Compliance Spec additions for UVC 1.5 (M1)

Ver	Changed by	Date	Description
0.1	Everyone	8.1.2012	Starting work on Compliance Spec
0.2	Everyone	8.2.2012	Locked the size of Encoding Unit Descriptor Added tests for reserved bits == 0 Finished Descriptors Finished Select Layer, Video Resolution, Min Frame Interval.
0.3	Maribel Figuera	9.24.2012	Added test #26 for Avg. Bit Rate and CPB size controls to verify device enforces that values are set cumulative per sub-bitstream. Replaced flag comparison typos with assignments. Added a couple of comments.
0.4	Stephen Cooper	11.9.2012	Updated assertion numbers
0.5	Maribel Figuera	11.20.2012	Made a few edits to align spec and implementation.
0.6	Maribel Figuera	12.7.2012	Updated Test Descriptions (section 6). Remaining edits under this section are: verify the 6.20.XX asserts referenced in the test descriptions and update them if necessary to 6.24.XX; update all links)
0.8	Stephen Cooper	12.18.2012	Merging in M2 Encoder Control assertions.
0.85	Stephen Cooper	2.4.2013	Added final three Encoder Control Assertions.
0.86	Stephen Cooper	3.12.2013	Established numbering for UVC 1.5 Test Descriptions. Created links between the Assertions and the Test Description tags. Populated the first few Test Descriptions.
0.9	Stephen Cooper	4.1.2013	Updated SelectLayer Test Description to match the updated Test Description template Added Profile & Toolset Test Description
0.91	Chandrashekhar Rao	5.1.2013	Matching the spec for the M1 CV tool
0.92	Group	5.8.2013	Update after the draft review by the group

Send questions to techadmin@usb.org.

Table of Contents

1 Scope

This document specifies assertions and test procedures for Video Class Devices version 1.5 supporting H.264 or VP8 encoding.

This testing is intended to be in addition to standard USB Compliance testing; assertions covered by the USBCV test document.

This testing applies to one configuration at a time. This testing covers the validation of the Encoding Unit Controls subset of the UVC 1.5 Specification.

The following tests are updated or added in the present compliance tool:

- Standard Video Streaming Interface Descriptor Assertions
- Camera Terminal Descriptor Assertions
- Processing Unit Descriptor Assertions
- Encoding Unit Descriptor Assertions
- H.264 Video Format Descriptor Assertions
- H.264 Video Frame Descriptor Assertions
- Encoding Unit Control Assertions
- Select Layer Control Assertions
- Video Resolution Control Assertions
- Profile and Toolset Control Assertions
- Minimum Frame Interval Control Assertions
- Slice Mode Control Assertions
- Rate Control Mode Control Assertions
- Average Bit Rate Control Assertions
- CPB Size Control Assertions
- Peak Bit Rate Control Assertions
- Quantization Parameter Control Assertions
- Quantization Parameter Range Control
- Sync Frame and Long Term Reference Control Assertions
- Long-Term Buffer Control

2 Related Documents

[1] USB Specification, Revision 2.0, April 27, 2000,
[2] USB Device Class Definition for Video Devices, Revision 1.1, June 1, 2005.
[3] UVC 1.5 Class Specification
[3] Unified Communication Specification and Interfaces for H.264 AVC and SVC UCConfig Modes V 1.1

3 Terms and Abbreviations

4 Test philosophy

The test philosophies adopted are as followings:

- Each control is tested independently.
- Minimum required dependencies utilized.
- Combined non-streaming tests with streaming tests.
- No bit stream or bit rate analysis.
- When a field is tested against an invalid value, all other fields in the control have valid values.

Note: that the USB Video Class Specification 1.5 has, Extension Units in section 4.2.2.5 instead of 4.2.2.4

4.1 Implementation

The test descriptions specified by this document will be integrated into the USBCV tool available from the USB-IF. It is intended that all devices that report a Video Class Interface will be required to pass this test in order to receive logo certification

4.2 High-speed, full-speed, and Super-Speed

For devices that support high-speed and full-speed operation, the test should be run twice: once in full-speed mode, and once in high-speed mode.

4.3 Multiple Configurations

For devices having more than one configuration, the test should be run once for each configuration. The test shall take the configuration index to be tested as an input parameter and report which configuration is being tested at the beginning of the test.

These need to be supported since UVC 1.5 recommends multiple configurations as a solution for backwards compatibility with UVC 1.1 and UVC 1.0.

4.4 Multiple VICs on one device

For devices having more than one VIC, the test should be run once. The test shall loop on every VIC to be tested and report which VIC is being tested inside the test.

This is also more likely with UVC 1.5. A device may want to support two different video formats (e.g. H.264 and VP8) with unique Encoding Unit Controls.

5 Assertions

All of these assertions below assume that **bcdUVC** in the Class VC Interface Descriptor is 1.50 unless otherwise noted.

5.1 Descriptor related assertions

5.1.1 Descriptor Basic Assertions

Class Video Control Interface Descriptor Assertions

Standard Video Streaming Interface Descriptor Assertions

6.3.x

Num	Assertion
6.3.2	Standard VS Interface Descriptor **bInterfaceProtocol** is not PC_PROTOCOL_15 This only applies to UVC 1.5, otherwise **bInterfaceProtocol** is undefined **Specification Ref:** USB Video Specification, Revision 1.5 Section 3.7.1 **Test Description:** TD 1.16

Camera Terminal Descriptor Assertions

6.2.x

Num	Assertion

6.2.63 Camera Terminal Descriptor **bLength** is not 18 (n = 3)
Specification Ref: Section 3.7.2.3 Camera Terminal Descriptor (Table 3-6)
Test Description: TD 1.8

Processing Unit Descriptor Assertions

Num	Assertion
6.2.100	Processing Unit Descriptor **bLength** is not 13 (n = 3) **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.7.2.5 (Table 3-8). **Test Description: TD 1.12**
6.2.103	The **Source** ID referred to in Processing Unit Descriptor does not exist **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.7.2.5 (Table 3-8). **Test Description: TD 1.12**
6.2.104	The **Source** ID referred to in the Processing Unit Descriptor refers to an Output Terminal. **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.7.2.5 (Table 3-8). **Test Description: TD 1.12**
6.2.105	Processing Unit Descriptor **bmControls** has reserved bits set. **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.7.2.5 (Table 3-8). **Test Description: TD 1.12**

Encoding Unit Descriptor Assertions

Num	Assertion
6.2.114	Encoding Unit Descriptor **bLength** is not 13 (n = 3). **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.5.2.6 (Table 3-9). **Test Description: TD 1.21**
6.2.115	Encoding Unit Descriptor **bSourceID** is 0. **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.5.2.6 (Table 3-9). **Test Description: TD 1.21**
6.2.116	Encoding Unit Descriptor, the Source ID referred to in Encoding Unit Descriptor does not exist. **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.5.2.6 (Table 3-9). **Test Description: TD 1.21**
6.2.117	Encoding Unit Descriptor, Source ID refers to an Output Terminal **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.5.2.6 (Table 3-9). **Test Description: TD 1.21**
6.2.118	Encoding Unit Descriptor, **bmControls** D20 – D23 are not 0. **Specification Ref:** USB Video Specification, Revision 1.5, Section 3.5.2.6 (Table 3-9). **Test Description: TD 1.21**

5.1.2 Advanced Descriptor Assertions

5.1.3 Format and Frame Descriptor Assertions

H.264 Video Format Descriptor Assertions

Num	**Assertion**
6.10.230	H264 Video Format Descriptor-- **bLength** is not 52. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.231	H264 Video Format Descriptor-- **bNumFrameDescriptors** is 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.232	H264 Video Format Descriptor -- **bDefaultFrameIndex** is larger than **bNumFrameDescriptors**. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.233	H264 Video Format Descriptor-- found H264 Video Frame Descriptor greater than **bNumFrameDescriptors**. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.234	H264 Video Format Descriptor found duplicate Frame indexes. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.235	H264 Video Format Descriptor **bNumFrameDescriptors** does not match the number of Descriptors. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.236	H.264 Video Format Descriptor **bmSupportedSliceModes** bits D4-D7,!= 0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.237	H.264 Video Format Descriptor **bmSupportedSyncFrameTypes** bits D7,!= 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.238	H.264 Video Format Descriptor **bResolutionScaling** > 4. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.239	H.264 Video Format Descriptor **Reserved1** != 0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.240	H.264 Video Format Descriptor **bmRateControlModes** bits D6-D7,!= 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15
6.10.241	H.264 Video Format Descriptor **bDescriptorSubtype !=** VS_FORMAT_H264_SIMULCAST and any wMaxMBPerSecXX field that supports more than one resolution != 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-1). **Test Description:** TD 3.15

H.264 Video Frame Descriptor Assertions

Num	**Assertion**
6.10.250	H264 Video Frame Descriptor **bLength** != 44 + (**bNumFrameIntervals** * 4). **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 **Test Description:** TD 3.16

Num	Assertion
6.10.251	H264 Video Frame Descriptor **dwFrameInterval**(1)..**dwFrameInterval**(n) not in increasing order. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 **Test Description:** TD 3.16
6.10.252	H.264 Video Frame Descriptor **dwMinBitRate** > **dwMaxBitRate** **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.253	H.264 Video Frame Descriptor **wWidth** or **wHeight** is not divisible by 2 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.254	H.264 Video Frame Descriptor greatest common denominator(**wSARwidth**, **wSARheight**) != 1 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.255	H.264 Video Frame Descriptor **bNumFrameIntervals**==0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.256	H.264 Video Frame Descriptor **dwDefaultFrameInterval** is not equal to any of the supported frame intervals for this Frame Descriptor. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.257	H.264 Video Frame Descriptor **wConstrainedToolset** != 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.258	**bmSupportedUsages** ==0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.259	Any of **bmSupportedUsages** bits D5-D7, D19-D23, and D26-D31 != 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.260	Any of **bmCapabilities** bits D7-D15 != 0. **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.261	**bmCapabilities** bits D4 == 1 and D3 == 0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.262	**bmSVCCapabilities** bits D14 – D31 != 0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16
6.10.263	**bmMVCCapabilities** bits D11 – D31 != 0 **Specification Ref:** USB H.264 Payload Specification, Revision 1.5, Section 3.1.2 (Table 3-2). **Test Description:** TD 3.16

5.2 Video control related assertions

5.2.1 Camera Terminal Control Assertions

5.2.2 Encoding Unit Control Assertions

Testing for Encoding Units are run for each usage as declared in **bmSupportedUsages**. For each test, **wLayerOrViewID** will be set to 0 unless otherwise specified in the test.
The following Encoding Unit Assertions apply to all Encoding Units.

