Universal Serial Bus

Video Class

Compliance Test Specification

Revision 1.1

October 11, 2013

Contributors

Abdul R. Ismail	Intel Corp.
Akihiro Tanabe	Canon Inc.
Allison Hicks	Texas Instruments
Anand Ganesh	Microsoft Corp.
Andy Hodgson	STMicroelectronics
Anshuman Saxena	Texas Instruments
Arnaud Glatron	Logitech Inc.
Bertrand Lee	Microsoft Corp.
Charng Lee	Sunplus Technology Co., Ltd
David Goll	Microsoft Corp.
Eric Luttmann	Cypress Semiconductor Corp.
Fernando Urbina	Apple Computer Inc.
Geert Knapen	Philips Electronics
Geraud Mudry	Logitech Inc.
Hiro Kobayashi	Microsoft Corp.
Jean-Michel Chardon	Logitech Inc.
Jeff Zhu	Microsoft Corp.
Ken-ichiro Ayaki	Fujifilm
Mitsuo Niida	Canon Inc.
Nobuo Kuchiki	Sanyo Electric Co., Ltd
Olivier Lechenne	Logitech Inc.
Paul Thacker	STMicroelectronics

Remy Zimmermann	Logitech Inc.
Shinichi Hatae	Canon Inc.
Steve Miller	STMicroelectronics
Tachio Ono	Canon Inc.
Takashi Sato	Philips Semiconductor
Yoichi Hirata	Matsushita Electric Industrial Co., Ltd

Please send questions to techadmin@usb.org.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

A LICENSE IS HEREBY GRANTED TO REPRODUCE AND DISTRIBUTE THIS SPECIFICATION FOR INTERNAL USE ONLY. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY OTHER INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

AUTHORS OF THIS SPECIFICATION DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. AUTHORS OF THIS SPECIFICATION ALSO DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

Revision History

Version	Date	Description
1.1	May 25, 2006	Initial release

Table of Contents

1 Scope

This document specifies assertions and test procedures for use with Video Class devices. At this time this specification has the following limitations:

- No validation is provided for streaming data (Video data obtained/sent through isochronous or bulk pipes). So payload headers and the data itself are not validated. The main reasons for this are the lack of support for isochronous pipes in the USBCV Framework and the fact that a lot of the expected behavior is subjective.
- No validation is provided for still images. The main reason for this is that to be thorough this test should really be ran while streaming data is being captured and/or sent. So this test will be covered when support for the streaming data test is added.
- Because of the dependency to streaming and still images, compliance to sections 4.3.1.4, 4.3.1.5, 4.3.1.6 and 4.3.1.7 of the USB Device Class Definition for Video Devices specification revision 1.1, are not tested for at this time.
- The tests for the VS_PROBE_CONTROL/VS_COMMIT_CONTROL, VS_STILL_PROBE_CONTROL/VS_STILL_COMMIT_CONTROL and the associated negotiation protocol are limited in scope because of the complexity and the subjectivity of testing coherency and compliance with behavioral requirements.
- No support for verification that Auto-Update controls send Control Change interrupts. The reason being that it is impossible to really know for sure when a device has changed an auto-update setting.

This testing is intended to be in addition to standard USB Compliance testing; assertions covered by the USBCV test document, for instance, are not covered here.

This testing applies to one configuration and all Video Interface Collection at a time. This testing covers the validation of the Video Class-specific Descriptors (including the possible dependencies between them) and of the Video Class-specific control requests (except those explicitly excluded above).

1.1 Open questions

1. How do we address streaming data tests?
2. How do we address still image tests?
3. Should we add support for auto-update behavior validation? If so, how?
4. How far do we want to go in the test of Formats? For example do we want to check Format coherency beyond the USB Video class requirements?

1.2 To be done

1. When all test assertions are documented and all tests described, add matrix listing tests and assertions, to allow for bidirectional mapping of tests and assertions.

2 Related Documents

USB Specification, Revision 2.0, April 27, 2000

USB Device Class Definition for Video Devices, Revision 1.1

USB Device Class Definition for Video Devices Uncompressed Payload, Revision 1.1

USB Device Class Definition for Video Devices Motion -JPEG Payload, Revision 1.1

USB Device Class Definition for Video Devices MPEG2-TS Payload, Revision 1.1

USB Device Class Definition for Video Devices DV Payload, Revision 1.1

USB Device Class Definition for Video Devices Frame-Based Payload, Revision 1.1

USB Device Class Definition fro Video Devices Stream-Based Payload, Revision 1.1

USB Device Class Definition for Video Devices Video Device Example, Revision 1.1

USB Device Class Definition for Video Devices Identifiers, Revision 1.1

USB Device Class Definition for Video Media Transport Terminal, Revision 1.1

3 Terms and Abbreviations

Term	Description
USB Video Specification	USB Device Class Defnition for Video Devices.
VIC	Video Interface Collection
IAD	Interface Association Descriptor
VC	Video Control

4 Test philosophy

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 versus full-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. If the configuration index is larger than the number of configurations available, then the test shall fail.

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.

5 Assertions

5.1 Descriptor related assertions

5.1.1 Descriptor Basic Assertions

Device Descriptor Assertions

Num	Assertion
6.1.1	Device Descriptor contains invalid bDeviceClass / bDeviceSubClass / bDeviceProtocol **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.2 **Test Description:** TD 1.1

Device Qualifier Descriptor Assertions

Num	Assertion
6.1.10	Device_Qualifier Descriptor contains invalid bDeviceClass / bDeviceSubClass / bDeviceProtocol **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.3 **Test Description:** TD 1.2

Interface Association Descriptor Assertions

Num	Assertion
6.1.20	No IAD is found but we have a VideoControl Interface and one or more associated Video Streaming Interface. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3
6.1.21	IAD bLength size is invalid. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3
6.1.22	IAD bInterfaceCount is 0 or 1. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3
6.1.23	IAD bFunctionProtocol is not PC_PROTOCOL_UNDEFINED **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3
6.1.24	IAD iFunction is missing **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.11 **Test Description:** TD 1.3

Interface Association Descriptor Assertions

Num	Assertion
6.1.25	IAD exists but the iFunction field of the IAD does the match the iInterface field of the Standard VC Interface Descriptor for this VIC **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.11 **Test Description:** TD 1.3
6.1.26	IAD bFirstInterface does not point to a Video Control Interface **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3
6.1.27	IAD bInterfaceCount some of the expected Video Streaming Interfaces are not Video Streaming Interfaces **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.6 **Test Description:** TD 1.3

Video Control Interface Descriptor Assertions

Num	Assertion
6.2.1	Video Control Interface has more than just default Endpoint 0 and one interrupt Endpoint **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.2 **Test Description:** TD 1.4
6.2.2	Video Control Interface bNumEndpoints does not match the set of Endpoints found. **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.2 **Test Description:** TD 1.4
6.2.3	Standard VC Interface Descriptor is missing **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.1 **Test Description:** TD 1.4
6.2.4	Standard VC Interface bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.1 **Test Description:** TD 1.4
6.2.5	Standard VC Interface Descriptor bNumEndpoints is more than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.1 **Test Description:** TD 1.4
6.2.6	Standard VC Interface Descriptor bInterfaceProtocol is not PC_PROTOCOL_UNDEFINED **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.1 **Test Description:** TD 1.4
6.2.7	Standard VC Interface Descriptor iInterface is missing **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.1 **Test Description:** TD 1.4

6.2.8 More the one Standard Video Control interface Descriptor has been in the VIC.

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.1

Test Description: TD 1.4

Class Video Control Interface Descriptor Assertions

Num	Assertion
6.2.20	Class VC Interface Descriptor is missing **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.1 **Test Description:** TD 1.5
6.2.21	Class VC Interface Descriptor is misplaced **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.5
6.2.22	Class VC Interface Descriptor bLength is invalid. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.5
6.2.23	Class VC Interface Descriptor wTotalLength is incorrect (after parsing following Descriptors) **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.5
6.2.24	Class VC Interface Descriptor baInterfaceNr(1)…baInterfaceNr(n) not sequential numbers **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.1 **Test Description:** TD 1.5
6.2.25	There is more than one Interface Header Descriptor. **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.1 **Test Description:** TD 1.5
6.2.26	Class VC Interface Descriptor bcdUVC is not 1.10 **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.1 **Test Description:** TD 1.5
6.2.27	Class VC Interface Descriptor, we found an invalid Descriptor while parsing the subsequent descriptors. **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.1 **Test Description:** TD 1.5
6.2.28	Class VC Interface Descriptor baInterfaceNr (1)…baInterfaceNr (n): at least one of these Interfaces is not a Video Streaming Interface. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.5

Input Terminal Descriptor Assertions

Num	Assertion
6.2.40	bUnitID or bTerminalID is 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.7, TD 1.8, TD 1.9, TD 1.10, TD 1.11, TD 1.12, TD 1.13

6.2.41 There is no Input terminal

Specification Ref: USB Video Specification, Revision 1.1, Section 2.3

Test Description: TD 1.6

Input Terminal Descriptor Assertions

Num **Assertion**

6.2.42 Input Terminal Descriptor bLength is invalid for this type

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.1

Test Description: TD 1.6

6.2.43 Input Terminal Descriptor wTerminalType is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.1

Test Description: TD 1.6

6.2.44 Input Terminal bAssocTerminal does not match the Terminal ID of the Output Terminal Found

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.1

Test Description: TD 1.6

Output Terminal Descriptor Assertions

Num **Assertion**

6.2.50 There is no Output Terminal

Specification Ref: USB Video Specification, Revision 1.1, Section 2.3

Test Description: TD 1.7

6.2.51 Output Terminal Descriptor bLength is invalid for this type

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.2

Test Description: TD 1.7

6.2.52 Output Terminal Descriptor wTerminalType is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.2

Test Description: TD 1.7

6.2.53 Output Terminal Descriptor bSourceID is 0

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 1.7

6.2.54 Output Terminal bAssocTerminal does not match the Terminal ID of the Input Terminal Found

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.2

Test Description: TD 1.7

6.2.55 The Source ID referred to in Output Terminal Descriptor is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 2.1

Camera Terminal Descriptor Assertions

Num **Assertion**

6.2.60 Camera Terminal Descriptor bLength is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.3

Test Description: TD 1.8

Camera Terminal Descriptor Assertions

Num	Assertion
6.2.61	Camera Terminal Descriptor bmControls has reserved bits set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2.3 **Test Description:** TD 1.8
6.2.62	Camera Terminal bAssocTerminal does not match the Terminal ID of the Output Terminal Found **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2.3 **Test Description:** TD 1.8

Media Transport Input Terminal Descriptor Assertions

Num	Assertion
6.2.70	Media Transport Input Terminal Descriptor bLength is invalid **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.1 **Test Description:** TD 1.9
6.2.71	Media Transport Input Terminal Descriptor bmControls, a bit D4 or above is set. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.1 **Test Description:** TD 1.9
6.2.72	Media Transport Input Terminal Descriptor bmTransportModes, a bit D34 or above is set. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.1 **Test Description:** TD 1.9
6.2.73	Media Transport Input Terminal bAssocTerminal does not match the Terminal ID of the Output Terminal Found **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.1 **Test Description:** TD 1.9

Media Transport Output Terminal Descriptor Assertions

Num	Assertion
6.2.80	Media Transport Output Terminal Descriptor bLength is invalid. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.2 **Test Description:** TD 1.10
6.2.81	Media Transport Output Terminal Descriptor bSourceID is 0. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.2 **Test Description:** TD 1.10

Media Transport Output Terminal Descriptor Assertions

Num	Assertion
6.2.82	Media Transport Output Terminal Descriptor bmControls, a bit D4 or above is set. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.2 **Test Description:** TD 1.10
6.2.83	Media Transport Output Terminal Descriptor bmTransportModes, a bit D34 or above is set. **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.2 **Test Description:** TD 1.10
6.2.84	Media Transport Output Terminal bAssocTerminal does not match the Terminal ID of the input Terminal Found **Specification Ref:** USB Device Class Definition for Video Media Transport Terminal, Revision 1.1, Section 3.2 **Test Description:** TD 1.10

Selector Unit Descriptor Assertions

Num	Assertion
6.2.90	Selector Unit Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2.4 **Test Description:** TD 1.11
6.2.91	Selector Unit Descriptor baSourceID(1)…baSourceID(p) some IDs are 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.11
6.2.92	Some Source IDs referred to in Selector Unit Descriptor are invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 2.1

Processing Unit Descriptor Assertions

Num	Assertion
6.2.100	Processing Unit Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2.5 **Test Description:** TD 1.12
6.2.101	Processing Unit Descriptor bSourceID is 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.12
6.2.102	Processing Unit Descriptor bmControls: D6 and D7 bits are both set **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.3.5 **Test Description:** TD 1.12
6.2.103	The Source ID referred to in Processing Unit Descriptor is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.12

6.2.104 Processing Unit Descripor bmControls: reserved bits are not set to zero.

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.5

Test Description: TD 1.12

6.2.105 Processing Unit Descripor bmVideoStandards: reserved bits are not set to zero.

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.5

Test Description: TD 1.12

Extension Unit Descriptor Assertions

Num **Assertion**

6.2.110 Extension Unit Descriptor bLength is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2.6

Test Description: TD 1.13

6.2.111 Extension Unit Descriptor bSourceID(0)…bSourceID(p) some Ids are set to 0

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 1.13

6.2.112 Some Source IDs referred to in Extension Unit Descriptors are invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 1.13

Standard Video Control Interrupt Endpoint Descriptor Assertions

Num **Assertion**

6.2.120 Standard VC Interrupt Endpoint Descriptor bLength is invalid.

Specification Ref: USB Video Specification, Revision 1.1, Section 3.8.2.1

Test Description: TD 1.14

6.2.121 Standard VC Interrupt Endpoint Descriptor bEndpointAddress direction is not IN

Specification Ref: USB Video Specification, Revision 1.1, Section 3.8.2.1

Test Description: TD 1.14

6.2.122 Standard VC Interrupt Endpoint Descriptor bEndpointAddress has some bits D6..4 set.

Specification Ref: USB Video Specification, Revision 1.1, Section 3.8.2.1

Test Description: TD 1.14

6.2.123 Standard VC Interrupt Endpoint Descriptor bmAttributes does not have D3=D2=0

Specification Ref: USB Video Specification, Revision 1.1, Section 3.8.2.1

Test Description: TD 1.14

Class Video Control Interrupt Endpoint Descriptor Assertions

Num **Assertion**

6.2.130 Class VC Interrupt Endpoint Descriptor is misplaced or missing

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 1.15

6.2.131 Class VC Interrupt Endpoint Descriptor bLength is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 3.8.2.2

Test Description: TD 1.15

Standard Video Streaming Interface Descriptor Assertions

Num	Assertion
6.3.1	Standard VS Interface Descriptor bNumEndpoints is more than 2 **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16
6.3.2	Standard VS Interface Descriptor bInterfaceProtocol is not PC_PROTOCOL_UNDEFINED **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.1 **Test Description:** TD 1.16
6.3.3	Video Streaming Interface has an invalid set of isochronous Endpoints (0 or more than 1 isochronous Endpoint in alternate setting different from 0) **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16
6.3.4	Video Streaming Interface has an isochronous Endpoint in alternate setting 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16
6.3.5	Video Streaming Interface has an invalid set of bulk Endpoints (1 bulk Video Endpoint is present in alternate Setting 0 and there is other alternate settings for this Interface, or 1 bulk Endpoint is present in alternate setting other than 0 but no isochronous Endpoint is present for that alternate setting, or there is 2 or more bulk Endpoint in an alternate setting.) **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16
6.3.6	Video Streaming Interface has more than one Header Descriptor **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2 **Test Description:** TD 1.16
6.3.7	Video Streaming Interface has Header, Format and Frame Descriptors in alternate setting <>0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2 **Test Description:** TD 1.16
6.3.8	Video Streaming Interface: The Bulk Endpoint should follow the Video Endpoint in Descriptor ordering. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.10.2.1 **Test Description:** TD 1.16
6.3.9	Video Streaming Interface: The Bulk Endpoint should follow the Video Endpoint in Descriptor Addressing. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.10.2.1 **Test Description:** TD 1.16
6.3.10	Video Streaming Interface has an invalid Endpoint in alternate setting other than 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16

Standard Video Streaming Interface Descriptor Assertions

Num	Assertion
6.3.11	Video Streaming Interface bNumEndpoints does not match the set of Endpoints found. **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.3 **Test Description:** TD 1.16

Class Specific Video Streaming Interface Input Header Descriptor Assertions

Num	Assertion
6.3.20	Input Header Descriptor is misplaced **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.1 **Test Description:** TD 1.17
6.3.21	Input Header Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.22	Input Header Descriptor wTotalLength is incorrect (after parsing following Descriptors) **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.23	Input Header Descriptor bEndpointAddress direction is not IN **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.24	Input Header Descriptor bmInfo D7…1 one of these bits is set **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.25	Input Header Descriptor bEndpointAddress has some bits D6..4 set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.26	Input Header Descriptor bTerminalLink is 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.17
6.3.27	Input Header Descriptor bStillCaptureMethod is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.28	Input Header Descriptor bTriggerSupport is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.29	Input Header Descriptor bTriggerUsage is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17

Class Specific Video Streaming Interface Input Header Descriptor Assertions

Num	Assertion
6.3.30	Input Header Descriptor bmaControls(1)…bmaControls(p) a reserved bit is set **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.31	Input Header Descriptor found an invalid Format / Frame Descriptor **Specification Ref:** Related payload specification document **Test Description:** TD 1.17
6.3.32	Input Header Descriptor found duplicate Format index **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.33	Input Header Descriptor found an invalid Still Image Descriptor **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.17
6.3.34	Input Header Descriptor found an invalid Color Matching Descriptor **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.17
6.3.35	Input Header Descriptor missing Still Image Frame Descriptor **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.17
6.3.36	Input Header Descriptor, the Endpoint referred to by bEndpointAddress cannot be found. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.37	Input Header Descriptor, we found an invalid Descriptor type while parsing the subsequent descriptors. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.1 **Test Description:** TD 1.17
6.3.38	The bTerminalLink referred to in Input Header Descriptor is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Sections 3.9.2.1 **Test Description:** TD 1.17

Class Specific Video Streaming Interface Output Header Descriptor Assertions

Num	Assertion
6.3.50	Output Header Descriptor is misplaced **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.1 **Test Description:** TD 1.18
6.3.51	Output Header Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18

Class Specific Video Streaming Interface Output Header Descriptor Assertions

Num	Assertion
6.3.52	Output Header Descriptor wTotalLength is incorrect (after parsing following Descriptors) **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18
6.3.53	Output Header Descriptor bEndpointAddress direction is not OUT **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18
6.3.54	Output Header Descriptor bEndpointAddress has some bits D6..4 set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18
6.3.55	Output Header Descriptor bTerminalLink is 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.7.2 **Test Description:** TD 1.18
6.3.56	Output Header Descriptor found an invalid Format / Frame Descriptor **Specification Ref:** Related payload specification document **Test Description:** TD 1.18
6.3.57	Output Header Descriptor found duplicate Format index **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18
6.3.58	Output Header Descriptor found an invalid Color Matching Descriptor **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.18
6.3.59	Output Header Descriptor, the Endpoint referred to by bEndpointAddress cannot be found. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.2 **Test Description:** TD 1.18
6.3.60	The bTerminalLink referred to in Output Header Descriptor is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Sections 3.9.2.2 **Test Description:** TD 1.18

Still Image Frame Descriptor Assertions

Num	Assertion
6.3.70	Still Image Frame Descriptor is misplaced **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19

Still Image Frame Descriptor Assertions

Num	Assertion
6.3.71	Still Image Frame Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19
6.3.72	Still Image Frame Descriptor bEndpointAddress direction is not IN **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19
6.3.73	Still Image Frame Descriptor bEndpointAddress has some bits D6..4 set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19
6.3.74	Still Image Frame Descriptor, bStillImageCaptureMethod is 2 and bEndpointAddress is not 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19
6.3.75	Still Image Frame Descriptor, the Endpoint referred to in bEndpointAdress can not be found **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.5 **Test Description:** TD 1.19

Color Matching Descriptor Assertions

Num	Assertion
6.3.80	Color Matching Descriptor is misplaced **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.20
6.3.81	Color Matching Descriptor bLength is invalid **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.20
6.3.82	Color Matching Descriptor bColorPrimaries is larger than 5 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.20
6.3.83	Color Matching Descriptor bTransferCharacteristics is larger than 7 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.20
6.3.84	Color Matching Descriptor bMatrixCoefficients is larger than 5 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.9.2.6 **Test Description:** TD 1.20

Standard VS Isochronous Video Data Endpoint Descriptor Assertions

Num	Assertion
6.3.90	Standard VS Isochronous Video Data Endpoint Descriptor bmAttributes: D3..2 is not 01 **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.10.1.1 **Test Description:** TD 1.21
6.3.91	Standard VS Isochronous Video Data Endpoint Descriptor bmAttributes has some bits D7..4 set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.10.1.1 **Test Description:** TD 1.21
6.3.92	Standard VS Isochronous Video Data Endpoint Descriptor bEndpointAddress has some bits D6..4 set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 3.10.1.1 **Test Description:** TD 1.21

5.1.2 Advanced Descriptor Assertions.

Descriptor Advanced Assertions

Num	Assertion

6.4.1 Device Topology has cycles

Specification Ref: USB Video Specification, Revision 1.1, Section 2.3

Test Description: TD 2.1

6.4.2 There are some duplicate Unit / Terminal IDs

Specification Ref: USB Video Specification, Revision 1.1, Section 3.7.2

Test Description: TD 2.1

6.4.20 A VS Interface reports that hardware triggers for still image capture are supported but the VC has no interrupt Endpoint

Specification Ref: USB Video Specification, Revision 1.1, Section 2.4.2.2

Test Description: TD 2.2

5.1.3 Format and Frame Descriptor Assertions

Uncompressed Video Format Descriptor Assertions

Num **Assertion**

6.10.1 Uncompressed Video Format Descriptor bLength is invalid

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.2 Uncompressed Video Format Descriptor guidFormat is not YUY2 nor NV12

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 2.2

Test Description: TD 3.1

6.10.3 Uncompressed Video Format Descriptor bNumFrameDescriptors is 0

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.4 Uncompressed Video Format Descriptor bDefaultFrameIndex is larger than bNumFrameDescriptors

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.5 Uncompressed Video Format Descriptor bmInterlaceFlags, bit D3 is set.

