Universal Serial Bus Device Class Definition for Video Devices

Revision 1.5
August 9, 2012

Contributors

Hans van Antwerpen	Cypress Semiconductor
Eric Luttmann	Cypress Semiconductor
David Roh	Dolby Labs
Choon Chng	Google Inc.
Pawel Osciak	Google Inc.
Ville-Mikko Rautio	Google Inc.
Van Duros	Immedia Semiconductor Inc.
Abdul R. Ismail	Intel Corp.
Bradley Saunders	Intel Corp.
Ygal Blum	Jungo
Yoav Nissim	Jungo
Jean-Michel Chardon	Logitech Inc.
Olivier Lechenne	Logitech Inc.
Geraud Mudry	Logitech Inc.
Chandrashekhar Rao	Logitech Inc.
Remy Zimmermann	Logitech Inc.
Chris Yokum	MCCI Corporation
Stephen Cooper	Microsoft Corp.
Maribel Figuera	Microsoft Corp.
Richard Webb	Microsoft Corp.
Anand Ganesh	Microsoft Corp.
David Goll	Microsoft Corp.
Hiro Kobayashi	Microsoft Corp.
Bertrand Lee	Microsoft Corp.
Jeff Zhu	Microsoft Corp.
Andrei Jefremov	Microsoft Corp.
Tim Vlaar	Point Grey Research
Mark Bohm	SMSC
John Sisto	SMSC
Will Harris	Texas Instruments
Grant Ley	Texas Instruments
Anshuman Saxena	Texas Instruments
Paul E. Berg	USB-IF

Please send comments via electronic mail to <video-chair>@usb.org

Revision History

Version	Date	Description
1.0	September 4, 2003	Initial release
1.0a	December 4, 2003	Table A.5: Added extension reference row Table A.6: Added USB extension row Table A.6: Defined VS_FORMAT_MPEG4SL as 0x0B Added section A,9.8 for selectors extensibility Table B.1, B.2, B3 & B4: Added extension row Added Appendix D. Revision history
1.1	June 1st, 2005	Added GET_INFO request for VideoStreaming interface Updated Table 4-56 Stream Error Code Control Updated Table 3-8 to correct the wMaxMultiplier description Add support for multiple clock frequencies (RR0033) Latency optimizations for Stream-based formats (RR0041) Define Probe/Commit controls for Render (RR0042) Add Analog Video Standard and Status Control (RR0044) Define constraints on RES value (RR0047) Detail behavior of Multiplier and Limit Multiplier (RR0048) Cosmetic changes to Terminal and Unit Descriptors (RR0049) Cosmetic and Functional changes(RR0050) Clarify usage of bits in GET_INFO (RR0051) Specify the meanings of Request Error Codes (RR0053) Allows support for a revision of a Payload specification to be made independently of the Core Specification. Added three fields to the Video Probe And Commit Control (RR0054). Specification of Absolute and Relative Control relationship. (RR057). Specification of Asynchronous Controls' behavior (RR0059). Allow 0 for Payload Version in Probe and Commit Control (RR0060). Modified Table 2-6 Extended Field of the Payload Header, Table 3-15 Payload Format Descriptor and Section 2.4.2.4 Still Image Capture. (RR0066) Remove "Driver" from Terms and abbreviation and added UVC. Updated section 2.4.3.7, change VDC to UVC. Update section 3.7.2 and Table 3-3: Change bcdVDC with bcdUVC. (RR0064) Corrected Statement regarding multiple clock support (RR0069). Removed auto-update side-effect from Probe/Commit Frame Interval field (RR0070). Updated Descriptor Size inTable 3-8 Processing Unit Descriptor (RR0072). Updated range of reserved values in Table 4-43 Analog Video

