the connection between input channel u and output channel v is non-programmable*
4. The *bmMixerControls* field stores the bit array row after row where the MSb of the first (highest) byte corresponds to the connection between input channel 1 and output channel 1.
5. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR (For any programmable/Non-Programmable)
   - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
   - ***wIndex*:** Entity ID = 0x08/Interface = Interface number
   - ***wLength*:** Desired
6. If the request does not succeed, then fail the test and throw the related assertion

3.2.32. BADD Feature Unit Control Test (BADD)

Assertions Verified in this Test

7.1.2.3#1

Test Procedure

1. **Initialize the test**.
2. For each BADD Function, parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.

   Mute Control Test

3. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Get for interface
   - ***bRequest*****:** CUR
   - ***wValue*****:** CS = FU_MUTE_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired
4. If the request does not succeed, then fail the test and throw the related assertion
5. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.
6. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Set/Get for interface
   - ***bRequest*****:** RANGE
   - ***wValue*****:** CS = FU_MUTE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired
7. If the control does not answer with STALL, then fail the test and throw the related assertion.

   Volume Control Test

8. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Get for interface
   - ***bRequest*****:** RANGE
   - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired
9. If the request does not succeed, then fail the test and throw the related assertion
10. Parse the parameter data block for get request, and verify

11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
12. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
13. If retrieved data should be same value we sent.
14. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
15. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
16. For all sub ranges Min value should be less than max value.
17. For all sub ranges max of one Sub range should be equal to another sub range.
18. If max and min are equal then RES should be 0.
19. The RES attribute of the Volume Control can only have positive values and range from 1/256db (0x0001) to +127.9961 dB (0x7FFF).
20. The code 0x8000 for the RES attribute shall never be reported as the MIN attribute value.
21. If any of one fails then fail the test, and throw the related assertion
22. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
23. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.33. BADD Clock Source Control Test (BADD)

Assertions Verified in this Test

7.1.2.4#1

Test Procedure

1. **Initialize the test**.
2. For each BADD Function, parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Sampling Frequency Control Test
4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = CS_SAM_FREQ_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Parse the parameter data block for get request, if the CUR attributes can range from 0 Hz (0x00000000) to 4,294,967,295 Hz (0xFFFFFFFF) then fail the test and throw the related assertion.

3.2.34. BADD Power Domain Control Test (BADD)

Assertions Verified in this Test

7.1.2.5#1, 4.3#1.0

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = 0x0A/Interface = Interface number
   - ***wLength*:** 1

4. If the request does not succeed, then fail the test and throw the related assertion
5. Verify the parameter data block for get request, if data is not positive values ranging from 0 to 2 then fail the test and throw the related assertion.
6. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0x0A/Interface = Interface number
    - ***wLength*:** 5
7. If the control does not answer with STALL, then fail the test and throw the related assertion.
8. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0x0A/Interface = Interface number
    - ***wLength*:** 1
    - In parametric block *bCur* Value 0.
9. Direct the user to measure the current manually. Save the result for future use.
10. Repeat steps 9 - 10 with *bCur* value = 1.
11. Repeat steps 9 - 10 with *bCur* value = 2.
12. Test fails if the power measured for *bCur* == 1 is not less than for *bCur* == 0.
13. Test fails if the power measured for *bCur* == 2 is not less than for *bCur* == 1.
14. Repeat steps 10-13 with EntityID = 0x0B.
15. Read *bmAttributes* field in configuration Descriptor, if bit 6 is set to 1, then skip the remaining steps.

3.2.35. Terminal Type Test (Full)

Assertions Verified in this Test

2.1#1, 2.4#3, 2.4#4, 2.5#1

Test Procedure

1. **Initialize the test**.
2. Parse all the input and output terminals with *bDescriptorType* == CS_INTERFACE and *bInterfaceSubClass* == INPUT_TERMINAL or OUTPUT_TERMINAL.

3. If the wTerminalType = 0x0101 or 0x100 or 0x1FF then store the *bTerminalID* of that terminals
4. Parse the class specific audio streaming descriptor with *bDescriptorType* == CS_INTERFACE and *bInterfaceSubClass* == AS_GENERAL. If not present skip the below step.
5. The bTerminalLink value should be one of the stored values.
6. If the wTerminalType == 0x400 to 0x405 then their *bAssocTerminal* value should be ID of I/p terminal if it o/p terminal & I/p terminal if it is o/p terminal and their wTerminalType should be same.
7. If the wTerminalType == 0x600 to 0x60A then their *bAssocTerminal* value should be ID of I/p terminal if it o/p terminal & I/p terminal if it is o/p terminal and their wTerminalType should be same.
8. Test fails if any value is not as specified.

3.2.36. Audio Format Test (Full)

Assertions Verified in this Test

2.3.1.6#1, 2.3.1.6#2, 2.3.3#1, 2.5#3, 2.5#4, 2.5#5, 2.5#6, 2.5#7, 2.5#9, 2.5#10, 2.5#11, 2.5#15, 2.5#15, 2.5#16, 2.3.1.3#1

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_STREAMING), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse the class specific audio streaming with *bDescriptorType* == CS_INTERFACE and *bInterfaceSubClass* == AS_GENERAL.
4. The bSubslotSize value should be either 1 or 2 or 3 or 4 or 8.
5. Find input or output terminal descriptor with *bTerminalID* = bTerminalLink
6. Read wTerminalType from that if it has values other than 0x100 or 0x101 or 0x1FF, then it is Type I or Type III format, otherwise it is Type IV format.
7. In bmFormats only one bit should be set in bits D0-D6 for Type 1 format.
8. bsubSlotSize value should either 1 or 2 or 3 or 4 or 8, if it is Type 1 format.
9. The bSubslotSize value should be 2, & bBitresolution should be ‘16’, if it is Type III format
10. The bSubslotSize value should be 0, & bBitresolution should be 0, if it is Type IV format
11. The bits 31..6 should be equal to 0.

12. If bmAuxProtocols value is 0 then bControlSize should be 0.
13. If bmAuxProtocols is not 0 & it is Type I, then bControlSize should not be be 0. And it should be 0 for remaining Types.
14. Test fails if any of the values are not as specified.

3.2.37. Audio Data Capture Test (Full)

Assertions Verified in this Test

2.4#1, 2.4#3, 2.4#4, 2.4#5, 2.4#6, 2.4#8, 2.4.1#1, 3.1#1, 3.1#2, 3.1#3, 2.3.1.1.1#1

Test Procedure

1. **Initialize the test**.
2. If bmAuxProtocols == 0 and bControlSize == 0, then skip this test.
3. Parse Standard Audio data endpoint with *bDescriptorType* = ENDPOINT, with bit 7 of *bEndpointAddress* is 1. If ENDPOINT not found skip the remaining steps
4. Poll the endpoint for streaming data.
5. From the retrieved data parse SIP Descriptor
6. Bits 15..3 of bmFlags must be equal to 0.
7. At least one bit of 2..0 of bmFlags must be 1.
8. If bit 0 is set then wHeaderLength must not be equal to 0.
9. Test fails if any of the values are not as specified.
10. In SIP Descriptor if wHeaderLength is not equal to 0 then parse the header descriptor that should be present after SIP Descriptor.
11. In Each sub header wSubHeaderID value should be 0x00 or 0x01.
12. HDCP Protocol sub header should be present at every HDCP_PACKET_HEADER_TIME (512ms).
13. Parse HDCP protocol sub header
14. The bOffset field should less than 16.
15. bReserved field should be 0x00.
16. Test fails if any of the values are not as specified.
17. Parse Standard Audio data endpoint with *bDescriptorType* = ENDPOINT, with bit 7 of *bEndpointAddress* is 0. If ENDPOINT not found skip the remaining steps
18. Write some data through that endpoint. If this fails fail the test and throw related assertion.

3.2.38. Enable Control Test (Full)

Assertions Verified in this Test

5.2#2, 5.2.1.4.1#1, 5.2.1.4.1#2, 5.2.1.4.1#3, 5.2.1.4.1#4, 5.2.1.4.1#5, 5.2.1.2#6, 5.2.1.3#1

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Parametric equalizer section effect Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wEffectType = PARAM_EQ_SECTION_EFFECT (PE), If no Parametric equalizer section effect Unit Descriptors found, skip the remaining steps. If a Parametric equalizer section effect Unit is found, retrieve the Unit ID and begin the test on the Control
4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = XX_ENABLE_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. If step 5 passes then issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = XX_ENABLE_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
7. If the request does not succeed, then fail the test and throw the related assertion
8. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = XX_ENABLE_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** 4.
9. If short packet is not returned fail the test and throw the related assertion.

10. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.
11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = XX_ENABLE_CONTROL/CN = non-0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
12. If the control does not answer with STALL, then fail the test and throw the related assertion.
13. Repeat steps 6 - 11 for wEffectType =
14. REVERBERATION EFFECT UNIT (RV)
15. MODULATION DELAY EFFECT UNIT (MD)
16. DYNAMIC RANGE COMPRESSOR EFFECT UNIT (DR).

3.2.39. Underflow Control Test (Full)

Assertions Verified in this Test

5.2.1.4.2#3, 5.2.1.4.2#4, 5.2.1.4.2#5, 5.2.1.4.2#6, 5.2.1.4.2#7, 5.2.1.2#4, 5.2.1.2#5, 5.2.1.12#1

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Input Terminal Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == INPUT_TERMINAL(TE) and, if no Input Terminal Unit Descriptors found, skip the remaining steps. If an Input Terminal Unit is found, retrieve the Unit ID and begin the test on the Control
4. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = XX_UNDERFLOW_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion

6. Verify the parameter data block for get request, if data is not either TRUE or FALSE then fail the test and throw the related assertion.
7. If data is TRUE, repeat steps 4 - 6.
8. Verify the parameter data block for get request, if data is not FALSE then fail the test and throw the related assertion.
9. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = XX_UNDERFLOW_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
10. If the control does not answer with STALL, then fail the test and throw the related assertion.
11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE/INTEN/CUR
    - ***wValue*:** CS = XX_UNDERFLOW_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = unknown ID or interface = unknown Interface number
    - ***wLength*:** Desired
12. If the control does not answer with STALL, then fail the test and throw the related assertion.
13. Repeat steps 3 - 12 for *bDescriptorSubType* ==
14. OUTPUT_TERMINAL(TE)
15. MIXER_UNIT(MU)
16. PARAMETRIC EQUALIZER SECTION EFFECT UNIT (PE)
17. REVERBERATION EFFECT UNIT (RV)
18. MODULATION DELAY EFFECT UNIT (MD)
19. DYNAMIC RANGE COMPRESSOR EFFECT UNIT (DR)
20. UP/DOWN-MIX PROCESSING UNIT (UD)
21. STEREO EXTENDER PROCESSING UNIT (ST_EXT)
22. EXTENSION UNIT (XU)

3.2.40. Overflow Control Test (Full)

Assertions Verified in this Test

5.2.1.1#1, 5.2.1.4.3#3, 5.2.1.4.3#4, 5.2.1.4.3#5, 5.2.1.4.3#6, 5.2.1.4.3#7, 5.2.1.12#1, 5.2#1, 5.2.1.1#1, 5.2.1.2#3