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.6 Uncompressed Video Format Descriptor bmInterlaceFlags D7..D6 are 11

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.7 Uncompressed Video Format Descriptor bCopyProtect is greater than 1

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

6.10.8 Uncompressed Video Format Descriptor found invalid Uncompressed Video Frame Descriptor

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.1

6.10.9 Uncompressed Video Format Descriptor found duplicate Frame indexes

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.1

6.10.10 Uncompressed Video Format Descriptor bNumFrameDescriptors does not match the number of Descriptors.

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.1

6.10.11 Uncompressed Video Format Descriptor bmInterlaceFlags D7..D6 are different than 00

Specification Ref: Uncompressed Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.1

Uncompressed Video Frame Descriptor Assertions

Num	Assertion
6.10.20	Uncompressed Video Frame Descriptor bLength is invalid **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.2
6.10.21	Uncompressed Video Frame Descriptor bmCapabilities is not 0 but VS Interface does not have an IN Video Endpoint or does not use Still Image Capture Method 1 **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.2
6.10.22	Uncompressed Video Frame Descriptor bmCapabilities is greater than 1 **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.2
6.10.23	Uncompressed Video Frame Descriptor dwDefaultFrameInterval is not consistent with supported intervals **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.2
6.10.24	Uncompressed Video Frame Descriptor dwMinFrameInterval > dwMaxFrameInterval **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.2.1 **Test Description:** TD 3.2
6.10.25	Uncompressed Video Frame Descriptor dwFrameIntervalStep > (dwMaxFrameInterval – dwMinFrameInterval) **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.2.1 **Test Description:** TD 3.2
6.10.26	Uncompressed Video Frame Descriptor dwFrameInterval(1)…dwFrameInterval(n) not in increasing order **Specification Ref:** Uncompressed Payload specification, Revision 1.1, Section 3.2.1 **Test Description:** TD 3.2

MJPEG Video Format Descriptor Assertions

Num	Assertion
6.10.40	MJPEG Video Format Descriptor bLength is invalid **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.41	MJPEG Video Format Descriptor bNumFrameDescriptors is 0 **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3

MJPEG Video Format Descriptor Assertions

Num	Assertion
6.10.42	MJPEG Video Format Descriptor bmFlags has some of the bits D7..1 set. **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.43	MJPEG Video Format Descriptor bmInterlaceFlags, bit D3 is set. **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.44	MJPEG Video Format Descriptor bDefaultFrameIndex is larger than bNumFrameDescriptors **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.45	MJPEG Video Format Descriptor bmInterlaceFlags D7..D6 is 11 **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.46	MJPEG Video Format Descriptor bCopyProtect is greater than 1 **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.3
6.10.47	MJPEG Video Format Descriptor found invalid MJPEG Video Frame Descriptor **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.3
6.10.48	MJPEG Video Format Descriptor found duplicate Frame indexes **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.3
6.10.49	MJPEG Video Format Descriptor bNumFrameDescriptors does not match the number of Descriptors. **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.3

MJPEG Video Frame Descriptor Assertions

Num	Assertion
6.10.60	MJPEG Video Frame Descriptor bLength is invalid **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4
6.10.61	MJPEG Video Frame Descriptor bmCapabilities is not 0 but VS Interface does not have an IN Video Endpoint or does not use Still Image Capture Method 1 **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4
6.10.62	MJPEG Video Frame Descriptor bmCapabilities is greater than 1 **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4

MJPEG Video Frame Descriptor Assertions

Num	Assertion
6.10.63	MJPEG Video Frame Descriptor dwDefaultFrameInterval is not consistent with supported intervals **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4
6.10.64	MJPEG Video Frame Descriptor dwMinFrameInterval > dwMaxFrameInterval **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4
6.10.65	MJPEG Video Frame Descriptor dwFrameIntervalStep > (dwMaxFrameInterval – dwMinFrameInterval) **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4
6.10.66	MJPEG Video Frame Descriptor dwFrameInterval(1)…dwFrameInterval(n) not in increasing order **Specification Ref:** MJPEG Payload specification, Revision 1.1, Section 3.1.2 **Test Description:** TD 3.4

MPEG2 TS Video Format Descriptor Assertions

Num	Assertion
6.10.110	MPEG2 TS Format Descriptor bLength is invalid. **Specification Ref:** MPEG2-TS Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.8
6.10.111	MPEG2 TS Format Descriptor bPacketLength!=bStrideLength while having only TSP Packets in the Stream. **Specification Ref:** MPEG2-TS Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.8

DV Format Descriptor Assertions

Num	Assertion
6.10.120	DV Format Descriptor bLength is invalid **Specification Ref:** DV Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.9

DV Format Descriptor Assertions

Num	Assertion
6.10.121	DV Format Descriptor bFormatType Bits D6..3 are set. **Specification Ref:** DV Payload specification, Revision 1.1, Section 3.1.1 **Test Description:** TD 3.9

Stream Based Video Format Descriptor Assertions

Num	Assertion
6.10.170	Stream Based Video Format Descriptor bLength is invalid **Specification Ref:** Stream Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.12
6.10.171	Stream Based Video Format Descriptor bFixedSize is >1. **Specification Ref:** Stream Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.12
6.10.172	Stream Based Video Format Descriptor dwPacketLength is not 0 while bFixedSize is set to zero. **Specification Ref:** Stream Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.12

Frame Based Video Format Descriptor Assertions

Num	Assertion
6.10.190	Frame Based Video Format Descriptor bLength is invalid **Specification Ref:** Frame Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.13
6.10.191	Frame Based Video Format Descriptor bNumFrameDescriptor is zero. **Specification Ref:** Frame Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.13
6.10.192	Frame Based Video Format Descriptor, reserved bits D7..6 in bmInterlaceFlags are set. **Specification Ref:** Frame Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.13
6.10.193	Frame Based Video Format Descriptor, reserved bits D3 in bmInterlaceFlags is set. **Specification Ref:** Frame Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.13
6.10.194	Frame Based Video Format Descriptor, bDefaultFrameIndex is greater than bNumFrameIndex. **Specification Ref:** Frame Based Payload specification, Revision 1.1, Section 3.1.1 **Test Description**: TD 3.13

6.10.195 Frame Based Video Format Descriptor, bCopyProtect is >1.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.13

6.10.196 Frame Based Video Format Descriptor, bVariableSize is >1.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.13

6.10.197 Frame Based Video Format Descriptor, the following bNumFrameDescriptors are not all Frame Based Frame Descriptors.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.13

6.10.198 Frame Based Video Format Descriptor, there is duplicated Frame Indexes among the following Frame Based Frame Descriptors.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.13

6.10.199 Frame Based Video Format Descriptor bNumFrameDescriptors does not match the number of Descriptors.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.1

Test Description: TD 3.13

Frame Based Video Frame Descriptor Assertions

Num **Assertion**

6.10.210 Frame Based Video Frame Descriptor bLength is invalid

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.211 Frame Based Video Frame Descriptor bmCapabilities is not 0 but VS Interface does not have an IN Video Endpoint or does not use Still Image Capture Method 1.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.212 Frame Based Video Frame Descriptor bmCapabilities is greater than 1

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.213 Frame Based Video Frame Descriptor, dwBytesPerLine is not set to zero while bVariableSize is set to 1 in the Format Descriptor.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.214 Frame Based Video Frame Descriptor dwDefaultFrameInterval is not consistent with supported intervals.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.215 Frame Based Video Frame Descriptor dwMinFrameInterval > dwMaxFrameInterval

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.216 Frame Based Video Frame Descriptor dwFrameIntervalStep > (dwMaxFrameInterval – dwMinFrameInterval).

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

6.10.217 Frame Based Video Frame Descriptor dwFrameInterval(1)…dwFrameInterval(n) not in increasing order.

Specification Ref: Frame Based Payload specification, Revision 1.1, Section 3.1.2

Test Description: TD 3.14

5.2 Video control related assertions

Video Control Assertions

Num	Assertion
6.20.1	Video Control responded NAK where STALL was expected **Specification Ref:** USB Video Specification, Revision 1.1, Section 4 & 2.4.4 **Test Description:** All TD.2x.x tests
6.20.2	Video Control did not respond STALL when an invalid value or an unsupported Request was set. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4 & 2.4.4 **Test Description:** All TD.2x.x tests
6.20.3	Found unexpected Video controls (invalid control selectors or on invalid units / terminals) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2 **Test Description:** All TD.20.x tests
6.20.4	An expected Video control is missing (in Interface / unit / terminal) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2 **Test Description:** All TD.20.x tests
6.20.5	An expected control did not behave properly **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.x **Test Description:** All TD.20.x tests
6.20.6	Video Control GET_INFO does not match actual supported requests **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.1.2 **Test Description:** TD 20.1 to TD 20.42
6.20.7	Video Control does not support all mandatory requests **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.x **Test Description:** TD 20.1 to TD 20.42
6.20.8	Video Control is asynchronous or auto update but No Interrupt Endpoint is present. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.x **Test Description:** TD 20.1 to TD 20.42
6.20.9	Video Control is asynchronous but Control Change interrupts are not generated on SET_CUR (timeout is 5s) **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.4 **Test Description:** TD 20.1 to TD 20.42
6.20.10	Video Control is synchronous but SET_CUR has completed in more than 10 ms **Specification Ref:** USB Video Specification, Revision 1.1, Section 2.4.4 **Test Description:** TD 20.1 to TD 20.42

6.20.11 Video Control default is not between MIN and MAX

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

Video Control Assertions

Num **Assertion**

6.20.12 Video Control default is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.13 Video Control GET_LEN is supported but does not match expected length

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.14 Video Control has a MIN>MAX

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.15 The requests does not return the correct size of parameter

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.16 Video Control GET_INFO have D3 set while the control is in Manual mode

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.17 Video Control GET_CUR reported Invalid Value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

6.20.18 Video Control GET_RES reported Invalid Value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: TD 20.1 to TD 20.42

Power Mode Control Assertions

Num **Assertion**

6.20.30 Power Mode Control setting invalid Power Mode

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.1

Test Description: TD 20.1

6.20.31 Power Mode Control GET_CUR reported invalid value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.1

Test Description: TD 20.1

Request Error Control Assertions

Num **Assertion**

6.20.40 Request Error Control Code GET_CUR reported invalid value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

6.20.41 Request Error Control Code did not report an Invalid Unit error after reading invalid unit

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

Request Error Control Assertions

Num **Assertion**

6.20.42 Request Error Control Code did not report an Invalid Control error after reading an invalid control

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

6.20.43 Request Error Control Code did not report an Invalid Request error after issuing an invalid request on a valid Control.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

6.20.44 Request Error Control Code did not report an Out Of Range error after issuing a SET_CUR Request with Out Of Bound value on a valid Control.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

6.20.45 Request Error Control Code did not clear after being read (should be 0 on second read)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.1.2

Test Description: TD 20.2

Scanning Mode Control Assertions

Num **Assertion**

6.20.50 Scanning Mode Control setting a value > 1 succeeded (GET_CUR reported bogus value).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.1

Test Description: TD 20.4

6.20.51 Scanning Mode Control GET_CUR reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.1

Test Description: TD 20.4

Auto-Exposure Mode Control Assertions

Num **Assertion**

6.20.60 Auto-Exposure Mode Control GET_RES returned invalid bitmap (0 or invalid bits set)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.2

Test Description: TD 20.5

6.20.61 Auto-Exposure Mode Control setting invalid bitmap (with modes not listed in GET_RES, invalid modes, or more than 1 bit set) succeeded (GET_CUR reported bogus value).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.2

Test Description: TD 20.5

6.20.62 Auto-Exposure Mode Control GET_CUR reported invalid value (0, invalid bits set, more than one bit set or modes not listed by GET_RES)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.2

Test Description: TD 20.5

Auto-Exposure Mode Control Assertions

Num	Assertion
6.20.63	Auto-Exposure Mode Control SET_CUR failed with a valid value **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.2 **Test Description:** TD 20.5
6.20.64	Exposure Time Absolute Control did not send any Control Change Interrupt when Auto Exposure Mode value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.2 **Test Description:** TD 20.5
6.20.65	Exposure Time Relative Control did not send any Control Change Interrupt when Auto Exposure Mode value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.2 **Test Description:** TD 20.5
6.20.66	Iris Absolute Control did not send any Control Change Interrupt when Auto Exposure Mode value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.2 **Test Description:** TD 20.5
6.20.67	Iris Relative Control did not send any Control Change Interrupt when Auto Exposure Mode value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.2 **Test Description:** TD 20.5

Auto-Exposure Priority Control Assertions

Num	Assertion
6.20.70	Auto-Exposure Priority Control setting a value > 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.3 **Test Description:** TD 20.6
6.20.71	Auto-Exposure Priority Control GET_CUR reported invalid value (>1) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.3 **Test Description:** TD 20.6
6.20.72	Auto-Exposure Priority Control SET_CUR failed with a valid value **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.3 **Test Description:** TD 20.6

Exposure Time (Absolute) Control Assertions

Num	Assertion

6.20.80 Exposure Time (Absolute) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.7

Exposure Time (Absolute) Control Assertions

Num **Assertion**

6.20.81 Exposure Time (Absolute) Control did not return STALL during SET_CUR while Auto-Exposure control was set to Auto or Aperture priority mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.7

6.20.82 Exposure Time (Absolute) Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.7

6.20.83 Exposure Time (Absolute) Control GET_INFO did not have D3 set while Auto-Exposure control was set to Auto or Aperture priority mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.7

6.20.84 Exposure Time (Absolute) Control GET_CUR reported reserved value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.7

Exposure Time (Relative) Control Assertions

Num **Assertion**

6.20.90 Exposure Time (Relative) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.8

6.20.91 Exposure Time (Relative) Control did not return STALL during SET_CUR while Auto-Exposure control was set to Auto or Aperture priority mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.5

Test Description: TD 20.8

6.20.92 Exposure Time (Absolute) Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.4

Test Description: TD 20.8

6.20.93 Exposure Time (Relative) Control GET_INFO did not have D3 set while Auto-Exposure control was set to Auto or Aperture priority mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.5

Test Description: TD 20.8

6.20.94 Exposure Time (Relative) Control GET_CUR reported reserved value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.5

Test Description: TD 20.8

Focus (Absolute) Control Assertions

Num	Assertion
6.20.100	Focus (Absolute) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.6 **Test Description:** TD 20.9
6.20.101	Focus (Absolute) Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.6 Test Description: TD 20.9
6.20.102	Focus (Absolute) Control GET_INFO did not have D2 and D3 set while Focus Auto Control was enabled. Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.6 Test Description: TD 20.9
6.20.103	Focus (Absolute) Control did not send any Control Change notification when the control from Automatic to manual Mode (or from Manual to Auto) Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.6 Test Description: TD 20.9
6.20.104	Focus Absolute is in Auto Mode but it still accepts SET_CUR requests Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.6 Test Description: TD 20.9

Focus (Relative) Control Assertions

Num	Assertion
6.20.110	Focus (Relative) Control setting bFocusRelative to other than 0xFF, 0 or 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.7 **Test Description:** TD 20.10
6.20.111	Focus (Relative) Control GET_MIN, GET_MAX, GET_RES and GET_DEF did not return 0 for bFocusRelative. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.7 **Test Description:** TD 20.10
6.20.112	Focus (Relative) Control SET_CUR succeeded with out-of-bound value for bSpeed (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.7 **Test Description:** TD 20.10
6.20.113	Focus (Relative) Control GET_INFO did not have D3 set while Focus is Auto Control **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.7 **Test Description:** TD 20.10

6.20.114 Focus (Relative) Control did not send any Control Change notification when the control from Automatic to manual Mode (or from Manual to Auto)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

Focus (Relative) Control Assertions

Num **Assertion**

6.20.115 Focus (Relative) Control did not send any Control Change Interrupt when Focus Relative value changed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

6.20.116 Focus (Relative) Control has a MIN>MAX for bSpeed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

6.20.117 Focus (Relative) Control has an invalid DEF value for bSpeed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

6.20.118 Focus (Relative) SET_CUR Failed with a valid value for bSpeed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

6.20.119 Focus (Relative) SET_CUR Failed with a valid value for bFocusRelative

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

6.20.120 Focus (Relative) is in Auto Mode but it still accepts SET_CUR requests

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.7

Test Description: TD 20.10

Focus Auto Control Assertions

Num **Assertion**

6.20.125 Focus Auto Control setting a value > 1 succeeded (GET_CUR reported bogus value).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.8

Test Description: TD 20.11

6.20.126 Focus Auto Control GET_CUR reported invalid value (>1).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.8

Test Description: TD 20.11

6.20.127 Focus Auto Control SET_CUR failed with a valid value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.8

Test Description: TD 20.11

Iris (Absolute) Control Assertions

Num	Assertion
6.20.130	Iris (Absolute) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.9 **Test Description:** TD 20.12

Iris (Absolute) Control Assertions

Num	Assertion
6.20.131	Iris (Absolute) Control did not return STALL during SET_CUR while Auto-Exposure control was set to Auto or Shutter priority mode. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.9 **Test Description:** TD 20.12
6.20.132	Iris (Absolute) Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.9 **Test Description:** TD 20.12
6.20.133	Iris (Absolute) Control GET_INFO did not have D3 set while Auto-Exposure control was set to Auto or Aperture Shutter mode. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.9 **Test Description:** TD 20.12
6.20.134	Iris (Absolute) Control GET_INFO did not have D2 and D3 set while Focus Auto Control was enabled. Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.9 Test Description: TD 20.12

Iris (Relative) Control Assertions

Num	Assertion
6.20.140	Iris (Relative) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.10 **Test Description:** TD 20.13
6.20.141	Iris (Relative) Control did not return STALL during SET_CUR while Auto-Exposure control was set to Auto or Shutter priority mode. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.10 **Test Description:** TD 20.13
6.20.142	Iris (Relative) Control SET_CUR succeeded with Out-Of-Bound value (GET_CUR reported Bogus Value). **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.10 **Test Description:** TD 20.13

6.20.143 Iris (Relative) Control GET_INFO did not have D3 set while Auto-Exposure control was set to Auto or Aperture priority mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.10

Test Description: TD 20.13

Zoom (Absolute) Control Assertions

Num **Assertion**

6.20.150 Zoom (Absolute) Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.11

Test Description: TD 20.14

Zoom (Absolute) Control Assertions

Num	Assertion
6.20.151	Zoom (Absolute) Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.11 **Test Description:** TD 20.14
6.20.152	Zoom (Absolute) Control GET_RES did not return 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.11 **Test Description:** TD 20.14

Zoom (Relative) Control Assertions

Num	Assertion
6.20.160	Zoom (Relative) Control setting bZoom to other than 0xFF, 0 or 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.161	Zoom (Relative) Control GET_MIN, GET_MAX, GET_RES and GET_DEF did not return 0 for bZoom. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.162	Zoom (Relative) Control SET_CUR succeeded with value other than 0 or 1 for bDigitalZoom (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.163	Zoom (Relative) Control GET_MIN, GET_MAX, GET_RES did not return 0 for bDigitalZoom. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.164	Zoom (Relative) Control SET_CUR succeeded with out-of-bound value for bSpeed (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.165	Zoom Absolute Control did not send any Control Change Interrupt when Focus Relative value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15
6.20.166	Zoom (Relative) Control has a MIN>MAX for bSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.12 **Test Description:** TD 20.15

6.20.167 Zoom (Relative) Control has an invalid DEF value for bSpeed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.12

Test Description: TD 20.15

Zoom (Relative) Control Assertions

Num **Assertion**

6.20.168 Zoom (Relative) Control SET_CUR failed with a valid value for bZoom

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.12

Test Description: TD 20.15

6.20.169 Zoom (Relative) Control SET_CUR failed with a valid value for bDigitalZoom

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.12

Test Description: TD 20.15

6.20.189 Zoom (Relative) Control SET_CUR failed with a valid value for bSpeed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.12

Test Description: TD 20.15

Pan Tilt (Absolute) Control Assertions

Num **Assertion**

6.20.170 Pan Tilt (Absolute) Control GET_MIN for dwPanAbsolute is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.171 Pan Tilt (Absolute) Control GET_MAX for dwPanAbsolute is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.172 Pan Tilt (Absolute) Control GET_DEF for dwPanAbsolute is not 0

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.173 Pan Tilt (Absolute) Control GET_MIN for dwTiltAbsolute is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.174 Pan Tilt (Absolute) Control GET_MAX for dwTiltAbsolute is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.175 Pan Tilt (Absolute) Control GET_DEF for dwTiltAbsolute is not 0

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.176 Pan Tilt (Absolute) Control SET_CUR succeeded with out-of-bound value for dwPanAbsolute (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

6.20.177 Pan Tilt (Absolute) Control SET_CUR succeeded with out-of-bound value for dwTiltAbsolute (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.1.13

Test Description: TD 20.16

Pan Tilt (Absolute) Control Assertions

Num	Assertion
6.20.178	Pan Tilt (Absolute) Control SET_CUR failed setting dwPanAbsolute and dwTiltAbsolute (error or a different value was returned during following GET_CUR) with a valid value **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.13 **Test Description:** TD 20.16
6.20.179	Pan Tilt (Absolute) Control has a MIN>MAX for dwPanAbsolute. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.13 **Test Description:** TD 20.16
6.20.180	Pan Tilt (Absolute) Control has a MIN>MAX for dwTiltAbsolute. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.13 **Test Description:** TD 20.16
6.20.181	Pan Tilt (Absolute) Control did not send any Control Change Interrupts when its value changed via PanTilt Relative Control. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.13 **Test Description:** TD 20.16

Pan Tilt (Relative) Control Assertions

Num	Assertion
6.20.190	Pan Tilt (Relative) Control setting bPanRelative to other than 0xFF, 0 or 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.191	Pan Tilt (Relative) Control GET_MIN, GET_MAX, GET_RES and GET_DEF did not return 0 for bPanRelative. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.192	Pan Tilt (Relative) Control SET_CUR succeeded with out-of-bound value for bPanSpeed (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.193	Pan Tilt (Relative) Control setting bTiltRelative to other than 0xFF, 0 or 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.194	Pan Tilt (Relative) Control GET_MIN, GET_MAX, GET_RES and GET_DEF did not return 0 for bTiltRelative. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17

Pan Tilt (Relative) Control Assertions

Num	Assertion
6.20.195	Pan Tilt (Relative) Control SET_CUR succeeded with out-of-bound value for bTiltSpeed (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.196	Pan Tilt (Relative) Control has a MIN>MAX for bPanSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.197	Pan Tilt (Relative) Control has a MIN>MAX for bTiltSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.198	PanTilt (Relative) Control has an invalid DEF value for bPanSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.199	PanTilt (Relative) Control has an invalid DEF value for bTiltSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.200	SET_CUR Failed with a valid value for bPanRelative **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.201	SET_CUR Failed with a valid value for bPanSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.202	SET_CUR Failed with a valid value for bTiltRelative **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17
6.20.203	SET_CUR Failed with a valid value for bTiltSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.14 **Test Description:** TD 20.17

Roll (Absolute) Control Assertions

Num	Assertion
6.20.210	Roll (Absolute) Control SET_CUR succeeded with out-of-bound value for dwRollAbsolute (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.15 **Test Description:** TD 20.18
6.20.211	Roll (Absolute) Control SET_CUR failed setting dwRollAbsolute (error or a different value was returned during following GET_CUR) to 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.15 **Test Description:** TD 20.18