		Standard Control (RR0072). Remove reference to “Vendor Unique Payload Format” in Table 4-47 Video Probe and Commit Controls (RR0072). Marked obsolete format and Frame descriptor type values as reserved (VS_FORMAT_MPEG1, VS_FORMAT_MPEG2PS, VS_FORMAT_MPEG4, VS_FORMAT_VENDOR, VS_FRAME_VENDOR) in Table A- 6 Video Class-Specific VS Interface Descriptor Subtypes (RR0072). Added new format and Frame descriptor types (VS_FORMAT_FRAME_BASED,VS_FRAME_FRAME_BASED, VS_FORMAT_STREAM_BASED) in Table A- 6 Video Class-Specific VS Interface Descriptor Subtypes (RR0072). Added new Processing Unit Controls (PU_ANALOG_VIDEO_STANDARD_CONTROL, PU_ANALOG_LOCK_STATUS_CONTROL) in Table A- 13 Processing Unit Control Selectors (RR0072).
1.5	June 6, 2012	Updated **bcdUVC** to 1.5. Updated Probe and Commit to include features of temporally encoded video that impact bandwidth negotiation, increasing the size of the VS_PROBE_CONTROL and VS_COMMIT_CONTROL. Added twenty Encoding Unit controls to facilitate control of device based video encoders. Added three new controls to the Camera Terminal: CT_FOCUS_SIMPLE_CONTROL, CT _WINDOW_CONTROL, and CT_REGION_OF_INTEREST_CONTROL. Added one new control to the Processing Unit: PU_CONTRAST_AUTO_CONTROL. Added solution for backward compatibility of UVC 1.5 devices. Added bit to GET_INFO to declare control as “Disabled for current Commit state” Removed reference section Device_Qualifier Descriptor and Other_Speed_Configuration Descriptor. Updated Data Format Classes (section 2.4.3.7) with temporally encoded video formats. Added Notations section. Added references to *USB Specification 3.0*. Moved discussion on Optical Zoom to appendix.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

This document describes the minimum capabilities and characteristics that a video streaming device must support to comply with the USB Video Class specification.

It defines and standardizes video streaming functionality on the USB, and contains all necessary information for a designer to build a USB-compliant device that incorporates video streaming functionality. It specifies the standard and class-specific descriptors that must be present in each USB video function. It further explains the use of class-specific requests that allow for full video streaming control. Finally, it explains how devices can be compliant with multiple versions of this specification to enable backwards compatibility.

Devices that conform to this specification will be referred to as USB Video Class devices.

1.2 Scope

The USB Device Class Definition for Video Devices applies to all devices or functions within composite devices that are used to manipulate video and video-related functionality. This would include devices such as desktop video cameras (or "webcams"), digital camcorders, analog video converters, analog and digital television tuners, and still-image cameras that support video streaming. This specification also applies to Video Devices that compress video using temporal encoders.

1.3 Related Documents

USB Specification Revision 3.0, November 12, 2008, www.usb.org
USB Specification Revision 2.0, April 27, 2000, www.usb.org
Interface Association Descriptor ECN, www.usb.org

1.4 Document Conventions

The following typographic conventions are used:

- *Italic* Documents references
- **Bold** Request fields
- UPPERCASE Constants

The following terms are defined:

- Expected
  a keyword used to describe the behavior of the hardware or software in the design models assumed by this specification. Other hardware and software design models may also be implemented
- May/Could
  a keyword that indicates flexibility of choice with no implied preference.

- Shall/Must
  keywords indicating a mandatory requirement. Designers are required to implement all such mandatory requirements.
- Should
  a keyword indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase is recommended.

1.4.1 Notations

The following notations are used in this specification and all associated video payload and example documents.

Notation	Description
SET_INTERFACE (n)	This notation indicates a SET_INTERFACE request as defined section 9.4.10 of *USB Specification 2.0* and *USB Specification 3.0* where **wValue** = n.
Control_Name(request_type)	This notation indicates a specific request_type being issued to a specific Control_Name.
EU_*_CONTROL(request_type)	This notation indicates a specific request_type issued to any Encoding Unit.

1.5 Terms and Abbreviations

Term	Description
Configuration	A collection of one or more interfaces that may be selected on a USB device.
Control	A logical object within an Entity that is used to manipulate a specific property of that Entity.
CT	Camera terminal.
Descriptor	Data structure used to describe a USB device capability or characteristic.
Device	USB peripheral.
Endpoint	Source or sink of data on a USB device.
Entity	A Unit, Terminal or Interface within the video function, each of which may contain Controls.
EOS	End of Slice. A bit marker in the video frame header that indicates the associated video payload contains the last bit in the current Slice.
GUID	Globally Unique Identifier. Also known as a universally unique identifier (UUID). The Guidgen.exe command line program from Microsoft is used to create a GUID. Guidgen.exe never produces the same GUID twice, no matter how many times it is run or how many different machines it runs on. Entities such as video formats that need to be uniquely identified have a GUID. Search www.microsoft.com for more information on GUIDs and Guidgen.exe.