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Input Terminal Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == INPUT_TERMINAL(TE) and, if no Input Terminal Unit Descriptors found, skip the remaining steps. If a Input Terminal Unit is found, retrieve the Unit ID and begin the test on the Control
4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = XX_OVERFLOW_CONTROL/CN = Desired Channel
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.
7. If data is TRUE, repeat steps 4 - 5.
8. Verify the parameter data block for get request, if data is not False then fail the test and throw the related assertion.
9. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** RANGE/INTEN
   - ***wValue*:** CS = XX_OVERFLOW_CONTROL/CN = Desired Channel
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
10. If the control does not answer with STALL, then fail the test and throw the related assertion.
11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = XX_OVERFLOW_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

12. If the control does not answer with STALL, then fail the test and throw the related assertion.
13. For extension unit issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = unsupported control/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
14. If the control pipe doesn't stall fail the test.
15. Repeat steps 6 - 11 for *bDescriptorSubType* ==
    a. OUTPUT_TERMINAL(TE)
    b. MIXER_UNIT(MU)
    c. PARAMETRIC EQUALIZER SECTION EFFECT UNIT (PE)
    d. REVERBERATION EFFECT UNIT (RV)
    e. MODULATION DELAY EFFECT UNIT (MD)
    f. DYNAMIC RANGE COMPRESSOR EFFECT UNIT (DR)
    g. UP/DOWN-MIX PROCESSING UNIT (UD)
    h. STEREO EXTENDER PROCESSING UNIT (ST_EXT)
    i. EXTENSION UNIT (XU)

3.2.41. Power Domain Control Test (Full)

Assertions Verified in this Test

5.2.1.4.4#1, 5.2.1.4.4#2, 5.2.1.4.4#3, 5.2.1.4.4#4, 5.2.1.4.4#5, 5.2.1.4.4#6, 5.2.1.4.4#7, 5.2.1.4.4#8, 3.14.4#9, 3.14.4#10, 3.14.4#11, 6.1#2, 6.1#21

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Power Domain Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == POWER_DOMAIN and, if no Power Domain Unit Descriptors found, skip the remaining steps. If a Power Domain Unit is found, retrieve the Unit ID and begin the test on the Control
4. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** 1

5. If the request does not succeed, then fail the test and throw the related assertion
6. Verify the parameter data block for get request, if data is not positive values ranging from 0 to 2 then fail the test and throw the related assertion.
7. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** 5
8. If the control does not answer with STALL, then fail the test and throw the related assertion.
9. Record the current time on the test computer, and save it for future use.
10. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** 1
    - In parametric block *bCur* Value 1. & note the time.
11. Now issue a CONTROL REQUEST with following values until we get value 0 and the note the time:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** 1
    - In parametric block *bCur* Value 0.
12. Now issue a CONTROL REQUEST with following values until we get value 0 and the note the time:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number

- ***wLength*****:** 1

13. Record the current time on the test computer.
    a. The difference between this time and the time recorded in step 10 should be less than or equal to *waRecoveryTime(1)* value.
14. Now do steps 10 – 14, setting *bCur* to 2 in all steps, and comparing the time recorded to *waRecoveryTime(2)* in step 14a.
15. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = AC_POWER_DOMAIN_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** 1
    - In parametric block *bCur* Value 0.
16. Direct the user to measure the current manually. Record the value for future use.
17. Repeat steps 16 - 17 with *bCur* value == 1 & 2.
18. Test fails if power measured when *bCur* = 1 is greater than or equal to when *bCur* = 0.
19. Test fails if power measured when *bCur* = 2 is greater than or equal to when *bCur* = 1.
20. If *bNumEndpoints* = 1 in audio control interface descriptor, then poll interrupt Endpoint for bInterval time. If the response is other than NACK fail the test and throw related assertion.

3.2.42. Latency Control Test (Full)

Assertions Verified in this Test

5.2.1.4.5#4, 5.2.1.4.5#5, 5.2.1.4.5#6, 5.2.1.4.5#7, 5.2.1.12#1

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find class specific header Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == HEADER and read bits 1..0 of *bmControls* field in it. If it is set to 00b skip this step.
    a. Parse descriptors to find Input Terminal Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == INPUT_TERMINAL(TE) and, if no Input

Terminal Unit Descriptors found, skip the remaining steps. If an Input Terminal Unit is found, retrieve the Unit ID and begin the test on the Control

b. Issue a CONTROL REQUEST with following values:

- ***bmRequestType***: Get for interface
- ***bRequest***: CUR
- ***wValue***: CS = XX_LATENCY_CONTROL/CN = 0
- ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
- ***wLength***: Desired

c. If the request does not succeed, then fail the test and throw the related assertion

d. Verify the parameter data block for get request, if data doesn't range from 0ns to 4,294,967,295ns in steps of 1ns then fail the test and throw the related assertion.

e. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType***: Set for interface
- ***bRequest***: RANGE/INTEN
- ***wValue***: CS = XX_LATENCY_CONTROL/CN = 0
- ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
- ***wLength***: Desired

f. If the control does not respond with STALL, then fail the test and throw the related assertion.

4. Repeat step 3 for the following *bDescriptorSubType* values:

- OUTPUT_TERMINAL(TE)
- MIXER_UNIT(MU)
- SELECTOR UNIT (SU)
- FEATURE UNIT (FU)
- PARAMETRIC EQUALIZER SECTION EFFECT UNIT (PE)
- REVERBERATION EFFECT UNIT (RV)
- MODULATION DELAY EFFECT UNIT (MD)
- DYNAMIC RANGE COMPRESSOR EFFECT UNIT (DR)
- UP/DOWN-MIX PROCESSING UNIT (UD)
- STEREO EXTENDER PROCESSING UNIT (ST_EXT)
- EXTENSION UNIT (XU)

3.2.43. Insertion Control Test (Full)

Assertions Verified in this Test

5.2.1.6.1#2, 5.2.1.6.1#3, 5.2.1.6.1#4, 5.2.1.6.1#5, 5.2.1.6.1#6, 5.2.1.6.1#7, 5.2.1.6.1#8, 5.2.1.6.1#9, 5.2.1.6.1#10

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Input Terminal Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == INPUT_TERMINAL and, if no Input Terminal Descriptors are found, skip the remaining steps.
4. Parse descriptors to find the connector descriptor ID in the input Terminal Descriptor. If no Connector ID is found, skip the remaining steps.
5. Now issue a CONTROL REQUEST with following Values:
    - ***bmRequestType*****:** 10100001B
    - ***bRequest*****:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*****:** *wConnectorsDescrID* and
    - ***wIndex*****:** 0 in higher byte & Interface number in lower byte
    - ***wLength*****:** 2
6. Fail if there is no ACK response or retrieved data is less than 2 bytes.
7. Parse the response and find the total length of the descriptor
8. Now issue a request with:
    - ***bmRequestType*****:** 10100001B
    - ***bRequest*****:** HIGH_CAPABILITY_DESCRIPTOR(0x06)
    - ***wValue*****:** *wConnectorsDescrID*
    - ***wIndex*****:** 0 in higher byte & Interface number in lower byte
    - ***wLength*****:** total length of the descriptor found in step 6.
9. Test fails if there is no ACK response.
10. retrieve the *bNrConnectors* and bmaAttributes(i) of each connectors.
11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = TE_INSERTION_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
12. If the request does not succeed, then fail the test and throw the related assertion
13. Read the parameter data block for get request, and verify

14. For each *bNrConnectors*, verify that if D2 of *bmaConAttributes(i)* is 0 then D[i-1] in bmaConInserted is also 0
15. Verify that if D2 of *bmaConAttributes(i)* is one then D[i-1] in bmaConInserted is either 0/one
16. Verify that *bSize* of response is equal to D[n..o]/8 of bmaConInserted.
17. Verify that the bits (*bSize* * 8 – *bNrConnectors*) on MSB of bmConINserted is always 0
18. If any one of fails then fail the test, and throw the related assertion.
19. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE/INTEN
    - ***wValue*:** CS = TE_INSERTION_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
20. If the control does not answer with STALL, then fail the test and throw the related assertion.
21. Repeat steps 2 - 13 for all Input and Output terminal descriptors in audio control interface descriptor.

3.2.44. Overload Control Test (Full)

Assertions Verified in this Test

5.2.1.6.2#2, 5.2.1.6.2#3, 5.2.1.6.2#4, 5.2.1.6.2#5

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Input Terminal Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == INPUT_TERMINAL and, if no Input Terminal Unit Descriptors found, skip the remaining steps. If a Input Terminal Descriptor is found, retrieve the Unit ID and begin the test on the Control
4. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = TE_OVERLOAD_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

5. If the request does not succeed, then fail the test and throw the related assertion
6. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.
7. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** RANGE/INTEN
    - ***wValue*****:** CS = TE_OVERLOAD_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
8. If the control does not answer with STALL, then fail the test and throw the related assertion.
9. Repeat steps 2 - 8 for all Input and Output terminal descriptors in audio control interface descriptor.

3.2.45. Mixer Unit Control Request Test (Full)

Assertions Verified in this Test

3.13.4#2, 4.5.2.5#4, 5.2.1.7.1#1, 5.2.1.7.1#2, 5.2.1.7.1#3, 5.2.1.7.1#4, 5.2.1.7.1#6, 5.2.1.7.1#7, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4, 5.2.1.7.1#5

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Mixer Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == MIXER_UNIT) and, if no Mixer Unit Descriptors found, skip the remaining steps.
4. If a Mixer Unit Descriptor is found, retrieve the Unit ID and *bmMixerControls* and begin the test on the Control
5. Mixer Control Test

    Note: The Mixer Unit descriptor reports which Controls are programmable in the bmMixerControls *bitmap field. This bitmap shall be interpreted as a two-dimensional bit array that has a row for each logical input channel and a column for each logical output channel. If a bit at position [u, v] is set to one, this means that the Mixer Unit contains a programmable Mixer Control that connects input channel u to output channel v. If bit [u, v] is set to 0, this indicates that the connection between input channel u and output channel v is non-programmable*

The bmMixerControls *field stores the bi t array row after row where the MSb of the first (highest) byte corresponds to the connection between input channel 1 and output channel 1.*

6. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR (For any programmable/Non-Programmable)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
7. If the request does not succeed, then fail the test and throw the related assertion
8. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE (For only programmable channel numbers)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
9. If the request does not succeed, then fail the test and throw the related assertion

    Note: As per specification, every input channel can virtually be mixed into all of the output channels. If n is the total number of logical input channels, contained in all the Clusters that are entering the Mixer Unit and m is the number of logical output channels, then there are (n · m) Mixer Controls in the Mixer Unit, some of which may not be programmable.

    Note: (n · m) shall be limited to 256.

    Mixer Control Number (MCN) = (𝑢–1).𝑚+(𝑣–1)

    The valid range for u is from one to n. The valid range for v is from one to m
10. Parse the parameter data block for get request, and verify
11. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE (For only programmable channel numbers)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
12. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE (For only programmable channel numbers)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number

- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR.