Roll (Absolute) Control Assertions

Num	Assertion
6.20.212	Roll (Absolute) Control GET_DEF for dwRollAbsolute is not 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.15 **Test Description:** TD 20.18

Roll (Relative) Control Assertions

Num	Assertion
6.20.220	Roll (Relative) Control setting bRollRelative to other than 0xFF, 0 or 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.221	Roll (Relative) Control GET_MIN, GET_MAX, GET_RES and GET_DEF did not return 0 for bRollRelative. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.222	Roll (Relative) Control SET_CUR succeeded with out-of-bound value for bSpeed (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.223	Roll (Relative) Control did not send any Control Change Interrupt when Focus Relative value changed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.224	Roll (Relative) Control has a MIN>MAX for bSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.225	Roll (Relative) Control has an invalid DEF value for bSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.226	Roll (Relative) Control SET_CUR Failed with a valid value for bFocusRelative **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19
6.20.227	Roll (Relative) Control SET_CUR Failed with a valid value for bSpeed **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.16 **Test Description:** TD 20.19

Privacy Control Assertions

Num	Assertion
6.20.230	Privacy Control GET_CUR reported invalid value (>1) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.17 **Test Description:** TD 20.20

Privacy Control Assertions

Num	Assertion
6.20.231	Privacy Control is not reported as AutoUpdate. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.17 **Test Description:** TD 20.20
6.20.232	Privacy Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.17 **Test Description:** TD 20.20
6.20.233	Privacy Control SET_CUR failed with a valid value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.17 **Test Description:** TD 20.20
6.20.234	Privacy Control did not send any Control Change Interrupt when bPrivacy value changed. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.1.17 **Test Description:** TD 20.20

Selector Unit Control Assertions

Num	Assertion
6.21.1	Selector Unit Control GET_MIN is not 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.2	Selector Unit Control GET_RES is not 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.3	Selector Unit Control GET_MAX is not consistent with number of Input Pins **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.4	Selector Unit Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.5	Selector Unit GET_CUR reported invalid value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.6	Selector Unit SET_CUR failed with a valid value **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21
6.21.7	A Selector Unit Descriptor is present but the Selector Unit Control is not supported **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.2 **Test Description:** TD 20.21

Backlight Compensation Control Assertions

Num	Assertion
6.22.1	Backlight Compensation Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.1 **Test Description:** TD 20.22
6.22.2	Backlight Compensation Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.1 **Test Description:** TD 20.22

Brightness Control Assertions

Num	Assertion
6.22.10	Brightness Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.2 **Test Description:** TD 20.23
6.22.11	Brightness Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.2 **Test Description:** TD 20.23
6.22.12	Brightness Control GET_RES reported other than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.2 **Test Description:** TD 20.23

Contrast Control Assertions

Num	Assertion
6.22.20	Contrast Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.3 **Test Description:** TD 20.24
6.22.21	Contrast Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.3 **Test Description:** TD 20.24
6.22.22	Contrast Control GET_RES reported other than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.3 **Test Description:** TD 20.24

Gain Control Assertions

Num	Assertion
6.22.30	Gain Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.4 **Test Description:** TD 20.25
6.22.31	Gain Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.4 **Test Description:** TD 20.25
6.22.32	Gain Control GET_RES reported other than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.4 **Test Description:** TD 20.25

Power Line Frequency Control Assertions

Num	Assertion
6.22.40	Power Line Frequency Control setting a value > 2 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.5 **Test Description:** TD 20.26
6.22.41	Power Line Frequency Control GET_CUR reported invalid value (>2) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.5 **Test Description:** TD 20.26
6.22.42	Power Line Frequency Control GET_DEF reported invalid value (!=1) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.5 **Test Description:** TD 20.26
6.22.43	Power Line Frequency Control SET_CUR did not success with valid value (<3) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.5 **Test Description:** TD 20.26

Hue Control Assertions

Num	Assertion
6.22.50	Hue Control GET_MIN for wHue is out-of-bound **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.6 **Test Description:** TD 20.27
6.22.51	Hue Control GET_MAX for wHue is out-of-bound **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.6 **Test Description:** TD 20.27
6.22.52	Hue Control GET_DEF for wHue is not 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.6 **Test Description:** TD 20.27
6.22.53	Hue Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to 0 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.6 **Test Description:** TD 20.27
6.22.54	Hue Control SET_CUR succeeded with out-of-bound value for wHue (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.6 **Test Description:** TD 20.27

Hue Auto Control Assertions

Num	Assertion
6.22.60	Hue Control GET_INFO did not have D3 set while Hue Auto control was set to 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.7 **Test Description:** TD 20.28
6.22.61	Hue Auto Control setting a value > 1 succeeded (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.7 **Test Description:** TD 20.28
6.22.62	Hue Auto Control GET_CUR reported invalid value (>1) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.7 **Test Description:** TD 20.28

6.22.63 Hue Auto Control GET_DEF reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.7

Test Description: TD 20.28

6.22.64 Hue Auto Control SET_CUR did not succeed with a value valid (<=1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.7

Test Description: TD 20.28

6.22.65 Hue Control did not send any Control Change Interrupt when Hue Auto value changed.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.7

Test Description: TD 20.28

Hue Auto Control Assertions

Num **Assertion**

6.22.66 Hue Control did not update his SET_CUR constraint when Hue Auto value changed.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.7

Test Description: TD 20.28

Saturation Control Assertions

Num **Assertion**

6.22.70 Saturation Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.8

Test Description: TD 20.29

6.22.71 Saturation Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.8

Test Description: TD 20.29

6.22.72 Saturation Control GET_RES reported other than 1

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.8

Test Description: TD 20.29

Sharpness Control Assertions

Num **Assertion**

6.22.80 Sharpness Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.9

Test Description: TD 20.30

6.22.81 Sharpness Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.9

Test Description: TD 20.30

6.22.82 Sharpness Control GET_RES reported other than 1

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.9

Test Description: TD 20.30

Gamma Control Assertions

Num **Assertion**

6.22.90 Gamma Control GET_MIN for wGamma is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.10

Test Description: TD 20.31

6.22.91 Gamma Control GET_MAX for wGamma is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.10

Test Description: TD 20.31

Gamma Control Assertions

Num **Assertion**

6.22.92 Gamma Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.10

Test Description: TD 20.31

6.22.93 Gamma Control SET_CUR succeeded with out-of-bound value for wGamma (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.10

Test Description: TD 20.31

White Balance Temperature Control Assertions

Num **Assertion**

6.22.100 White Balance Temperature Control GET_MIN for wWhiteBalanceTemperature is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.11

Test Description: TD 20.32

6.22.101 White Balance Temperature Control GET_MAX for wWhiteBalanceTemperature is out-of-bound

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.11

Test Description: TD 20.32

6.22.102 White Balance Temperature Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.11

Test Description: TD 20.32

6.22.103 White Balance Temperature Control SET_CUR succeeded with out-of-bound value for wWhiteBalanceTemperature (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.11

Test Description: TD 20.32

White Balance Temperature Auto Control Assertions

Num **Assertion**

6.22.110 White Balance Temperature Control GET_INFO did not have D3 set while White Balance Temperature Auto control was set to 1

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

6.22.111 White Balance Temperature Auto Control setting a value > 1 succeeded (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

6.22.112 White Balance Temperature Auto Control GET_CUR reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

White Balance Temperature Auto Control Assertions

Num **Assertion**

6.22.113 White Balance Temperature Auto Control GET_DEF reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

6.22.114 SET_CUR did not Succeed with a valid value (GET_CUR Reported Bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

6.22.115 White Balance Temperature Control did not send any Control Change Interrupt when White Balance Temperature Auto value changed.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

6.22.116 White Balance Temperature Control did not accept SET request after have been set to manual Mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.12

Test Description: TD 20.33

White Balance Component Control Assertions

Num **Assertion**

6.22.120 White Balance Component Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set wWhiteBalanceBlue and wWhiteBalanceRed to default (GET_DEF)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.121 White Balance Component Control SET_CUR succeeded with out-of-bound value for wWhiteBalanceBlue (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.122 White Balance Component Control SET_CUR succeeded with out-of-bound value for wWhiteBalanceRed (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.123 White Balance Component Control has a MIN>MAX for wWhiteBalanceBlue

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.124 White Balance Component Control has a MIN>MAX for wWhiteBalanceRed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.125 White Balance Component Control has an invalid DEF value for wWhiteBlanceBlue

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

White Balance Component Control Assertions

Num Assertion

6.22.126 White Balance Component Control has an invalid DEF value for wWhiteBlanceRed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.127 White Balance Component SET_CUR failed with a valid value for wWhiteBalanceBlue

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

6.22.128 White Balance Component SET_CUR failed with a valid value for wWhiteBalanceRed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.13

Test Description: TD 20.34

White Balance Component Auto Control Assertions

Num Assertion

6.22.130 White Balance Component Control GET_INFO did not have D3 set while White Balance Component Auto control was set to 1

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.131 White Balance Component Auto Control setting a value > 1 succeeded (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.132 White Balance Component Auto Control GET_CUR reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.133 White Balance Component Auto Control GET_DEF reported invalid value (>1)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.134 SET_CUR did not Succeed with a valid value (GET_CUR Reported Bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.135 White Balance Component Control did not send any Control Change Interrupt when White Balance Component Auto value changed.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

6.22.136 White Balance Component Control did not accept SET request after have been set to manual Mode.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.2.2.3.14

Test Description: TD 20.35

Digital Multiplier Control Assertions

Num	Assertion
6.22.140	Digital Multiplier Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.15 **Test Description:** TD 20.36
6.22.141	Digital Multiplier Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.15 **Test Description:** TD 20.36
6.22.142	Digital Multiplier Control GET_RES reported other than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.15 **Test Description:** TD 20.36

Digital Multiplier Limit Control Assertions

Num	Assertion
6.22.150	Digital Multiplier Limit Control SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.16 **Test Description:** TD 20.37
6.22.151	Digital Multiplier Limit Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.16 **Test Description:** TD 20.37
6.22.152	Digital Multiplier Limit Control GET_RES reported other than 1 **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.16 **Test Description:** TD 20.37

Analog Video Standard Control Assertions

Num	Assertion
6.22.155	Analog Video Standard Control returned an invalid value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.17 **Test Description:** TD 20.43

Analog Video Lock Status Control Assertions

Num	Assertion
6.22.157	Analog Video Lock Status Control returned an invalid value. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.3.18 **Test Description:** TD 20.44

Extension Unit: Extension Unit Control Assertions

Num	Assertion
6.22.160	An expected Extension Unit Control is missing. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.4 **Test Description:** TD 20.38
6.22.161	An Extension Unit Control number should not be supported(not listed in bmControls) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.4 **Test Description:** TD 20.38
6.22.162	Extension Unit Control number should not be supported(no control Selector above bNumControl should be supported) **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.2.2.4 **Test Description:** TD 20.38

Media Transport Terminal: Transport Control Assertions

Num	Assertion
6.22.170	Transport Control SET_CUR failed with a valid Mode(error or a different value was returned during following GET_CUR. **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39
6.22.171	Transport Control SET_CUR succeeded with reserved value for Playback Mode (GET_CUR reported bogus value) **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39
6.22.172	Transport Control SET_CUR succeeded with reserved value for Rewind Mode (GET_CUR reported bogus value) **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39
6.22.173	Transport Control SET_CUR succeeded with reserved value for Record Mode (GET_CUR reported bogus value) **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39
6.22.174	Transport Control SET_CUR succeeded with reserved value for Eject Mode (GET_CUR reported bogus value) **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39
6.22.175	Transport Control SET_CUR succeeded with reserved value for Status Mode (GET_CUR reported bogus value) **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1 **Test Description:** TD 20.39

6.22.176 Transport Control SET_CUR succeeded with a Status Mode Value.

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1

Test Description: TD 20.39

6.22.177 Transport Control GET_CUR reported an invalid Value (The mode is not listed in bmTransportMode).

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1

Test Description: TD 20.39

6.22.178 Transport Control is not reported as Asynchronous.

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1

Test Description: TD 20.39

Media Transport Terminal: Transport Control Assertions

Num **Assertion**

6.22.179 Transport Control is not reported as Autoupdate.

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1

Test Description: TD 20.39

6.22.180 Transport Control SET_CUR succeeded with an unsupported Mode (not listed in bmTransportModes).

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.1

Test Description: TD 20.39

Media Transport Terminal: Absolute Track Number (ATN) Control Assertions

Num **Assertion**

6.22.190 Absolute Track Number Control GET_CUR reported invalid value for bmMediumType (one of the reserved values have been returned).

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

6.22.191 Absolute Track Number Control GET_CUR reported invalid value for dw_ATNData (one of the reserved values have been returned).

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

6.22.192 Absolute Track Number Control SET_CUR failed (error or a different value was returned during following GET_CUR).

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

6.22.193 Absolute track Number Control SET_CUR succeeded with reserved value for bmMediumType (GET_CUR reported bogus value)

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

6.22.194 Absolute track Number Control SET_CUR succeeded with reserved value for dwATN_Data (GET_CUR reported bogus value)

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

6.22.195 Absolute Track Number is not reported as Asynchronous

Specification Ref: USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2

Test Description: TD 20.40

Media Transport Terminal: Absolute Track Number (ATN) Control Assertions

Num	Assertion
6.22.196	Absolute Track Number is not reported as Autoupdate **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2 **Test Description:** TD 20.40
6.22.197	Absolute Track Number: Transport Control did not send any Control Change Interrupt when ATN Information value changed **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.2 **Test Description:** TD 20.40

Media Transport Terminal: Media Information Control Assertions

Num	Assertion
6.22.200	Media Information Control GET_CUR reported invalid value for bmMediaType (one of the reserved values have been returned). **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.3 **Test Description:** TD 20.41
6.22.201	Media Information Control GET_CUR reported invalid value for bmWriteProtect (one of the reserved value is returned). **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.3 **Test Description:** TD 20.41

Media Transport Terminal: Time Code Information Control Assertions

Num	Assertion
6.22.210	Time Code Information Control, SET_CUR Failed with a valid value. **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.4 **Test Description:** TD 20.42
6.22.211	Time Code Information Control is not reported as Asynchronous. **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.4 **Test Description:** TD 20.42
6.22.212	Time Code Information Control is not reported as AutoUpdate. **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.4 **Test Description:** TD 20.42
6.22.213	Time Code Information Control Transport Control did not send any Control Change Interrupt when Time Code Information value changed. **Specification Ref:** USB Device Class for Video Media Transport Terminal, Revision 1.1, Section 4.1.3.4 **Test Description:** TD 20.42

5.3 Streaming control related assertions

Probe and Commit Controls Assertions

Num	Assertion
6.23.1	The Max Payload Transfer Size specified by the device is not achievable with the Endpoints MaxPacketSizes specified in the Alternate Settings **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.2	The Max Payload Transfer Size specified by the Probe Control is not lower than the previous when we cycle through Frame Interval. This will lead in Negotiation Loops. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.3	The Max Payload Transfer Size specified by the Probe Control is not lower than the previous when we cycle through KeyFrameRate. This will lead in Negotiation Loops. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.4	The Max Payload Transfer Size specified by the Probe Control is not lower than the previous when we cycle through PFrameRate. This will lead in Negotiation Loops. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.5	The Max Payload Transfer Size specified by the Probe Control is not lower than the previous when we cycle through wCompQuality. This will lead in Negotiation Loops. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.6	The Max Payload Transfer Size specified by the Probe Control is not lower than the previous when we cycle through dwCompWindowSize. This will lead in Negotiation Loops. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.7	The Format is stream based. The VS Header shall not advertise that it supports Frame Interval as a parameter. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1
6.23.8	The Format is stream based. The VS Header shall not advertise that it supports Key frame as a parameter. **Specification Ref:** USB Video Specification, Revision 1.1, Section 4.3.1.1.1 **Test Description:** TD 23.1

6.23.9 The Format is stream based. The VS Header shall not advertise that it supports P Frame as a parameter

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.10 The Frame Interval step cannot be 0. Use Discrete Frame Intervals if there is only one frame interval.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.11 MIN > MAX for wKeyFrameRate.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.12 The Probe Control has an invalid default value for wKeyFrameRate

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.13 The RES value of the control is not valid for wKeyFrameRate.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.14 The RES value is 0 for wKeyFrameRate but MIN is not equal to MAX.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.15 MIN > MAX for wPFrameRate.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.16 The Probe Control has an invalid default value for wPFrameRate.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.17 The RES value of the control is not valid for wPFrameRate.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.18 The RES value is 0 for wPFrameRate but MIN is not equal to MAX.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.19 MIN > MAX for wCompQuality.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.20 The Probe Control has an invalid default value for wCompQuality.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.21 The RES value of the control is not valid for wCompQuality.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1

Test Description: TD 23.1

6.23.22 The RES value is 0 for wCompQuality but MIN is not equal to MAX.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.23 MIN > MAX for wCompWindowSize.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.24 The Probe Control has an invalid default value for wCompWindowSize.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.25 The RES value of the control is not valid for wCompWindowSize.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.26 The RES value is 0 for wCompWindowSize but MIN is not equal to MAX.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.27 The Probe Control does not implement all mandatory requests.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.28 Probe Control GET_MIN failed
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.29 Probe Control GET_MAX failed
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.30 Probe Control GET_RES failed
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.31 Probe Control GET_DEF failed
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

6.23.32 Probe Control GET_CUR failed
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.1.1
Test Description: TD 23.1

Still Probe and Still Commit Controls Assertions

Num Assertion

6.23.50 The Still Probe Control does not implement all mandatory requests.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2
Test Description: TD 23.2

6.23.51 Still Probe Control GET_MIN failed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.52 Still Probe Control GET_MAX failed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.53 MIN > MAX for bCompressionIndex.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.54 MIN > MAX for dwMaxVideoFrameSize.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.55 Still Probe Control GET_DEF failed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.56 DEF is not between MIN and MAX for bCompressionIndex.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.57 DEF is not between MIN and MAX for dwMaxVideoFrameSize.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.58 Still Probe Control GET_CUR failed

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.59 The Max Payload Transfer Size specified by the device is not achievable with the Endpoints MaxPacketSizes specified in the Alternate Settings

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.60 The Max Payload Transfer Size specified by the Still Probe Control is not lower than the previous when we cycle through bCompressionIndex. This will lead in Negotiation Loops.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.61 The dwMaxVideoFrameSize returned by the Device is not achievable with the values specified in the Still Image Frame Descriptor.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.62 SET_CUT to Still Commit Control did not succeed.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

6.23.63 The values returned by the Still Probe Control do not match any value in the Still Image Frame Descriptor.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.2

Test Description: TD 23.2

Common Stream Header Assertions

Num	Assertion
6.23.80	The FID bit does not toggle at the beginning of each frame. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5
6.23.81	The EOF bit is set for the first Stream Header of the Frame. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5
6.23.82	PTS is advertised as present but the PTS value is either NULL or inconsistent with the Clock Frequency. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5
6.23.83	SCR is advertised as present but the SCR value is either NULL or inconsistent with the Clock Frequency. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5
6.23.84	STI bit is set in a Non-Still Video Stream Header. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5, TD23.7, TD23.8
6.23.85	The RES bit is set while the value is Reserved. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5, TD23.7, TD23.8
6.23.86	The EOF bit is not set in the last Stream Header of a frame. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5
6.23.87	The FID bit is not constant trough all the Stream Headers of the current Frame. **Specification Ref:** USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2 **Test Description:** TD 23.3, TD23.4, TD23.5

6.23.88 The PTS field is present but the PTS value is not equal to the PTS value of the first Stream Header of the current Frame.

Specification Ref: USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD 23.3, TD23.4, TD23.5

6.23.89 The ERR bit is set but the Stream Error Code Control indicates that there is no error (returns 0).

Specification Ref: USB Video Class Payload Specification (MJPEG, DV, Uncompressed, MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD 23.3, TD23.4, TD23.5, TD23.7, TD23.8

Stream Based Format Specific Stream Header Assertions

Num **Assertion**

6.23.100 The PTS field is not set to zero.

Specification Ref: USB Video Class Payload Specification (MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD23.7, TD23.8

6.23.101 The SCR field is not set to zero

Specification Ref: USB Video Class Payload Specification (MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD23.7, TD23.8

6.23.102 EOH bit is not set to 1.

Specification Ref: USB Video Class Payload Specification (MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD23.7, TD23.8

6.23.103 The bit D0 of the bmFramingInfo in the Probe and Commit structure is not set but the FID bit is set (to indicate the beginning of a codec specific segment).

Specification Ref: USB Video Class Payload Specification (MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD23.7, TD23.8

6.23.104 The bit D1 of the bmFramingInfo in the Probe and Commit structure is not set but the EOF bit is set (to indicate the end of a codec specific segment).