5.2.2.1 Encoding Unit Control Assertions

These assertions apply to all Encoding Unit Controls.

Num	**Assertion**
6.24.40	Encoding Unit Control request that is not GET_LEN and is required, did not respond STALL when issued before a successful VS_COMMIT_CONTROL (SET_CUR). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.41	Encoding Unit Control request did not respond STALL with bRequestErrorCode = 6 (Invalid Control) when not supported by **bmControls** or **bmControlsRuntime** . **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 ***Test Description:*** Applies to Test Descriptions for all Encoding Unit Controls
6.24.42	Encoding Unit Control SET_CUR request did not respond STALL with bRequestErrorCode **=** “Wrong State” when supported by **bmRuntimeControls** but not supported by **bmControls** and issued while not streaming. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.43	Encoding Unit Control SET_CUR request did not respond STALL with bRequestErrorCode **=** “Wrong State” when supported by **bmControls** but not supported by **bmControlsRuntime**.and issued while streaming. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.44	GET_LEN request to supported Encoding Unit Control results in a STALL. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.45	GET_INFO request, when issued after successful VS_COMMIT_CONTROL (SET_CUR) to a supported Encoding Unit Control results in a STALL. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.46	GET_INFO returns (D0==0 and D1==0). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.1.2 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.47	GET_INFO returns (D4==1), but DUT has no interrupt endpoint **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.1.2 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.48	Encoding Unit Control SET_CUR request with invalid **wLayerOrViewID** does not STALL. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls

Num	Assertion
6.24.49	Encoding Unit Control: Any of GET_CUR, GET_MIN, GET_MAX, GET_DEF, and GET_RES requests did not STALL when **wLayerOrViewID** is wildcard mask 0x1C00. This assertion does not apply to the EU_SELECT_LAYER_CONTROL. This test only applies if SET_CUR to Select Layer with **wLayerOrViewID** = 0x1C00 succeeded. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.50	Encoding Unit Control GET_CUR, if supported, does not equal GET_DEF when requested immediately after successful VS_COMMIT_CONTROL (SET_CUR). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.51	GET_CUR, SET_CUR do not STALL when GET_INFO D5 = 1. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.52	GET_CUR does not return the last successful SET_CUR. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.53	Encoding Unit Control SET_CUR responded with STALL when using value returned by GET_DEF request. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.54	Encoding Unit SET_CUR responded with STALL when using value returned by GET_MIN request. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls
6.24.55	Encoding Unit SET_CUR responded with STALL when using value returned by GET_MAX request. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4 **Test Description:** Applies to Test Descriptions for all Encoding Unit Controls

5.2.2.2 Select Layer Control Assertions

Num	Assertion
6.24.60	Select Layer Control GET_CUR immediately following successful VS_COMMIT_CONTROL(SET_CUR) != 0. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.2 **Test Description:** TD 21.1
6.24.61	Select Layer Control GET_CUR does not return the previous successful SET_CUR. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.2 **Test Description:** TD 21.1
6.24.63	Select Layer Control SET_CUR responded with STALL when **wLayerOrViewID** is 0. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.2 **Test Description:** TD 21.1
6.24.64	Select Layer Control SET_CUR does not STALL when bits 13-15 of **wLayerOrViewID** != 0. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.2 **Test Description:** TD 21.1
6.24.65	Select Layer Control GET_CUR returns bits 13-15 of **wLayerOrViewID** != 0 **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.2 **Test Description:** TD 21.1

5.2.2.3 Video Resolution Control Assertions

Num	Assertion
6.24.80	Video Resolution Control GET_LEN != 4. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.81	Video Resolution Control After Commit, GET_CUR returns **wWidth** or **wHeight** > those specified by Frame Descriptor select with COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.82	Video Resolution Control SET_CUR did not STALL with out-of-bound value for **wWidth** (GET_CUR reflects out-of-bounds value). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.83	Video Resolution Control SET_CUR did not STALL with out-of-bound value for **wHeight** (GET_CUR reflects out-of-bounds value). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.84	Video Resolution Control SET_CUR failed with values for **wWidth** and **wHeight** equal to those specified by Frame Descriptor selected with COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.85	Video Resolution Control GET_MAX returned a **wWidth** greater than the **wWidth** specified by Frame Descriptor selected with COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.86	Video Resolution Control GET_MAX returned a **wHeight** greater than the **wHeight** specified by Frame Descriptor selected with COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.87	When bResolutionScaling == 3, GET_MIN returns a **wWidth** less than the minimum **wWidth** specified in the Frame Descriptor list. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.88	When bResolutionScaling == 3, GET_MIN returns a **wHeight** less than the minimum **wHeight** specified in the Frame Descriptor list. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.89	Video Resolution Control SET_CUR responded with STALL when **wLayerOrViewID** is a valid wildcard mask = 0x1C00. This test only applies if SET_CUR to Select Layer with wLayerOrViewID = 0x1C00 succeeded. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.90	Video Resolution Control SET_CUR responded with STALL when using a valid (lower and/or equal) resolution as defined by **bResolutionScaling.** **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2

Num	Assertion
6.24.91	Video Resolution Control SET_CUR does not STALL on odd valued **wWidth/wHeight.** **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.92	SET_CUR failed to STALL when **wWidth** > **wWidth** returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2
6.24.93	SET_CUR failed to STALL when **wHeight** > **wHeight** returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.3 **Test Description:** TD 21.2

5.2.2.4 Profile and Toolset Control Assertions

Test for the Profile and Toolset Control are performed after a successful (VS_COMMIT_CONTROL(SET_CUR) (USB Video Specification, Revision 1.5, Section 4.2.2.4.1). These tests are performed before streaming if **bmControls** bit D11 in the EU descriptor is set to 1 and bit D5 in GET_INFO is set to 0 (USB Video Specification, Revision 1.5, Section 4.1.2). These tests are also performed while streaming if **bmControlsRuntime** bit D11 in the EU descriptor is set to 1 and bit D5 in GET_INFO is set to 0 (USB Video Specification, Revision 1.5, Section 4.1.2).

Num	Assertion
6.24.100	Profile and Toolset Control GET_LEN != 5. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3
6.24.101	Profile and Toolset Control SET_CUR did not respond with STALL when new **wProfile** does not support the current **bUsage** as established in Probe & Commit as defined in the Frame Descriptor for the new **wProfile**. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3
6.24.102	Profile and Toolset Control SET_CUR did not respond with STALL when new **wProfile** is not in list of supported **wProfile** as determined by Frame Descriptors for the current video Format. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3
6.24.103	Profile and Toolset Control SET_CUR did not respond with STALL when new **wProfile** requires a different resolution than is currently resolution as established in Probe & Commit. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3
6.24.104	Profile and Toolset Control SET_CUR did not respond with STALL when new **wProfile** requires a change to **LevelIDC** **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3
6.24.105	Profile and Toolset Control SET_CUR did not respond with STALL when new **bmSettings** includes an unsupported bit as established by **bmCapabilities** of the negotiated Frame Descriptor. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.4 **Test Description:** TD 21.3

5.2.2.5 Minimum Frame Interval Control Assertions

Num	Assertion
6.24.120	Minimum Frame Interval Control, GET_LEN != 4. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4
6.24.121	Minimum Frame Interval Control, Minimum Frame Interval Control GET_MIN returns a value less than the **dwFrameInterval** specified by COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4
6.24.122	Minimum Frame Interval Control, Minimum Frame Interval Control GET_MAX returned a value less than the **dwFrameInterval** specified by COMMIT. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4
6.24.123	Minimum Frame Interval Control, Minimum Frame Interval Control SET_CUR protocol STALLed with wildcard mask (0x1C00). **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4
6.24.124	Minimum Frame Interval Control, Minimum Frame Interval Control For any supported frame interval (as specified in the Frame Descriptor for the committed Video Resolution) SET_CUR resulted in STALL. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4
6.24.125	Minimum Frame Interval Control, SET_CUR did not protocol STALL with **dwFrameInterval** < **dwFrameInterval** from Commit. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.5 **Test Description:** TD 21.4

5.2.2.6 Slice Mode Control Assertions

If a device supports this control, it must support at least one wSlideMode.

Num	Assertion
6.24.140	GET_LEN does not equal 4. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.141	GET_MAX for the **wSliceMode** field support and check for “Maximum number of MBs per slice mode” = D0. If it is not supported then skip the next three tests. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.142	The slice mode control (**wSliceMode**=Maximum number of MBs per slice mode) GET_CUR return **wSliceConfig** != the valid set values of **wSliceConfigSetting**. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5

Num	**Assertion**
6.24.143	The slice mode control SETCUR did not STALL when **wSliceMode** is “Maximum number of MBs per slice mode” and **wSliceConfigSetting** > **wSliceConfigSetting** returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.144	The slice mode control SET_CUR did not STALL when **wSliceMode** is “Maximum number of MBs per slice mode” and **wSliceConfigSetting** < **wSliceConfigSetting** returned by GET_MIN. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.145	GET_MAX for the **wSliceMode** field support and check for “Maximum Target compressed size per slice mode” = D1. If it is not supported then skip the next three tests. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.146	The slice mode control (**wSliceMode**=Target compressed size per slice mode) GET_CUR return **wSliceConfig** != the valid set values of **wSliceConfigSetting**. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.147	The slice mode control SET_CUR did not STALL when **wSliceMode** is “Maximum Target compressed size per slice mode” and **wSliceConfigSetting** > **wSliceConfigSetting** returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.148	The slice mode control SET_CUR did not STALL when **wSliceMode** is “Maximum Target compressed size per slice mode” and **wSliceConfigSetting** < **wSliceConfigSetting** returned by GET_MIN. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.149	GET_MAX for the **wSliceMode** field support and check for “Number of slices per frame mode” = D2. If it is not supported then skip the next three tests. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.150	The slice mode control (**wSliceMode**=Number of slices per frame mode) GET_CUR return **wSliceConfig** != the valid set values of **wSliceConfigSetting** . **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.151	The slice mode control SET_CUR did not STALL when **wSliceMode** is “Number of slices per frame mode” and **wSliceConfigSetting** > **wSliceConfigSetting** returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.152	The slice mode control SET_CUR did not STALL when **wSliceMode** is “Number of slices per frame mode” and **wSliceConfigSetting** < **wSliceConfigSetting** returned by GET_MIN. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5
6.24.153	GET_MAX for the **wSliceMode** field support and check for “Number of MB rows per slice mode” = D3. If it is not supported then skip the next three tests. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.6 **Test Description:** TD 21.5