13. If retrieved data is the same value we sent:
14. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
15. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
16. For all sub ranges Min value should be less than max value.
17. For all sub ranges max of one Sub range should be equal to another sub range.
18. If max and min are equal then RES should be 0.
19. The code 0x8000 for the RES attribute shall never be reported as the MIN attribute value.
20. If any one fails then fail the test, and throw the related assertion.
21. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE (For only programmable channel numbers)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value 0x8000.
22. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR (For Non-Programmable channel number)
    - ***wValue*:** CS = MU_MIXER_CONTROL/MCN = Mixer Control Number
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
23. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.46. Selector Unit Control Request Test (Full)

Assertions Verified in this Test

5.2.1.8.1#1, 5.2.1.8.1#2, 5.2.1.8.1#3, 5.2.1.8.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.

3. Parse descriptors to find Selector Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == SELECTOR_UNIT and, if no Selector Unit Descriptors found, skip the remaining steps. If a Selector Unit Descriptor is found, retrieve the Unit ID and bNrPins and begin the test on the Control
4. Selector Control Test
5. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = SU_SELECTOR_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
6. If the request does not succeed, then fail the test and throw the related assertion
7. Parse the parameter data block for get request, and verify the CUR attribute of the Selector Control is from one up to the number of Input pins (bNrPins) of the Selector Unit
8. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = SU_SELECTOR_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
9. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.47. Feature unit Control Request Test (Full)

Assertions Verified in this Test

3.14.1#2, 3.14.1.1#1, 5.2.1.9.1#1, 5.2.1.9.1#2, 5.2.1.9.1#3, 5.2.1.9.2#1, 5.2.1.9.2#2, 5.2.1.9.2#3, 5.2.1.9.2#4, 5.2.1.9.2#5, 5.2.1.9.2#6, 5.2.1.9.2#7, 5.2.1.9.3#1, 5.2.1.9.3#2, 5.2.1.9.3#3, 5.2.1.9.3#4, 5.2.1.9.3#5, 5.2.1.9.4#1, 5.2.1.9.4#2, 5.2.1.9.4#3, 5.2.1.9.4#4, 5.2.1.9.4#5, 5.2.1.9.5#1, 5.2.1.9.5#2, 5.2.1.9.5#3, 5.2.1.9.5#4, 5.2.1.9.6#1, 5.2.1.9.6#2, 5.2.1.9.6#3, 5.2.1.9.6#4, 5.2.1.9.6#5, 5.2.1.9.6#6, 5.2.1.9.6#7, 5.2.1.9.7#1, 5.2.1.9.7#2, 5.2.1.9.7#3, 5.2.1.9.8#1, 5.2.1.9.8#2, 5.2.1.9.8#3, 5.2.1.9.9#1, 5.2.1.9.9#2, 5.2.1.9.9#3, 5.2.1.9.10#1, 5.2.1.9.10#2, 5.2.1.9.10#3, 5.2.1.9.11#1, 5.2.1.9.11#2, 5.2.1.9.11#3, 5.2.1.9.11#4, 5.2.1.9.12#1, 5.2.1.9.12#2, 5.2.1.9.12#3, 5.2.1.9.12#4, 5.2.1.9.13#1, 5.2.1.9.13#2, 5.2.1.9.13#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.

2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Feature Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == FEATURE_UNIT and, if no Feature Unit Descriptors found, skip the remaining steps. If a Feature Unit Descriptor is found, retrieve the Unit ID and begin the test on the Control

Mute Control Test

4. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = FU_MUTE_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.
7. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = FU_MUTE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
8. If the control does not answer with STALL, then fail the test and throw the related assertion.

Volume Control Test

9. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
10. If the request does not succeed, then fail the test and throw the related assertion
11. Parse the parameter data block for get request, and verify

12. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
13. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
14. If retrieved data is the same value we sent:
15. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
16. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
17. For all sub ranges Min value should be less than max value.
18. For all sub ranges max of one Sub range should be equal to another sub range.
19. If max and min are equal then RES should be 0.
20. the RES attribute of the Volume Control can only have positive values and range from 1/256db (0x0001) to +127.9961 dB (0x7FFF).
21. The code 0x8000 for the RES attribute shall never be reported as the MIN attribute value.
22. If any one fails then fail the test, and throw the related assertion
23. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
24. If the control does not answer with STALL, then fail the test and throw the related assertion.

Bass Control Test

25. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_BASS_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

26. If the request does not succeed, then fail the test and throw the related assertion

27. Parse the parameter data block for get request, and verify

28. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_BASS_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

29. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_BASS_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

30. If retrieved data is the same value we sent:

    a. For each max value, if it is less than 0x7F add RES value to it and do step (i) with the calculated value, it should be STALL.

    b. For each min value, if it is less than 0xFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.

    c. For all sub ranges Min value should be less than max value.

    d. For all sub ranges max of one Sub range should be equal to another sub range.

    e. If max and min are equal then RES should be 0.

    f. the RES attribute of the Bass Control can only have positive values and range from 0.25 dB (0x01) to +31.75 dB (0x7F) if any one fails then fail the test, and throw the related assertion

    g. Test fails if any of the values are not as specified

31. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** other than CUR/RANGE/INTEN
- ***wValue*:** CS = FU_VOLUME_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

32. If the control does not answer with STALL, then fail the test and throw the related assertion.

MID Control Test

33. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_MID_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

34. If the request does not succeed, then fail the test and throw the related assertion

35. Parse the parameter data block for get request, and verify

36. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_MID_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired, and
- wCUR value from retrieved valid range value

37. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_MID_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

38. If retrieved data is the same value we sent:

    a. For each max value, if it is less than 0x7F add RES value to it and do step (i) with the calculated value, it should be STALL.

b. For each min value, if it is less than 0xFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.

c. For all sub ranges Min value should be less than max value.

d. For all sub ranges max of one Sub range should be equal to another sub range.

e. If max and min are equal then RES should be 0.

f. the RES attribute of the Bass Control can only have positive values and range from 0.25 dB (0x01) to +31.75 dB (0x7F) if any one fails then fail the test, and throw the related assertion

g. Test fails if any of the values are not as specified

39. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** other than CUR/RANGE/INTEN
- ***wValue*****:** CS = FU_MID_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

40. If the control does not answer with STALL, then fail the test and throw the related assertion.

Treble Control Test

41. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = FU_TREBLE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

42. If the request does not succeed, then fail the test and throw the related assertion

43. Parse the parameter data block for get request, and verify

44. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_TREBLE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

45. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_TREBLE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

46. If retrieved data is the same value we sent:
47. For each max value, if it is less than 0x7F add RES value to it and do step (i) with the calculated value, it should be STALL.
48. For each min value, if it is less than 0xFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
49. For all sub ranges Min value should be less than max value.
50. For all sub ranges max of one Sub range should be equal to another sub range.
51. If max and min are equal then RES should be 0.
52. the RES attribute of the Bass Control can only have positive values and range from 0.25 dB (0x01) to +31.75 dB (0x7F) if any one fails then fail the test, and throw the related assertion
53. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** other than CUR/RANGE/INTEN
- ***wValue*:** CS = FU_ TREBLE _CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

54. If the control does not answer with STALL, then fail the test and throw the related assertion.

Graphic Equalizer Test

55. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_ GRAPHIC_EQUALIZER_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

56. If the request does not succeed, then fail the test and throw the related assertion
57. Parse the parameter data block for get request, and verify
58. count the no of bits set in bmBandsPresent field

59. the *wLength* is equal to 4 + count, else fail the test
60. the D30 and D31 of bmBandsPresent are 0. If not fail the test
61. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = FU_GRAPHIC_EQUALIZER_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
62. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = FU_GRAPHIC_EQUALIZER_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
63. If retrieved data is the same value we sent:
64. For each max value, if it is less than 0x7F add RES value to it and do step (i) with the calculated value, it should be STALL.
65. For each min value, if it is less than 0xFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
66. For all sub ranges Min value should be less than max value.
67. For all sub ranges max of one Sub range should be equal to another sub range.
68. If max and min are equal then RES should be 0.
69. the CUR, MIN and MAX attributes of the Graphic Equalizer Control can range from +31.75 dB (0x7F) down to -32.00b dB (0x80) in steps of 0.25 dB (0x01). if any one fails then fail the test, and throw the related assertion
70. Test fails if any of the values are not as specified
71. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = FU_GRAPHIC_EQUALIZER_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

72. If the control does not answer with STALL, then fail the test and throw the related assertion.

Automatic Gain Control Test

73. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_AUTOMATIC_GAIN_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

74. If the request does not succeed, then fail the test and throw the related assertion

75. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.

76. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_AUTOMATIC_GAIN_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

77. If the control does not answer with STALL, then fail the test and throw the related assertion.

Delay Control Test

78. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_DELAY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

79. If the request does not succeed, then fail the test and throw the related assertion

80. Parse the parameter data block for get request, and verify

81. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_DELAY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number

- ***wLength*****:** Desired/wCUR value from retrieved valid range value

82. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_DELAY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
83. If retrieved data should be same value we sent:
84. For each max value, if it is less than 0xFFFFFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
85. For each min value, if it is less than 0x00000000 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
86. For all sub ranges Min value should be less than max value.
87. For all sub ranges max of one Sub range should be equal to another sub range.
88. If max and min are equal then RES should be 0.
89. Test fails if any of the values are not as specified
90. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** other than CUR/RANGE/INTEN
    - ***wValue*****:** CS = FU_DELAY_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
91. If the control does not answer with STALL, then fail the test and throw the related assertion.

Bass Boost Control Test

92. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = FU_BASS_BOOST_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
93. If the request does not succeed, then fail the test and throw the related assertion

94. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.

95. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_BASS_BOOST_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

96. If the control does not answer with STALL, then fail the test and throw the related assertion.

Loudness Control Test

97. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = FU_LOUDNESS_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

98. If the request does not succeed, then fail the test and throw the related assertion

99. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.

100. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_LOUDNESS_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

101. If the control does not answer with STALL, then fail the test and throw the related assertion.

Input Gain Control Test

102. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = FU_INPUT_GAIN__CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)

- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

103. If the request does not succeed, then fail the test and throw the related assertion

104. Parse the parameter data block for get request, and verify

105. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_INPUT_GAIN_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

106. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_INPUT_GAIN_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired, and
- wCUR value from retrieved valid range value

107. If retrieved data should be same value we sent:

108. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.

109. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.

110. For all sub ranges Min value should be less than max value.

111. For all sub ranges max of one Sub range should be equal to another sub range.

112. If max and min are equal then RES should be 0.

113. the RES attribute of the Input Gain Pad control can only have positive values and range from 1/256 dB (0x0001) to +127.9961 dB (0x7FFF). If it fails then fail the test and throw the related assertion

114. Test fails if any of the values are not as specified

115. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** other than CUR/RANGE/INTEN

- ***wValue*****:** CS = FU_INPUT_GAIN__CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

116. If the control does not answer with STALL, then fail the test and throw the related assertion.

117. Input Gain Pad Control Test

118. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** CUR/RANGE
- ***wValue*****:** CS = FU_INPUT_GAIN_PAD_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

119. If the request does not succeed, then fail the test and throw the related assertion

120. Parse the parameter data block for get request, and verify

121. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_GAIN_PAD_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

122. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_GAIN_PAD_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired, and
- wCUR value from retrieved valid range value

123. If retrieved data should be same value we sent:

124. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.