Specification Ref: USB Video Class Payload Specification (MPEG2TS, MPEG1SS/MPEP2PS), Revision 1.1, Section 2.2

Test Description: TD23.7, TD23.8

Video Streaming Controls Assertions

Num **Assertion**

6.30.1 Video Streaming Control did not respond STALL when an invalid value or an unsupported Request was set.

Specification Ref: USB Video Specification, Revision 1.1, Section 4 & 2.4.4

Test Description: All TD.23.10 to 23.13 tests

6.30.2 An expected Video Streaming control is missing.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3

Test Description: All TD.23.10 to 23.13 tests

6.30.3 Video Streaming Control GET_INFO does not match actual supported requests

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

Video Streaming Control Assertions

Num **Assertion**

6.30.4 Video Streaming Control does not support all mandatory requests

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.x

Test Description: All TD.23.10 to 23.13 tests

6.30.5 Video Streaming Control is asynchronous or auto update but No Interrupt Endpoint is present.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.x

Test Description: All TD.23.10 to 23.13 tests

6.30.6 Video Streaming Control is asynchronous but Control Change interrupts are not generated on SET_CUR (timeout is 5s)

Specification Ref: USB Video Specification, Revision 1.1, Section 2.4.4

Test Description: All TD.23.10 to 23.13 tests

6.30.7 Video Streaming Control is synchronous but SET_CUR has completed in more than 10 ms

Specification Ref: USB Video Specification, Revision 1.1, Section 2.4.4

Test Description: All TD.23.10 to 23.13 tests

6.30.8 Video Streaming Control default is not between MIN and MAX

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.9 Video Streaming Control default is invalid

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.10 Video Streaming Control GET_LEN is supported but does not match expected length

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.11 Video Streaming Control has a MIN>MAX

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.12 Video Streaming Control: The requests does not return the correct size of parameter

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.13 Video Streaming Control GET_CUR reported Invalid Value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

6.30.14 Video Streaming Control GET_RES reported Invalid Value

Specification Ref: USB Video Specification, Revision 1.1, Section 4.1.2

Test Description: All TD.23.10 to 23.13 tests

Synch Delay Control Assertions

Num **Assertion**

6.30.30 Synch Delay Control SET_CUR succeeded with out-of-bound value (GET_CUR reported bogus value)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.3

Test Description: TD 23.10

6.30.31 Synch Delay Control is supported and SET_CUR failed (error or a different value was returned during following GET_CUR) to set to default (GET_DEF) value or any other valid value in the range.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.3

Test Description: TD 23.10

Still Image Trigger Control Assertions

Num **Assertion**

6.30.50 Still Image Trigger Control, there is no support for Still Image Capture (No VS Input Header, no IN Endpoint) but the Control is supported.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.51 Still Image Trigger Control, the device advertises Still Image Capture method 1 and the control is supported.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.52 Still Image Trigger Control, SET_CUR succeeded with an Out Of Bound value for the control.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.53 Still Image Trigger Control, the device advertises Still Image Capture Method 2 and a SET_CUR succeeded with a value of 2 (corresponding to Still Image Capture Method 3)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.54 Still Image Trigger Control, the device advertises Still Image Capture Method 3 and a SET_CUR succeeded with a value of 1 (corresponding to Still Image Capture Method 2)

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.55 Still Image Trigger Control, a SET_CUR failed with a value of 0.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.56 Still Image Trigger Control, a SET_CUR failed a valid value corresponding to the Still Image Capture Method used.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.57 Still Image Trigger Control, the control did not send a Still Image when set to transmit Still Image.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.58 Still Image Trigger Control, the control did not automatically return to the normal operation mode after transmission of the Still Image.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.59 Still Image Trigger Control, resetting the control to Normal Operation during transmission of a Still Image did not result in the abortion of the transmission of the Image.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

6.30.60 Still Image Trigger Control, the Stream Error Code Control did not answer Still Image Capture Error (0x07) when the transmission of the still Image has been aborted.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.4

Test Description: TD 23.11

Generate Key Frame Control Assertions

Num **Assertion**

6.30.70 Generate Key Frame Control, there is no IN endpoint and the control is supported.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.5

Test Description: TD 23.12

6.30.71 Generate Key Frame Control, a SET_CUR succeeded with an Out Of Bound value (2).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.5

Test Description: TD 23.12

6.30.72 Generate Key Frame Control, a SET_CUR failed with a valid value (1).

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.5

Test Description: TD 23.12

6.30.73 Generate Key Frame Control, the control was set to Generate Key Frame but no Key Frame has been generated.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.5

Test Description: TD 23.12

6.30.74 Generate Key Frame Control, the control did not automatically return to the normal operation mode after transmission of the Key Frame.

Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.5

Test Description: TD 23.12

Update Frame Segment Control Assertions

Num **Assertion**

6.30.90 Update Frame Segment Control, there is no IN endpoint and the control is supported.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.6
Test Description: TD 23.13

Update Frame Segment Control Assertions

Num **Assertion**

6.30.91 Update Frame Segment Control, a SET_CUR succeeded with an Out Of Bound value.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.6
Test Description: TD 23.13

6.30.92 Update Frame Segment Control, a SET_CUR failed with a valid value.
Specification Ref: USB Video Specification, Revision 1.1, Section 4.3.1.6
Test Description: TD 23.13

6 Description of tests

6.1 Organization of tests

Every test is independent. We decided to implement all test independently in order for a user to be able to run every test individually without any side effect from the other tests. Another reason for that choice was to remain consistent with other Tests implemented in the USB Command Verifier Test Tool.

6.2 Output Format:

All tests output their results to the same text base log file. Each test shall output a header in the form:

```
*********************************************************************

Test: xx.yy       (Configuration# cc / VIC# zz)

Test description: <description of test as listed in this specification>

Date: MM/DD/YY      Start Time: hh:mm:ss (hh is in 24 hour Format)

*********************************************************************
```

Where "xx.yy" is the test number, "cc" the 1 based index of the configuration under test and "zz" the 1 based index of the VIC under test.

If the test runs multiple iterations of itself with different parameters/device states, the test should output a sub-header for each iteration in the following form:

```
**************************************************************************

Iteration# x

Iteration Description: <list what makes the iteration different>

Date: MM/DD/YY      Start Time: hh:mm:ss (hh is in 24 hour Format)

**************************************************************************
```

Where "x" is the iteration number.

The (sub)-header should be followed by a list of assertions (as defined above) in the form of:

```
-> 6.xx.yyy: <description of assertion as listed in this specification>
```

Where "6.xx.yyy" is the assertion number.

Finally the test should log (for each iteration):

```
<Passed>  End Time: hh:mm:ss (hh is in 24 hour Format)
```

Or

```
<FAILED>  End Time: hh:mm:ss (hh is in 24 hour Format)
```

Depending on its success or failure.

6.3 General Procedures

In general, that is unless specified otherwise like for sub-tests in hierarchical tests, the Init procedure (see below) is always called before a test to restore the device state to a known state.

When an Endpoint is stalled (except for protocol stalls on the Video Control Endpoint), the reset Endpoint procedure (see below) shall be called.

If a device is frozen, the “Unfreeze” device procedure (see below) shall be called.

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

The test shell shall take the 1-based index of the configuration to be tested as an input parameter and report which configuration is being tested at the beginning of the test. If the configuration index is larger than the number of configurations available, then the test shall fail.

6.5 Test details

6.5.1 Descriptor tests

6.5.1.1 Basic Descriptor Tests

TD 1.1 **Device Descriptor Test**

This test verifies that the device Descriptor is 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. Get the device Descriptor by issuing a 'get device Descriptor' command with a length of 18 bytes
4. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
5. 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 4.
6. Parse the configuration Descriptor for Interface Association Descriptors (IAD).
7. If no IAD are found, then verify that bDeviceClass==0, bDeviceSubClass==0 and bDeviceProtocol==0.
8. If at least one IAD is found, then verify that bDeviceClass==0xEF, bDeviceSubClass==0x02 and bDeviceProtocol==0x01.
9. If either 7 or 8 fail the test and throw related assertion (6.1.1).

TD 1.2 **Device_Qualifier Descriptor Test**

This test verifies that the device_qualifier Descriptor is 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. Get the device Descriptor by issuing a 'get device_qualifier Descriptor' command with a length of 10 bytes. If this request fails exit the test with no errors.
4. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
5. 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 4.
6. Parse the configuration Descriptor for Interface Association Descriptors (IAD).
7. If no IAD are found, then verify that bDeviceClass==0, bDeviceSubClass==0 and bDeviceProtocol==0.
8. If at least one IAD is found, then verify that bDeviceClass==0xEF, bDeviceSubClass==0x02 and bDeviceProtocol==0x01.
9. If either 7 or 8 fail the test and throw related assertion (*6.1.10*).

TD 1.3 **Interface Association Descriptor test**

This test verifies that the IAD corresponding to the VIC being tested is 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 Interface Association Descriptors (IAD).
6. Verify that that if no IAD is found, we do not have a VideoControl Interface with one or more Video Streaming Interfaces associated, if not fail the test and throw the related assertion(6.1.20).
7. When the IAD corresponding to the current VIC is found (bFunctionClass==CC_VIDEO and bFunctionSubClass==SS_VIDEO_INTERFACE_COLLECTION) check that:
    a. bLength==8. If not, fail the test and throw the related assertion (*6.1.21*)
    b. bInterfaceCount >= 2, if not fail the test and throw related assertion (6.1.22)
    c. bFunctionProtocol==PC_PROTOCOL_UNDEFINED, if not fail the test and throw related assertion (6.1.23)
8. Try to get the string Descriptor identified by iFunction with LANGID=0x409 by issuing a Get String Descriptor request. If the request fails, fail the test and throw related assertion (6.1.24).
9. Fetch the next Descriptor (Video Control Interface Descriptor) and check that:
    a. The iInterface field matches the iFunction field of our IAD. If not, fail the test and throw the related assertion (6.1.25).
    b. The bInterfaceNumber of the Video Control Interface matches the bFirstInterface field of our IAD. If not, fail the test and throw the related assertion (6.1.26).
10. Fetch all the VS Interface Descriptors belonging to the VIC and verify that their number matches the bInterfaceCount of our IAD. If not, fail the test and throw the related assertion (6.1.27).

TD 1.4 **Video Control Interface Test**

This test verifies that the Video Control Interface does not have incorrect Endpoints.

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 start of the Video Control I/F (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOCONTROL). If this Interface cannot be found exit the test without failing. Another test will fail later. For the rest of the test only parse Descriptors belonging to this Interface.
7. Parse Descriptors looking for Endpoint Descriptors.
8. If more than one interrupt Endpoint Descriptors are found or if other Endpoint Descriptors are found, fail the test and throw related assertion (6.2.1).
9. On the VCI Descriptor, check that:
    a. bLength==9. If not, fail the test and throw the related assertion (6.2.4).
    b. bNumEndpoints<=1, if not fail the test and throw related assertion (6.2.5).
    c. bNumEndpoints match the number of physical Endpoints. If not, fail the test and throw the related assertion (6.2.2).
    d. bInterfaceProtocol==PC_PROTOCOL_UNDEFINED if not, fail the test and throw related assertions (6.2.6).
    e. Try to get the string Descriptor identified by iInterface with LANGID=0x409 by issuing a Get String Descriptor request. If the request fails, fail the test and throw related assertion (6.2.7).

TD 1.5 **Class Video Control Interface Descriptor Test**

This test verifies that the Class Video Control Interface Descriptor corresponding to the VIC being tested is 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 (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_HEADER). If the Descriptor cannot be found then report this Descriptor as missing (assertion) and exit the test with a failure (6.2.20).
7. The Class Video Control Interface Descriptor found should be just after the Standard Video Control Interface Descriptor. If not fail the test and throw related assertion (6.2.21). Keep the position (with the offset from the beginning of the descriptors) on the Class Video Control Interface Descriptor found, but continue parsing to the end of the VIC. If another Class Video Control Interface Descriptor is found, fail the test and throw related assertion (6.2.21).
8. For the Class Video Control Interface Descriptor found check that:
    a. bLength is correct by verifying that bLength==12+ bInCollection. If not fail the test and throw related assertion (6.2.22).
    b. bcdUVC == 1.10 or bcdUVC == 1.0. If bcdUVC == 1.0, display a warning that the specification has been replaced by specification 1.1.
    If not, fail the test and throw the related assertion (6.2.26).
    c. wTotalLength is correct. This is done by parsing all the Descriptors of type Unit or Terminal that follow this Descriptor and adding their length together (t). For parsing we have to check that bDescriptorType==CS_INTERFACE and (bDescriptorSubType==VC_INPUT_TERMINAL or bDescriptorSubType==VC_OUTPUT_TERMINAL or bDescriptorSubType==VC_SELECTOR_UNIT_TERMINAL or bDescriptorSubType==VC_PROCESSING_UNIT or bDescriptorSubType==VC_EXTENSION_UNIT). Stop parsing when reaching a Descriptor that is not a Unit or Terminal Descriptor (bDescriptorType==ENDPOINT). wTotalLength should be: bLength + t. If not fail the test and throw related assertion (6.2.23 or 6.2.27).
    d. Verify that baInterfaceNr(1)…baInterfaceNr(n)[(x+1)== x + 1] are all sequential numbers. If not fail the test and throw related assertion (6.2.24).
    e. Parse the descriptor to find Interface header Descriptors, if more than one found, fail the test and throw the related assertion (6.2.25).
    f. For each baInterfaceNr verify that it corresponds to a VS Interface (bDescriptorType==INTERFACE and bInterfaceSubClass=SC_VIDEOSTREAMING and bAlternateSetting=0). We count only alternate setting 0 since it is the only alternate setting always present. If not, fail the test and throw the related assertion (6.2.28).

TD 1.6 **Input Terminal Descriptor Test.**

This test verifies that Input Terminal 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 Input Terminal Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_INPUT_TERMINAL. If no found, fail the test and throw the related assertion (6.2.41).
8. For each Input Terminal Descriptor verify that:
    a. bLength==8+x (X TO FIND IT DEPENDS ON THE INPUT TERMINAL TYPE). If not fail the test and throw related assertion (6.2.42).
    b. bAssocTerminal refers to an existing Output terminal. If not, fail the test and throw the related assertion (6.2.44).
    c. bTerminalID!=0. If not fail the test and throw related assertion(6.2.40).
    d. wTerminalType==ITT_VENDOR_SPECIFIC or wTerminalType==ITT_CAMERA or wTerminalType==ITT_MEDIA_TRANSPORT_INPUT. If not fail the test and throw the related assertion (6.2.43).

TD 1.7 **Output Terminal Descriptor Test**

This test verifies that Output Terminal 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 an Output Terminal Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_OUTPUT_TERMINAL). If not found, fail the test and throw the related assertion (6.2.50).
8. For the Output Terminal Descriptor found, verify that:
    a. bLength==9+x (X TO FIND IT DEPENDS ON THE INPUT TERMINAL TYPE). If not fail the test and throw related assertion (6.2.51).
    b. bTerminalID!=0. If not fail the test and throw related assertion (6.2.40).
    c. bAssocTerminal refers to an existing input Terminal. If not, fail the test and throw the related assertion (6.2.54).
    d. wTerminalType==OTT_VENDOR_SPECIFIC or wTerminalType==OTT_DISPLAY or wTerminalType==OTT_MEDIA_TRANSPORT_OUTPUT. If not fail the test and throw the related assertion (6.2.52).
    e. bSourceID!=0. If not fail the test and throw related assertion (6.2.53).
    f. Fetch the Unit or Terminal Descriptor corresponding to bSourceID. If not found, fail the test and throw the related assertion (6.2.55).

TD 1.8 **Camera Terminal Descriptor test**

This test verifies that Camera Terminal 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 Camera Terminal Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_INPUT_TERMINAL and wTerminalType==ITT_CAMERA).
8. For each Camera Terminal Descriptor verify that:
    a. bLength==15+bControlSize and bControlSize<=3. If not fail the test and throw related assertion (6.2.60).
    b. bTerminalID!=0. If not fail the test and throw related assertion (6.2.40).
    c. bAssocTerminal refers to an existing Output terminal. If not, fail the test and throw the related assertion (6.2.62).
    d. If bControlSize==3, check that bit 15*8+19 to (15+n)*8 == 0.If not, fail the test and throw the related assertion (6.2.61)

TD 1.9 **Media Transport Input Terminal Descriptor test**

This test verifies that Camera Terminal 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 Media Transport Input Terminal Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_INPUT_TERMINAL and wTerminalType==ITT_MEDIA_TRANSPORT_INPUT).
8. For each Media Transport Input Terminal Descriptor verify that:
    a. bLength==10+bControlSize+bTransprotModesize. If not fail the test and throw related assertion (6.2.70).
    b. bTerminalID!=0. If not fail the test and throw related assertion (6.2.40).
    c. bAssocTerminal refers to an existing Output terminal. If not, fail the test and throw the related assertion (6.2.73).
    d. In bmControls, no bits D4 or above is set. If not, fail the test and throw the related assertion (6.2.71).
    e. In bmTransportModes, no bits D34 or above is set. If not, fail the test and throw the related assertion (6.2.72).

TD 1.10 **Media Transport Output Terminal Descriptor test**

This test verifies that Camera Terminal 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 Media Transport Output Terminal Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_INPUT_TERMINAL and wTerminalType==ITT_MEDIA_TRANSPORT_OUTPUT).
8. For each Media Transport Output Terminal Descriptor verify that:
    a. bLength==11+bControlSize+bTransprotModesize. If not fail the test and throw related assertion (6.2.80).
    b. bTerminalID!=0. If not fail the test and throw related assertion (6.2.40).
    c. bAssocTerminal refers to an existing Input terminal. If not, fail the test and throw the related assertion (6.2.84).
    d. bSourceID!=0. If not, fail the test and throw the related assertion (6.2.81).
    e. In bmControls, no bits D4 or above is set. If not, fail the test and throw the related assertion (6.2.82).
    f. In bmTransportModes, no bits D34 or above is set. If not, fail the test and throw the related assertion (6.2.83).

TD 1.11 Selector Unit Descriptor Test

This test verifies that Selector 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 Selector Unit Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_SELECTOR_UNIT).
8. For each Selector Unit Descriptor verify that:
    a. bLength==6+bNrInPins. If not fail the test and throw related assertion (6.2.90).
    b. bUnitID!=0. If not fail the test and throw related assertion (6.2.40).
    c. baSourceID(1)!=0….. baSourceID(bNrInPins)!=0. If not fail the test and throw related assertion (6.2.91).
    d. For every baSourceID(1)..baSourceID(bNrInPins), fetch the Unit or Terminal Descriptor corresponding to baSourceID. If not found, fail the test and throw the related assertion (6.2.92).

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. For UVC 1.0 devices:
    bLength==9+bControlSize and bControlSize<=2. If not fail the test and throw related assertion (6.2.100).
    For UVC 1.1 devices:
    bLength==10+bControlSize and bControlSize<=3. 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. For UVC 1.1 devices: bmControls D18..D23 are set to zero. If not fail the test and throw related assertion (6.2.104)
    f. For UVC 1.1 devices: bmVideoStandards D6-D7 are set to zero. If not fail the test and throw related assertion (6.2.105)
    g. Fetch the Unit or Terminal Descriptor corresponding to bSourceID. If not found, fail the test and throw the related assertion (6.2.103).

TD 1.13 Extension Unit Descriptor Test

This test verifies that Extension 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 Extension Unit Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_EXTENSION_UNIT).
8. For each Extension Unit Descriptor verify that:
    a. bLength==24+bNrInPins+bControlSize. If not fail the test and throw related assertion (6.2.110).
    b. bUnitID!=0. If not fail the test and throw related assertion (6.2.40).
    c. baSourceID(1)!=0….. baSourceID(bNrInPins)!=0. If not fail the test and throw related assertion (6.2.111).
    d. For every baSourceID(1)..baSourceID(bNrInPins), fetch the Unit or Terminal Descriptor corresponding to baSourceID. If not found, fail the test and throw the related assertion (6.2.112).

TD 1.14 Standard VC Interrupt Endpoint Descriptor Test

This test verifies that the Standard VC Interrupt Endpoint Descriptor in the VIC being tested is 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 the Standard VC Interrupt Endpoint Descriptor (bDescriptorType==ENDPOINT).
8. For the Standard VC Interrupt Endpoint Descriptor found check that:
    a. D7 in bEndpointAddress==1. If not fail the test and throw related assertion (6.2.121).
    b. D6..4 in bEndpointAddress==0. If not, fail the test and throw the related assertion (6.2.122).
    c. bLength=7. If not, fail the test and throw related assertions (6.2.120).
    d. D3..2==00 and D1..0==00 in bmAttributes. If not fail the test and throw related assertion (6.2.123).

TD 1.15 Class VC Interrupt Endpoint Descriptor

This test verifies that the Class Specific VC Interrupt Endpoint Descriptor in the VIC being tested is 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 Interrupt Endpoint Descriptor (bDescriptorType==CS_ENDPOINT and bDescriptorSubType==EP_INTERRUPT).
7. The Class Video Control Interrupt Endpoint Descriptor found should be just after the Standard Video Control Interrupt Endpoint Descriptor. If not fail the test and throw related assertion (6.2.130).
8. For the Class VC Interrupt Endpoint Descriptor found check that:
    a. bLength==5. If not, fail the test and throw related assertion (6.2.131).

TD 1.16 VS Interface Descriptor Test

This test verifies general rules on layout of the Descriptors for a VS Interface and its alternate settings specified in the Video Class Specification.

Device States For Test

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

This test is then run once for each configuration and once for each VS Interface.

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).
7. For each Standard VS Interface found check that:
    a. bNumEndpoints<=2. If not, fail the test and throw related assertion (6.3.1).
    b. bNumEndpoints match the physical set of Endpoints. If not fail the test and throw the related assertion (6.3.11)
    b. bInterfaceProtocol==PC_PROTOCOL_UNDEFINED. If not, fail the test and throw related assertion (6.3.2).
    c. If bAlternateSettings==0, parse Descriptors belonging to the class VS Interface Descriptor to find an isochronous Endpoint (bDescriptorType ==ENDPOINT and bmAttributes==0101). If found, fail the test and throw the related assertion (6.3.4).
    d. If bAlternateSettings==0, parse Descriptors belonging to the class specific VS Interface Descriptor to find a bulk Endpoint (bDescriptorType==ENDPOINT and bmAttributes==10). If found only one verify that the Interface has no other alternate setting, if other alternate setting fail the test and throw the related assertion. If more than one bulk Endpoint, fail the test and throw related assertion (6.3.5).
    e. If bAlternateSettings==0, parse Descriptors belonging to the class VS Interface Descriptor to find an Input or Output Header (bDescriptorType ==CS_INTERFACE and (bDescriptorSubType ==VS_INPUT_HEADER or VS_OUTPUT_HEADER)). If more than one or no found, fail the test and throw the related assertion (6.3.6).
    f. If bAlternateSettings!=0, parse Descriptors belonging to the class specific VS Interface Descriptor to find Endpoint Descriptors (bDescriptorType==ENDPOINT). If more than two Descriptors or zero Descriptors found, fail the test and throw the related assertion (6.3.1). If two Descriptors found, check that bmAttributes==0101 in one of them and bmAttributes==10 in the other, if not, fail the test and throw related assertion (6.3.3). Verify also that the bulk Endpoint follow the Video Endpoint in Descriptor Ordering and Descriptor Addressing (6.3.8 and 6.3.9). If not, fail the test and throw the related assertions. If only one found, check that bmAttributes==1010, if not fail the test and throw related assertion (6.3.10).
    g. If bAlternateSettings!=0, Parse Descriptors belonging to the class specific VS Interface Descriptor to find Header, Format or Frame Descriptors (bDescriptorType ==CS_INTERFACE and (bDescriptorSubType==VS_*_HEADER or bDescriptorSubType==VS_FORMAT_* or bDescriptorSubType==VS_FRAME_*).If found fail the test and throw related assertion (6.3.7).

TD 1.17 **Class VS Interface Input Header Descriptor.**

This test verifies that input Header and subsequent Format/Frame related Descriptors are compliant with the UVC specification.

The assumption in this test is that the Format Indexes of the Format Descriptors are all sequential Numbers.