Num **Assertion**

6.24.154 The slice mode control (**wSliceMode**=Number of MB rows per slice mode) GET_CUR return **wSliceConfig** != the valid set values of **wSliceConfigSetting**.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.6

Test Description: TD 21.5

6.24.155 The slice mode control SET_CUR did not STALL when **wSliceMode** is “Number of MB rows per slice mode” and **wSliceConfigSetting** > **wSliceConfigSetting** returned by GET_MAX.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.6

Test Description: TD 21.5

6.24.156 The slice mode control SET_CUR did not STALL when **wSliceMode** is “Number of MB rows per slice mode” and **wSliceConfigSetting** < **wSliceConfigSetting** returned by GET_MIN.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.6

Test Description: TD 21.5

6.24.157 When calling GET_MAX, bits D4-D15 of **wSliceMode** are not set to zero.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.6

Test Description: TD 21.5

5.2.2.7 Rate Control Mode Control Assertions

Num **Assertion**

6.24.160 Rate Control Mode Control GET_LEN != 1

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.7

Test Description: TD 21.6

6.24.161 Rate Control Mode Control After Commit, GET_CUR returns **bRateControlMode** different than that one specified by **bmRateControlModes** selected with COMMIT **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.7

Test Description: TD 21.6

6.24.162 Rate Control Mode Control SET_CUR did not STALL with a **bRateControlMode** marked as unsupported by **bmSupportedRateControlModes** in the Video Format Descriptor.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.7

Test Description: TD 21.6

6.24.163 Rate Control Mode Control SET_CUR responded with STALL when using a valid **bRateControlMode** as defined by **bmSupportedRateControlModes** in the Video Format Descriptor.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.7

Test Description: TD 21.6

5.2.2.8 Average Bit Rate Control Assertions

Test for all values defined by **bmSupportedRateControlModes** for the current Video Format Descriptor.

Num	Assertion
6.24.180	Average Bitrate Control, GET_LEN != 4. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7

If EU_RATE_CONTROL_MODE_CONTROL **bRateControlMode** = 3 (ConstantQP)

Num	Assertion
6.24.181	SET_CUR did not STALL with “Wrong State” **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7

Else

Num	Assertion
6.24.182	Average Bit Rate Control GET_MIN returned a value less than the **dwMinBitRate** specified by the current Video Frame Descriptor. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7
6.24.183	Average Bit Rate Control GET_MAX returned a value greater than the **dwMaxBitRate** specified by the current Video Frame Descriptor. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7
6.24.184	Average Bit Rate Control SET_CUR failed to set a valid Average Bit Rate **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7
6.24.185	SET_CUR failed to protocol STALL with out of range bit rate **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.8 **Test Description:** TD 21.7

5.2.2.9 CPB Size Control Assertions

Num	Assertion
6.24.200	CPB Size Control, GET_LEN != 4. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.9 **Test Description:** TD 21.8
6.24.201	CPB Size Control, SET_CUR value below GET_MIN did not STALL **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.9 **Test Description:** TD 21.8
6.24.202	CPB Size Control, SET_CUR value above GET_MAX did not STALL **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.9 **Test Description:** TD 21.8

6.24.203 CPB Size Control, SET_CUR did not protocol STALL when **bRateControlMode** is Constant QP = 3.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.9

Test Description: TD 21.8

6.24.204 CPB Size Control, SET_CUR failed to set a valid CPB Size

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.9

Test Description: TD 21.8

Valid means CPG size is not out of range when considered per sub-bitstream.

5.2.2.10 Peak Bit Rate Control Assertions

Num **Assertion**

6.24.220 Peak Bit Rate Control, GET_LEN != 4.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

If **bmRateControlModes** for the current stream = 2 (Constant bit rate)

6.24.221 Peak Bit Rate Control SET_CUR did not STALL with "Wrong State"

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

Else If **bmRateControlModes** for the current stream = 3 (Constant QP)

6.24.222 Peak Bit Rate Control SET_CUR did not STALL with "Wrong State"

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

Else

6.24.223 Peak Bit Rate Control GET_MIN returned a value less than the **dwMinBitRate** specified by the current Video Frame Descriptor.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

6.24.224 Peak Bit Rate Control GET_MAX returned a value greater than the **dwMaxBitRate** specified by the current Video Frame Descriptor.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

6.24.225 Peak Bit Rate Control SET_CUR failed to set the value returned by GET_CUR.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

6.24.226 Peak Bit Rate Control SET_CUR failed to protocol STALL with an out of range peak bit rate as determined by **dwMinBitRate** and **dwMaxBitRate**

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.10

Test Description: TD 21.9

5.2.2.11 Quantization Parameter Control Assertions

Num **Assertion**

6.24.240 Quantization Parameter Control, GET_LEN != 6

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.11

Test Description: TD 21.10

6.24.242 Quantization Parameter Control, SET_CUR failed to set a valid set of Quantization Parameters

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.11

Test Description: TD 21.10

6.24.243 SET_CUR did not protocol STALL when **bRateControlMode** is not Constant QP = 3

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.11

Test Description: TD 21.10

6.24.244 Limits on QP value (0 - ??? based on color depth?)

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.11

Test Description: TD 21.10

6.24.246 SET_CUR failed to stall with invalid value for **wQpPrime_I, wQpPrime_P,** or **wQpPrime_B**

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.11

Test Description: TD 21.10

5.2.2.12 Quantization Parameter Range Control

Quantization Parameter Range Control tests shall be run against each layer as declared in **wLayerOrViewID.**

Num **Assertion**

6.24.260 GET_LEN does not equal 2

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.261 SET_CUR request does not protocol STALL with an error code of **bRequestErrorCode** = “Wrong state” if **bRateControlMode** is Constant QP.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.262 GET_MAX request reports higher **bMinQp** than GET_MIN.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.263 GET_MIN request reports lower **bMaxQp** than GET_MAX.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.264 GET_DEF reports lower **bMinQp** or higher **bMaxQp** than **bMinQP** and **bMaxQp** returned by GET_MAX.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.265 SET_CUR protocol stalls with valid values for **bMinQp** and **bMaxQp**.

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

6.24.266 SET_CUR does not protocol STALL with invalid value (as bounded by GET_MAX, GET_MIN and GET_RES).

Specification Ref: USB Video Specification, Revision 1.5, Section 4.2.2.4.12

Test Description: TD 21.11

5.2.2.13 Sync Frame and Long Term Reference Control Assertions

Num	Assertion
6.24.280	GET_CUR **bSyncFrameType** after COMMIT is not among **bmSupportedSyncFrameTypes** specified in the current Video Format Descriptor **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12
6.24.281	SET_CUR with unsupported **bSynchFrameType** does not fail **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12
6.24.282	SET_CUR **wSyncFrameInterval** outside GET_MIN, GET_MAX does not STALL (if **bSyncFrameType** is not GDR) **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12
6.24.283	SET_CUR **bGradualDecoderRefresh** != 0 does not STALL when **bSyncFrameType** != 6 **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12
6.24.284	Sync Frame and Long Term Reference Control failed SET_CUR failed to set a valid Sync mode and valid Sync Interval **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12
6.24.285	Sync Frame and Long Term Reference Control failed SET_CUR failed to set a valid GDR mode **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.13 **Test Description:** TD 21.12

5.2.2.14 Long-Term Buffer Control

Num	Assertion
6.24.300	Long-Term Buffer Control GET_LEN != 2 **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.14 **Test Description:** TD 21.13
6.24.301	Long-term Buffer Control SET_CUR did not protocol STALL with with ErrorCode equal to invalid out-of-range when **bNumHostControlLTRBuffers** value is greater than that returned by GET_MAX. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.14 **Test Description:** TD 21.13
6.24.302	Long-term Buffer Control SET_CUR did not protocol STALL with ErrorCode equal to invalid out-of-range when **bTrustMode** value > 1. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.14 **Test Description:** TD 21.13
6.24.303	Long-term Buffer Control SET_CUR responded with STALL when using within range values for **bTrustMode** and **bNumHostControlLTRBuffers** as defined by the [0, GET_MAX]. **Specification Ref:** USB Video Specification, Revision 1.5, Section 4.2.2.4.14 **Test Description:** TD 21.13

5.2.2.15 LTR Picture Control

5.2.2.16 LTR Validation Control

6 Description of tests

6.1 Organization of tests

6.2 Output Format

6.3 General Procedures

In general, that is unless specified otherwise like for sub-tests in hierarchical tests, the Init

6.3.1 Init procedure

Before each test (unless specified otherwise) do the following:

1. Send a Device Reset to set the device back in Default mode
2. Check that the device is alive by reading the Device Descriptor
3. If the device is not alive, execute the “Unfreeze” device procedure

6.3.2 Reset Endpoint procedure

If an Endpoint is stalled, do the following:

1. Send a reset command to the stalled Endpoint
2. Check Endpoint state to see if it is now functional
3. If not, execute the Init procedure and fail the current test

6.3.3 “Unfreeze” device procedure

If the device is frozen and does not respond to the Init procedure, do the following:

1. Cycle the USB Port, to make sure the device power is turned off.
2. Re-enumerate the device
3. Check that the device is alive by reading the Device Descriptor
4. If the device still does not respond turn off the power to the device and exit the test with a failure.

6.4 Test parameters

NOTE:

- The current document specifies the 1-based index that is stored in the Configuration descriptor, and used by the SET_CONFIGURATION request.
- The actual test framework uses the 0-based index that is used by the GET_DESCRIPTOR request.

6.5 Test details

6.5.1 Descriptor tests

6.5.1.1 Basic Descriptor Tests

TD 1.12

Processing Unit Descriptor Test

This test verifies that Processing Unit Descriptors in the VIC being tested are compatible with the Video Class specification.

Device States For Test

This test is run once for each of the following device states: Addressed and Configured.

Overview of Test Steps

The test software performs the following steps.