125. For each min value, if it is less than 0xFFFFsubtract RES value to it and do step (i) with the calculated value, it should be STALL.

126. For all sub ranges Min value should be less than max value.

127. For all sub ranges max of one Sub range should be equal to another sub range.

128. If max and min are equal then RES should be 0.

129. the RES attribute of the Input Gain Pad control can only have positive values and range from 1/256 dB (0x0001) to +127.9961 dB (0x7FFF). if fails then fail the test, and throw the related assertion

130. Test fails if any of the values are not as specified

131. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** other than CUR/RANGE/INTEN
- ***wValue*****:** CS = FU_INPUT_GAIN_PAD_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

132. If the control does not answer with STALL, then fail the test and throw the related assertion.

Phase Inverter Control Test

133. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = FU_PHASE_INVERTER_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number, w
- Length = Desired

134. If the request does not succeed, then fail the test and throw the related assertion

135. Verify the parameter data block for get request, if data is not either TRUE or False then fail the test and throw the related assertion.

136. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = FU_PHASE_INVERTER_CONTROL/CN = Desired Channel (not from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

137. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.48. Effect Unit Control Request Test: PARAMETRIC EQUALIZER SECTION EFFECT UNIT (Full)

Assertions Verified in this Test

5.2.1.10.1.2#1, 5.2.1.10.1.2#2, 5.2.1.10.1.2#3, 5.2.1.10.1.3#1, 5.2.1.10.1.3#2, 5.2.1.10.1.3#3, 5.2.1.10.1.3#4, 5.2.1.10.1.4#1, 5.2.1.10.1.4#2, 5.2.1.10.1.4#3, 5.2.1.10.1.4#4, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Parametric equalizer section effect Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wEffectType = PARAM_EQ_SECTION_EFFECT(PE), If no Parametric equalizer section effect Unit Descriptors found, skip the remaining steps. If a Parametric equalizer section effect Unit is found, retrieve the Unit ID and begin the test on the Control
4. Center Frequency Control Test
5. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = PE_CENTER_FREQ_CONTROL/CN = Desired Channel
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
6. If the request does not succeed, then fail the test and throw the related assertion
7. Parse the parameter data block for get request.
8. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = PE_CENTRAL_FREQ_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired/wCUR value from retrieved valid range value
9. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR

- ***wValue*****:** CS = PE_CENTRAL_FREQ_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired, and
- wCUR value from retrieved valid range value

10. If retrieved data is the same value we sent:
11. For each max value, if it is less than 0xFFFFFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
12. For each min value, if it is less than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
13. For all sub ranges Min value should be less than max value.
14. For all sub ranges max of one Sub range should be equal to another sub range.
15. If max and min are equal then RES should be 0.
16. Test fails if any of the values are not as specified

Qfactor Control Test

17. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = PE_QFACTOR_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

18. If the request does not succeed, then fail the test and throw the related assertion
19. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = PE_QFACTOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired, and
- wCUR value from retrieved valid range value

20. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = PE_QFACTOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number

- ***wLength*****:** Desired, and
- wCUR value from retrieved valid range value

21. If retrieved data is the same value we sent:
22. For each max value, if it is less than 0xFFFFFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
23. For each min value, if it is less than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
24. For all sub ranges Min value should be less than max value.
25. For all sub ranges max of one Sub range should be equal to another sub range.
26. If max and min are equal then RES should be 0.
27. Test fails if any of the values are not as specified

Gain Control Test

28. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = PE_GAIN_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
29. If the request does not succeed, then fail the test and throw the related assertion
30. Parse the parameter data block for get request
31. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = PE_GAIN_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired, and
    - wCUR value from retrieved valid range value
32. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = PE_GAIN_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired, and
    - wCUR value from retrieved valid range value

33. If retrieved data is the same value we sent:
34. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
35. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
36. For all sub ranges Min value should be less than max value.
37. For all sub ranges max of one Sub range should be equal to another sub range.
38. If max and min are equal then RES should be 0.
39. If the RES attribute of the Gain Control does not have positive values and range from 1/256 dB (0x0001) to +127.9961 dB (0x7FFF) then fail the test and throw the related assertion.
40. Test fails if any of the values are not as specified

3.2.49. Effect Unit Control Request Test: REVERBERATION EFFECT UNIT (Full)

Assertions Verified in this Test

5.2.1.10.2.2#2, 5.2.1.10.2.2#3, 5.2.1.10.2.2#4, 5.2.1.10.2.2#5, 5.2.1.10.2.3#1, 5.2.1.10.2.3#2, 5.2.1.10.2.3#3, 5.2.1.10.2.4#1, 5.2.1.10.2.4#2, 5.2.1.10.2.4#3, 5.2.1.10.2.5#1, 5.2.1.10.2.5#2, 5.2.1.10.2.5#3, 5.2.1.10.2.5#4, 5.2.1.10.2.6#1, 5.2.1.10.2.6#2, 5.2.1.10.2.6#3, 5.2.1.10.2.7#1, 5.2.1.10.2.7#2, 5.2.1.10.2.7#3, 5.2.1.10.2.8#1, 5.2.1.10.2.8#2, 5.2.1.10.2.8#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Reverberation effect Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wEffectType = REVERBERATION_EFFECT. If no Reverberation effect Unit Descriptors are found, skip the remaining steps. If a Reverberation effect Unit is found, retrieve the Unit ID and begin the test on the Control

   Type Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** RANGE

- ***wValue*****:** CS = RV_TYPE_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

5. If the request does not succeed, then fail the test and throw the related assertion
6. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = RV_TYPE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
7. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = RV_TYPE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
8. If retrieved data is the same value we sent:
9. For each max value, if it is less than 255 add RES value to it and do step (i) with the calculated value, it should be STALL.
10. For each min value, if it is less than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.
11. For all sub ranges Min value should be less than max value.
12. For all sub ranges max of one Sub range should be equal to another sub range.
13. If max and min are equal then RES should be 0.
14. Test fails if any of the values are not as specified
15. Parse the parameter data block for get request, if the CUR, MIN, and MAX attributes of the Type Control is not from 0 to 255 or if The RES attribute of the Type Control can not have a value of 1. then fail the test and throw the related assertion.
16. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** other than CUR/RANGE/INTEN
    - ***wValue*****:** CS = RV_TYPE_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired

17. If the control does not answer with STALL, then fail the test and throw the related assertion.

Level Control Test

18. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = RV_LEVEL_CONTROL/CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

19. If the request does not succeed, then fail the test and throw the related assertion

20. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_LEVEL_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

21. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_LEVEL_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

22. If retrieved data is the same value we sent:

23. For each max value, if it is less than 0xFF add RES value to it and do step (i) with the calculated value, it should be STALL.

24. For each min value, if it is less than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.

25. For all sub ranges Min value should be less than max value.

26. For all sub ranges max of one Sub range should be equal to another sub range.

27. If max and min are equal then RES should be 0.

28. Test fails if any of the values are not as specified

29. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** other than CUR/RANGE/INTEN

- ***wValue*****:** CS = RV_LEVEL_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

30. If the control does not answer with STALL, then fail the test and throw the related assertion.

Time Control Test

31. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = RV_TIME_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

32. If the request does not succeed, then fail the test and throw the related assertion

33. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = RV_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

34. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = RV_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

35. If retrieved data is the same value we sent:

36. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.

37. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.

38. For all sub ranges Min value should be less than max value.

39. For all sub ranges max of one Sub range should be equal to another sub range.

40. If max and min are equal then RES should be 0.

41. Test fails if any of the values are not as specified

42. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** other than CUR/RANGE/INTEN
- ***wValue*:** CS = RV_TIME_CONTROL/CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

43. If the control does not answer with STALL, then fail the test and throw the related assertion.

Delay Feedback Control Test

44. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_TYPE_CONTROL/CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

45. If the request does not succeed, then fail the test and throw the related assertion

46. Parse the response and verify that bCUR value is either 6 or 7, then test below assertions if not skip them

47. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = RV_FEEDBACK_CONTROLand CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

48. If the request does not succeed, then fail the test and throw the related assertion

49. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_FEEDBACK_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

50. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_FEEDBACK_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number

- ***wLength*:** Desired/wCUR value from retrieved valid range value

51. If retrieved data is the same value we sent:
52. For each max value, if it is less than 0xFF add RES value to it and do step (i) with the calculated value, it should be STALL.
53. For each min value, if it is less than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.
54. For all sub ranges Min value should be less than max value.
55. For all sub ranges max of one Sub range should be equal to another sub range.
56. If max and min are equal then RES should be 0.
57. Test fails if any of the values are not as specified
58. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = RV_FEEDBACK_CONTROLand CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

59. If the control does not answer with STALL, then fail the test and throw the related assertion.

Pre-Delay Control Test

60. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = RV_PREDELAY_CONTRO/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

61. If the request does not succeed, then fail the test and throw the related assertion
62. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = RV_PREDELAY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value

63. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR

- ***wValue*****:** CS = RV_PREDELAY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

64. If retrieved data is the same value we sent:
65. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
66. For each min value, if it is less than 0x0 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
67. For all sub ranges Min value should be less than max value.
68. For all sub ranges max of one Sub range should be equal to another sub range.
69. If max and min are equal then RES should be 0.
70. Test fails if any of the values are not as specified
71. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set/Get for interface
- ***bRequest*****:** other than CUR/RANGE/INTEN
- ***wValue*****:** CS = RV_PREDELAY_CONTRO/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

72. If the control does not answer with STALL, then fail the test and throw the related assertion.
73. Density Control Test
74. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = RV_DENSITY_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

75. If the request does not succeed, then fail the test and throw the related assertion
76. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = RV_DENSITY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

77. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS RV_DENSITY_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
78. If retrieved data is the same value we sent:
79. For each max value, if it is less than 0x64 add RES value to it and do step (i) with the calculated value, it should be STALL.
80. For each min value, if it is less than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.
81. For all sub ranges Min value should be less than max value.
82. For all sub ranges max of one Sub range should be equal to another sub range.
83. If max and min are equal then RES should be 0.
84. Test fails if any of the values are not as specified
85. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** CUR/RANGE/INTEN
    - ***wValue*:** CS = RV_DENSITY_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/bCUR value greater than 0x64.
86. If the control does not answer with STALL, then fail the test and throw the related assertion.