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 VS Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubClass==VS_INPUT_HEADER).). If this Header cannot be found exit the test without failing. Another test has been performed, so we will have an output header instead.
7. The Class VS Interface Input Header Descriptor found should be just after the Standard VS Interface Descriptor. If not fail the test and throw related assertion (6.3.20).
8. For the Input Header found, check that:
    a. bLength==13+(bNumFormats*bControlSize). If not, fail the test and throw the related assertions (6.3.21).
    b. D7 in bEndpointAddress==1. If not, fail the test and throw the related assertion (6.3.23).
    c. D6..4 in bEndpointAddress==0. If not, fail the test and throw related assertion (6.3.25)
    d. bEndpointAddress refers to an existing Endpoint. If not, fail the test and throw the related assertion (6.3.36)
    e. D7..D1 in bmInfo==0. If not, fail the test and throw the related assertion (6.3.24).
    f. wTotalLength==bLength+x where x is the sum of the bLength of the subsequent Descriptors. To find the accurate subsequent Descriptors, parse all Descriptors where bDescriptorType=CS_INTERFACE and stop when you find a Standard VS Interface Descriptor (bDescriptorType==INTERFACE). If not, fail the test and throw related assertions (6.3.22).
    g. bTerminalLink!=0. If not, fail the test and throw related assertion (6.3.26).
    h. bStillCaptureMethod<=3. If not, fail the test and throw related assertions (6.3.27).
    i. bTriggerSupport<=1. If not, fail the test and throw related assertions (6.3.28).
    j. bTriggerUsage<=1. If not, fail the test and throw related assertions (6.3.29).
    k. Bits D6..(bControlSize*8-1) of bmaControls(1)..bmaControls(bNumFormats) == 0. If not, fail the test and throw the related assertion (6.3.30).
    l. Verify that bTerminalLink refers to an existing Output Terminal. If not, fail the test and throw the related assertion (6.3.38).

TD 1.17 **Test 1.17 Continuation**

m. Parse all subsequent Descriptors in sequence with bDescriptorType==CS_INTERFACE, for all Formats found we will perform the tests depending on the type of the Format Descriptor Found:

i. For all Formats: check that bFormatIndex is less than bNumFormats and that they are all different. If not, fail the test and throw the related assertion (6.3.32).

ii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_UNCOMPRESSED or VS_FORMAT_MJPEG, parse all subsequent Descriptors until another Format is found. If a Still Image Frame Descriptor found, verify that it is unique and that bStillCaptureMethod is 2 or 3. If not, fail the test and throw the related assertion (6.3.35).
If bStillCaptureMethod is 0 or 1, verify that there is no Still Image Frame Descriptor. If not, fail the test and throw the related assertion (6.3.33).
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion (6.3.34).
If other than Frame Uncompressed (or MJPEG depending on the first condition), Still Image, Color matching Descriptors are found, fail the test and throw related assertion (6.3.31).

iii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_FRAME_BASED, parse all subsequent Descriptors until another Format is found. If a Still Image Frame Descriptor found, verify that it is unique and that bStillCaptureMethod is 2 or 3. If not, fail the test and throw the related assertion (6.3.35).
If bStillCaptureMethod is 0 or 1, verify that there is no Still Image Frame Descriptor. If not, fail the test and throw the related assertion (6.3.33).
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion (6.3.34).
If other than Frame Based, Still Image, Color matching Descriptors are found, fail the test and throw related assertion (6.3.31).

iv. For all Descriptors found with bDescriptorSubType=VS_FORMAT_VENDOR, verify that it is followed only by Frame Vendor Descriptors. If not, fail the test and throw related assertion (6.3.31).

v. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG1, parse all subsequent Descriptors until another Format is found.
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion (6.3.34).
If other than another Format, Color matching Descriptors are found, fail the test and throw related assertion (6.3.31).

vi. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG2TS, verify that it is followed only by another Format Descriptor. If not, fail the test and throw related assertion (6.3.31).

vii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG2PS, verify that it is followed only by another Format Descriptor. If not, fail the test and throw related assertion (6.3.31).

viii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_STREAM_BASED, verify that it is followed only by another Format Descriptor. If not, fail the test and throw related assertion (6.3.31).

ix. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG4SL, verify that it is followed only by another Format Descriptor. If not, fail the test and throw related assertion (6.3.31).

x. For all Descriptors found with bDescriptorSubType=VS_FORMAT_DV, verify that it is followed only by Frame DV Descriptors. If not, fail the test and throw related assertion (6.3.31).

TD 1.18 **Class VS Interface Output Header Descriptor.**

This test verifies that Output Header and subsequent Format/Frame related Descriptors are compliant with the UVC 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 VS Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDecsriptorSubClass==VS_OUTPUT_HEADER). If this Header cannot be found exit the test without failing. Another test has been performed, so we will have an output header instead.
7. The Class VS Interface Output Header Descriptor found should be just after the Standard VS Interface Descriptor. If not fail the test and throw related assertion (6.3.50).
8. For the Input Header found, check that:
    a. bLength==8. If not, fail the test and throw the related assertions (6.3.51).
    b. D7 in bEndpointAddress==0. If not, fail the test and throw the related assertion (6.3.53).
    c. D6..4 in bEndpointAddress==0. If not, fail the test and throw the related assertion (6.3.54).
    d. bEndpointAddress refers to an existing Endpoint. If not, fail the test and throw the related assertion (6.3.59).
    e. wTotalLength==bLength+x where x is the sum of the bLength of the subsequent Descriptors. To find the accurate subsequent Descriptors, parse all Descriptors where bDescriptorType=CS_INTERFACE and stop when you find a Standard VS Interface Descriptor (bDescriptorType==INTERFACE). If not, fail the test and throw related assertions (6.3.52).
    f. bTerminalLink!=0. If not, fail the test and throw related assertion (6.3.55).
    g. bTerminalLink refers to an existing Input Terminal. If not fail the test and throw the related assertion (6.3.60).

TD 1.18 **Class VS Interface Output Header Descriptor (Continuation).**

h. Parse all subsequent Descriptors in sequence with bDescriptorType==CS_INTERFACE, for all Formats found we will perform the tests depending on the type of the Format Descriptor Found:

i. For all Formats: check that bFormatIndex is les than bNumFormats and that they are all different. If not, fail the test and throw the related assertion (6.3.57).

ii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_UNCOMPRESSED or VS_FORMAT_MJPEG, parse all subsequent Descriptors until another Format is found. If a Still Image Frame Descriptor found, verify that it is unique and that bStillCaptureMethod is 2 or 3. If not, fail the test and throw the related assertion (6.3.56).
If bStillCaptureMethod is 0 or 1, verify that there is no Still Image Frame Descriptor. If not, fail the test and throw the related assertion (6.3.56).
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion (6.3.58).
If other than Frame Uncompressed (or MJPEG depending on the first condition), Still Image, Color matching Descriptors are found, fail the test and throw related assertion (6.3.56).

iii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_FRAME_BASED, parse all subsequent Descriptors until another Format is found. If a Still Image Frame Descriptor found, verify that it is unique and that bStillCaptureMethod is 2 or 3. If not, fail the test and throw the related assertion (6.3.56). If bStillCaptureMethod is 0 or 1, verify that there is no Still Image Frame Descriptor. If not, fail the test and throw the related assertion (6.3.56).
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion (6.3.58).
If other than Frame Base Frame, Still Image, Color matching Descriptors are found, fail the test and throw related assertion (6.3.56).

iv. For all Descriptors found with bDescriptorSubType=VS_FORMAT_VENDOR, verify that it is followed only by Frame Vendor Descriptors. If not, fail the test and throw related assertion (6.3.56).

v. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG1, parse all subsequent Descriptors until another Format is found.
If a Color Matching Descriptor is found, verify that it is unique and well placed. If not, fail the test and throw the related assertion.
If other than Format, Color matching Descriptors are found, fail the test and throw related assertion (6.3.56).

vi. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG2TS, verify that it is followed only by another Format Descriptors. If not, fail the test and throw related assertion (6.3.56).

vii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG2PS, verify that it is followed only by another Format Descriptors. If not, fail the test and throw related assertion (6.3.56).

viii. For all Descriptors found with bDescriptorSubType=VS_FORMAT_MPEG4SL, verify that it is followed only by another Descriptors. If not, fail the test and throw related assertion (6.3.56).

ix. For all Descriptors found with bDescriptorSubType=VS_FORMAT_DV, verify that it is followed only by Frame DV Descriptors. If not, fail the test and throw related assertion (6.3.56).

TD 1.19 **Still Image Frame Descriptor Test.**

This test verifies that Format and place of a Still Image 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 a Still Image Frame Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_STILL_IMAGE_FRAME).
7. The Still Image Frame Descriptor found should be just after the Frame Descriptor of a Format Group. If not fail the test and throw related assertion (6.3.70). For each Still Image Frame Descriptor found, check that:
    a. bLength==10+(4*bNumImageSizePatterns)-4+bNumCompressionPattern. If not, fail the test and throw the related assertion (6.3.71).
    b. D7 in bEndpointAddress==1. If not, fail the test and throw the related assertion (6.3.72).
    c. D6..4 in bEndpointAddress==0. If not, fail the test and throw the related assertion (6.3.73).
    d. If bStillCaptureMethod==2, bEndpointAddress==0. If not, fail the test and throw the related assertion (6.3.74).
    e. bEndpointAddress refers to an existing Endpoint. If not, fail the test and throw the related assertion (6.3.75).

TD 1.20 **Color Matching Descriptor test.**

This test verifies that Format and place of a Color Matching 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 Color Matching Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_COLORFORMAT).
7. The Color Matching Descriptor found should be just after the Frame Descriptor of a Format Group. If not fail the test and throw related assertion (6.3.80). For each Color Matching Descriptor found, check that:
    a. bLength==6. If not, fail the test and throw the related assertion (6.3.81).
    b. bColorPrimaries<=5. If not, fail the test and throw the related assertion (6.3.82).
    c. bTransferCharacteristics<=7. If not, fail the test and throw related assertions (6.3.83).
    d. bMatrixCoefficients<=5. If not, fail the test and throw related assertion (6.3.84).

TD 1.21 **Standard VS Isochronous Video data Endpoint Test.**

This test verifies that the Standard Vs Isochronous Video data Endpoint is compliant with 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 Isochronous Video data Endpoint (bDescriptorType==ENDPOINT and D1..0 in bmAttributes==01).
7. For every Isochronous Video Data Endpoint found, check that:
    a. D3..2 in bmAttributes==01. If not, fail the test and throw the related assertion (6.3.90).
    b. D7..4 in bmAttributes==0. If not, fail the test and throw the related assertion (6.3.91).
    c. D6..4 in bEndpointAddress==0. If not, fail the test and throw the related assertion (6.3.92).

6.5.1.2 Advanced Descriptor Tests

TD 2.1 **Device Topology Test:**

This test verifies that there is no loop between the Units and Terminals of the device.

To test the topology we have to sketch a graph of the topology.

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 Unit or Terminal Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VC_* TERMINAL or VC_*_UNIT). Create a vector of visited Units or Terminal, call it ListOfUnitAndTerminals and add the bTerminalID or bUnitID of the descriptor to this vector. This vector will be used to check if there are no duplicate Ids. If not, fail the test and throw the related assertion (6.4.2).
8. While Parsing of all the Unit and Terminal Descriptors, we initialize a tree representing the topology of the Video Function. A node of the tree is composed by an ID, a pointer on another node Son and a pointer on another node Brother. The son is the first Input Pin of the Unit/Terminal corresponding to the node; every other Input Pin will be Brother of the Son of the Node. For each Descriptor encountered, depending on the type of the descriptor, we create the node and his sons and update the tree.
9. When the tree is complete (at the end of the parsing of the descriptors), we can test the topology. For each Unit/ Terminal verify that his Id is not in the trees of his sons. If not, fail the test and throw the related assertions (6.4.1).

TD 2.2 **Control supporting hardware Trigger Test.**

This test verifies that that if a VS supports hardware Triggers for Still Image, the VC has an Interrupt Endpoint.

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 a Class Specific VC Interrupt Endpoint Descriptor (bDescriptorType==CS_ENDPOINT and bDescriptorSubType==EP_INTERRUPT). If found one, store that information, go to the end of the test and pass it. If no Interrupt Endpoint Descriptor found, go on next step.
7. Parse the Descriptors to find Video Streaming Interfaces and their alternate Setting 0 Descriptor (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bAlternateSettings==0).
8. For each VS found, parse the Descriptors to find a Class-specific Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_INPUT_HEADER). If found, check the bStillCaptureMethod field. If (bStillCaptureMethod==2 or bStillCaptureMethod==3), an interrupt Endpoint is needed, so fail the test and throw the related assertion (6.4.3).

6.5.1.3 Video Format and Frame Descriptor Tests

TD 3.1 **Uncompressed Video Format Descriptor Test.**

This test verifies that the uncompressed 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 Uncompressed Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_UNCOMPRESSED).
8. For each Uncompressed Video Format Descriptor found check that:
    a. bLength==27. If not, fail the test and throw the related assertion (6.10.1).
    b. guidFormat==YUY2 or guidFormat==NV12. If not, fail the test and throw related assertion (6.10.2).
    c. bNumFrameDescriptors!=0. If not, fail the test and throw related assertion (6.10.3).
    d. bDefaultFrameIndex<=bNumFrameDescriptors. If not, fail the test and throw related assertions (6.10.4).
    e. Bits D3 in bmInterlaceFlags is not set. If not, fail the test and throw related assertion (6.10.5).
    f. For UVC 1.0 devices: Bits D7..6!=11in bmInterlaceFlags. If not, fail the test and throw related assertion (6.10.6).
    For UVC 1.1 device: Bits D7..6 != 00. . If not, fail the test and throw related assertion (6.10.11).
    g. bCopyProtect<=1. If not, fail the test and throw related assertion (6.10.7).
9. Parse the next bNumFrameDescriptors Descriptors, store their Frame indexes (bFrameIndex) and check that they are all Uncompressed Frame Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FRAME_UNCOMPRESSED). If not, fail the test and throw related assertion (6.10.10).
10. Check that all the Frame indexes stored are unique. If not, fail the test and throw related assertion (6.10.9).
11. Parse the next Descriptor and check that it is not an Uncompressed Frame Descriptor. If not, fail the test and throw the related assertion (6.10.8).

TD 3.2 **Uncompressed Video Frame Descriptor Test.**

This test verifies that the Uncompressed 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 iff bStillCaptureMethod==1, flag==0 otherwise.
8. Parse the Descriptors to find every Uncompressed Video Format Descriptor (bDescriptorType==CS_INTERFACE&bDescriptorSubType==VS_FORMAT_UNCOMPRESSED
9. Repeat the following steps for every Uncompressed Video Format Descriptor found.
    a. Parse the next bNumFrameDescriptors Descriptors and verify that they are FRAME UNCOMPRESSED.
    b. For each Frame Descriptor found, check that:
    c. bLength==38 if bFrameIntervalType==0 or bLength==26+4*bFrameIntervalType if bFrameIntervalType!=0. If not, fail the test and throw the related assertion (6.10.20).
    d. bmCapabilities!=0 if flag==1 and bmCapabilities==0 otherwise. If not, fail the test and throw related assertion (6.10.21).
    e. bmCapabilities<=1. If not, fail the test and throw related assertion (6.10.22).
    f. If bFrameIntervalType==0, check that:
        i. dwDefaultFrameInterval==dwMinInterval + j*dwFrameIntervalStep with j integer and dwDefaultFrameInterval<dwMaxFrameInterval (6.10.23).
        ii. dwMinFrameInterval<=dwMaxFrameInterval. If not, fail the test and throw related assertion (6.10.24).
        iii. dwFrameIntervalStep<=(dwMaxFrameInterval-dwMinFrameInterval). If not, fail the test and throw related assertion (6.10.25).
    g. If bFrameIntervalType!=0, check that :
        i. dwDefaultFrameInterval is in the set dwFrameInterval(1)…dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.23).
        ii. dwFrameInterval(1)<=..<=dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.26)

.

TD 3.3 **MJPEG Video Format Descriptor Test.**

This test verifies that the MJPEG 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 MJPEG Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_MJPEG) .
8. For each MJPEG Video Format Descriptor found check that:
    a. bLength==11. If not, fail the test and throw the related assertion (6.10.40).
    b. bNumFrameDescriptors!=0. If not, fail the test and throw related assertion (6.10.41).
    c. Bits D7..1 in bmFlags is not set. If not, fail the test and throw related assertions (6.10.42).
    d. Bits D7..6!=11 in bmInterlaceFlags. If not, fail the test and throw related assertion (6.10.45).
    e. Bit D3 in bmInterlaceFlags is not set. If not, fail the test and throw related assertion (6.10.43).
    f. bDefaultFrameIndex<=bNumFrameIndex. If not, fail the test and throw the related assertion (6.10.44).
    g. bCopyProtect<=1. If not, fail the test and throw related assertion (6.10.46).
9. Parse the next bNumFrameDescriptors, store their Frame indexes (bFrameIndex) and check that they are all MJPEG Frame Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FRAME_MJPEG). If not, fail the test and throw related assertion (6.10.47).
10. Check that all the Frame indexes stored are unique. If not, fail the test and throw related assertion (6.10.48).
11. Parse the next Descriptor and check that it is not an MJPEG Frame Descriptor. If not, fail the test and throw the related assertion (6.10.49).

TD 3.4 **MJPEG Video Frame Descriptor Test.**

This test verifies that the MJPEG 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 iff bStillCaptureMethod==1, flag==0 otherwise.
8. Parse the Descriptors to find every MJPEG Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_MJPEG)
9. Repeat the following steps for every MJPEG Video Format Descriptor found.
    a. Parse the next bNumFrameDescriptors Descriptors and verify that they are FRAME MJPEG.
    b. For each Descriptor found, check that:
    c. bLength==38 if bFrameIntervalType==0 or bLength==26+4*bFrameIntervalType if bFrameIntervalType!=0. If not, fail the test and throw the related assertion (6.10.60).
    d. bmCapabilities!=0 if flag==1 and bmCapabilities==0 otherwise. If not, fail the test and throw related assertion (6.10.61)
    e. bmCapabilities<=1. If not, fail the test and throw related assertion (6.10.62).
    f. If bFrameIntervalType==0, check that:
        i. dwDefaultFrameInterval==dwMinInterval + j*dwFrameIntervalStep with j integer and dwDefaultFrameInterval<dwMaxFrameInterval (6.10.63).
        ii. dwMinFrameInterval<=dwMaxFrameInterval. If not, fail the test and throw related assertion (6.10.64).
        iii. dwFrameIntervalStep<=(dwMaxFrameInterval-dwMinFrameInterval). If not, fail the test and throw related assertion (6.10.65).
    g. If bFrameIntervalType!=0, check that :
        i. dwDefaultFrameInterval is in the set dwFrameInterval(1)…dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.63).
        ii. dwFrameInterval(1)<=..<=dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.66).

TD 3.8 MPEG2 TS Format Descriptor Test.

This test verifies that the MPEG2 TS 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 Format Descriptors have been already checked in test 1.17 and 1.18).
7. Parse the Descriptors to find every MPEG2TS Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_MPEG4SL) .
8. For each MPEG2TS Video Format Descriptor found check that:
    a. bLength==7. If not, fail the test and throw the related assertion (6.10.110).
    b. If bDataOffset==0, bStrideLength==bPacketLength. If not fail the test and throw related assertion (6.10.111).

TD 3.9 **DV Payload Descriptor Test.**

This test verifies that the DV Payload 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 DV Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_DV) .
8. For each DV Video Format Descriptor found check that:
    a. bLength==9. If not, fail the test and throw the related assertion (6.10.120).
    b. Bits D6..3 in bFormatType are not set. If not fail the test and throw related assertion (6.10.121).

TD 3.12 **Stream Based Video Format Descriptor Test.**

This test verifies that the Stream Based 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 Stream Based Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_STREAM_BASED).
8. For each Stream Based Video Format Descriptor found check that:
    a. bLength==25. If not, fail the test and throw the related assertion (6.10.170).
    b. bFixedSize<=1. If not, fail the test and throw the related assertion (6.10.171).
    c. If bFixedSize=0, verify that dwPacketLength equals 0. If not, fail the test and throw related assertion (6.10.172).

TD 3.13 **Frame Based Video Format Descriptor Test.**

This test verifies that the Frame Based 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 Frame Based Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_FRAME_BASED).
8. For each MJPEG Video Format Descriptor found check that:
    a. bLength==28. If not, fail the test and throw the related assertion (6.10.190).
    b. bNumFrameDescriptors!=0. If not, fail the test and throw related assertion (6.10.191).
    c. Bits D7..6 are reserved in bmInterlaceFlags. If not, fail the test and throw related assertion (6.10.192).
    d. Bit D3 in bmInterlaceFlags is not set. If not, fail the test and throw related assertion (6.10.193).
    e. bDefaultFrameIndex<=bNumFrameIndex. If not, fail the test and throw the related assertion (6.10.194).
    f. bCopyProtect<=1. If not, fail the test and throw related assertion (6.10.195).
    g. bVariableSize<=1. If not, fail the test and throw the related assertion (6.10.196).
9. Parse the next bNumFrameDescriptors, store their Frame indexes (bFrameIndex) and check that they are all Frame Based Frame Descriptors (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FRAME_FRAME_BASED). If not, fail the test and throw related assertion (6.10.197).
10. Check that all the Frame indexes stored are unique. If not, fail the test and throw related assertion (6.10.198).
11. Parse the next Descriptor and check that it is not a Frame Based Frame Descriptor. If not, fail the test and throw the related assertion (6.10.199).

TD 3.14 Frame Based Video Frame Descriptor Test.

This test verifies that the Frame Based 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 iff bStillCaptureMethod==1, flag==0 otherwise.
8. Parse the Descriptors to find every Vendor Video Format Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_FORMAT_FRAME_BASED)
9. Repeat the following steps for every Frame Based Video Format Descriptor found. Store the value of the bVariableSizeField, if set to 1 set a flag.
    a. Parse the next bNumFrameDescriptors Descriptors and verify that they are all Frame Based Frame Descriptors.
    b. For each Descriptor found, check that:
        a. bLength==38 if bFrameIntervalType==0 or bLength==26+4*bFrameIntervalType if bFrameIntervalType!=0. If not, fail the test and throw the related assertion (6.10.210).
        b. bmCapabilities!=0 if flag==1 and bmCapabilities==0 otherwise. If not, fail the test and throw related assertion (6.10.211)
        c. bmCapabilities<=1. If not, fail the test and throw related assertion (6.10.212).
        d. If bVariableSize is set to 1 in the Format Descriptor, then verifies that dwBytesPerLine is set to 0. If not, fail the test and throw the related assertion (6.10.213).
        e. If bFrameIntervalType==0, check that:
            i. dwDefaultFrameInterval==dwMinInterval + j*dwFrameIntervalStep with j integer and dwDefaultFrameInterval<dwMaxFrameInterval . If not , fail the test and throw the relater assertion (6.10.214).
            ii. dwMinFrameInterval<=dwMaxFrameInterval. If not, fail the test and throw related assertion (6.10.215).
            iii. dwFrameIntervalStep<=(dwMaxFrameInterval-dwMinFrameInterval). If not, fail the test and throw related assertion (6.10.216).
        f. If bFrameIntervalType!=0, check that :
            i. dwDefaultFrameInterval is in the set dwFrameInterval(1)…dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.214).
            ii. dwFrameInterval(1)<=..<=dwFrameInterval(n). If not, fail the test and throw related assertion (6.10.217).