1. Execute the Init procedure
2. Put the device in the desired state
3. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
4. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
5. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
6. Parse the Descriptors to find the Class Video Control Interface Descriptor.
7. Parse the Descriptors following the Class Video Control Interface Descriptor (starting bLength after), and stopping wTotalLength after. Look for all Processing Unit Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_PROCESSING_UNIT).
8. For each Processing Unit Descriptor verify that:
    a. bLength==9+bControlSize and bControlSize<=2. If not fail the test and throw related assertion (6.2.100).
    b. bUnitID!=0. If not fail the test and throw related assertion (6.2.40).
    c. bSourceID!=0. If not fail the test and throw related assertion (6.2.101).
    d. bmControls does not have both D6 and D7 bits set. If not fail the test and throw related assertion (6.2.102).
    e. Fetch the Unit or Terminal Descriptor corresponding to bSourceID. If not found, fail the test and throw the related assertion (6.2.103).

TD 1.21

Encoding Unit Descriptor Test.

This test verifies that the Encoding Unit Descriptor is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Addressed and Configured.

Overview of Test Steps

The test software performs the following steps.

1. Execute the Init procedure
2. Put the device in the desired state
3. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
4. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
5. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
6. Parse the Descriptors to find the Class Video Control Interface Descriptor.
7. Parse the Descriptors following the Class Video Control Interface Descriptor (starting bLength after), and stopping wTotalLength after. Look for all Processing Unit Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_ENCODING_UNIT).
8. For each Encoding Unit Descriptor assert if:

a. 6.2.114, Encoding Unit Descriptor bLength is not 13 (n = 3)
b. 6.2.115, Encoding Unit Descriptor bSourceID is 0.
c. 6.2.116, Encoding Unit, the Source ID referred to in Encoding Unit Descriptor does not exist.
d. 6.2.117, the Source ID referred to in the Encoding Unit Descriptor refers to an Output Terminal.
e. 6.2.118, bmControls D20 – D23 are not 0.

Advanced Descriptor Tests

6.5.1.2 Video Format and Frame Descriptor Tests

TD 3.15

H.264 Video Format Descriptor Test.

This test verifies that the H.264 Video Format Descriptor is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Addressed and Configured.

Overview of Test Steps

The test software performs the following steps.

1. Execute the Init procedure
2. Put the device in the desired state
3. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
4. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
5. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
6. Parse the Descriptors to find all the Standard VS Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING and bAlternateSettings==0). Run the next step for every Standard VS Interface and Alternate Setting 0. (Place of the Format Descriptors have been already checked in test 1.17 and 1.18).
7. Parse the Descriptors to find every H.264 Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_H264).
8. For each H.264 Video Format Descriptor found check that:
    a. bLength==52. If not, fail the test and throw the related assertion (6.10.230).
    b. bNumFrameDescriptors!=0. If not, fail the test and throw related assertion (6.10.231).
    c. bDefaultFrameIndex<=bNumFrameDescriptors. If not, fail the test and throw related assertions (6.10.232).
9. Parse the next bNumFrameDescriptors Descriptors, store their Frame indexes (bFrameIndex) and check that they are all H.264 Frame Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FRAME_H264). If not, fail the test and throw related assertion (6.10.233).
10. Check that all the Frame indexes stored are unique. If not, fail the test and throw related assertion (6.10.234).
11. Parse the next Descriptor and check that it is not an H.264 Frame Descriptor. If not, fail the test and throw the related assertion (6.10.235).
12. For each H.264 Video Format Descriptor found fail the test if:
    a. 6.10.236, H.264 Video Format Descriptor bmSupportedSliceModes bits D4-D7,!= 0.
    b. 6.10.237, H.264 Video Format Descriptor bmSupportedSyncFrameTypes bits D7,!= 0.
    c. 6.10.238, H.264 Video Format Descriptor bResolutionScaling > 4.
    d. 6.10.239, H.264 Video Format Descriptor Reserved1 != 0
    e. 6.10.240, H.264 Video Format Descriptor bmRateControlModes bits D6-D7,!= 0.
    f. 6.10.241 H.264 Video Format Descriptor bDescriptorSubtype != VS_FORMAT_H264_SIMULCAST and any wMaxMBPerSecXX field that supports more than one resolution != 0.

TD 3.16

H.264 Video Frame Descriptor Test.

This test verifies that the H.264 Video Frame Descriptor is compliant with the USBVC specification.

Device States For Test

This test is run once for each of the following device states: Addressed and Configured.

Overview of Test Steps

The test software performs the following steps.

1. Execute the Init procedure
2. Put the device in the desired state
3. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
4. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
5. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
6. Parse the Descriptors to find all the Standard VS Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING and bAlternateSettings==0).Run the next step for every Standard VS Interface and Alternate Setting 0. (Place of the Frame Descriptors have been already checked in test 1.17 and 1.18).
7. Parse the Descriptors to find a Class VS Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_INPUT_HEADER). If not found put a flag=0 (will be used later to test the Frame Descriptors). If found, put flag=1 if bStillCaptureMethod==1, flag=0 otherwise.
8. Parse the Descriptors to find every H.264 Video Format Descriptor (bDescriptorType==CS_INTERFACE&bDescriptorSubType==VS_FORMAT_H264
9. Repeat the following steps for every H.264 Video Format Descriptor found.
    a. Parse the next bNumFrameDescriptors Descriptors and verify that they are FRAME H264.
    b. For each H.264 Frame Descriptor found, check that:
        i. Verify bLength==44+4*bNumFrameIntervals. If not, fail the test and throw the related assertion (6.10.250).
        ii. wWidth and wHeight are both even numbers. If not, fail the test and throw related assertion (6.10.253).
        iii. GCD (wSARwidth, wSARheight) == 1, if not, fail the test and throw related assertion (6.10.254).
        iv. Check wProfile and bLevelIDC against bit rates.
        v. dwMinBitRate <= dwMaxBitRate, if not fail the test and throw related assertion (6.10.252).
        vi. If bNumFrameIntervals==0, throw related assertion (6.10.255).
        vii. If bNumFrameIntervals!=0, check that :
            1. dwDefaultFrameInterval is in the set dwFrameInterval(1)<…<dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.256).
               dwFrameInterval(1)<=..<=dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.251).

        viii. wConstrainedToolset == 0, if not, fail the test and throw related assertion (6.10.257).
        ix. bmSupportedUsages != 0, if not, fail the test and throw related assertion (6.10.258).
        x. `bmSupportedUsages bits D5-D7, D19-D23, and D26-D31 are all == 0,` if not, fail the test and throw related assertion (6.10.259).
        xi. `bmCapabilities bits D7-D15  == 0,` if not, fail the test and throw related assertion (6.10.260).

        xii. `bmCapabilities bits D4 == 0 when D3 == 0,` if not, fail the test and throw related assertion (6.10.261).

xiii. `bmSVCCapabilities bits D14 – D31 == 0,` if not, fail the test and throw related assertion (6.10.262).

xiv. `bmMVCCapabilities bits D11 – D31 == 0,` if not, fail the test and throw related assertion (6.10.263).

6.5.2 Video Control tests

6.5.2.1 Interface Control Tests

6.5.2.2 Unit and Terminal Control tests

6.5.2.2.1 Camera Terminal Control tests

6.5.2.2.2 Selector Unit Control tests

6.5.2.2.3 Processing Unit Control tests

6.5.2.2.4 Extension Unit Control tests

6.5.2.2.5 Media Transport Terminal Control tests

6.5.2.2.6 Encoding Unit Control tests

TD 21.1

Select Layer ID Control Test

This test verifies that the Select Layer ID Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_SELECT_LAYER_CONTROL
Bitmap index of **bmControls** and **bmRuntimeControls**: D0
Size of control parameter block: 2
Mandatory requests: GET_CUR, SET_CUR, GET_LEN, GET_INFO

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

6.24.21

1. Initialize
    a. Execute the “Initialize for Encoding Unit” procedure
    b. Put the device in the desired state
2. For each Encoding Unit found
3. Check absence:
    a. Check the **bmControls** and **bmRuntimeControls** fields of the Encoding Unit Descriptor. If D0==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.24.41 and 6.20.3).
    b. If the Encoding Unit supports no controls, skip to the next Encoding Unit if it exists, else end the test.
4. If D0==1 in either the **bmControls** or the **bmRuntimeControls** field (Select Layer Control is supported), do the following
5. Check GET_LEN
    a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.24.44 and 6.20.7).
    b. Verify that the packet length is 2. If not, fail the test and throw the related assertion (6.20.15).
    c. Verify that the returned value is 2. If not, fail the test and throw the related assertion (6.20.13).
6. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7 and 6.24.45).
    b. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
7. For each streaming **bInterfaceNumber**, determine if it is connected to the selected Unit ID.
    a. If it is not then skip to next interface.
8. For each **bFormatIndex**, if **bDescriptorSubtype** == (VS_FORMAT_H.264, VS_FORMAT_H.164_SIMULCAST, VS_FORMAT_VP8 or , do the following
9. For each **bFrameIndex**, do the following
10. Set the default **dwFrameInterval**,.
11. For each supported **bUsage**, do the following
12. Perform simple Probe and Commit
    a. Issue a GET_DEF request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.31).

    b. Issue a SET_CUR request using the values returned by GET_DEF to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.33).
    c. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.32).
    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.23.38).