Hi-Freq Roll-Off Control Test

87. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = RV_HIFREQ_ROLLOFF_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
88. If the request does not succeed, then fail the test and throw the related assertion
89. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR

- ***wValue*:** CS = RV_HIFREQ_ROLLOFF_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

90. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = RV_HIFREQ_ROLLOFF_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

91. If retrieved data is the same value we sent:
92. For each max value, if it is less than 0xFFFFFFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
93. For each min value, if it is less than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
94. For all sub ranges Min value should be less than max value.
95. For all sub ranges max of one Sub range should be equal to another sub range.
96. If max and min are equal then RES should be 0.
97. Test fails if any of the values are not as specified
98. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** other than CUR/RANGE/INTEN
- ***wValue*:** CS = RV_HIFREQ_ROLLOFF_CONTROL/CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

99. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.50. Effect Unit Control Request Test: MODULATION DELAY EFFECT UNIT (Full)

Assertions Verified in this Test

5.2.1.10.3.2#1, 5.2.1.10.3.2#2, 5.2.1.10.3.2#3, 5.2.1.10.3.2#4, 5.2.1.10.3.3#1, 5.2.1.10.3.3#2, 5.2.1.10.3.3#3, 5.2.1.10.3.4#1, 5.2.1.10.3.4#2, 5.2.1.10.3.4#3, 5.2.1.10.3.5#1, 5.2.1.10.3.5#2, 5.2.1.10.3.5#3, 5.2.1.10.3.6#1, 5.2.1.10.3.6#2, 5.2.1.10.3.6#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Modulation Delay effect Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wEffectType = MODULATION_DELAY_EFFECT, If no Modulation Delay effect Unit Descriptors found, skip the remaining steps. If a Modulation Delay effect Unit is found, retrieve the Unit ID and begin the test on the Control

   Balance Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Get for interface
   - ***bRequest*****:** RANGE
   - ***wValue*****:** CS = MD_BALANCE_CONTROL/CN = Desired Channel
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Set for interface
   - ***bRequest*****:** CUR
   - ***wValue*****:** CS = MD_BALANCE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired/wCUR value from retrieved valid range value
7. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Get for interface
   - ***bRequest*****:** CUR
   - ***wValue*****:** CS = MD_BALANCE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired/wCUR value from retrieved valid range value
8. If retrieved data is the same value we sent.
9. Do step (i) with the value greater than +100, it should be STALL.
10. Do step (i) with the value less than -100, it should be STALL.
11. For all sub ranges Min value should be less than max value.

12. For all sub ranges max of one Sub range should be equal to another sub range.
13. If max and min are equal then RES should be 0.
14. Test fails if any of the values are not as specified
15. If the RES attribute of the Balance Control can only have positive values and range from 1% (0x01) to 100% (0x64) then fail the test and throw the related assertion.
16. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** other than CUR/RANGE/INTEN
    - ***wValue*****:** CS = MD_BALANCE_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
17. If the control does not answer with STALL, then fail the test and throw the related assertion.

    Rate Control Test
18. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = MD_RATE_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
19. If the request does not succeed, then fail the test and throw the related assertion
20. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = MD_RATE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
21. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = MD_RATE_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
22. If retrieved data is the same value we sent:

23. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
24. For each min value, if it is less than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
25. For all sub ranges Min value should be less than max value.
26. For all sub ranges max of one Sub range should be equal to another sub range.
27. If max and min are equal then RES should be 0.
28. Test fails if any of the values are not as specified
29. Parse the parameter data block for get request, if theRES attributes of the Rate Control can range from 0Hz (0x0000) to 255.9961Hz (0xFFFF) then fail the test and throw the related assertion.
30. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = MD_RATE_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
31. If the control does not answer with STALL, then fail the test and throw the related assertion.

Depth Control Test

32. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = MD_DEPTH_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
33. If the request does not succeed, then fail the test and throw the related assertion
34. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = MD_DEPTH_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
35. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface

- ***bRequest*:** CUR
- ***wValue*:** CS = MD_DEPTH_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

36. If retrieved data is the same value we sent:
37. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
38. For each min value, if it is less than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.
39. For all sub ranges Min value should be less than max value.
40. For all sub ranges max of one Sub range should be equal to another sub range.
41. If max and min are equal then RES should be 0.
42. Test fails if any of the values are not as specified
43. Parse the parameter data block for get request, if the CUR, MIN, MAX, and RES attributes for the Depth Control can range from 1ms (0x0000) to 255.9961ms (0xFFFF) in steps of 1/256ms (0x0001) then fail the test and throw the related assertion.
44. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = MD_DEPTH_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

45. If the control does not answer with STALL, then fail the test and throw the related assertion.

Time Control Test

46. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = MD_TIME_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

47. If the request does not succeed, then fail the test and throw the related assertion
48. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR

- ***wValue*****:** CS = MD_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

49. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = MD_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value

50. If retrieved data is the same value we sent:
51. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
52. For each min value, if it is less than RES value, subtract RES value to it and do step (i) with the calculated value, it should be STALL.
53. For all sub ranges Min value should be less than max value.
54. For all sub ranges max of one Sub range should be equal to another sub range.
55. If max and min are equal then RES should be 0.
56. Test fails if any of the values are not as specified
57. Parse the parameter data block for get request, if the CUR, MIN, MAX, and RES attributes of the Time Control can not range from 0ms (0x0000) to 255.9961ms (0xFFFF) in steps of 1/256ms (0x0001) then fail the test and throw the related assertion.
58. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** other than CUR/RANGE/INTEN
    - ***wValue*****:** CS = MD_TIME_CONTROL/CN = Desired Channel
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired

59. If the control does not answer with STALL, then fail the test and throw the related assertion.

Feedback Level Control Test

60. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = MD_FEEDBACK_CONTROL/CN = Desired Channel

- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

61. If the request does not succeed, then fail the test and throw the related assertion
62. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = MD_FEEDBACK_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
63. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = MD_FEEDBACK_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
64. If retrieved data is the same value we sent:
65. For each max value, if it is less than 0xFF add RES value to it and do step (i) with the calculated value, it should be STALL.
66. For each min value, if it is greater than RES value subtract RES value to it and do step (i) with the calculated value, it should be STALL.
67. For all sub ranges Min value should be less than max value.
68. For all sub ranges max of one Sub range should be equal to another sub range.
69. If max and min are equal then RES should be 0.
70. Test fails if any of the values are not as specified
71. Parse the parameter data block for get request, if the CUR, MIN, MAX, and res attributes of the Feedback Level Control is not from 0% to 255% then fail the test and throw the related assertion.
72. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = MD_FEEDBACK_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

73. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.51. Effect Unit Control Request Test: DYNAMIC RANGE COMPRESSOR EFFECT UNIT (Full)

Assertions Verified in this Test

5.2.1.10.4.2#1, 5.2.1.10.4.2#2, 5.2.1.10.4.2#3, 5.2.1.10.4.3#1, 5.2.1.10.4.3#2, 5.2.1.10.4.3#3, 5.2.1.10.4.3#4, 5.2.1.10.4.4#1, 5.2.1.10.4.4#2, 5.2.1.10.4.4#3, 5.2.1.10.4.4#4, 5.2.1.10.4.5#1, 5.2.1.10.4.5#2, 5.2.1.10.4.5#3, 5.2.1.10.4.6#1, 5.2.1.10.4.6#2, 5.2.1.10.4.6#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Dynamic Range Compressor effect Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wEffectType = MODULATION_DELAY_EFFECT, If no Dynamic Range Compressor effect Unit Descriptors found, skip the remaining steps. If a Dynamic Range Compressor effect Unit is found, retrieve the Unit ID and begin the test on the Control

   Compression Ratio Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = DR_COMPRESSION_RATIO_CONTROL/CN = Desired Channel
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = DR_COMPRESSION_RATIO_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired/wCUR value from retrieved valid range value
7. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = DR_COMPRESSION_RATIO_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

8. If retrieved data is the same value we sent:
9. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
10. For each min value, if it is greater than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
11. For all sub ranges Min value should be less than max value.
12. For all sub ranges max of one Sub range should be equal to another sub range.
13. If max and min are equal then RES should be 0.
14. Test fails if any of the values are not as specified
15. Parse the parameter data block for get request, if the CUR, MIN, MAX, and RES attributes of the Compression Ration Control is not from 0 (0x0000) to 255.9961 (0xFFFF) in steps of 1/256 (0x0001) then fail the test and throw the related assertion.
16. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = DR_COMPRESSION_RATIO_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

17. If the control does not answer with STALL, then fail the test and throw the related assertion.

    MaxAmpl Control Test

18. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = DR_MAXAMPL_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

19. If the request does not succeed, then fail the test and throw the related assertion
20. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set for interface

- ***bRequest*:** CUR
- ***wValue*:** CS = DR_MAXAMPL_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired/wCUR value from retrieved valid range value

21. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = DR_MAXAMPL_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
22. If retrieved data is the same value we sent:
23. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
24. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
25. For all sub ranges Min value should be less than max value.
26. For all sub ranges max of one Sub range should be equal to another sub range.
27. If max and min are equal then RES should be 0.
28. Test fails if any of the values are not as specified
29. Parse the parameter data block for get request, the CUR, MIN, and MAX attributes of the MaxAmpl Control can not range from -128.0000 dB (0x8000) to +127.9961 Db (0x7FFF) in steps of 1/256 dB (0x0001) or if the RES attribute of the MaxAmpl Control can not have positive values and range from 1/256 dB (0x0001) to +127.9961 dB (0x7FFF) then fail the test and throw the related assertion.
30. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = DR_MAXAMPL_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
31. If the control does not answer with STALL, then fail the test and throw the related assertion.

    Threshold Control Test

32. Now issue a CONTROL REQUEST with following value:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** RANGE
- ***wValue*:** CS = DR_TRESHOLD_CONTROL/CN = Desired Channel
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

33. If the request does not succeed, then fail the test and throw the related assertion
34. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = DR_THRESHOLD_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
35. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = DR_THRESHOLD_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
36. If retrieved data is the same value we sent:
37. For each max value, if it is less than 0x7FFF add RES value to it and do step (i) with the calculated value, it should be STALL.
38. For each min value, if it is less than 0xFFFF subtract RES value to it and do step (i) with the calculated value, it should be STALL.
39. For all sub ranges Min value should be less than max value.
40. For all sub ranges max of one Sub range should be equal to another sub range.
41. If max and min are equal then RES should be 0.
42. Test fails if any of the values are not as specified
43. Parse the parameter data block for get request, the CUR, MIN, and MAX attributes of the Threshold Control can not range from -128.0000 dB (0x8000) to +127.9961 dB (0x7FFF) in steps of 1/256 dB (0x0001) or if the RES attribute of the Threshold Control can not have positive values and range from 1/256 dB (0x0001) to +127.9961 dB (0x7FFF) then fail the test and throw the related assertion.
44. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface

- ***bRequest*****:** other than CUR/RANGE/INTEN
- ***wValue*****:** CS = DR_TRESHOLD_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

45. If the control does not answer with STALL, then fail the test and throw the related assertion.

Attack Time Control Test

46. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** RANGE
- ***wValue*****:** CS = DR_ATTACK_TIME_CONTROL/CN = Desired Channel
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

47. If the request does not succeed, then fail the test and throw the related assertion

48. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Set for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = DR_ATTACK_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

49. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*****:** Get for interface
- ***bRequest*****:** CUR
- ***wValue*****:** CS = DR_ATTACK_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid range value

50. If retrieved data is the same value we sent:

51. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.