6.5.2 Video Control tests

6.5.2.1 Interface Control Tests

TD 20.1 **Power Mode Control Test**

This Test verifies that the Power Mode Control is compliant with 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.

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).
7. Retrieve the Interface number of this Video Control Interface and begin test on the Control. This control is mandatory.
8. 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 (6.20.31). 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.
9. Verify that the control supports all mandatory requests: SET_CUR,GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported: GET_MIN, GET_MAX, GET_RES, GET_LEN, and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported(D0 and D1 of the returned bit). If not, fail the test and throw the related assertion (6.20.6).
    b. If the Control is Autoupdate or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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).
    d. 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).
11. Issue a SET_CUR request with an invalid Power Mode value. Verify that the Power Mode 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 (6.20.30).

TD 20.2 **Request Error Code Control Test:**

This Test verifies that Request Error Code Control is compliant with 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.

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).
7. Retrieve the Interface number of this Video Control Interface and begin test on the Control. This control is mandatory.
8. 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. 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.
9. Verify that the control supports all mandatory requests: SET_CUR,GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded*. Verify also that all other requests are not supported: GET_MIN, GET_MAX, GET_RES, GET_LEN, and GET_DEF. Issue all these requests and verify that the device answers STALL. If not, fail the test and throw the related assertion. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request)**.If one of the conditions(* and **) above is not met, fail the test and throw the related assertions.
10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported(D0 and D1 of the returned bit mask). If not, fail the test and throw the related assertion.
    b. If the Control is Autoupdate or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion.
    c. 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.
    d. 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.
11. Issue a SET_CUR request with an invalid Power Mode value. Verify that the Power Mode Control answers STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion. 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.

6.5.2.2 Unit and Terminal Control tests

6.5.2.2.1 Camera Terminal Control tests

TD 20.4 Scanning Mode Control Test

This Test verifies that the Scanning Mode Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D0==0, 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. Do Step 8 to 11 only if D0==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value<=1). If not, fail the test and throw the related assertion (6.20.4 and 6.20.52). 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 (6.20.42).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported: GET_MIN, GET_MAX, GET_RES, GET_LEN, and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0 and D1 of the returned bit). If not, fail the test and throw the related assertion (6.20.6).
    b. If the Control is Autoupdate or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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).
    d. 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).
11. Issue a SET_CUR request with an invalid bScanningMode value (value>1). Verify that the Scanning Mode 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 (6.20.51).

TD 20.5 Auto-Exposure Mode Control Test

This Test verifies that the Auto-Exposure Mode Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D1==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 12 only if D1==1 (Control is supported).
8. Issue a GET_RES request and verify that the value is valid (values supported by the control, no bits over D3 set). If not, fail the test and throw the related assertion (6.20.4 and 6.20.60).
9. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (D0 or D1 or D2 or D3 set, but no other bits set, check with values returned by RES attribute). If not, fail the test and throw the related assertion (6.20.62). 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 (6.20.42).
10. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR and GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported: GET_MIN, GET_MAX, and GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.5

11. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported(D0 and D1 of the returned bit). If not, fail the test and throw the related assertion (6.20.6).
    b. If the Control is Autoupdate or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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).
    d. 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).
12. Issue a GET_DEF request and verify that value returned is valid (corresponds to only one of the modes listed in the RES value). If not, fail the test and throw the related assertion (6.20.12).
13. Issue a SET_CUR request with an invalid bAutoExposureMode value (0, invalids bits set defined in RES, more than one bit set). Verify that the AutoExposure Mode 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 (6.20.61).
14. Issue a SET_CUR request with a valid value (according to RES). Issue a GET_CUR requests and verify that the value have been set. If not, fail the test and throw the related assertion (6.20.63). Verify also that Control Change Interrupt have been sent by the controls affected by the changes. If not, fail the test and throw the related assertions (6.20.64, 6.20.65, 6.20.66, 6.20.67)

TD 20.6 Auto-Exposure Priority Control Test

This Test verifies that the Auto-Exposure Priority Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D2==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 12 only if D2==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value<=1). If not, fail the test and throw the related assertion (6.20.4 and 6.20.71). 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 (6.20.42).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported: GET_MIN, GET_MAX, GET_LEN, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.6

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported(D0 and D1 of the returned bit). If not, fail the test and throw the related assertion (6.20.6).
    b. If the Control is Autoupdate or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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).
    d. 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).
11. Issue a SET_CUR request with an invalid bAutoExposurePriority value (value >1). Verify that the AutoExposure Priority 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 (6.20.70).
12. Issue a GET_CUR request and then issue a SET_CUR requests with a value different than the value returned by the GET_CUR request. Verify that the value has been set by issuing a GET_CUR request. If not fail the test and throw the related assertion (6.20.72).

TD 20.7 Exposure Time (Absolute) Control Test

This Test verifies that the Exposure Time (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D3==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 18 only if D3==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value!=0). If not, fail the test and throw the related assertion (6.20.84). 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 (6.20.42 and 6.20.4).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.7

10. If the Auto Exposure mode Control is supported and is in Auto Mode (D1 set in bAutoExposureMode) or in Aperture Priority Mode (D3 set in bAutoExposureMode), issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.83).
11. If the control is in Auto Mode. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x01 to disable the auto Mode. Verify that an asynchronous notification is sent by the Exposure Time (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.64).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
    f. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the dwExposureTimeAbsolute value. Verify that MIN<MAX. If not fail the test and throw the related assertion (6.20.14)
14. Issue a GET_DEF Request and verify that the value returned is between Min and Max. If not fail the test and throw the related assertion (6.20.12).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.20.80).
16. If SET_CUR is supported. Issue a SET_CUR Request with valid values. Verify that the values have been set. If not, fail the test and throw the related assertions (6.20.80).
17. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.82).
18. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x02 or 0x04 to enable the auto Mode. Verify that an asynchronous notification is sent by the Exposure Time (Absolute) Control to notify of the change (6.20.64). If not, fail the test and throw the related assertion.
19. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.83).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.81).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.8 Exposure Time (Relative) Control Test

This Test verifies that the Exposure Time (Relative) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D4==0, 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 (6.20.42 and 6.20.3). Do Step 8 to 18 only if D4==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value==0 or 1 or 0xFF). If not, fail the test and throw the related assertion (6.20.94). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.8

10. If the Auto Exposure mode Control is supported and is in Auto Mode (D1 set in bAutoExposureMode) or in Aperture Priority Mode (D3 set in bAutoExposureMode), issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.93).
11. If the control is in Auto Mode. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x00 to disable the auto Mode. Verify that an asynchronous notification is sent by the Exposure Time (Relative) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.65).
12. Issue a GET_INFO request:
    a. 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).
    b. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. 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)
    e. 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).
13. Issue a SET_CUR Request with an invalid value (value != 0, or 1 or 0xFF). Verify that the answer is STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.92).
14. Issue a GET_CUR request to the Exposure Time(Absolute) Control to store the current value of the Control.
15. Issue a SET_CUR request with a valid value (value==0 or 1 or 0xFF). Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the Exposure Time (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.90).
16. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x01 or 0x04 to enable the auto Mode. Verify that an asynchronous notification is sent by the Exposure Time (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.65).
17. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.93).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.91).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.9 Focus (Absolute) Control Test

This Test verifies that the Focus (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D5==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 18 only if D5==1 (Control is supported).
8. Issue a GET_CUR request. Verify that the request completed with success. If not, fail the test and throw the related assertion (6.20.4). 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 (6.20.42).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.9

10. In the Camera Terminal Descriptor, if D17 in bmControls is set. If the Focus, Auto Control is supported and is in Auto Mode (bFocusAuto=1) issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.102).
11. If the control is in Auto Mode. Issue a set Request to Focus, Auto Control with a bFocusAuto value of 0 to disable the auto Mode. Verify that an asynchronous notification is sent by the Focus (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.103).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
    f. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the dwFocusAbsolute value. Verify that MIN<MAX. If not fail the test and throw the related assertion (6.20.14)
14. Issue a GET_DEF Request and verify that the value returned is between Min and Max. If not fail the test and throw the related assertion (6.20.11).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.20.100).
16. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.101).
17. Issue a set Request to the Focus, Auto Control with a bFocusAuto value of 1 to enable the auto Mode. Verify that an asynchronous notification is sent by the Focus (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.103).
18. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.102).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.104).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.10 Focus (Relative) Control Test

This test verifies that the Focus (Relative) 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 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 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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors if found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D6==0, 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 (6.20.42 and 6.20.3). Do Step 8 to 19 only if D6==1 (Control is supported).
8. 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 . 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 (6.20.42 and 6.20.4).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.10

10. In the Camera Terminal Descriptor, if D17 in bmControls is set. If the Focus, Auto Control is supported and is in Auto Mode (bFocusAuto=1) issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.113).
11. If the control is in Auto Mode. Issue a set Request to Focus, Auto Control with a bFocusAuto value of 0 to disable the auto Mode. Verify that an asynchronous notification is sent by the Focus (Relative) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.114).
12. Issue a GET_INFO request:
    a. 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).
    b. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. 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)
    e. 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).
13. Issue a SET_CUR request with an invalid bFocusRelative value(other than 0xFF, 0x00, or 0x01). Verify that the Focus (Relative) 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.
14. Issue GET_MIN, GET_MAX,GET_RES,GET_DEF requests and verify that they return 0 for bFocusRelative. If not, fail the test and throw the related assertions (6.20.111).
15. Issue a SET_CUR request with an invalid bSpeed value(issue GET_MIN and GET_MAX requests to get the range of bSpeed values). Verify that the Focus (Relative) 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.
16. Issue a SET_CUR request with valid values. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the Focus (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.115).
18. Issue a set Request to the Focus, Auto Control with a bFocusAuto value of 1 to enable the auto Mode. Verify that an asynchronous notification is sent to the Focus (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.114).
19. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.113).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.120).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.11 Focus Auto Control Test

This Test verifies that the Focus Auto Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D17==0, 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 (6.20.42 and 6.20.3). Do Step 8 to 12 only if D17==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value<=1). If not, fail the test and throw the related assertion (6.20.126). 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 (6.20.42 and 6.20.4).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX, and GET_RES. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.11

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a SET_CUR request with an invalid bFocusAuto value (value>1). Verify that the Focus Auto 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 (6.20.125).
12. Issue a GET_DEF request and verify that it is a valid value. If not, fail the test and throw the related assertion (6.20.12).
13. Issue a SET_CUR request with a value valid but different from the current one (retrieved with a GET_CUR). Verify that the value have been set. If not, fail the test and throw the related assertion (6.20.127). Verify also that Control Change notification have been sent by the Focus Absolute and Relative Controls (6.20.103 and 6.20.114)

TD 20.12 Iris (Absolute) Control Test

This Test verifies that the Iris (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D7==0, 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 (6.20.4 and 6.20.42). Do Step 8 to 19 only if D7==1 (Control is supported).
8. Issue a GET_CUR request. 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 (6.20.3 and 6.20.42).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.12

10. If the Auto Exposure mode Control is supported and is in Auto Mode (D1 set in bAutoExposureMode) or in Shutter Priority Mode (D2 set in bAutoExposureMode), issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.133).
11. If the control is in Auto Mode. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x01 to disable the auto Mode. Verify that an asynchronous notification is sent by the Iris (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.66).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
    f. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the dwIrisAbsolute value. Verify that MIN<MAX. If not, fail the test and throw the related assertion (6.20.14).
14. Issue a GET_DEF Request and verify that the value returned is between Min and Max. If not fail the test and throw the related assertion (6.20.11).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.20.130).
16. Issue a SET_CUR Request with valid values. Verify that the values have been set. If not, fail the test and throw the related assertions (6.20.130).
17. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.132).
18. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x01 or 0x02 to enable the auto Mode or Shutter Priority Mode. Verify that an asynchronous notification is sent by the Iris (Absolute) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.66).
19. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.133).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.131).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.13 Iris (Relative) Control Test

This Test verifies that the Iris (Relative) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D8==0, 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 (6.20.42 and 6.20.3). Do Step 8 to 15 only if D8==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (value==0 or 1 or 0xFF). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also if other requests are supported GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.13

10. If the Auto Exposure mode Control is supported and is in Auto Mode (D1 set in bAutoExposureMode) or in Shutter Priority Mode (D2 set in bAutoExposureMode), issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.20.143).
11. If the control is in Auto Mode. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x00 to disable the auto Mode. Verify that an asynchronous notification is sent to the Iris (Relative) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.67).
12. Issue a GET_INFO request:
    a. 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).
    b. Verify that D3 is not set in the GET_INFO answer. If not, fail the test and throw the related assertion (6.20.16).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. 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)
    e. 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).
13. Issue a SET_CUR Request with an invalid value (value != 0, or 1 or 0xFF). Verify that the answer is STALL. Check that the Request Error Code is correct. If not, fail the test and throw the related assertion (6.20.44 and 6.20.142).
14. Issue a GET_CUR request to the Iris (Absolute) Control to store the current value of the Control.
15. Issue a SET_CUR request with a valid value (value==0 or 1 or 0xFF). Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.20.140). Verify also that an asynchronous notification is sent to the Iris (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.66).
16. Issue a set Request to the Auto-Exposure Mode Control with a bAutoExposureMode value of 0x01 or 0x02 to enable the auto Mode or Shutter Priority Mode. Verify that an asynchronous notification is sent by the Iris (Relative) Control to notify of the change. If not, fail the test and throw the related assertion (6.20.67).
17. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.20.143).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.20.141).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.14 Zoom (Absolute) Control Test

This Test verifies that the Zoom (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D9==0, 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 (6.20.42 and 6.20.3). Do Step 8 to 16 only if D9==1 (Control is supported).
8. Issue a GET_CUR request. 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.14

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wObjectiveFocalLength value . Verify that MIN<MAX. If not fail the test and throw the related assertion (6.20.14)
12. Issue a GET_DEF Request and verify that the value returned is between Min and Max. If not fail the test and throw the related assertion (6.20.11).
13. Issue a GET_RES request. Verify that the value returned is 1 for wObjectiveFocalLength. If not, fail the test and throw the related assertion (6.20.152).
14. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.20.150).
15. If SET_CUR is supported. Issue a SET_CUR Request with valid values. Verify that the values have been set. If not, fail the test and throw the related assertion (6.20.150).
16. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.151).

TD 20.15 Zoom (Relative) Control Test

This test verifies that the Zoom (Relative) 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 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 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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors if found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D10==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 15 only if D10==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (bZoom==0 or 1 or 0xFF, bDigitalZoom==0 or 1,). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.15

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a SET_CUR request with an invalid bZoom value(other than 0xFF, 0x00, or 0x01). Verify that the Zoom (Relative) 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 (6.20.160).
12. Issue a SET_CUR request with an invalid bDigitalZoom value(other than 0x00, or 0x01). Verify that the Zoom (Relative) 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 (6.20.162).
13. Issue GET_MIN, GET_MAX,GET_RES,GET_DEF requests and verify that they return 0 for bZoom and bDigitalZoom. If not, fail the test and throw the related assertions (6.20.161 and 6.20.163). Verify also that MIN<MAX for bSpeed. If not, fail the test and throw the related assertion (6.20.166). Verify also that DEF fro bSpeed is between MIN and MAX. If not, fail the test and throw the related assertion. (6.20.167)
14. Issue a SET_CUR request with an invalid bSpeed value (issue GET_MIN and GET_MAX requests to get the range of bSpeed values). Verify that the Zoom (Relative) 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 (6.20.164).
15. Issue a GET_CUR request to the Zoom (Absolute) Control to store the current value.
16. Issue a SET_CUR Request with valid values for bZoom. Verify that the values have been set. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the Zoom (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.165 and 6.20.168).
17. Issue a SET_CUR Request with valid values for bDigitalZoom. Verify that the values have been set. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the Zoom (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.165 and 6.20.169).
18. Issue a SET_CUR Request with valid values for bSpeed. Verify that the values have been set. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the Zoom (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.165 and 6.20.189).

TD 20.16 Pan Tilt (Absolute) Control Test

This Test verifies that the Pan Tilt (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D11==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 14 only if D11==1 (Control is supported).
8. Issue a GET_CUR request. 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.16

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the dwPanAbsolute and dwTiltAbsolute value. Verify that Min and Max are in the range –180*3600 to +180*3600 for dwPanAbsolute (6.20.170 and 6.20.171). Verify also that Min and Max are in the range –180*3600 to +180*3600 for dwTiltAbsolute (6.20.173 and 6.20.174).
12. Verify also that MIN<MAX for dwPanAbsolute and dwTiltRelative. If not, fail the test and throw the relative assertion (6.20.179 and 6.20.180).
13. Issue a GET_DEF Request and verify that the DEF value is 0 for dwPanAbsolute and dwTiltAbsolute. If not fail the test and throw the related assertion (6.20.172 and 6.20.175).
14. If SET_CUR is supported. Issue a SET_CUR request with values equal to the default values. Verify that the request succeeded and issue a GET_CUR Request to verify that the values have been set. If not, fail the test and throw the related assertion (6.20.178). Repeat the same procedure for values inside the range.
15. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound values (MIN-1, MAX+1, LONG_MIN, LONG_MAX) for dwPanAbsolute. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.20.176).
16. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound values (MIN-1, MAX+1, LONG_MIN, LONG_MAX) for dwTiltAbsolute. Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.20.177).

TD 20.17 **Pan Tilt (Relative) Control Assertion.**

This test verifies that the Pan Tilt (Relative) 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 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 Config 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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors if found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D12==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 23 only if D12==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.17

11. Issue a SET_CUR request with an invalid bPanRelative value (other than 0xFF, 0x00, or 0x01). Verify that the Pan Tilt (Relative) 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 (6.20.190).
12. Issue GET_MIN, GET_MAX, GET_RES and GET_DEF requests and verify that they return 0 for bPanRelative and for bTitltRelative. If not, fail the test and throw the related assertions (6.20.191 and 6.20.194).
13. Verify also that MIN<MAX for bPanSpeed. If not, fail the test and throw the related assertion (6.20.196).
14. Verify that MIN<MAX for bTiltSpeed. If not, fail the test and throw the related assertion (6.20.197).
15. Verify that DEF value is between MIN and MAX for bPanSpeed. If not, fail the test and throw the related assertion (6.20.198).
16. Verify that DEF value is between MIN and MAX for bTiltSpeed. If not, fail the test and throw the related assertion (6.20.199).
17. Issue a SET_CUR request with an invalid bPanSpeed value (issue GET_MIN and GET_MAX requests to get the range of bPanSpeed values). Verify that the Pan Tilt (Relative) 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 (6.20.192).
18. Issue a GET_CUR and store the value returned. Issue a SET_CUR request with an invalid bTiltRelative value(other than 0xFF, 0x00, or 0x01). Verify that the Pan Tilt (Relative) Control answers STALL Check that the Request Error Code is correct and by issuing a GET_CUR that the value is unchanged. 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 (6.20.193).
19. Issue a GET_CUR and store the value returned. Issue a SET_CUR request with an invalid bTiltSpeed value (issue GET_MIN and GET_MAX requests to get the range of bPanSpeed values). Verify that the Pan Tilt (Relative) Control answers STALL. Check that the Request Error Code is correct and by issuing a GET_CUR that the value is unchanged. 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 (6.20.195).
20. Issue a SET_CUR Request with valid values bPanRelative. Verify that the value has been updated. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the PanTilt (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.200 and 6.20.181).
21. Issue a SET_CUR Request with valid values for bPanSpeed. Verify that the value has been updated. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the PanTilt (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.201 and 6.20.181).
22. Issue a SET_CUR Request with valid values for bTiltRelative. Verify that the value has been updated. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the PanTilt (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.202 and 6.20.181).
23. Issue a SET_CUR Request with valid values for bTiltSpeed. Verify that the value has been updated. If not, fail the test and throw the related assertion. Verify also that an asynchronous notification is sent to the PanTilt (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.203 and 6.20.181).

TD 20.18 Roll (Absolute) Control Test

This Test verifies that the Pan Tilt (Absolute) Control is compliant with 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.

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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors is found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D13==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 14 only if D13==1 (Control is supported).
8. Issue a GET_CUR request. 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.18

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wRollAbsolute value. Verify that Min and Max are in the range –180 to +180 and MIN<MAX. If not fail the test and throw the related assertion (6.20.11).
12. Issue a GET_DEF Request and verify that the value returned is 0. If not fail the test and throw the related assertion (6.20.212).
13. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion. Repeat the same procedure for values inside the range (6.20.211).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.20.210).

TD 20.19 **Roll (Relative) Control Assertion.**

This test verifies that the Roll (Relative) 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 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 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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors if found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D14==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 18 only if D14==1 (Control is supported).
8. 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. If the request did not success, check the Request Error Code Control and if its value is 0x06(Invalid Control), then fail the test and throw the related assertion (6.20.4 and 6.20.42).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.19

11. Issue a SET_CUR request with an invalid bRollRelative value (other than 0xFF, 0x00, or 0x01). Verify that the Pan Tilt (Relative) 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 (6.20.220).
12. Issue GET_MIN, GET_MAX,GET_RES,GET_DEF requests and verify that they return 0 for bRollRelative. If not, fail the test and throw the related assertions (6.20.221).
13. Verify also that MIN<MAX for bSpeed. If not, fail the test and throw the related assertion (6.20.224).
14. Verify that DEF value is between MIN and MAX fro bSpeed. If not, fail the test and throw the related assertion (6.20.225).
15. Issue a SET_CUR request with an invalid bSpeed value(issue GET_MIN and GET_MAX requests to get the range of bSpeed values). Verify that the Roll (Relative) 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 (6.20.222).
16. Issue a SET_CUR Request to the Roll (Absolute) Control to store its Current Value.
17. Issue a SET_CUR Request with valid values for bRollRelative. Verify that the value have been set. If not, fail the test and throw the related assertion (6.20.226) Verify also that an asynchronous notification is sent to the Roll (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.223).
18. Issue a SET_CUR Request with valid values for bSpeed. Verify that the value have been set. If not, fail the test and throw the related assertion (6.20.227) Verify also that an asynchronous notification is sent to the Roll (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.223).

TD 20.20 Privacy Control Assertion.

This test verifies that the Privacy 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 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 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 a Camera Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL and wTerminalType == ITT_CAMERA). If no Camera Terminal Descriptors if found, stop the test. If a camera Terminal is found, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Camera Terminal Descriptor. If D15==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D15==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (0 or 1). If not, fail the test and throw the related assertion (6.20.230). 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 (6.20.42 and 6.20.4).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_MIN, GET_MAX, GET_RES, GET_DEF and GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.20

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If D3==0 in GET_INFO. Fail the test and throw the related assertion (6.20.231).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
    f. If the control is synchronous and SET_CUR supported, 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).
11. If SET_CUR is supported. Issue a SET_CUR request with an invalid bPrivacy value (other than 0x00, or 0x01). Verify that the Privacy 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 (6.20.232).
12. Issue a SET_CUR Request with valid values for bSpeed. Verify that the value have been set. If not, fail the test and throw the related assertion (6.20.233) Verify also that an asynchronous notification is sent to the Roll (Absolute) Control and that the value has been updated. If not fail the test and throw the related assertion (6.20.234).