13. Check for control disabled by COMMIT.
    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
14. Check lack of support of the control when not streaming
    a. Check the **bmControls** field of the Encoding Unit Descriptor. If D0==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not, fail the test and throw the related assertions (6.24.42 and 6.20.3).
    b. If not supported, skip non-streaming tests
15. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests:
    b. GET_CUR, GET_DEF, GET_MAX, and GET_MIN. Issue all these requests and verify that the request succeeded. If the request did not succeed, fail the test and throw the related assertion (6.20.7).
16. For the requests issued in the previous step, confirm that the retrieved packets are of the correct length.(6.20.15)
17. Check that Select Layer Control GET_CUR immediately following successful VS_COMMIT_CONTROL(SET_CUR) == 0. If != 0, fail the test and throw the related assertion (6.24.60)
18. Check that Select Layer Control GET_CUR returns the previous successful SET_CUR. If it does not, then fail the test and throw the related assertion (6.24.61)
19. Check that Select Layer Control SET_CUR succeeds when wLayerOrViewID is 0. If it responds with a STALL, then fail the test and throw the related assertion (6.24.63)
20. Check that Select Layer Control SET_CUR responds with STALL when bits 13-15 of wLayerOrViewID != 0. It it does not STALL, then fail the test and throw the related assertion (6.24.64)
21. Check that Select Layer Control GET_CUR returns bits 13-15 of wLayerOrViewID == 0. If !=0, then fail the test and throw the related assertion (6.24.65).
22. Issue a SET_INTERFACE request
23. Start isoch stream
24. Check lack of support for the control when streaming
    a. Check the **bmRuntimeControls** field of the Encoding Unit Descriptor. If D0==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not, fail the test and throw the related assertions (6.20.3**).**
    b. If not supported, skip streaming tests (and stop streaming).Continue with next probe & commit setting.
25. Repeat steps 15-21
26. Stop isoch stream
27. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.2

Video Resolution Control Test

This test verifies that the Video Resolution Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_CPB_SIZE_CONTROL

Bitmap index of bmControls and bmRuntimeControls: D2

Size of control parameter block: 4

Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_RES, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize
    a. Execute the “Initialize for Encoding Unit”procedure
    b. Put the device in the desired state
2. For each Encoding Unit found
3. Check absence:
    a. Check the **bmControls** and **bmRuntimeControls** fields of the Encoding Unit Descriptor. If D2==0 in both fields,
        i. Issue a GET_CUR request on the Control and verify that the answer is STALL and the Request Error Code Control is set to 0x06 (invalid control). If not fail the test and throw the related assertions (6.24.41 and 6.20.3).
        ii. Go to the next Encoding Unit if it exists.
    b. If D1==1 in either the **bmControls** or the **bmRuntimeControls** field (Control is supported) continue the test
4. Check GET_LEN
    a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.24.44 and 6.20.7).
    b. Verify that the packet length is 2. If not, fail the test and throw the related assertion (6.20.15).
    c. Verify that the returned value is 4. If not, fail the test and throw the related assertion (6.20.13).
5. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7 and 6.24.45).
    b. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    c.
6. For each streaming **bInterfaceNumber**, determine if it is connected to the selected Encoding Unit ID.
    a. If it is not skip to the next interface
    b. Otherwise
7. For each **bFormatIndex** that references a descriptor which is compatible with the Encoding Unit, do the following
8. For each **bFrameIndex** that references a descriptor which is compatible with the Encoding Unit, do the following
9. Set the default **dwFrameInterval**,.
10. For each supported **bUsage**, do the following
11. For each supported **bmRateControlModes**, do the following
12. Perform simple Probe and Commit
    a. Issue a GET_DEF request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.31).
    b. Issue a SET_CUR request using the values returned by GET_DEF to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.33).
    c. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.23.32).

    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.23.38).
13. Check for control disabled by COMMIT.

    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.

14. Check lack of support of the control when not streaming
    a. Check the **bmControls** field of the Encoding Unit Descriptor. If D2==0, issue a GET_CUR request on the Control and verify that the answer is STALL~~.~~ If not fail the test and throw the related assertions (6.24.42 and 6.20.3).
    b. If not supported, skip non-streaming tests
15. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7)..If not STALL, fail the test and throw the related assertion (6.20.43).
16. Check retrieved packets are of the correct length. (6.24.80 and 6.20.15)
17. Check that Video Resolution Control after Commit, GET_CUR returns **wWidth** and **wHeight** <= those specified by Frame Descriptor select with COMMIT. If **wWidth** or **wHeight** > than those specified by Frame Descriptor selected on Commit, then fail the test and throw the related assertion (6.24.81)
18. Check that Video Resolution Control SET_CUR responds with STALL for an out-of-range value for **wWidth** (GET_CUR reflects out-of-range value). If it does not, then fail the test and throw the related assertion (6.24.82)
19. Check that Video Resolution Control SET_CUR responds with STALL for an out-of-range value for **wHeight** (GET_CUR reflects out-of-range value). If it does not, then fail the test and throw the related assertion (6.24.83)
20. Check that Video Resolution Control SET_CUR succeeds with values for **wWidth** and **wHeight** equal to those specified by Frame Descriptor selected with COMMIT. If it fails, then fail the test and throw the related assertion (6.24.84)
21. Check that Video Resolution Control GET_MAX returns a **wWidth** no greater than the **wWidth** specified by Frame Descriptor selected with COMMIT. If GET_MAX returns a **wWidth** greater than the **wWidth** specified by Frame Descriptor selected with COMMIT, then fail the test and throw the related assertion (6.24.85)
22. Check that Video Resolution Control GET_MAX returned a **wHeight** no greater than the **wHeight** specified by Frame Descriptor selected with COMMIT. If GET_MAX returns a **wHeight** greater than the **wHeight** specified by Frame Descriptor selected with COMMIT, then fail the test and throw the related assertion (6.24.86)
23. Check that when bResolutionScaling == 3, GET_MIN returns a **wWidth** not lower than the minimum **wWidth** specified in the Frame Descriptor list. If the **wWidth** returned on GET_MIN is lower than the minimum **wWidth** specified in the Frame Descriptor list, then fail the test and throw the related assertion (6.24.87)
24. Check that when bResolutionScaling == 3, GET_MIN returns a **wHeight** not lower than the minimum **wHeight** specified in the Frame Descriptor list. If the **wHeight** returned on GET_MIN is lower than the minimum **wHeight** specified in the Frame Descriptor list, then fail the test and throw the related assertion (6.24.88)
25. Check that Video Resolution Control SET_CUR succeeds when **wLayerOrViewID** is a valid wildcard mask = 0x1C00. This test only applies if SET_CUR to Select Layer with **wLayerOrViewID** = 0x1C00 succeeded. If Video Resolution Control SET_CUR responds with a STALL, then fail the test and throw the related assertion (6.24.89)
26. Check that Video Resolution Control SET_CUR succeeds when using a valid (lower and/or equal) resolution as defined by **bResolutionScaling**. If Video Resolution Control SET_CUR responds with a STALL, then fail the test and throw the related assertion (6.24.90)
27. Check that Video Resolution Control SET_CUR responds with a STALL on odd valued **wWidth/wHeight**. If Video Resolution Control SET_CUR fails to STALL, then fail the test and throw the related assertion (6.24.91)
28. Check that SET_CUR responds with a STALL when **wWidth** > **wWidth** returned by GET_MAX. It it fails to STALL, then fail the test and throw the related assertion (6.24.92).
29. Check that SET_CUR responds with a STALL when **wHeight** > **wHeight** returned by GET_MAX. It it fails to STALL, then fail the test and throw the related assertion (6.24.93)
30. Issue a SET_INTERFACE request
31. Start isoch stream
32. Check lack of support for the control when streaming
    a. Check the **bmRuntimeControls** field of the Encoding Unit Descriptor. If D2==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3).

   - b. If not supported, skip streaming tests (and stop streaming).).Continue with next probe & commit setting.

33. Repeat steps 15-29
34. Stop isoch stream
35. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.4

Minimum Frame Interval Control Test

This test verifies that the Minimum Frame Interval Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_MIN_FRAME_INTERVAL_CONTROL
Bitmap index of bmControls and bmRuntimeControls: D3
Size of control parameter block: 4
Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize
   - a. Execute the Init procedure
   - b. Put the device in the desired state
2. Get configuration
   - a. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
   - b. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
3. Get VICs
   - a. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
4. Get VCI
   - a. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find Encoding Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_ENCODING_UNIT. If no Encoding Unit Descriptors if found, stop the test. If an Encoding Unit is found, retrieve the Unit ID and begin the test on the Control.
5. For each Encoding Unit found
6. Check absence:
   - a. Check the bmControls and bmRuntimeControls fields of the Encoding Unit Descriptor. If D3==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
   - b. If control is absent, skip to the next Encoding Unit if it exists, else end the test.
7. If D3==1 in either the bmControls or the bmRuntimeControls field (Control is supported), do the following
8. Check GET_LEN
   - a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).

    b. Verify that the returned value is 4. If not, fail the test and throw the related assertion (6.20.15).

9. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Check if disabled by auto update
    c. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If asynch, check for presence of interrupt endpoint
    e. If the Control is Asynchronous or Autoupdate, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
10. For each streaming bInterfaceNumber, determine if it is connected to the selected Unit ID.
    a. If it is not, then make sure VS_COMMIT_CONTROL issued on this interface does not enable Encoding Units controls, and skip to next interface.
    b. Otherwise
11. For each bFormatIndex that references a descriptor which is compatible with the Encoding Unit, do the following
12. For each bFrameIndex that references a descriptor which is compatible with the Encoding Unit, do the following
13. Set the default dwFrameInterval,.
14. For each supported bUsage, do the following
15. For each supported bmRateControlModes, do the following
16. Perform simple Probe and Commit
    a. Issue a SET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    b. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    c. If the above Probe settings have been changed, the Probe setting is not supported, so skip to next Probe setting.
    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.24.6).
17. Check for control disabled by COMMIT.
    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
18. Check lack of support of the control when not streaming
    a. Check the bmControls field of the Encoding Unit Descriptor. If D3==0, issue a GET_CUR request on the Control and verify that the answer is STALL If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip non-streaming tests
19. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, and GET_DEF. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7.If not STALL, fail the test and throw the related assertion (6.20.43).
20. Check retrieved packets are of the correct length. (6.24.120)

21. Check GET_CUR and SET_CUR
    a. If the control is Asynchronous, verify that Control Change Interrupts are generated on SET_CUR (Timeout 5s). Issue a SET_CUR request and wait during 5 s for a Control Change Interrupt. If no Control Change Interrupt, fail the test and throw the related assertion (6.20.9)
    b. If the control is synchronous, issue a GET_CUR request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.17). If the request did not success, check the Request Error Code Control (issue a GET_CUR Request) and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).

   c. If the control is synchronous, issue a SET_CUR Request and verifies that it completes in less than 10 ms. If not fail the test and throw the related assertion (6.20.10).
22. Check range constraints
   a. Issue GET_MIN, GET_MAX, and GET_DEF requests and verify that they meet the following constraints:
   b. MIN >= committed dwFrameInterval (from commit packet) , if not throw related assertion (6.24.12)
   c. MAX >= committed dwFrameInterval (from commit packet) , if not throw related assertion (6.24.122)
   d. MIN <= MAX, if not throw related assertion (6.20.14)
   e. DEF >= MIN, if not throw related assertion (6.20.11)
   f. DEF <= MAX, if not throw related assertion (6.20.11)
23. Check out of bound value
   a. Issue a SET_CUR request with out-of-bound dwFrameInterval value. Verify that the Minimum Frame Interval Control answers STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.44). If the Control answer is different from STALL, issue a GET_CUR request. If the Control reports the bogus value, fail the test and throw the related assertion (6.22.11).
24. Check range of values
   a. Issue a SET_CUR Request with valid values for dwFrameInterval. The valid values are from the dwFrameInterval list in the Frame descriptor. Verify that the value have been set. If not, fail the test and throw the related assertion (6.24.124).
25. Issue a SET_INTERFACE request
26. Start isoch stream (Currently the CV Test Tool does not perform tests during streaming. Steps 26 -29 are not run)
27. Check lack of support for the control when streaming
   a. Check the bmRuntimeControls field of the Encoding Unit Descriptor. If D3==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
   b. If not supported, skip streaming tests (and stop streaming).
28. Repeat steps 19-25
29. Stop isoch stream
30. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.7