52. For each min value, if it is greater than 0x0 subtract RES value to it and do step (i) with the calculated value, it should be STALL.

53. For all sub ranges Min value should be less than max value.

54. For all sub ranges max of one Sub range should be equal to another sub range.

55. If max and min are equal then RES should be 0.

56. Test fails if any of the values are not as specified

57. Parse the parameter data block for get request, if the CUR, MIN, MAX, and RES attributes of the Attack Time Control can not range from 0 ms (0x0000) to 255.9961 ms (0xFFFF) in steps of 1/256 ms (0x0001). then fail the test and throw the related assertion.
58. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = DR_ATTACK_TIME_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
59. If the control does not answer with STALL, then fail the test and throw the related assertion

    Release Time Control

60. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = DR_RELEASE_TIME_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
61. If the request does not succeed, then fail the test and throw the related assertion
62. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = DR_RELEASE_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
63. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = DR_RELEASE_TIME_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
64. If retrieved data is the same value we sent:
65. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.

66. For each min value, if it is greater than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
67. For all sub ranges Min value should be less than max value.
68. For all sub ranges max of one Sub range should be equal to another sub range.
69. If max and min are equal then RES should be 0.
70. Test fails if any of the values are not as specified
71. Parse the parameter data block for get request, if the CUR, MIN, MAX, and RES attributes of the Attack Time Control can range from 0ms (0x0000) to 255.9961 ms (0xFFFF) in steps of 1/256 ms (0x0001) then fail the test and throw the related assertion.
72. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set/Get for interface
    - ***bRequest*:** other than CUR/RANGE/INTEN
    - ***wValue*:** CS = DR_RELEASE_TIME_CONTROL/CN = Desired Channel
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
73. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.52. Processing Unit Control Request Test: Up/Down-Mix Processing Unit (Full)

Assertions Verified in this Test

5.2.1.11.1.1#1, 5.2.1.11.1.1#2, 5.2.1.11.1.1#3

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Up/Down-Mix Processing Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wProcessType = UP/DOWNMIX_PROCESS, If no Up/Down-Mix Processing Unit Descriptors found, skip the remaining steps. If a Up/Down-Mix Processing Unit is found, retrieve the Unit ID and bNrModes and begin the test on the Control

   Mode Select Control Test

4. Now issue a CONTROL REQUEST with following values:

- ***bmRequestType*:** Set/Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = UD_MODE_SELECT_CONTROL/CN = 0
- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** Desired

5. If the request does not succeed, then fail the test and throw the related assertion
6. Parse the parameter data block for get request, if the CUR attribute of the Mode Select Control is not from one to the number of modes supported by the Entity (bNrModes) then fail the test and throw the related assertion.
7. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = UD_MODE_SELECT_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
8. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.53. Processing Unit Control Request Test: Stereo Extender Processing Unit (Full)

Assertions Verified in this Test

5.2.1.11.1.2#1, 5.2.1.11.1.2#2, 5.2.1.11.1.2#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Stereo Exntender Processing Unit Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == EFFECT_UNIT and wProcessType = STEREO_EXTENDER_PROCESS, If no Stereo Exntender Processing Unit Descriptors found, skip the remaining steps. If a Stereo Exntender Processing Unit is found, retrieve the Unit ID and begin the test on the Control

   Width Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** RANGE

- ***wValue*****:** CS = ST_EXT_WIDTH_CONTROL/CN = 0
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

5. If the request does not succeed, then fail the test and throw the related assertions
6. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Set for interface
   - ***bRequest*****:** CUR
   - ***wValue*****:** CS = ST_EXT_WIDTH_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired/wCUR value from retrieved valid range value
7. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*****:** Get for interface
   - ***bRequest*****:** CUR
   - ***wValue*****:** CS = ST_EXT_WIDTH_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*****:** Desired/wCUR value from retrieved valid range value
8. If retrieved data is the same value we sent:
9. For each max value, if it is less than 0XFF add RES value to it and do step (i) with the calculated value, it should be STALL.
10. For each min value, if it is greater than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
11. For all sub ranges Min value should be less than max value.
12. For all sub ranges max of one Sub range should be equal to another sub range.
13. If max and min are equal then RES should be 0.
14. Test fails if any of the values are not as specified
15. Parse the parameter data block for get request, if the CUR, MIN, MAX and RES attributes of the Width Control is from 0 to 255 then fail the test and throw the related assertion.
16. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set/Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = ST_EXT_WIDTH_CONTROL/CN = Non 0
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired

17. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.54. Clock Source Control Request Test (Full)

Assertions Verified in this Test

4.5.2.12#2, 5.2.1.13.1#1, 5.2.1.13.1#2, 5.2.1.13.1#3, 5.2.1.13.1#6, 5.2.1.13.2#1, 5.2.1.13.2#2, 5.2.1.13.2#3, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4, 5.2.2.1.1#2, 5.2.2.1.1#4, 5.2.2.1.1#5

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Stereo Clock Source Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == CLOCK_SOURCE, If no Clock Source Descriptors found, skip the remaining steps. If a Clock Source is found, retrieve the Unit ID and begin the test on the Control

   Sampling Frequency Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR/RANGE,
   - ***wValue*:** CS = CS_SAM_FREQ_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Check D0 bit of *bmAttributes*, whether clock is 0 (External) or 1 (Internal). If it is Internal, issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = CS_SAM_FREQ_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired/wCUR value from retrieved valid CUR value
7. If it is External, issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR

- ***wValue*****:** CS = CS_SAM_FREQ_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired/wCUR value from retrieved valid CUR value. If the request does not not completed with STALL, then fail the test/throw the related assertion

8. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = CS_SAM_FREQ_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired/wCUR value from retrieved valid range value
9. If retrieved data is the same value we sent:
10. For each max value, if it is less than 0xFFFFFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
11. For each min value, if it is greater than 0x00, subtract RES value to it and do step (i) with the calculated value, it should be STALL.
12. For all sub ranges Min value should be less than max value.
13. For all sub ranges max of one Sub range should be equal to another sub range.
14. If max and min are equal then RES should be 0.
15. Test fails if any of the values are not as specified
16. Parse the Descriptors to find the CLASS-SPECIFIC AS INTERFACE DESCRIPTOR (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == AS_GENERAL), Retrieve the bTerminalLink and bSubslotSize from CLASS-SPECIFIC AS INTERFACE DESCRIPTOR. Check in *bmControls* whether Active Alternate Setting Control is present or not. If it is not present skip the remaining steps.
17. If it is present, find the descriptors with ID's associated with bTerminalLink and calculate the number of channels corresponding to the descriptors associated with bTerminalLink.
18. Parse the Descriptors to find the STANDARD ENDPOINT DESCRIPTOR (*bDescriptorType*=USB_ENDPOINT_DESCRIPTOR_TYPE), Retrieve the bInterval from STANDARD ENDPOINT DESCRIPTOR.
19. Gets the Device speed. Calculate BW(BandWidth) according to formula for UsbHighSpeed/UsbFullSpeed.
20. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** RANGE

- ***wValue*****:** CS = CS_SAM_FREQ_CONTROL/CN = 0
- ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*****:** Desired

21. If the control does answer with STALL, then fail the test and throw the related assertion.
22. Now for each range values, if BandWidth(BW) > (1024(UsbHighSpeed)/1023(UsbFullSpeed)), then Set the current Interface and AlternateSetting not equal to 0.
23. Now set sampling frequency for which BW exceed according to device speed by issuing a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR,
    - ***wValue*****:** CS = CS_SAM_FREQ_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** 4, Data = range value
24. If the request is completed with STALL, then issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = AS_ACT_ALT_SETTING_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*****:** Desired
25. If the request is not completed with STALL, then fail the test and throw the related assertion
26. Check whether Alternate Setting is set to 0 or not. If it is not set to 0, then fail the test and throw the related assertion.
27. If it is set to 0, then issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR,
    - ***wValue*****:** CS = AS_VAL_ALT_SETTINGS_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** 1.
28. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR,
    - ***wValue*****:** CS = AS_VAL_ALT_SETTINGS_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** 1+size.

29. Now check the bit corresponding to current alternate setting in bmValidAltSettings, it should be set to 0. If it is not set to 0, then fail the test and throw the related assertion.

    Clock Validity Control Test

30. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR,
    - ***wValue*:** CS = CS_CLOCK_VALID_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** 1

31. If the request does not succeed, then fail the test and throw the related assertion

32. Parse the parameter data block for get request, if the CUR attributes is either TRUE or FALSE then fail the test and throw the related assertion.

33. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = CS_CLOCK_VALID_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

34. If the control does not answer with STALL, then fail the test and throw the related assertion

35. Now issue a CONTROL REQUEST with following values:

    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = CS_CLOCK_VALID_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired

36. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.55. Clock Selector Control Request Test (Full)

Assertions Verified in this Test

5.2.1.14.1#1, 5.2.1.14.1#2, 5.2.1.14.1#3

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.

3. Parse descriptors to find Stereo Clock Selector Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == CLOCK_SELECTOR.
   a. If no Clock Selector Descriptors found, skip the remaining steps.
4. Retrieve the Unit ID and bNrInPins and begin the test on the Control

   Clock Selector Control Test

5. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR,
   - ***wValue*:** CS = CX_CLOCK_SELECTOR_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
6. If the request does not succeed, then fail the test and throw the related assertion
7. Parse the parameter data block for get request, if the CUR attribute of the Clock Selector Control is not from one up to the number of Clock Input Pins of the Clock Selector (bNrInPins) then fail the test and throw the related assertion.
8. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = CX_CLOCK_SELECTOR_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
9. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.56. Clock Multiplier Control Request Test (Full)

Assertions Verified in this Test

5.2.1.15.1#1, 5.2.1.15.1#2, 5.2.1.15.1#3, 5.2.1.15.1#4, 5.2.1.15.2#1, 5.2.1.15.2#2, 5.2.1.15.2#3, 5.2.1.15.2#4, 5.2.1.1#2, 5.2.1.1#3, 5.2.1.1#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse descriptors to find Stereo Clock Multiplier Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == CLOCK_MULTIPLIER, If no Clock Multiplier

Descriptors found, skip the remaining steps. If a Clock Multiplier is found, retrieve the Unit ID and begin the test on the Control

Numerator Control Test

4. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = CM_NUMERATOR_CONTROL/CN = 0
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired
5. If the request does not succeed, then fail the test and throw the related assertion
6. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Set for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = CM_NUMERATOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired/wCUR value from retrieved valid range value
7. Now issue a CONTROL REQUEST with following values:
   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = CM_NUMERATOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
   - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength*:** Desired/wCUR value from retrieved valid range value
8. If retrieved data should be same value we sent.
9. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.
10. For each min value, if it is greater than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
11. For all sub ranges Min value should be less than max value.
12. For all sub ranges max of one Sub range should be equal to another sub range.
13. If max and min are equal then RES should be 0.
14. Test fails if any of the values are not as specified

15. Parse the parameter data block for get request, if the CUR, MIN, MAX and RES of the Numerator Control is not in the range 0x0001 to 0xFFFF then fail the test and throw the related assertion.
16. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR/RANGE
    - ***wValue*:** CS = CM_NUMERATOR_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
17. If the control does not answer with STALL, then fail the test and throw the related assertion

    Denominator Control Test
18. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = CM_DENAMIRATOR_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
19. If the request does not succeed, then fail the test and throw the related assertion
20. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = CM_DENOMINATOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
21. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = CM_DENOMINATOR_CONTROL/CN = Desired Channel (from 0 to the number of logical channels in the Cluster)
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired/wCUR value from retrieved valid range value
22. If retrieved data should be same value we sent.
23. For each max value, if it is less than 0xFFFF add RES value to it and do step (i) with the calculated value, it should be STALL.