6.5.2.2.2 Selector Unit Control Tests

TD 20.21 Selector Unit Control Test.

This test verifies that the Selector Unit 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 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 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 Selector Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_SELECTOR_UNIT. If no Selector Unit Descriptors if found, stop the test. If a Selector Unit is found, retrieve the Unit ID and begin the test on the Control.
7. If a selector Unit is present, then this control has to be supported.
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (1 up to the number of Input Pins of the selector Unit bNrInPins). If not, fail the test and throw the related assertion (6.21.5). 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 (6.21.7 and 6.20.42).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.21

11. Issue a GET_RES request, verify whether it is supported or not. If it is not supported, verify that the control answers STALL and check the Request Error Code Control (value 0x07). If the request is supported, verify that the returned value is 1. if not fail the test and throw the related assertion (6.21.2)
12. Issue GET_MIN, GET_MAX requests and verify that they return 1 for minimum and bNrInPins for Maximum as the bSelector value. If not, fail the test and throw the related assertions (6.21.1 and 6.21.3).
13. Issue a SET_CUR request with out-of-bound bSelector value (0,MAX+1). Verify that the Selector Unit Control answers STALL. Check that the Request Error Code is correct (6.20.44). If not, fail the test and throw the related assertion. 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 (6.21.4).
14. Issue a GET_CUR request to store the current value. Then issue a SET_CUR request with a value different from the stored value but valid. Issue a GET_CUR request to verify that the value has been set. If not, fail the test and throw the related assertion (6.21.6).

6.5.2.2.3 Processing Unit Control Tests

TD 20.22 BackLight Compensation Control Test.

This test verifies that the Backlight Compensation 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 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 Config 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D8==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D8==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.22

11. Issue GET_MIN, GET_MAX and GET_DEF requests and verify that they return a DEF value inside the Min-Max Range. Verify also that MIN<=MAX. If not, fail the test and throw the related assertions (6.20.14 and 6.20.11).
12. Issue a GET_RES request and check that the returned value is valid. If not, fail the test and throw the related assertion (6.20.17).
13. Issue a SET_CUR request with out-of-bound wBackLightCompensation value. Verify that the BackLight Compensation 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 (6.22.1).
14. Issue a SET_CUR Request with valid values for wBacklightCompensation. Verify that the value have been set. If not, fail the test and throw the related assertion (6.22.2).

TD 20.23 **Brightness Control Test.**

This test verifies that the Brightness 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D0==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D0==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.23

11. Issue a GET_RES request, verify that the returned value is 1. If not fail the test and throw the related assertion (6.22.12).
12. Issue GET_MIN, GET_MAX and GET_DEF requests and verify that they return a DEF value inside the Min-Max Range. Verify also that MIN<=MAX. If not, fail the test and throw the related assertions (6.20.12 and 6.20.11).
13. Issue a SET_CUR request with out-of-bound wBrigtness value. Verify that the Brightness 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).
14. Issue a SET_CUR Request with valid values for wBrightness. Verify that the value have been set. If not, fail the test and throw the related assertion (6.22.10).

TD 20.24 **Contrast Control Test.**

This test verifies that the Contrast 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D1==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D1==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.24

11. Issue a GET_RES request, verify that the returned value is 1. If not fail the test and throw the related assertion (*6.22.22*).
12. Issue GET_MIN, GET_MAX and GET_DEF requests and verify that they return a DEF value inside the Min-Max Range. Verify also that MIN<=MAX. If not, fail the test and throw the related assertions (6.20.11 and 6.20.12).
13. Issue a SET_CUR request with out-of-bound wContrast value. Verify that the Contrast 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 (6.22.21).
14. Issue a SET_CUR Request with valid values for wContrast. Verify that the value have been set. If not, fail the test and throw the related assertion (6.22.20).

TD 20.25 **Gain Control Test.**

This test verifies that the Gain 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D9==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D9==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.25

11. Issue a GET_RES request, verify that the returned value is 1. If not fail the test and throw the related assertion (6.22.32).
12. Issue GET_MIN, GET_MAX and GET_DEF requests and verify that they return a DEF value inside the Min-Max Range. Verify also that MIN<=MAX. If not, fail the test and throw the related assertions (6.20.11 and 6.20.12).
13. Issue a SET_CUR request with out-of-bound wGain value. Verify that the Gain 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 (6.22.31).
14. Issue a SET_CUR Request with valid values for wBrightness. Verify that the value have been set. If not, fail the test and throw the related assertion (6.22.30).

TD 20.26 **Power Line Frequency Control Test.**

This test verifies that the Power Line Frequency 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 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 Config 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D10==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D10==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (bPowerLineFrequency==0,1 or 2). If not, fail the test and throw the related assertion (6.22.41). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX and GET_RES. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).

TD 20.26

11. Issue a GET_DEF request, verify that the returned value is 0,1 or 2. If not fail the test and throw the related assertion (6.22.42)
12. Issue a SET_CUR with an invalid value (other than 0,1 or 2). Verify that the Gain 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 (6.22.40).
13. Issue a SET_CUR with a valid value. Verify that the request succeeded and that the correct value has been set. If not, fail the test and throw the related assertion (6.22.43).

TD 20.27 **Hue Control Test.**

This test verifies that the Hue 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D2==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D2==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_DEF and GET_RES. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.27

10. In the Processing Unit Descriptor, if D11 in bmControls is set. If the Hue, Auto Control is supported and is in Auto Mode (bHueAuto=1) issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.22.60).
11. If the control is in Auto Mode. Issue a set Request to Hue, Auto Control with a bHueAuto value of 0 to disable the auto Mode. Verify that an asynchronous notification is sent to the Hue Control to notify of the change. If not, fail the test and throw the related assertion (6.22.65).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the wHue value. Verify that Min and Max are in the range –18000 to +18000 and that MIN<=MAX. If not, fail the test and throw the related assertion (*6.22.50*, 6.22.51 and 6.20.12).
14. Issue a GET_DEF Request and verify that the value returned is 0. If not fail the test and throw the related assertion (6.22.52).
15. Issue a GET_RES Request and verify that the value returned is valid. If not fail the test and throw the related assertion (6.20.18).
16. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion. Repeat this test with all the values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeed (6.22.53).
17. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.54).
18. Issue a set Request to the Hue, Auto Control with a bHueAuto value of 1 to enable the auto Mode. Verify that an asynchronous notification is sent to the Hue Control to notify of the change. If not, fail the test and throw the related assertion (6.22.65).
19. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.22.60).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.22.66).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.28 **Hue, Auto Control Test.**

This test verifies that the Hue, Auto 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D11==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D11==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (bHueAuto==0 or 1). If not, fail the test and throw the related assertion (6.22.62). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX, and GET_RES. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.28

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a SET_CUR with a value of 1, verify that the request succeeded by issuing a GET_CUR request and then try to issue a SET_CUR request to the Hue Control. If the Hue Control answer STALL. Check that the Request Error Code is correct. Check that the request did not succeed. If not, fail the test and throw the related assertion (6.22.66). Verify also that an asynchronous notification has been sent to the Hue Control to notify of the change. If not, fail the test and throw the related assertion (6.22.65).
12. Issue a GET_DEF Request and verify that the value returned is 0or 1. If not fail the test and throw the related assertion (6.22.63).
13. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(>1). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.61).

TD 20.29 **Saturation Control Test.**

This test verifies that the Saturation 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D3==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D3==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.29

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wSaturation value. Verify that MIN<=MAX. If not, fail the test and throw the related assertion (6.20.12). Issue a GET_DEF Request and verify that the value returned between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
12. Issue a GET_RES request and verify that the answer is 1. If not, fail the test and throw the related assertion (6.22.72).
13. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.70). Repeat this test with all values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeed (6.22.70).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.22.71 and 6.20.44).

TD 20.30 **Sharpness Control Test.**

This test verifies that the Sharpness 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D4==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D4==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.30

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wSharpness value. Verify that MIN<=MAX. If not, fail the test and throw the related assertion (6.20.12) Issue a GET_DEF Request and verify that the value returned between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
12. Issue a GET_RES request and verify that the answer is 1. If not, fail the test and throw the related assertion (6.22.82).
13. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.80). Repeat this test with all values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeed (6.22.80).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.22.81).

TD 20.31 **Gamma Control Test.**

This test verifies that the Gamma 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D5==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D5==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.31

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a GET_RES Request and verify that the value returned is valid. If not fail the test and throw the related assertion (6.20.18).
12. Issue GET_MIN and GET_MAX request to store Min and Max values for the wGamma value. Verify that min is >=1 and MAX is <=500 and MIN<=MAX. If not, fail the test and throw the related assertions (6.22.90 and 6.22.91). Verify that MIN<MAX. If not, fail the test and throw the related assertion (6.20.12) Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
13. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.92). Repeat this test with all values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeed (6.22.92).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.93).

TD 20.32 **White Balance Temperature Control Test.**

This test verifies that the White Balance Temperature 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D6==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D6==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (in the range of 2800 to 6500). 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.32

10. In the Processing Unit Descriptor, if D12 in bmControls is set. If the White Balance Temperature, Auto Control is supported and is in Auto Mode (wWhiteBalanceTemperatureAuto=1) issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.22.110).
11. If the control is in Auto Mode. Issue a set Request to White balance Temperature, Auto Control with a wWhiteBalanceTemperatureAuto value of 0 to disable the auto Mode. Verify that an asynchronous notification is sent to the White Balance Temperature Control to notify of the change. If not, fail the test and throw the related assertion (6.22.115).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the wWhiteBalanceTemperature value. Verify that min is <=2800 and MAX is >=6500 and that MIN<=MAX. if not, fail the test and throw the related assertions (6.20.12, 6.22.100 and 6.22.101). Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
14. Issue a GET_RES Request and verify that the value returned is valid. If not fail the test and throw the related assertion (6.20.18).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.102). Repeat this test with all values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeed (6.22.102).
16. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.103).
17. Issue a set Request to the Whit Balance Temperature, Auto Control with a wWhiteBalanceTemperatureAuto value of 1 to enable the auto Mode. Verify that an asynchronous notification is sent to the White Balance Temperature Control to notify of the change. If not, fail the test and throw the related assertion (6.22.115).
18. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.22.110).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.22.116).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.33 White Balance Temperature, Auto Control Test.

This test verifies that the White Balance Temperature, Auto 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D12==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D12==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (0 or 1). If not, fail the test and throw the related assertion (6.22.112). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX, and GET_RES. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.33

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a GET_DEF Request and verify that the value returned is 0 or 1. If not fail the test and throw the related assertion (6.22.113).
12. Issue a SET_CUR Request with value of 1 (Auto Mode enabled) and now issue a SET_CUR on the White Balance Temperature Control and verify that the request does not success, device answers STALL. If not, fail the test and throw the related assertion (6.22.116).
13. Issue a SET_CUR request with a value equal to a value different from the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.114). Verify also that an asynchronous notification has been sent by the White Balance Temperature Control to notify of the change. If not, fail the test and throw the related assertion (6.22.115).
14. If SET_CUR is supported. Issue a SET_CUR request with invalid value(value >1). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.111).

TD 20.34 **White Balance Component Control Test.**

This test verifies that the White Balance Component 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D7==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D7==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (in the range of 2800 to 6500). 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.34

10. In the Processing Unit Descriptor, if D13 in bmControls is set. If the White Balance Component, Auto Control is supported and is in Auto Mode (wWhiteBalanceComponentAuto=1) issue a GET_INFO Request and verify that D2 and D3 are set. If not, fail the test and throw the related assertion (6.22.130).
11. If the control is in Auto Mode. Issue a set Request to White Balance Component, Auto Control with a wWhiteBalanceComponentAuto value of 0 to disable the auto Mode. Verify that an asynchronous notification is sent by the White Balance Component Control to notify of the change. If not, fail the test and throw the related assertion (6.22.135).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    d. If the control is Asynchronous and SET_CUR supported, 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)
    e. If the control is synchronous and SET_CUR supported, 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).
13. Issue GET_MIN and GET_MAX request to store Min and Max values for the wWhiteBalanceBlue and wWhiteBalanceRed values. Verify that MIN<=MAX. If not, fail the test and throw the related assertions (6.22.123 and 6.22.124) Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertions (6.22.125 and 6.22.126).
14. Issue a GET_RES Request and verify that the value returned is valid. If not fail the test and throw the related assertion (6.20.18).
15. If SET_CUR is supported. Issue a SET_CUR request with values equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertions (6.22.120). Repeat this test with all values in the Min-Max Range, fail the test and throw the related assertion if the test does not succeeds (6.22.127 and 6.22.128).
16. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound values for wWhiteBalanceBlue and wWhiteBalanceRed (MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertions (6.22.121 and 6.22.122).
17. Issue a set Request to the Whit Balance Temperature, Auto Control with a wWhiteBalanceTemperatureAuto value of 1 to enable the auto Mode. Verify that an asynchronous notification is sent to the White Balance Component Control to notify of the change. If not, fail the test and throw the related assertion (6.22.135).
18. Issue a GET_INFO request:
    a. Verify that the answer of the GET_INFO specifies that the control is in Auto Mode. If not, fail the test and throw the related assertion (6.22.130).
    b. Verify that the SET_CUR Request result in STALL. Check that the request error code is correct. If not fail the test and throw the related assertion (6.22.136).
    c. Since the control is in Auto Mode, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.35 White Balance Component, Auto Control Test.

This test verifies that the White Balance Component, Auto 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D13==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D13==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (0 or 1). If not, fail the test and throw the related assertion (6.22.132). 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN, GET_MIN, GET_MAX, and GET_RES. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.35

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue a GET_DEF Request and verify that the value returned is 0 or 1. If not fail the test and throw the related assertion (6.22.133).
12. Issue a SET_CUR Request with value of 1 (Auto Mode enabled) and now issue a SET_CUR on the White Balance Temperature Control and verify that the request does not success, device answers STALL. If not, fail the test and throw the related assertion (6.22.136).
13. Issue a SET_CUR request with a value equal to a value different from the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.134). Verify also that an asynchronous notification has been sent to the Hue Control to notify of the change. If not, fail the test and throw the related assertion (6.22.135).
14. If SET_CUR is supported. Issue a SET_CUR request with invalid value(value >1). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.131).

TD 20.36 Digital Multiplier Control Test

This test verifies that the Digital Multiplier 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D14==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D14==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.36

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wMultiplierStep value. Verify that MIN<=MAX. If not, fail the test and throw the related assertion (6.20.12). Issue a GET_DEF Request and verify that the value returned between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
12. Issue a GET_RES request and verify that the answer is 1. If not, fail the test and throw the related assertion (6.22.142).
13. Issue a GET_CUR request on the Digital Multiplier Limit Control (If supported). Update then the range of valid values according to the value retrieved from the Digital Multiplier Limit Control.
14. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.140). Repeat this test with all values in the Min-LimitMax Range, fail the test and throw the related assertion if the test does not succeed.
15. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value(MIN-1, MAX+1, LONG_MIN, LONG_MAX and Zlimit+1). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.20.44 and 6.22.141).

TD 20.37 Digital Multiplier Limit Control Test

This test verifies that the Digital Multiplier Limit 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D15==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D15==1 (Control is supported).
8. 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).
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.38

10. Issue a GET_INFO request:
    a. 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).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    c. 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)
    d. 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).
11. Issue GET_MIN and GET_MAX request to store Min and Max values for the wMultiplierLimit value. Verify that MIN<MAX. If not, fail the test and throw the related assertion (6.20.12). Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertion (6.20.11).
12. Issue a GET_RES Request and verify that it returns 1. If not, fail the test and throw the related assertion (6.22.152).
13. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.22.150).
14. Issue a SET_CUR Request to the Digital Multiplier Control to set its value to Max. Issue now a SET_CUR request to the Digital Multiplier Limit Control. Verify that the request succeeded. If not, fail the test and throw the related assertion (6.22.150). Verify also that the a Control Change notification is sent by the Digital Multiplier Limit Control to notify that its value has been updated according to new limit. If not, fail the test and throw the related assertion (). Repeat the same procedure for values inside the range in decreasing order.
15. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (MIN-1, MAX+1, LONG_MIN, LONG_MAX). Verify that the device answers STALL. Check that the Request Error Code is correct. If not fail the test and throw the related assertion (6.22.151).

TD 20.43 Analog Video Standard Control Test

This test verifies that the Analog Video Standard 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 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 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
7. Check the bmControls field of the Processing Unit Descriptor. If D16==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D16==1 (Control is supported).
8. 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 (6.22.155). 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).
9. Verify that the control supports all mandatory requests: GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. Verify that Get requests is supported (D1==1). If not, fail the test and throw the related assertion (6.20.6).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.44 **Analog Video Lock Status Control Test**

This test verifies that the Analog Video Lock Status 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 performs the following steps.

11. Execute the Init procedure
12. Put the device in the desired state
13. Issue a Get configuration Descriptor command for the selected configuration with a length of 9 bytes.
14. 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.
15. 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.
16. 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 Processing Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_PROCESSING_UNIT. If no Processing Unit Descriptors if found, stop the test. If a Processing Unit is found, retrieve the Unit ID and begin the test on the Control.
17. Check the bmControls field of the Processing Unit Descriptor. If D17==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D17==1 (Control is supported).
18. 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 (6.22.157). 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).
19. Verify that the control supports all mandatory requests: GET_CUR and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
20. Issue a GET_INFO request:
    c. Verify that Get requests is supported (D1==1). If not, fail the test and throw the related assertion (6.20.6).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

6.5.2.2.4 Extension Unit Control Test

TD 20.38 Extension Unit Control Test

This test verifies that the Extension Unit 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 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 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 Extension Unit Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_EXTENSION_UNIT. If no Extension Unit Descriptors if found, stop the test. If an Extension Unit is found, retrieve the Unit ID and begin the test on the Control.
7. For every Control specified in bmControls and with a value less than bNumControls, execute steps 8 to 11. For every other Control verify that they are not supported. If not, fail the test and throw the related assertions (6.22.161 and 6.22.162).
8. Issue a GET_LEN request. If the request completed with success, verifies that value returned is valid. If not, fail the test and throw the related assertion (6.20.13). 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 (6.22.160 and 6.20.42).
9. The LEN value is now the length of Parameter returned in every request, verify that every supported request return the correct Length. If not, fail the test and throw the related assertion (6.20.15).
10. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
11. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If the control is Auto-Update or Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

6.5.2.2.5 Media Transport Terminal Control Tests

TD 20.39 **Transport Control Test**

This test verifies that the Transport 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 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 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 Input Or Output Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL or bDescriptorSubType == VC_OUTPUT_TERMINAL. If no Input or Output Terminal Descriptors are found, stop the test. For every Input or Output Terminal Descriptor found, check that wTerminalType=ITT_MEDIA_TRANSPORT_INPUT or wTerminalType=OTT_MEDIA_TRANSPORT_TERMINAL_OUTPUT. If we found an MTT, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Media Terminal Descriptor. If D0==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D0==1 (Control is supported).
8. 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 (6.22.177). 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.39

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If SET_CUR is supported, verify that the control is reported as Asynchronous (D4 set in GET_INFO). If not, fail the test and throw the related assertion (6.22.178).
    d. Verify that the control is reported as Auto Update (D3 set in GET_INFO). If not, fail the test and throw the related assertion (6.22.179).
    e. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    f. If the control is Asynchronous and SET_CUR supported, 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)
11. If SET_CUR is supported, retrieve the bmTransportMode Bitmap. For every Transport Mode supported, issue a SET_CUR with the value of the Transport Mode. If the control is Autoupdate or Asynchronous, verify that the correct Control Change have been issued by the Control. If not, fail the test and throw the related assertion. Verify that the request succeeded by issuing a GET_CUR. If not, fail the test and throw the related assertion (6.22.170).
12. If SET_CUR is supported, for every Transport Mode that is not supported, issue a SET_CUR with the value of the Transport Mode. Verify that the request result in s STALL and that no Control Change Interrupt is sent. If not, fail the test and throw the related assertion (6.22.180).
13. If SET_CUR is supported, for every reserved value of bmTransportMode, issue a SET_CUR with the reserved value. Verify that the request result in s STALL and that no Control Change Interrupt is sent. If not, fail the test and throw the related assertion (6.22.171, 6.22.172, 6.22.173, 6.22.174, 6.22.175).
14. If SET_CUR is supported, for every Status Mode value of bmTransportMode, issue a SET_CUR with the status value. Verify that the request result in s STALL and that no Control Change Interrupt is sent. If not, fail the test and throw the related assertion (6.22.176).

TD 20.40 Absolute Track Number (ATN) Control Test

This test verifies that the Absolute Track Number (ATN) 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 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 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 Input Or Output Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL or bDescriptorSubType == VC_OUTPUT_TERMINAL. If no Input or Output Terminal Descriptors are found, stop the test. For every Input or Output Terminal Descriptor found, check that wTerminalType=ITT_MEDIA_TRANSPORT_INPUT or wTerminalType=OTT_MEDIA_TRANSPORT_TERMINAL_OUTPUT. If we found an MTT, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Media Terminal Descriptor. If D1==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D1==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (no reserved bits Set in bmMediumType, no reserved bits in dwATN_Data). If not, fail the test and throw the related assertion (6.22.190 and 6.22.191). 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.40

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If SET_CUR is supported, verify that the control is reported as Asynchronous (D4 set in GET_INFO). If not, fail the test and throw the related assertion (6.22.195).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
11. If SET_CUR is supported. Issue a SET_CUR request with a correct dwATN_Data value to move the media to a different position. If the control is Autoupdate or Asynchronous, verify that the correct Control Change have been issued by the Controls (Transport and ATN). If not, fail the test and throw the related assertion (6.22.197). Verify that the request succeeded by issuing a GET_CUR. If not, fail the test and throw the related assertion (6.22.192).
12. If SET_CUR is supported. Issue a SET_CUR request with an invalid dwATN_Data value (some of the reserved bits D24..D31 set).Verify that the request results in STALL and that no Control Change Interrupt is sent. If not, fail the test and throw the related assertion (6.22.194).
13. If SET_CUR is supported. Issue a SET_CUR request with an invalid bmMediumType value (some of the reserved bits D4..D7 set).Verify that the request results in STALL and that no Control Change Interrupt is sent. If not, fail the test and throw the related assertion (6.22.193).