Average Bitrate Control Test

This test verifies that the Average Bit Rate Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_AVERAGE_BITRATE_CONTROL

Bitmap index of bmControls and bmRuntimeControls: D6

Size of control parameter block: 4

Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_RES, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize

    a. Execute the Init procedure
    b. Put the device in the desired state
2. Get configuration
    a. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
    b. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
3. Get VICs
    a. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
4. Get VCI
    a. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find Encoding Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_ENCODING_UNIT. If no Encoding Unit Descriptors if found, stop the test. If an Encoding Unit is found, retrieve the Unit ID and begin the test on the Control.
5. For each Encoding Unit found
6. Check absence:
    a. Check the bmControls and bmRuntimeControls fields of the Encoding Unit Descriptor. If D6==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL .If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If control is absent, skip to the next Encoding Unit if it exists, else end the test.
7. If D6==1 in either the bmControls or the bmRuntimeControls field (Control is supported), do the following
8. Check GET_LEN
    a. Issue a GET_LEN request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Verify that the returned value is 4. If not, fail the test and throw the related assertion (6.20.15).
9. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Check if disabled by auto update
    c. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If asynch, check for presence of interrupt endpoint
    e. If the Control is Asynchronous or Autoupdate, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
10. For each streaming bInterfaceNumber, determine if it is connected to the selected Unit ID.
    a. If it is not, then make sure VS_COMMIT_CONTROL issued on this interface does not enable Encoding Units controls, and skip to next interface.
    b. Otherwise
11. For each bFormatIndex that references a descriptor which is compatible with the Encoding Unit, do the following
12. For each bFrameIndex that references a descriptor which is compatible with the Encoding Unit, do the following
13. Set the default dwFrameInterval,.
14. For each supported bUsage, do the following
15. For each supported bmRateControlModes, do the following
16. Perform simple Probe and Commit
    a. Issue a SET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).

   b. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
   c. If the above Probe settings have been changed, the Probe setting is not supported, so skip to next Probe setting.
   d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.24.6).

17. Check for control disabled by COMMIT.
   a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
   b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
18. Check disallowed Constant QP constraint
   a. If bmRateControl mode for this stream is constant QP, verify that the request stalls~~.~~ If not, fail the test and throw the related assertion (6.24.7).
   b. Skip to next Probe/Commit setting
19. Check lack of support of the control when not streaming
   a. Check the bmControls field of the Encoding Unit Descriptor. If D6==0, issue a GET_CUR request on the Control and verify that the answer is STALL If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
   b. If not supported, skip non-streaming tests
20. Check support of mandatory requests
   a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7If not STALL, fail the test and throw the related assertion (6.20.43).
21. Check retrieved packets are of the correct length. (6.24.180)
22. Check GET_CUR and SET_CUR
   a. If the control is Asynchronous, verify that Control Change Interrupts are generated on SET_CUR (Timeout 5s). Issue a SET_CUR request and wait during 5 s for a Control Change Interrupt. If no Control Change Interrupt, fail the test and throw the related assertion (6.20.9)
   b. If the control is synchronous, issue a GET_CUR request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.17). If the request did not success, check the Request Error Code Control (issue a GET_CUR Request) and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).
   c. If the control is synchronous, issue a SET_CUR Request and verifies that it completes in less than 10 ms. If not fail the test and throw the related assertion (6.20.10).
23. In addition to performing the following tests with wLayerOrViewID set to 0, the tests are also repeated with wLayerOrViewID set to (1<<7) when the stream has multiple temporal layers.
24. Check range constraints
   a. Issue GET_MIN, GET_MAX, GET_RES, and GET_DEF requests and verify that they meet the following constraints:
25. If Average Bit Rate Control GET_MIN returned a value less than the dwMinBitRate specified by the Video Frame descriptor selected by COMMIT, then fail the test and throw the related assertion (6.24.182).
26. If Average Bit Rate Control GET_MAX returned a value greater than the dwMaxBitRate specified by the Video Frame descriptor selected by COMMIT, then fail the test and throw the related assertion (6.24.183).
27. Check
   a. MIN <= MAX, if not throw related assertion (6.20.14)
   b. DEF >= MIN, if not throw related assertion (6.20.11)
   c. DEF <= MAX, if not throw related assertion (6.20.11)
   d. RES != 0 and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
   e. RES != 0 and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
28. Check out of bound value

    a. Issue a SET_CUR request with out-of-bound dwAverageBitRate value. Verify that the Average Bit Rate Control answers STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.44). If the Control answer is different from STALL, issue a GET_CUR request. If the Control reports the bogus value, fail the test and throw the related assertion (6.24.185).

29. Check range of values
    a. Issue a SET_CUR Request with valid values for dwAverageBitRate. Verify that the value have been set. If not, fail the test and throw the related assertion (6.24.184).
30. Issue a SET_INTERFACE request
31. Start isoch stream
32. Check lack of support for the control when streaming
    a. Check the bmRuntimeControls field of the Encoding Unit Descriptor. If D6==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip streaming tests (and stop streaming).
33. Repeat steps 19-26
34. Stop isoch stream
35. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.8

CPB Size Control Test

This test verifies that the CPB Size Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_CPB_SIZE_CONTROL

Bitmap index of bmControls and bmRuntimeControls: D7

Size of control parameter block: 4

Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_RES, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize
    a. Execute the Init procedure
    b. Put the device in the desired state
2. Get configuration
    a. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
    b. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
3. Get VICs
    a. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
4. Get VCI

    a. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find Encoding Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_ENCODING_UNIT. If no Encoding Unit Descriptors if found, stop the test. If an Encoding Unit is found, retrieve the Unit ID and begin the test on the Control.

5. For each Encoding Unit found
6. Check absence:
    a. Check the bmControls and bmRuntimeControls fields of the Encoding Unit Descriptor. If D7==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If control is absent, skip to the next Encoding Unit if it exists, else end the test.
7. If D7==1 in either the bmControls or the bmRuntimeControls field (Control is supported), do the following
8. Check GET_LEN
    a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Verify that the returned value is 4. If not, fail the test and throw the related assertion (6.20.15).
9. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Check if disabled by auto update
    c. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If asynch, check for presence of interrupt endpoint
    e. If the Control is Asynchronous or Autoupdate, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
10. For each streaming bInterfaceNumber, determine if it is connected to the selected Unit ID.
    a. If it is not, then make sure VS_COMMIT_CONTROL issued on this interface does not enable Encoding Units controls, and skip to next interface.
    b. Otherwise
11. For each bFormatIndex that references a descriptor which is compatible with the Encoding Unit, do the following
12. For each bFrameIndex that references a descriptor which is compatible with the Encoding Unit, do the following
13. Set the default dwFrameInterval,.
14. For each supported bUsage, do the following
15. For each supported bmRateControlModes, do the following
16. Perform simple Probe and Commit
    a. Issue a SET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    b. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    c. If the above Probe settings have been changed, the Probe setting is not supported, so skip to next Probe setting.
    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.24.6).
17. Check for control disabled by COMMIT.
    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
18. Check disallowed Constant QP constraint
    a. If bmRateControl mode for this stream is constant QP, verify that the request. If not, fail the test and throw the related assertion (6.24.203).

    b. Skip to next Probe/Commit setting

19. Check lack of support of the control when not streaming
    a. Check the bmControls field of the Encoding Unit Descriptor. If D7==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip non-streaming tests
20. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).If not STALL, fail the test and throw the related assertion (6.20.43).
21. Check retrieved packets are of the correct length.(6.24.200)
22. Check GET_CUR and SET_CUR
    a. If the control is Asynchronous, verify that Control Change Interrupts are generated on SET_CUR (Timeout 5s). Issue a SET_CUR request and wait during 5 s for a Control Change Interrupt. If no Control Change Interrupt, fail the test and throw the related assertion (6.20.9)
    b. If the control is synchronous, issue a GET_CUR request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.17). If the request did not success, check the Request Error Code Control (issue a GET_CUR Request) and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).
    c. If the control is synchronous, issue a SET_CUR Request and verifies that it completes in less than 10 ms. If not fail the test and throw the related assertion (6.20.10).
23. In addition to performing the following tests with wLayerOrViewID set to 0, the tests are also repeated with wLayerOrViewID set to (1<<7) when the stream has multiple temporal layers.
24. Check range constraints
    a. Issue GET_MIN, GET_MAX, GET_RES, and GET_DEF requests and verify that they meet the following constraints:
    b. MIN <= MAX, if not throw related assertion (6.20.14)
    c. DEF >= MIN, if not throw related assertion (6.20.11)
    d. DEF <= MAX, if not throw related assertion (6.20.11)
    e. RES != 0 and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    f. RES != 0 and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
25. Check out of bound value
26. Issue a SET_CUR request with out-of-bound dwCPBSize value. Verify that the Average Bit Rate Control answers STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.44). If the Control answer is different from STALL, issue a GET_CUR request. If the Control reports the bogus value, fail the test and throw the related assertion (6.22.11). Check range of values
    a. Issue a SET_CUR Request with valid values for dwCpbSize. Verify that the value have been set. If not, fail the test and throw the related assertion (6.24.204).
27. Issue a SET_INTERFACE request
28. Start isoch stream
29. Check lack of support for the control when streaming
    a. Check the bmRuntimeControls field of the Encoding Unit Descriptor. If D7==0, issue a GET_CUR request on the Control and verify that the answer is If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip streaming tests (and stop streaming).
30. Repeat steps 19-26
31. Stop isoch stream
32. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.10

Quantization Parameter Control Test

This test verifies that the Quantization Parameter Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_QUANTIZATION_PARAMS_CONTROL