24. For each min value, if it is greater than 0x00 subtract RES value to it and do step (i) with the calculated value, it should be STALL.
25. For all sub ranges Min value should be less than max value.
26. For all sub ranges max of one Sub range should be equal to another sub range.
27. If max and min are equal then RES should be 0.
28. Test fails if any of the values are not as specified
29. Parse the parameter data block for get request, if the CUR, MIN, MAX and RES attributes for the Denominator Control can range from 1 (0x0001) to 2^16 – 1 (0xFFFF) then fail the test and throw the related assertion.
30. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR/RANGE
    - ***wValue*:** CS = CM_DENAMIRATOR_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
    - ***wLength*:** Desired
31. If the control does not answer with STALL, then fail the test and throw the related assertion

3.2.57. Audio Streaming Control Test (Full)

Assertions Verified in this Test

5.2.2.1.1#1, 5.2.2.1.1#6, 5.2.2.1.1#8, 5.2.2.1.1#9, 5.2.2.1.1#10, 5.2.2.1.2#1, 5.2.2.1.2#2, 5.2.2.1.2#3, 5.2.2.1.2#4, 5.2.2.1.2#5, 5.2.2.1.2#6, 5.2.2.1.2#7, 5.2.2.1.2#8, 5.2.2.1.2#9, 5.2.2.1.2#10, 5.2.2.1.2#11, 5.2.2.1.2#12, 5.2.2.1.3#1, 5.2.2.1.3#2, 5.2.2.1.3#3, 5.2.2.1.3#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_STREAMING), Retrieve the Interface number of this Audio Streaming Interface descriptor.
3. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AS_ACT_ALT_SETTING_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0/Interface = Interface number
    - ***wLength*:** Desired
4. If the request does not succeed, then fail the test and throw the related assertion

5. Verify the parameter data block for get request, the value of an Active Alternate Setting Control CUR attribute shall only be either the last set Alternate Setting or 0 if not then fail the test and throw the related assertion.
6. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = AS_ACT_ALT_SETTING_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** Desired
7. If the control does not answer with STALL, then fail the test and throw the related assertion.
8. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = AS_ACT_ALT_SETTING_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** Desired
9. If the control does not answer with STALL, then fail the test and throw the related assertion.
10. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Get for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = AS_VAL_ALT_SETTINGS_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** Desired
11. If the request does not succeed, then fail the test and throw the related assertion
12. Read the *bSize* and bmValidAltSettings and verify that, for each bit in *bSize* value is
13. D0 is always set for AS0
14. D1….. Dm is corresponding set/reset positions for alternative settings.
15. Test fails if any of the values/response is not as specified.
16. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** CUR
    - ***wValue*****:** CS = AS_VAL_ALT_SETTINGS_CONTROL/CN = 0
    - ***wIndex*****:** Entity ID = 0/Interface = Interface number
    - ***wLength*****:** Desired
17. If the control does not answer with STALL, then fail the test and throw the related assertion.

18. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AS_AUDIO_DATA_FORMAT_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0/Interface = Interface number
    - ***wLength*:** Desired
19. If the request does not succeed, then fail the test and throw the related assertion
20. Read the bmFormats from response and verify that only one bit can be set. If not fail the test.
21. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** CUR
    - ***wValue*:** CS = AS_AUDIO_DATA_FORMAT_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0/Interface = Interface number
    - ***wLength*:** Desired
22. If the control does not answer with STALL, then fail the test and throw the related assertion.
23. Now issue a CONTROL REQUEST with following values:
    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = AS_AUDIO_DATA_FORMAT_CONTROL/CN = 0
    - ***wIndex*:** Entity ID = 0/Interface = Interface number
    - ***wLength*:** Desired
24. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.58. Endpoint Control Request Test

Assertions Verified in this Test

5.2.2.2.1#1, 5.2.2.2.1#2, 5.2.2.2.1#3, 5.2.2.2.2#1, 5.2.2.2.2#2, 5.2.2.2.2#3, 5.2.2.2.2#4, 5.2.2.2.3#1, 5.2.2.2.3#2, 5.2.2.2.3#3, 5.2.2.2.3#4

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Endpoint Descriptor (*bDescriptorType* == ENDPOINT), Retrieve the endpoint number(*bEndpointAddress*[3:0]).

   Pitch Control Test

3. Issue a CONTROL REQUEST with the following values:

- ***bmRequestType*:** Ser/Get for interface
- ***bRequest*:** CUR
- ***wValue*:** CS = EP_PITCH_CONTROL/CN = 0
- ***wIndex*:** 0 in MSB/Endpoint Number in LSB
- ***wLength*:** Desired

4. If the request does not succeed, then fail the test and throw the related assertion
5. Verify the parameter data, The value of a Pitch Control CUR attribute shall be either TRUE (1) or FALSE (0): if not fail the test and throw the related assertion.
6. Issue a CONTROL REQUEST with the following values:

   - ***bmRequestType*:** Set/Get for interface
   - ***bRequest*:** RANGE
   - ***wValue*:** CS = EP_PITCH_CONTROL/CN = 0
   - ***wIndex*:** 0 in MSB/Endpoint Number in LSB
   - ***wLength*:** Desired

7. If the control does not answer with STALL, then fail the test and throw the related assertion.

   Data Overrun Control Test

8. Issue a CONTROL REQUEST with the following values:

   - ***bmRequestType*:** Get for interface
   - ***bRequest*:** CUR
   - ***wValue*:** CS = EP_DATA_OVERRUN_CONTROL/CN = 0
   - ***wIndex*:** 0 in MSB/Endpoint Number in LSB
   - ***wLength*:** Desired

9. If the request does not succeed, then fail the test and throw the related assertion
10. Verify the parameter data, The value of a Pitch Control CUR attribute shall be either TRUE (1) or FALSE (0): if not fail the test and throw the related assertion.
11. Issue a CONTROL REQUEST with the following values:

    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** RANGE
    - ***wValue*:** CS = EP_DATA_OVERRUN_CONTROL/CN = 0
    - ***wIndex*:** 0 in MSB/Endpoint Number in LSB
    - ***wLength*:** Desired

12. If the control does not answer with STALL, then fail the test and throw the related assertion.

    Data Underrun Control test

13. Issue a CONTROL REQUEST with the following values:

    - ***bmRequestType*:** Get for interface
    - ***bRequest*:** CUR

- ***wValue*****:** CS = EP_DATA_UNDERRUN_CONTROL/CN = 0
- ***wIndex*****:** 0 in MSB/Endpoint Number in LSB
- ***wLength*****:** Desired

14. If the request does not succeed, then fail the test and throw the related assertion
15. Verify the parameter data, The value of a Pitch Control CUR attribute shall be either TRUE or FALSE if not fail the test and throw the related assertion.
16. Issue a CONTROL REQUEST with the following values:
    - ***bmRequestType*****:** Set for interface
    - ***bRequest*****:** RANGE
    - ***wValue*****:** CS = EP_DATA_UNDERRUN_CONTROL/CN = 0
    - ***wIndex*****:** 0 in MSB/Endpoint Number in LSB
    - ***wLength*****:** Desired
17. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.59. Memory Requests Test (Full)

Assertions Verified in this Test

5.2.3.1#1, 5.2.3.1#4, 5.2.3.1#5, 6.1#3, 6.1#21

Test Procedure

1. **Initialize the test**.
2. Issue a MEMORY REQUEST with following values:
    - <figure out from code>
3. If the request does not succeed, then fail the test and throw the related assertion
4. Issue a CONTROL REQUEST with the following values:
    - ***bmRequestType*****:** 0010001B
    - ***bRequest*****:** MEM
    - ***wValue*** a 0-based offset value
    - ***wIndex*****:** Entity ID = UnknownID/Interface = wrong Interface number or 0 in MSB/Endpoint Number in LSB
    - ***wLength*****:** Desired
5. If the control does not answer with STALL, then fail the test and throw the related assertion.
6. Issue a MEMORY REQUEST with following values:
    - ***bmRequestType*****:** 0010001B
    - ***bRequest*****:** INTEN
    - ***wValue*** a 0-based offset value

- ***wIndex*****:** Entity ID = AnyID/Interface = Interface number or 0 in MSB/Endpoint Number in LSB
- ***wLength*****:** 1
- data as 0/1

7. If the request does not succeed, then fail the test and throw the related assertion
8. If *bNumEndpoints* == 1 in audio control interface descriptor, then poll interrupt Endpoint for bIntervalTime. If the response is other than NACK, fail the test and throw related assertion.
9. Issue a CONTROL REQUEST with the following values:
    - ***bmRequestType*****:** 0010001B
    - ***bRequest*****:** INTEN
    - ***wValue*** a 0-based offset value
    - ***wIndex*****:** Entity ID = AnyId/Interface = interface number or 0 in MSB/Endpoint Number in LSB
    - ***wLength*****:** 1/data other than 0/1.
10. If the control does not answer with STALL, then fail the test and throw the related assertion.

3.2.60. Class Specific String Request Control Test (Full)

Assertions Verified in this Test

5.2.3.2#1, 5.2.3.2#2, 5.2.3.2#3, 5.2.3.2#4, 5.2.3.2#5, 5.2.3.2#6, 5.2.3.2#7

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse the Audio Control Interface descriptor and retrieve the wStrDescrID of required entity of audio function.
4. If wStrDescrID is less than 255, fail the test and throw related assertion.
5. Now issue a class specific string request with following data:
    - <figure out from code>
6. If the request does not succeed, then fail the test and throw the related assertion
7. Repeat step 5 with *wLength* greater than the length of the string descriptor. If more bytes are returned than the length of the string descriptor, fail the test and throw related assertion.
8. Now issue a class specific string request with following data but provide unknown wStrDescrId/Interface numbers:
    - <figure out from code>

9. If the control does not answer with STALL, then fail the test and throw the related assertion.
10. Repeat step 5 with *wValue* 0 in higher byte and standard string descriptor ID in the lower byte. If there is no ACK response, fail the test and throw related assertion.