TD 20.41 Media Information Control Test

This test verifies that the Media Information 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 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 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 Input Or Output Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL or bDescriptorSubType == VC_OUTPUT_TERMINAL. If no Input or Output Terminal Descriptors are found, stop the test. For every Input or Output Terminal Descriptor found, check that wTerminalType=ITT_MEDIA_TRANSPORT_INPUT or wTerminalType=OTT_MEDIA_TRANSPORT_TERMINAL_OUTPUT. If we found an MTT, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Media Terminal Descriptor. If D2==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D2==1 (Control is supported).
8. Issue a GET_CUR request. If the request completed with success, verifies that value returned is valid (no reserved bits Set in bmMediaType, no reserved bits in bmWriteProtect). If not, fail the test and throw the related assertion (6.22.200 and 6.22.201). 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).
10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported(D0==1 and D1==0). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==0(SET_CUR Request not supported) verify that the SET_CUR Request is not supported. If not fail the test and throw the related assertion (6.20.5).
    c. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).

TD 20.42 Time Code Information Control Test

This test verifies that the Time Code Information 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 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 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 Input Or Output Terminal Descriptor (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VC_INPUT_TERMINAL or bDescriptorSubType == VC_OUTPUT_TERMINAL. If no Input or Output Terminal Descriptors are found, stop the test. For every Input or Output Terminal Descriptor found, check that wTerminalType=ITT_MEDIA_TRANSPORT_INPUT or wTerminalType=OTT_MEDIA_TRANSPORT_TERMINAL_OUTPUT. If we found an MTT, retrieve the Terminal ID and begin the test on the Control.
7. Check the bmControls field of the Media Terminal Descriptor. If D3==0, 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 (6.20.3 and 6.20.42). Do Step 8 to 13 only if D3==1 (Control is supported).
8. Issue a GET_CUR request. 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).
9. Verify that the control supports all mandatory requests: GET_CUR, GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.20.7). Verify also that if other requests are supported SET_CUR, GET_LEN, GET_MIN, GET_MAX, GET_RES and GET_DEF. If the answers is STALL, then check the Request Error Code Control (issue a GET_CUR Request) and verifies that the Request Error Code is 0x07 (Invalid request).If not, fail the test and throw the related assertion (6.20.43).

TD 20.42

10. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==Optional). If not, fail the test and throw the related assertion (6.20.6).
    b. If D1==1 in GET_INFO. Verify that the SET_CUR Request is supported. If not fail the test and throw the related assertion (6.20.7).
    c. If SET_CUR is supported, verify that the control is reported as Asynchronous (D4 set in GET_INFO). If not, fail the test and throw the related assertion (6.22.211).
    d. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.20.8).
    e. If the control is Asynchronous and SET_CUR supported, 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)
11. If SET_CUR is supported. Issue a SET_CUR request with a correct bdcFrame, bcdSecond, bcdMinute, bcdHour value to move the media to a different position. If the control is Autoupdate or Asynchronous, verify that the correct Control Change have been issued by the Controls (Transport and Media Information). If not, fail the test and throw the related assertion (6.22.213). Verify that the request succeeded by issuing a GET_CUR. If not, fail the test and throw the related assertion (6.22.210).

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.

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).

TD 23.1

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).

TD 23.1

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).

TD 23.2 Video Still Probe and Still Commit Controls Test

This test verifies that the Still Probe and Still 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.

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_*).Parse the next Descriptor for Still Image Frame Descriptor. If a Still Image Frame descriptor has been found associated with the Format, execute steps 8 to 14.
8. Verify that the Still 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 6.23.50)
9. For every Frame descriptor found, repeat steps 9 to 14 :
10. Issue a SET_CUR to Still Probe Control to initialize Format and Frame Index.
11. Issue a GET_MIN to Still Probe Control and store the Min values for bCompressionIndex and dwMaxVideoFrameSize. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.51). Verify that the returned values correspond to the fields in the Still Image Frame Descriptor. If not fail the test and throw the related assertion (6.23.63).
12. Issue a GET_MAX to Still Probe Control and store the Max values for bCompressionIndex and dwMaxVideoFrameSize. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.52). Verify that the returned values correspond to the fields in the Still Image Frame Descriptor. If not fail the test and throw the related assertion (6.23.63).Verify that MIN<=MAX for every parameter. If not, fail the test and throw the related assertions (6.23.53 and 6.23.54).
13. Issue a GET_DEF to Still Probe Control and store the DEF values for bCompressionIndex and dwMaxVideoFrameSize. Verify that the Request succeeded. If not, fail the test and throw the related assertion (6.23.55). Verify that the returned values correspond to the fields in the Still Image Frame Descriptor. If not fail the test and throw the related assertion (6.23.63).Verify that MIN<=DEF<=MAX for every parameter. If not, fail the test and throw the related assertions (6.23.56 and 6.23.57).

14. For every bCompressionIndex (retrieved in the corresponding field (bCompression(i)) of the Still Image Frame Descriptor):
    a. Issue a SET_CUR to Still Probe Control with the current values of Format/Frame indexes and bCompressionIndex, all other fields are set to 0.
    b. Then issue a GET_CUR to Still Probe Control to retrieve the negotiated values. Verify that the Request succeeded, if not fail the test and throw the related assertion (6.23.58). Save the MaxPayloadTransferSize Negotiated
    c. Verify that the MaxPaylaodTransferSize is achievable with the values provided in the alternate setting. If not, fail the test and throw the related assertion (6.23.59).
    d. Verify that the dwMaxVideoFrameSize specified by the Device correspond to on of the sizes specified in the Still Image Frame Descriptor. If not, fail the test and throw the related assertion (6.23.61).
    e. Verify that the negotiated MaxPayloadTransferSize is lower than the previous negotiated one. If not, fail the test and throw the related assertions (6.23.60).
    f. Issue a Commit to the Still Commit Control with the negotiated values and verify that the request succeeds. If not fail the test and throw the related assertion. (6.23.62)

TD 23.3 MJPEG Payload Header Validation Test

This test verifies that the MJPEG payload Header 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 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 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 MJPEG Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_MJPEG).
7. Parse the following Descriptor to find every MJPEG Frame descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FRAME_MJPEG). For every Frame Descriptor found do the following.
8. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 10. Otherwise repeat step 8 with decreasing bandwidth requirements.
9. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame. Parse all frame Header by monitoring FID bits of BFH[0], stop parsing when the bit toggles. If the bit never toggles through all the Frames, fail the test and throw the related assertion (6.23.80).
    a. Verify that EOF bit of BFH[0] is not set for this Frame. If not, fail the test and throw the related assertion (6.23.81).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.82).
    c. If the SCR bit of BFH[0] is set, verify that the SCR field (6 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.83).
    d. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.3

10. Parse every following Stream header until a header with EOF bit set is found. If no Header with EOF bit set is found, fail the test and throw the related assertion (6.23.86). For every Stream Header parsed:
    a. Verify that the FID field of BFH[0] is constant (No toggle). If not, fail the test and throw the related assertion (6.23.87).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the PTS value of the first Stream Header. If not, fail the test and throw the related assertion (6.23.88).
    c. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    d. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.4 Uncompressed Payload Header Validation Test

This test verifies that the Uncompressed payload Header 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 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 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 Uncompressed Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_UNCOMPRESSED).
7. Parse the following Descriptor to find every Uncompressed Frame descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FRAME_UNCOMPRESSED). For every Frame Descriptor found do the following.
8. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 10. Otherwise repeat step 8 with decreasing bandwidth requirements.
9. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame. Parse all frame Header by monitoring FID bits of BFH[0], stop parsing when the bit toggles. If the bit never toggles through all the Frames, fail the test and throw the related assertion (6.23.80).
    a. Verify that EOF bit of BFH[0] is not set for this Frame. If not, fail the test and throw the related assertion (6.23.81).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.82).
    c. If the SCR bit of BFH[0] is set, verify that the SCR field (6 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.83).
    d. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.4

10. Parse every following Stream header until a header with EOF bit set is found. If no Header with EOF bit set is found, fail the test and throw the related assertion (6.23.86). For every Stream Header parsed:
    a. Verify that the FID field of BFH[0] is constant (No toggle). If not, fail the test and throw the related assertion (6.23.87).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the PTS value of the first Stream Header. If not, fail the test and throw the related assertion (6.23.88).
    c. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    d. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.5 DV Payload Header Validation Test

This test verifies that the DV payload Header 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 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 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 DV Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_DV).
7. Parse the following Descriptor to find every DV Frame descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FRAME_DV). For every Frame Descriptor found do the following.
8. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 10. Otherwise repeat step 8 with decreasing bandwidth requirements.
9. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame. Parse all frame Header by monitoring FID bits of BFH[0], stop parsing when the bit toggles. If the bit never toggles through all the Frames, fail the test and throw the related assertion (6.23.80).
    a. Verify that EOF bit of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.81).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.82).
    c. If the SCR bit of BFH[0] is set, verify that the SCR field (6 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.83).
    d. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.5

10. For every Stream Header parsed:
    a. Verify that the EOF bit is never set. If not fail the test and throw the related assertion (6.23.81).
    b. Verify that the FID field of BFH[0] is constant (No toggle). If not, fail the test and throw the related assertion (6.23.87).
    c. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the PTS value of the first Stream Header. If not, fail the test and throw the related assertion (6.23.88).
    d. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    e. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    f. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.6 Frame based Payload Header Validation Test

This test verifies that the Frame based payload Header 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 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 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 Frame Based Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_FRAME_BASED).
7. Parse the following Descriptor to find every Frame Based Frame descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FRAME_FRAME_BASED). For every Frame Descriptor found do the following.
8. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 10. Otherwise repeat step 8 with decreasing bandwidth requirements.
9. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame. Parse all frame Header by monitoring FID bits of BFH[0], stop parsing when the bit toggles. If the bit never toggles through all the Frames, fail the test and throw the related assertion (6.23.80).
    a. Verify that EOF bit of BFH[0] is not set for this Frame. If not, fail the test and throw the related assertion (6.23.81).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.82).
    c. If the SCR bit of BFH[0] is set, verify that the SCR field (6 bytes) is set and is consistent with the dwClockFrequency value specified in the stored Probe Commit value. If not, fail the test and throw the related assertion (6.23.83).
    d. Verify also that the STI field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.84).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.6

10. Parse every following Stream header until a header with EOF bit set is found. If no Header with EOF bit set is found, fail the test and throw the related assertion (6.23.86). For every Stream Header parsed:
    a. Verify that the FID field of BFH[0] is constant (No toggle). If not, fail the test and throw the related assertion (6.23.87).
    b. If the PTS bit of BFH[0] is set, verify that the PTS field (4 bytes) is set and is consistent with the PTS value of the first Stream Header. If not, fail the test and throw the related assertion (6.23.88).
    c. Verify also that the STI field of BFH[0] is not set. If not, fail the test and throw the related assertion (6.23.84).
    d. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    e. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).

TD 23.7 MPEG2 TS Payload Header Validation Test

This test verifies that the MPEG2 TS payload Header 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 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 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 MPEG2 TS Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_MPEG2TS).
7. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 11. Otherwise repeat step 8 with decreasing bandwidth requirements.
8. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame.
9. For every Stream header verify that:
    a. Verify that the PTS field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.100).
    b. Verify that the SCR field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.101).
    c. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).
    d. Verify that the STI field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.84).
    e. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    f. Verify that EOH is set to 1. If not fail the test and throw the related assertion (6.23.102).
10. If D0 of bmFramingInfo is not set in the stored Commit Structure, parse all Parse all Stream Headers by monitoring FID bit of BFH[0]. Verify that the FID is never set. If not, fail the test and throw the related assertion (6.23.103).
11. If D1 of bmFramingInfo is not set in the stored Commit Structure, parse all Parse all Stream Headers by monitoring EOF bit of BFH[0]. Verify that the EOF is never set. If not, fail the test and throw the related assertion (6.23.104).
12.

TD 20.9 Stream Based Payload Header Validation Test

This test verifies that the Stream Based payload Header 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 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 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 Stream Based Format Descriptors (bDescriptorType == CS_INTERFACE and bDescriptorSubType == VS_FORMAT_STREAM_BASED).
7. Issue a Probe and Commit negotiation to set the compression parameters and retrieve the MaxPaylaodTransferSize. Attempt to select the corresponding alternate setting to start the stream. If the Alternate setting selection succeeds, store the value returned by the Commit Control and perform steps 9 to 11. Otherwise repeat step 8 with decreasing bandwidth requirements.
8. Poll data from the Device for a complete duration of at least two Full Frames. This will ensure that we will have stored an entire Frame.
9. For every Stream header verify that:
    a. Verify that the PTS field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.100).
    b. Verify that the SCR field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.101).
    c. Verify that the RES field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.85).
    d. Verify that the STI field of BFH[0] is set to zero. If not, fail the test and throw the related assertion (6.23.84).
    e. If the ERR bit of BFH[0] is set, issue a SET_CUR to the Stream Error Code Control and verify that the value returned is different from 0. If not, fail the test and throw the related assertion (6.23.89).
    f. Verify that EOH is set to 1. If not fail the test and throw the related assertion (6.23.102).
10. If D0 of bmFramingInfo is not set in the stored Commit Structure, parse all Stream Headers by monitoring FID bit of BFH[0]. Verify that the FID is never set. If not, fail the test and throw the related assertion (6.23.103).
11. If D1 of bmFramingInfo is not set in the stored Commit Structure, parse all Parse all Stream Headers by monitoring EOF bit of BFH[0]. Verify that the EOF is never set. If not, fail the test and throw the related assertion (6.23.104).

TD23.10 **Synch Delay Control Test**

This test verifies that the Synch Delay 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 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 Video Streaming Interface Descriptors (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING).
7. For every Video Streaming Interface found:
8. Issue a GET_CUR request to the Synch Delay Control. If the control is supported execute steps 9 to 15. If the Device answered STALL, the control is not supported and end of the test.
9. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_LEN, GET_RES and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.30.4). Verify also that GET_LEN answers the correct Length (2). If not, fail the test and throw the related assertion (6.30.10). Verify also that all requests return the correct size of parameter. If not, fail the test and throw the related assertion (6.30.12).
10. Verify that the value returned by a GET_CUR request is between MIN and MAX. If not, fail the test and throw the related assertion (6.30.13).
11. Issue a GET_INFO request:
    e. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.30.3).
    f. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.30.5).
    g. 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 (6.30.6)
    h. 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 (6.30.7).
12. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (MIN-1, MAX+1, CHAR_MIN, CHAR_MAX). Verify that the device answers STALL. If not fail the test and throw the related assertion (6.30.30).

TD 23.10

13. Issue GET_MIN and GET_MAX request to store Min and Max values for the wMultiplierLimit value. Verify that MIN<MAX. If not, fail the test and throw the related assertion (6.30.11). Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertion (6.30.8).
14. Issue a GET_RES Request and verify that (MAX-MIN)/RES is an integral number. If not, fail the test and throw the related assertion (6.30.14).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.30.31). Repeat this step for all the values in the Range.

TD 23.11 Still Image Trigger Control Test

This test verifies that the Still Image Trigger 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 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 a Class Specific VC Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_INPUT_HEADER). If no input Header found, issue a GET_CUR request to the Still Image Trigger Control and verify that the device answers Stall. If not, fail the test and throw the related assertion (6.30.50), exit the test. If an Input Header was found, check the bStillCaptureMethod field. If the method of capture is method 1, issue a GET_CUR to the Still Image Trigger Control and verify that the control is not supported. If not, fail the test and throw the related assertion (6.30.51), exit the test. If the method is method 2 or 3, execute steps 7 to 17.
7. Parse the Descriptors to find Video Streaming Interface Descriptors (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING).
8. For every Video Streaming Interface found:
9. Issue a GET_CUR request to the Still Image Trigger Control. If the Device answered STALL, fail the test and throw the related assertion (6.30.2).
10. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.30.4). Verify also that if GET_LEN is supported, it answers the correct Length (1). If not, fail the test and throw the related assertion (6.30.10). Verify also that all requests return the correct size of parameter. If not, fail the test and throw the related assertion (6.30.12).
11. Verify that the value returned by a GET_CUR request is 0, 1 or 2. If not, fail the test and throw the related assertion (6.30.13).
12. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.30.3).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.30.5).
    c. 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 (6.30.6)
    d. 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 (6.30.7).

TD23.11

13. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (3). Verify that the device answers STALL. If not fail the test and throw the related assertion (6.30.52).
14. Issue a Probe and Commit negotiation to the Device and select the negotiated alternate setting in order to ensure that the Device is Streaming.
15. If the Still capture Method is 2, issue a SET_CUR to the Still Image Control with a value of 2. Verify that the device answers STALL. If not, fail the test and throw the related assertion (6.30.53). If the Still capture Method is 3, issue a SET_CUR to the Still Image Control with a value of 1. Verify that the device answers STALL. If not, fail the test and throw the related assertion (6.30.54). Issue a SET_CUR request with a value of 0. Verify that the request succeeded. If not fail the test and throw the related assertion (6.30.55).
16. If the Still Image Capture Method is 2, issue a SET_CUR request with a value equal to 1. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.30.56). Verify that the Still Image is sent. If not fail the test and throw the related assertion (6.30.57). Verify also that the control returns to a value of 0 after transmission of the image (Issue a GET_CUR to check the value). If not, fail the test and throw the related assertion (6.30.58).
17. If the Still Image Capture is 3, issue a SET_CUR request with a value equal to 2. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.30.56). Verify that the Still Image is sent. If not fail the test and throw the related assertion (6.30.57). Verify also that the control returns to a value of 0 after transmission of the image (Issue a GET_CUR to check the value). If not, fail the test and throw the related assertion (6.30.58).
18. Issue a SET_CUR with the value corresponding to the right Still Image Capture Method. Immediately after issue a SET_CUR with a value of 0 to abort the transmission of the Still Image. Verify that the Request succeeded and that the transmission has stopped. If not, fail the test and throw the related assertion (6.30.59). Verify also that the Stream Error Code control answers 0x07 (Still Image Capture error). If not, fail the test and throw the related assertion (6.30.60).

TD 23.12 Generate Key Frame Control Test

This test verifies that the Generate Key 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 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 a Class Specific VC Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_INPUT_HEADER). If no input Header found, issue a GET_CUR request to the Generate Key Frame Control and verify that the device answers Stall. If not, fail the test and throw the related assertion (6.30.70), exit the test. If an Input Header was found, execute steps 7 to 17.
7. In the Input Header, store the Format Indexes of the Formats that support the Control (bit D4 set in bmaControls(i) for Format Index i).
8. Parse the Descriptors to find Video Streaming Interface Descriptors (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING).
9. For every Video Streaming Interface found and every Format Index found for this interface. Issue a Probe and Commit Negotiation for the corresponding Format Index in order to start a Stream.
10. If in the Input Header Generate Key Frame Control is not supported for the corresponding Format, issue a GET_CUR request and verify that the Device answer Stall. If not, fail the test and throw the related assertion (6.30.2) If the Control is supported for this Format, repeat steps 11 to 17:
11. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, and GET_INFO. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.30.4). Verify also that if GET_LEN is supported, it answers the correct Length (1). If not, fail the test and throw the related assertion (6.30.10). Verify also that all requests return the correct size of parameter. If not, fail the test and throw the related assertion (6.30.12).
12. Verify that the value returned by a GET_CUR request is 0 or 1. If not, fail the test and throw the related assertion (6.30.13).

TD 23.12

13. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.30.3).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.30.5).
    c. 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 (6.30.6)
    d. 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 (6.30.7).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (2). Verify that the device answers STALL. If not fail the test and throw the related assertion (6.30.71).
15. If SET_CUR is supported. Issue a SET_CUR request with a value of 1. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.30.72). Verify that a Key Frame has been transmitted. If not fail the test and throw the related assertion (6.30.73). Verify by issuing a GET_CUR to the Generate Key Frame Control to verify that the control is automatically reset to 0 after transmission of the Key Frame. If not, fail the test and throw the related assertion (6.30.74).

TD 23.13 Update Frame Segment Control Test

This test verifies that the Update Frame Segment 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 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 a Class Specific VC Interface Input Header Descriptor (bDescriptorType==CS_INTERFACE and bDescriptorSubType==VS_INPUT_HEADER). If no input Header found, issue a GET_CUR request to the Update Frame Segment Control and verify that the device answers STALL. If not, fail the test and throw the related assertion (6.30.90), exit the test. If an Input Header was found, execute steps 7 to 15.
7. In the Input Header, store the Format Indexes of the Formats that support the Control (bit D5 set in bmaControls(i) for Format Index i).
8. Parse the Descriptors to find Video Streaming Interface Descriptors (bDescriptorType==INTERFACE and bInterfaceClass==CC_VIDEO and bInterfaceSubClass==SC_VIDEOSTREAMING).
9. For every Video Streaming Interface found and every Format Index found for this interface. Issue a Probe and Commit Negotiation for the corresponding Format Index in order to start a Stream.
10. If in the Input Header Update Frame Segment Control is not supported for the corresponding Format, issue a GET_CUR request and verify that the Device answer Stall. If not, fail the test and throw the related assertion (6.30.2) If the Control is supported for this Format, repeat steps 11 to 15:
11. Verify that the control supports all mandatory requests: SET_CUR, GET_CUR, GET_INFO, GET_MIN, GET_MAX, GET_RES, and GET_DEF. Issue all those request and verify that the request succeeded. If not, fail the test and throw the related assertion (6.30.4). Verify also that if GET_LEN is supported, it answers the correct Length (2). If not, fail the test and throw the related assertion (6.30.10). Verify also that all requests return the correct size of parameter. If not, fail the test and throw the related assertion (6.30.12).
12. Verify that the value returned by a GET_CUR request is between MIN and MAX. If not, fail the test and throw the related assertion (6.30.13).

TD 23.13

13. Issue a GET_INFO request:
    a. Verify that Set and Get requests are supported (D0==1 and D1==1). If not, fail the test and throw the related assertion (6.30.3).
    b. If the Control is Asynchronous, verify that a Status Interrupt Endpoint is present. If not, fail the test and throw the related assertion (6.30.5).
    c. 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 (6.30.6)
    d. 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 (6.30.7).
14. If SET_CUR is supported. Issue a SET_CUR request with out-of-bound value (MIN-1, MAX+1, SHORT_MIN, SHORT_MAX). Verify that the device answers STALL. If not fail the test and throw the related assertion (6.30.91).
16. Issue GET_MIN and GET_MAX request to store Min and Max values for the wMultiplierLimit value. Verify that MIN<MAX. If not, fail the test and throw the related assertion (6.30.11). Issue a GET_DEF Request and verify that the value returned is between the Min-Max Range. If not fail the test and throw the related assertion (6.30.8).
17. Issue a GET_RES Request and verify that (MAX-MIN)/RES is an integral number. If not, fail the test and throw the related assertion (6.30.14).
15. If SET_CUR is supported. Issue a SET_CUR request with a value equal to the default value. Verify that the request succeeded and issue a GET_CUR Request to verify that the value has been set. If not, fail the test and throw the related assertion (6.30.92). Repeat this step for all the values in the Range.