Bitmap index of bmControls and bmRuntimeControls: D9

Size of control parameter block: 6

Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_RES, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize
    a. Execute the Init procedure
    b. Put the device in the desired state
2. Get configuration
    a. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
    b. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
3. Get VICs
    a. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
4. Get VCI
    a. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find Encoding Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_ENCODING_UNIT. If no Encoding Unit Descriptors if found, stop the test. If an Encoding Unit is found, retrieve the Unit ID and begin the test on the Control.
5. For each Encoding Unit found
6. Check absence:
    a. Check the bmControls and bmRuntimeControls fields of the Encoding Unit Descriptor. If D9==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If control is absent, skip to the next Encoding Unit if it exists, else end the test.
7. If D9==1 in either the bmControls or the bmRuntimeControls field (Control is supported), do the following
8. Check GET_LEN
    a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Verify that the returned value is 6. If not, fail the test and throw the related assertion (6.20.15).
9. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Check if disabled by auto update
    c. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If asynch, check for presence of interrupt endpoint

    e. If the Control is Asynchronous or Autoupdate, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

10. For each streaming bInterfaceNumber, determine if it is connected to the selected Unit ID.
    a. If it is not, then make sure VS_COMMIT_CONTROL issued on this interface does not enable Encoding Units controls, and skip to next interface.
    b. Otherwise
11. For each bFormatIndex that references a descriptor which is compatible with the Encoding Unit, do the following
12. For each bFrameIndex that references a descriptor which is compatible with the Encoding Unit, do the following
13. Set the default dwFrameInterval,.
14. For each supported bUsage, do the following
15. For each supported bmRateControlModes, do the following
16. Perform simple Probe and Commit
    a. Issue a SET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    b. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    c. If the above Probe settings have been changed, the Probe setting is not supported, so skip to next Probe setting.
    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.24.6).
17. Check for control disabled by COMMIT.
    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
18. Check mandatory Constant QP constraint
    a. If bmRateControl mode for this stream is not constant QP, verify that the request. If not, fail the test and throw the related assertion (6.24.243).
    b. Skip to next Probe/Commit setting
19. Check lack of support of the control when not streaming
    a. Check the bmControls field of the Encoding Unit Descriptor. If D9==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip non-streaming tests
20. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). If not STALL, fail the test and throw the related assertion (6.20.43).
21. Check retrieved packets are of the correct length.(6.24.240)
22. Check GET_CUR and SET_CUR
    a. If the control is Asynchronous, verify that Control Change Interrupts are generated on SET_CUR (Timeout 5s). Issue a SET_CUR request and wait during 5 s for a Control Change Interrupt. If no Control Change Interrupt, fail the test and throw the related assertion (6.20.9)
    b. If the control is synchronous, issue a GET_CUR request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.17). If the request did not success, check the Request Error Code Control (issue a GET_CUR Request) and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).
    c. If the control is synchronous, issue a SET_CUR Request and verifies that it completes in less than 10 ms. If not fail the test and throw the related assertion (6.20.10).
23. Check range constraints
    a. Issue GET_MIN, GET_MAX, GET_RES, and GET_DEF requests and verify that they meet the following constraints:

b. If P-frame not supported, wQpPrime_P = 0xFFFF, if not throw related assertion ()

c. If B-frame not supported, wQpPrime_B = 0xFFFF, if not throw related assertion ()

d. If H.264, check H.264 fields

    i. wQpPrime_I QP'Y MIN <= MAX, if not throw related assertion (6.24.244)
    ii. wQpPrime_I qPoffset 1 MIN <= MAX, if not throw related assertion (6.24.244)
    iii. wQpPrime_I qPoffset 2 MIN <= MAX, if not throw related assertion (6.24.244)
    iv. wQpPrime_P QP'Y RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    v. wQpPrime_P qPoffset 1 RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    vi. wQpPrime_P qPoffset 2 RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    vii. wQpPrime_B QP'Y RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    viii. wQpPrime_B qPoffset 1 RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    ix. wQpPrime_B qPoffset 2 RES != 0xFFFF and MIN <= MAX, if not throw related assertion (6.24.244)
    x. wQpPrime_I QP'Y MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xi. wQpPrime_I qPoffset 1 MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xii. wQpPrime_I qPoffset 2 MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xiii. wQpPrime_P QP'Y RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xiv. wQpPrime_P qPoffset 1 RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xv. wQpPrime_P qPoffset 2 RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xvi. wQpPrime_B QP'Y RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xvii. wQpPrime_B qPoffset 1 RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xviii. wQpPrime_B qPoffset 2 RES != 0xFFFF and MIN <= DEF <= MAX, if not throw related assertion (6.20.11)
    xix. wQpPrime_I QP'Y RES !=0 and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    xx. wQpPrime_I QP'Y RES !=0 and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
    xxi. wQpPrime_I qPoffset 1 RES !=0 and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    xxii. wQpPrime_I qPoffset 1 RES !=0 and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
    xxiii. wQpPrime_I qPoffset 2 RES !=0 and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    xxiv. wQpPrime_I qPoffset 2 RES !=0 and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
    xxv. wQpPrime_P QP'Y RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    xxvi. wQpPrime_P QP'Y RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
    xxvii. wQpPrime_P qPoffset 1 RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
    xxviii. wQpPrime_P qPoffset 1 RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
    xxix. wQpPrime_P qPoffset 2 RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)

        xxx. wQpPrime_P qPoffset 2 RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
        xxxi. wQpPrime_B QP’Y RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
        xxxii. wQpPrime_B QP’Y RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
        xxxiii. wQpPrime_B qPoffset 1 RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
        xxxiv. wQpPrime_B qPoffset 1 RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)
        xxxv. wQpPrime_B qPoffset 2 RES !=0 and RES != 0xFFFF and (DEF – MIN) / RES is integral, if not throw related assertion (6.20.11)
        xxxvi. wQpPrime_B qPoffset 2 RES !=0 and RES != 0xFFFF and (MAX – MIN) / RES is integral, if not throw related assertion (6.20.13)

24. Check out of bound value
    a. Issue a SET_CUR request with out-of-bound wQpPrime value. Verify that the Quantization Parameters Control answers STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.44). If the Control answer is different from STALL, issue a GET_CUR request. If the Control reports the bogus value, fail the test and throw the related assertion (6.22.11).
25. Check range of values
    a. Issue a SET_CUR Request with valid values for wQpPrime. Verify that the value have been set. If not, fail the test and throw the related assertion (6.24.242).
26. Check SET_CUR/GET_CUR value
    a. Issue a GET_DEF request to get valid value. Issue a SET_CUR request and verify
27. Issue a SET_INTERFACE request
28. Start isoch stream
29. Check lack of support for the control when streaming
    a. Check the bmRuntimeControls field of the Encoding Unit Descriptor. If D9==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip streaming tests (and stop streaming).
30. Repeat steps 19-27
31. Stop isoch stream
32. Issue a SET_INTERFACE request with bAltSetting==0.

TD 21.11

Synchronization and Long Term Reference Frame Control Test

This test verifies that the Synchronization and Long Term Reference Frame Control is compliant with the USBVC Specification.

Device States For Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software has the following UVC Control parameters.

Control selector: EU_SYNC_REF_FRAME_CONTROL

Bitmap index of bmControls and bmRuntimeControls: D10

Size of control parameter block: 4

Mandatory requests: GET_CUR, SET_CUR, GET_MIN, GET_MAX, GET_LEN, GET_INFO, GET_DEF

The test software performs the following steps. For each test, the value of wLayerOrViewID is always set to its default (0) except when the test indicates a different value.

1. Initialize
    a. Execute the Init procedure
    b. Put the device in the desired state
2. Get configuration
    a. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
    b. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
3. Get VICs
    a. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
4. Get VCI
    a. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find Encoding Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_ENCODING_UNIT. If no Encoding Unit Descriptors if found, stop the test. If an Encoding Unit is found, retrieve the Unit ID and begin the test on the Control.
5. For each Encoding Unit found
6. Check absence:
    a. Check the bmControls and bmRuntimeControls fields of the Encoding Unit Descriptor. If D10==0 in both fields, issue a GET_CUR request on the Control and verify that the answer is STALL~~.~~ If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If control is absent, skip to the next Encoding Unit if it exists, else end the test.
7. If D10==1 in either the bmControls or the bmRuntimeControls field (Control is supported), do the following
8. Check GET_LEN
    a. Issue a GET_LEN request:and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Verify that the returned value is 4. If not, fail the test and throw the related assertion (6.20.15).
9. Check GET_INFO
    a. Issue a GET_INFO request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7).
    b. Check if disabled by auto update
    c. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If asynch, check for presence of interrupt endpoint
    e. If the Control is Asynchronous or Autoupdate, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
10. For each streaming bInterfaceNumber, determine if it is connected to the selected Unit ID.
    a. If it is not, then make sure VS_COMMIT_CONTROL issued on this interface does not enable Encoding Units controls, and skip to next interface.
    b. Otherwise
11. For each bFormatIndex that references a descriptor which is compatible with the Encoding Unit, do the following
12. For each bFrameIndex that references a descriptor which is compatible with the Encoding Unit, do the following
13. Set the default dwFrameInterval,.
14. For each supported bUsage, do the following
15. For each supported bmRateControlModes, do the following
16. Perform simple Probe and Commit
    a. Issue a SET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).

    b. Issue a GET_CUR request to VS_PROBE_CONTROL. If request fails, throw the related assertion (6.24.6).
    c. If the above Probe settings have been changed, the Probe setting is not supported, so skip to next Probe setting.
    d. Issue a SET_CUR request to VS_COMMIT_CONTROL. If request fails, throw the related assertion (6.24.6).