3.2.61. L1 & L2 Tests (Full)

Assertions Verified in this Test

3.14.5#1, 3.14.5#2, 3.14.5#3, 3.14.5#4, 3.14.4#7, 3.14.4#8

Test Procedure

1. **Initialize the test**.
2. Read *bmAttributes* field in the current Configuration Descriptor.
3. **Retrieve the full BOS Descriptor**.
4. Parse the BOS descriptor for a USB 2 Extension Descriptor (*bDescriptorType* = DEVICECAPABILITY and *bDevCapabilityType* = USB 2.0 EXTENSION (02H)).
5. Check bit D6 of the *bmAttributes* field in the current Configuration Descriptor:
    a. If D6 == 0:
        i. If no USB 2 Extension Descriptor was found in step 4, then fail the test.
        ii. If a USB 2 Extension Descriptor was found, check bit D1 of its *bmAttributes* field:
            - If D1 == 0, then fail the test.
6. If no USB 2 Extension Descriptor was found, or if one was found and D1 of its *bmAttributes* field is not set to 1, then skip all remaining steps.
7. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
8. Parse descriptors to find Power Domain Descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == POWER_DOMAIN).
9. If no Power Domain Descriptor is found, then skip all remaining steps.
10. For each Power Domain Unit:
    a. Issue a CONTROL REQUEST with the following values:
        - ***bmRequestType***: Set for interface
        - ***bRequest***: CUR
        - ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
        - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
        - ***wLength***: 1

- In Parametric data Block, *bCur* = 2.

b. Confirm that the last CONTROL REQUEST succeeded by issuing another CONTROL REQUEST with the following values:

- ***bmRequestType***: Get for interface
- ***bRequest***: CUR
- ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
- ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
- ***wLength***: 1

i. Fail the test if the data returned is not equal to 2.

c. Now send an L2 Request to the DUT.

d. Bring the DUT back to L0.

e. Issue a CONTROL REQUEST with the following values:

- ***bmRequestType***: Get for interface
- ***bRequest***: CUR
- ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
- ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
- ***wLength***: 1

i. Fail the test if the data returned is not equal to 2.

3.2.62. INTEN Block Test (Full)

Assertions Verified in this Test

5.2.1.1#6, 5.2.1.1#7, 5.2.1.1#8, 5.2.1.1#9, 5.2.1.3#2

Test Procedure

1. **Initialize the test**.
2. Parse the Descriptors to find the Audio Control Interface Descriptor (*bDescriptorType* == INTERFACE and *bInterfaceSubClass* == AUDIO_CONTROL), Retrieve the Interface number of this Audio Control Interface descriptor.
3. Parse the Descriptors to find the Class Specific header descriptor (*bDescriptorType* == CS_INTERFACE and *bDescriptorSubType* == HEADER).
4. Retrieve the total length from the header descriptor.
5. Issue a CONTROL REQUEST with the following values:

- ***bmRequestType*:** Get for interface
- ***bRequest*:** INTEN
- ***wValue*:** CS = 0/CN = 0

- ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number
- ***wLength*:** 1

6. Test fails if response is not either success or STALL.
7. If the response of the step 5 is success:
    a. Issue CONTROL REQUEST with following values:
        - ***bmRequestType***: Set for interface
        - ***bRequest***: INTEN
        - ***wValue***: CS = 0/CN = 0
        - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
        - ***wLength***: 1
        - ***bINTEN*** = 0
    b. Issue CONTROL REQUEST with following values:
        - ***bmRequestType***: Get for interface
        - ***bRequest***: INTEN
        - ***wValue***: CS = 0/CN = 0
        - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
        - ***wLength***: 1
    c. Test fails if response data is not 0.
    d. Issue CONTROL REQUEST with following values:
        - ***bmRequestType***: Set for interface
        - ***bRequest***: INTEN
        - ***wValue***: CS = 0/CN = 0
        - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
        - ***wLength***: 1
        - ***bINTEN*** = 1
    e. Issue CONTROL REQUEST with following values:
        - ***bmRequestType***: Get for interface
        - ***bRequest***: INTEN
        - ***wValue***: CS = 0/CN = 0
        - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
        - ***wLength***: 1
    f. Test fails if response data is not 1.
8. Issue CONTROL REQUEST with following values:
    - ***bmRequestType*:** Set for interface
    - ***bRequest*:** INTEN
    - ***wValue*:** CS = 0/CN = 0
    - ***wIndex*:** Entity ID = *bUnitId*/Interface = Interface number

- ***wLength*****:** 1
- ***bINTEN*** = 2

9. The response should be STALL.
10. Test fails if any of the values are not as specified.

3.2.63. BADD Latency Control Test (BADD)

Assertions Verified in this Test

BADD6.2.2.2#1

Test Procedure

1. **Initialize the test**.
2. For every Terminal and Unit defined in the BADD Profile (as defined by *bFunctionSubtype*):

   Note: The required values for XX_LATENCY_CONTROL and Entity Id for each profile are given in the following steps.

   a. Issue a CONTROL REQUEST with following values:

      - ***bmRequestType***: Get for interface
      - ***bRequest***: CUR
      - ***wValue***: CS = XX_LATENCY_CONTROL/CN = 0
      - ***wIndex***: Entity ID = *bUnitId* (for Units) or bTerminalId (for Terminals)/Interface = interface number
      - ***wLength***: 4

   b. If the request did not succeed, then fail the test and throw the related assertion

   c. Verify the parameter data block for get request, if data doesn't range from 0ns to 4,294,967,295ns in steps of 1ns then fail the test and throw the related assertion.

   d. Now issue a CONTROL REQUEST with following values:

      - ***bmRequestType***: Set for interface
      - ***bRequest***: RANGE
      - ***wValue***: CS = XX_LATENCY_CONTROL/CN = 0
      - ***wIndex***: Entity ID = *bUnitId* (for Units) or bTerminalId (for Terminals)/Interface = interface number
      - ***wLength***: 14

   e. If the control does not answer with STALL, then fail the test and throw the related assertion.

3. If *bFunctionSubtype* == GENERIC_I/O (0x20:

   a. **Determine BADD Generic I/O Profile Subtype**.

b. If the Function is Generic Out, then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2

c. If the Function is Generic In, then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5

d. If the Function is Generic I/O, then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5

4. If *bFunctionSubtype* == HEADPHONE (0x21), then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2

5. If *bFunctionSubtype* == SPEAKER (0x22), then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2

6. If *bFunctionSubtype* == MICROPHONE (0x23), then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5

7. If *bFunctionSubtype* == HEADSET (0x24), then perform step 2 for each of the following sets of values:

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3

- CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
- CS = MU_LATENCY_CONTROL (0x04), Entity Id = 8
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5
- CS = FU_LATENCY_CONTROL (0x10), Entity Id = 7

8. If *bFunctionSubtype* == HEADSET_ADAPTER (0x25), then perform step 2 for each of the following sets of values:

   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
   - CS = MU_LATENCY_CONTROL (0x04), Entity Id = 8
   - CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2
   - CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5
   - CS = FU_LATENCY_CONTROL (0x10), Entity Id = 7

9. If *bFunctionSubtype* == SPEAKERPHONE (0x26), then perform step 2 for each of the following sets of values:

   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 1
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 4
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 3
   - CS = TE_LATENCY_CONTROL (0x05), Entity Id = 6
   - CS = MU_LATENCY_CONTROL (0x04), Entity Id = 8
   - CS = FU_LATENCY_CONTROL (0x10), Entity Id = 2
   - CS = FU_LATENCY_CONTROL (0x10), Entity Id = 5

3.2.64. BADD L1 & L2 Tests (BADD)

Assertions Verified in this Test

3.14.5#1, 3.14.5#2, 3.14.5#3, 3.14.5#4, 3.14.4#7, 3.14.4#8

Test Procedure

1. **Initialize the test**.
2. Read *bmAttributes* field in the current Configuration Descriptor.
3. **Retrieve the full BOS Descriptor**.
4. Parse the BOS descriptor for a USB 2 Extension Descriptor (*bDescriptorType* = DEVICECAPABILITY and *bDevCapabilityType* = USB 2.0 EXTENSION (02H)).
5. Check bit D6 of the *bmAttributes* field in the current Configuration Descriptor:

   a. If D6 == 0:

i. If no USB 2 Extension Descriptor was found in step 4, then fail the test.

ii. If a USB 2 Extension Descriptor was found, check bit D1 of its *bmAttributes* field:

- If D1 == 0, then fail the test.

6. If no USB 2 Extension Descriptor was found, or if one was found and D1 of its *bmAttributes* field is not set to 1, then skip all remaining steps.
7. For every Terminal and Unit defined in the BADD Profile (as defined by *bFunctionSubtype*):

   Note: The required values of bUnitId *for each profile are given in the following steps.*

   a. Issue a CONTROL REQUEST with the following values:

   - ***bmRequestType***: Set for interface
   - ***bRequest***: CUR
   - ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
   - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength***: Desired
   - In Parametric data Block, *bCur* = 2.

   b. Now send an L2 Request to the DUT.

   c. Bring the DUT back in L0.

   d. Issue a CONTROL REQUEST with the following values:

   - ***bmRequestType***: Set for interface
   - ***bRequest***: CUR
   - ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
   - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength***: Desired

   e. Fail the test if the data returned is not equal to 2.

   f. Issue a CONTROL REQUEST with the following values:

   - ***bmRequestType***: Set for interface
   - ***bRequest***: CUR
   - ***wValue***: CS = AC_POWER_DOMAIN_CONTROL/CN = 0
   - ***wIndex***: Entity ID = *bUnitId*/Interface = Interface number
   - ***wLength***: Desired

   g. Fail the test if the data returned is not equal to 2.

8. If *bFunctionSubtype* == GENERIC_I/O (0x20:

   a. **Determine BADD Generic I/O Profile Subtype**.

   b. If the Function is Generic Out, then perform step 7 for each of the following sets of values:

- *bUnitId* = 10

c. If the Function is Generic In, then perform step 7 for each of the following sets of values:

- *bUnitId* = 11

d. If the Function is Generic I/O, then perform step 7 for each of the following sets of values:

- *bUnitId* = 10
- *bUnitId* = 11

9. If *bFunctionSubtype* == HEADPHONE (0x21), then perform step 7 for each of the following sets of values:
   - *bUnitId* = 10
10. If *bFunctionSubtype* == SPEAKER (0x22), then perform step 7 for each of the following sets of values:
    - *bUnitId* = 10
11. If *bFunctionSubtype* == MICROPHONE (0x23), then perform step 7 for each of the following sets of values:
    - *bUnitId* = 11
12. If *bFunctionSubtype* == HEADSET (0x24), then perform step 7 for each of the following sets of values:
    - *bUnitId* = 10
    - *bUnitId* = 11
13. If *bFunctionSubtype* == HEADSET_ADAPTER (0x25), then perform step 7 for each of the following sets of values:
    - *bUnitId* = 10
    - *bUnitId* = 11
14. If *bFunctionSubtype* == SPEAKERPHONE (0x26), then perform step 7 for each of the following sets of values:
    - *bUnitId* = 10
    - *bUnitId* = 11