17. Check for control disabled by COMMIT.
    a. Request GET_INFO. If it does not succeed, throw related assertion (6.24.5) and skip to next probe/commit setting.
    b. If it succeeds and D5==1 (control is disabled), skip to next probe/commit setting.
18. Check lack of support of the control when not streaming
    a. Check the bmControls field of the Encoding Unit Descriptor. If D10==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip non-streaming tests
19. Check support of mandatory requests
    a. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_LEN, GET_INFO, GET_MIN, GET_MAX, and GET_DEF. Issue all those requests and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). If the answer is STALL. If not STALL, fail the test and throw the related assertion (6.20.43).
20. Check retrieved packets are of the correct length.(6.20.15)
21. Check GET_CUR and SET_CUR
    a. If the control is Asynchronous, verify that Control Change Interrupts are generated on SET_CUR (Timeout 5s). Issue a SET_CUR request and wait during 5 s for a Control Change Interrupt. If no Control Change Interrupt, fail the test and throw the related assertion (6.20.9)
    b. If the control is synchronous, issue a GET_CUR request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.17). If the request did not success, check the Request Error Code Control (issue a GET_CUR Request) and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).
    c. If the control is synchronous, issue a SET_CUR Request and verifies that it completes in less than 10 ms. If not fail the test and throw the related assertion (6.20.10).
22. Check range constraints
    a. Issue GET_MIN, GET_MAX, and GET_DEF requests and verify that they meet the following constraints:
    b. wSyncFrameInterval MIN <= MAX, if not throw related assertion (6.20.14)
    c. bGradualDecoderRefresh MIN <= MAX, if not throw related assertion (6.20.14)
    d. wSyncFrameInterval DEF >= MIN, if not throw related assertion (6.20.11)
    e. wSyncFrameInterval DEF <= MAX, if not throw related assertion (6.20.11)
    f. bGradualDecoderRefresh DEF >= MIN, if not throw related assertion (6.20.11)
    g. bGradualDecoderRefresh DEF <= MAX, if not throw related assertion (6.20.11)
23. Test supported Sync Types (walk through bmSupportedSyncFrameType in Format descriptor)
    a. Case 0 = Reset
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. If supported, verify SyncFrameInterval values succeed. (6.24.284)
    b. Case 1 = IDR
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. if supported, verify SyncFrameInterval values succeed. (6.24.284)
    c. Case 2 = LTR IDR
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. If supported, verify SyncFrameInterval values succeed. (6.24.284)
    d. Case 3 = random access I-frame
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. If supported, verify SyncFrameInterval values succeed. (6.24.284)

    e. Case 4 = random access LTR I-frame
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. If supported, verify SyncFrameInterval values succeed. (6.24.284)
    f. Case 5 = LTR P-frame
        i. If supported, Set GDR != 0, require STALL (6.24.283)
        ii. If supported, verify SyncFrameInterval values succeed. (6.24.284)
    g. Case 6 = GDR
        i. If supported, verify GDR values within bounds succeed. (6.24.285)
        ii. If supported, verify SyncFrameInterval values succeed.
    h. Other

24. Issue a SET_INTERFACE request
25. Start isoch stream
26. Check lack of support for the control when streaming
    a. Check the bmRuntimeControls field of the Encoding Unit Descriptor. If D10==0, issue a GET_CUR request on the Control and verify that the answer is STALL. If not fail the test and throw the related assertions (6.20.3 and 6.20.42).
    b. If not supported, skip streaming tests (and stop streaming).
27. Repeat steps 19-23
28. Stop isoch stream
29. Issue a SET_INTERFACE request with bAltSetting==0.

6.5.3 Video Streaming Control Tests

TD 23.1

Video Probe and Commit Controls Test

This test verifies that the Probe and Commit Controls are compliant with the USBVC Specification.

Device States for Test

This test is run once for each of the following device states: Configured.

Overview of Test Steps

The test software performs the following steps.

(this is a copy of the original to be updated)

1. Execute the Init procedure
2. Put the device in the desired state
3. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
4. Get the Configuration Descriptor for the selected configuration by issuing a Get Configuration Descriptor command with a length of wTotalLength from the data returned in step 3.
5. Parse the configuration Descriptor for VICs. Execute all the steps below for each VIC found in the configuration Descriptor. For the rest of the test only parse Descriptors belonging to the current tested VIC.
6. Parse the Descriptors to find the Video Control Interface Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL).Retrieve the Interface number of this Video Control Interface and parse descriptors to find every Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_*.
7. For every Format Descriptor found, if the format is a frame based format, parse the descriptors to find every associated Frame descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FRAME_*).
8. Verify that the Probe Control supports all mandatory and Request and that the Data Length returned has a correct size. If not, fail the test and throw the related assertions (6.20.15 and 6.23.27)

9. For every Frame (or Format if the Format is not a frame based format) descriptor found, repeat steps 9 to 14 :
10. Get the negotiable fields by retrieving the bmaControls[bFormatIndex] field of the Header Descriptor for the currently tested format. If the Format is Stream-based, verify that the Header does not advertise that the Format support Frame Interval, KeyFrameRate or PFrameRate as a parameter. If not, fail the test and throw the related assertions (6.23.7, 6.23.8 and 6.23.9).
11. Issue a SET_CUR to Probe Control to initialize Format and Frame Index.
12. For every Frame Interval (if applicable, i.e. if the Format is Frame-based):
    a. Issue a SET_CUR in Probe Control with the current values of Format/Frame indexes and Frame Interval (If applicable), all other fields are set to 0.
    b. Issue a GET_MIN to probe Control and store the Min values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.28).
    c. Issue a GET_MAX to probe control and store the Max values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.29). Verify that MIN<=MAX for every parameter. If not, fail the test and throw the related assertions (6.23.11, 6.23.15, 6.23.19 and 6.23.23).
    d. Issue a GET_DEF to probe control and store the Def values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.30). Verify that the value is valid. If not fail the test and throw assertion the related assertion (6.23.12, 6.23.16, 6.23.20 and 6.23.24). Verify the values against the one specified in the Frame/Format Descriptor. If they are not valid, fail the test and throw the related assertions.
    e. Issue a GET_RES to probe control and store the RES value for compression field. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.31). Verify that the RES values are consistent with MIN and MAX. If not, fail the test and throw the related assertion (6.23.13, 6.23.17, 6.23.21, 6.23.25). Verify also that if RES=0, then MIN=MAX. if not, fail the test and throw the related assertions (6.23.14, 6.23.18, 6.23.22, 6.23.26).
    f. For every possible value of the compression parameters, issue a SET_CUR to Probe Control with the bmHint set to zero since we cycle only through Frame Intervals. Then issue a GET_CUR to retrieve the negotiated values. Verify that the Request succeeded, if not fail the test and throw the related assertion (6.23.32). Save the MaxPayloadTransferSize negotiated
    g. Verify that the MaxPaylaodTransferSize is achievable with the values provided in the alternate setting. If not, fail the test and throw the related assertion (6.23.1).
    h. Verify that the negotiated MaxPaylaodTransferSize is lower than the previous negotiated one. If not, fail the test and throw the related assertions (6.23.2, 6.23.3, 6.23.4, 6.23.5 and 6.23.6).
13. For every Frame Interval (if applicable, i.e. if the Format is Frame-based):
    a. Issue a SET_CUR in Probe Control with the current values of Format/Frame indexes and Frame Interval (If applicable), all other fields are set to 0.
    b. Issue a GET_MIN to probe Control and store the Min values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.28).
    c. Issue a GET_MAX to probe control and store the Max values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.29). Verify that MIN<=MAX for every parameter. If not, fail the test and throw the related assertions (6.23.11, 6.23.15, 6.23.19 and 6.23.23).
    d. Issue a GET_DEF to probe control and store the Def values for compression fields. Verify that the value is valid. If not fail the test and throw assertion the related assertion (6.23.30). Verify that the value is valid. If not fail the test and throw assertion the related assertion (6.23.12, 6.23.16, 6.23.20 and 6.23.24). Verify the values against the one specified in the Frame/Format Descriptor. If they are not valid, fail the test and throw the related assertions.

   e. Issue a GET_RES to probe control and store the RES value for compression field. Verify that the RES values are consistent with MIN and MAX. If not, fail the test and throw the related assertion (6.23.31). Verify that the RES values are consistent with MIN and MAX. If not, fail the test and throw the related assertion (6.23.13, 6.23.17, 6.23.21, 6.23.25). Verify also that if RES=0, then MIN=MAX. if not, fail the test and throw the related assertions (6.23.14, 6.23.18, 6.23.22, 6.23.26).
   f. For every negotiable parameter for the current Format/Frame: KeyFrameRate, PFrameRate, CompQuality, CompWindowSize.
      i. For every Value between MAX and MIN, by decrement of RES:
         1. For every possible value of the compression parameters, issue a SET_CUR to Probe Control with the bmHint set to fixed/Variable according to the parameter we will cycle through. Then issue a GET_CUR to retrieve the negotiated values. Verify that the Request succeeded, if not fail the test and throw the related assertion (6.23.32). Save the MaxPayloadTransferSize negotiated
         2. Verify that the MaxPaylaodTransferSize is achievable with the values provided in the alternate setting. If not, fail the test and throw the related assertion (6.23.1).
         3. Verify that the negotiated MaxPaylaodTransferSize is lower than the previous negotiated one. If not, fail the test and throw the related assertions (6.23.2, 6.23.3, 6.23.4, 6.23.5 and 6.23.6).

14. For every Stream-Based Format Descriptor:
   a. Issue a SET_CUR in Probe Control with the current values of Format Index, all other fields being set to 0.
   b. Issue a GET_MIN to probe Control and store the Min values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.28).
   c. Issue a GET_MAX to probe control and store the Max values for compression fields. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.29). Verify that MIN<=MAX for every parameter. If not, fail the test and throw the related assertions (6.23.11, 6.23.15, 6.23.19 and 6.23.23).
   d. Issue a GET_DEF to probe control and store the Def values for compression fields. Verify that the value is valid. If not fail the test and throw assertion the related assertion (6.23.30). Verify that the value is valid. If not fail the test and throw assertion the related assertion (6.23.12, 6.23.16, 6.23.20 and 6.23.24). Verify the values against the one specified in the Frame/Format Descriptor. If they are not valid, fail the test and throw the related assertions.
   e. Issue a GET_RES to probe control and store the RES value for compression field. Verify that the RES values are consistent with MIN and MAX. If not, fail the test and throw the related assertion (6.23.31). Verify that the RES values are consistent with MIN and MAX. If not, fail the test and throw the related assertion (6.23.13, 6.23.17, 6.23.21, 6.23.25). Verify also that if RES=0, then MIN=MAX. if not, fail the test and throw the related assertions (6.23.14, 6.23.18, 6.23.22, 6.23.26).
   f. For every negotiable parameter for the current Format/Frame: CompQuality, CompWindowSize.
      i. For every Value between MAX and MIN, by decrement of RES:
         1. For every possible value of the compression parameters, issue a SET_CUR to Probe Control with the bmHint set to fixed/Variable according to the parameter we will cycle through. Then issue a GET_CUR to retrieve the negotiated values. Verify that the Request succeeded, if not fail the test and throw the related assertion (6.23.32). Save the MaxPayloadTransferSize negotiated
         2. Verify that the MaxPaylaodTransferSize is achievable with the values provided in the alternate setting. If not, fail the test and throw the related assertion (6.23.1).
         3. Verify that the negotiated MaxPaylaodTransferSize is lower than the previous negotiated one. If not, fail the test and throw the related assertions (6.23.2, 6.23.3, 6.23.4, 6.23.5 and 6.23.6).