Host	Computer system where a Host Controller is installed.
Host Controller	Hardware that connects a Host to the USB.
Host Software	Generic term for a collection of drivers, libraries and/or applications that provide operating system support for a device.
IAD	Interface Association Descriptor. This is used to describe that two or more interfaces are associated to the same function. An 'association' includes two or more interfaces and all of their alternate setting interfaces.
IDR	Instantaneous Decoder Refresh. An I-Frame where no frames after the IDR frame can rely on frames before the IDR frame.
Interface	An Entity representing a collection of zero or more endpoints that present functionality to a Host.
IT	Input Terminal.
Dependency Layer	The same as dependency representation in the H.264 specification and in the VP8 payload specification.
Multicast Streaming	Two or more independent video streams originating from the same source (Video Function) sent over separate video streaming interfaces..
OT	Output Terminal.
Payload Transfer	In the context of the USB Video Class, a Payload Transfer is a unit of data transfer common to bulk and isochronous endpoints. Each Payload Transfer includes a Payload Header followed by Payload Data. For isochronous endpoints, a Payload Transfer is contained in the data transmitted during a single (micro) frame: up to 1023 bytes for a full-speed endpoint; up to 1024 bytes for a high-speed endpoint; and up to 3072 bytes for a high-speed/high-bandwidth endpoint. For bulk endpoints, a Payload Transfer is contained in the data transmitted in a single bulk transfer (which may consist of multiple bulk data transactions).
Payload Data	Format-specific data contained in a Payload Transfer (excluding the Payload Header).
Payload Header	A header at the start of each Payload Transfer that provides data framing and encapsulation information.
PTS	Presentation Time Stamp. This is the source clock time in native device clock units when the raw frame capture begins.
PU	Processing Unit.
Request	A mechanism supported by the video function for the host software to interact with a Control within an Entity.
QP	Quantization Parameter. This is an index used to derive a scaling matrix for video encoding.
Sample Transfer	A sample transfer is composed of one or more payload transfer(s) representing a video sample.
SCR	Source Clock Reference. This is a two part value that contains (1) the source clock time when the video sample leaves the device and (2) the

	current 1 KHz SOF counter.
SEI	Supplemental Enhancement Information as defined in the H.264 specification.
Simulcast Stream	Simulcast Streams are multiple concurrent, independently coded bit streams from the same source streamed over single Video Streaming Interface.
SOF	Start of Frame. Transmitted by the USB host every millisecond, the SOF token contains an 11 bit incremental frame number.
STC	Source Time Clock. The clock used by the data source that governs the sampling of video (or related) data.
SU	Selector Unit.
TD	Terminal Descriptor.
Terminal	An Entity representing a source (Input Terminal) or sink (Output Terminal) for data flowing into or out of a video function.
UD	Unit Descriptor.
Unit	An Entity representing a transformation of data flowing through a video function.
USB	Universal Serial Bus.
USB Transaction	See USB 2.0 Chapter 5.
USB Transfer	See USB 2.0 Chapter 5.
UVC	USB Video Class.
VC	VideoControl; refers to the interface used for video function control.
VIC	Video Interface Collection; refers to the collection of VideoControl and VideoStreaming interfaces within the same video function.
VS	VideoStreaming; refers to the interface(s) used for video stream transport.
XU	Extension Unit.

2 Functional Characteristics

The video function is located at the interface level in the device class hierarchy. It consists of a number of interfaces grouping related pipes that together implement the interface to the video function.

Video functions are addressed through their video interfaces. Each video function has a single VideoControl (VC) interface and can have several VideoStreaming (VS) interfaces. The VideoControl (VC) interface is used to access the device controls of the function whereas the VideoStreaming (VS) interfaces are used to transport data streams into and out of the function. The collection of the single VideoControl interface and the VideoStreaming interfaces that belong to the same video function is called the Video Interface Collection (VIC). An Interface Association Descriptor (IAD) is used to describe the Video Interface Collection.

2.1 Video Interface Class

The Video Interface class groups all functions that can interact with USB-compliant video data streams. All functions that convert between analog and digital video domains can be part of this class. In addition, those functions that transform USB-compliant video data streams into other USB-compliant video data streams can be part of this class. Even analog video functions that are controlled through USB belong to this class.

In fact, for a video function to be part of this class, the only requirement is that it exposes one VideoControl Interface. No further interaction with the function is mandatory, although most functions in the video interface class will support one or more optional VideoStreaming interfaces for consuming or producing one or more video data streams.

The Video Interface class code is assigned by the USB. For details, see section A.1 "Video Interface Class Code".

2.2 Video Interface Subclass and Protocol

The Video Interface class is divided into subclasses as identified by the Interface Subclass code. The following two interface subclasses are defined in this specification.

- VideoControl Interface
- VideoStreaming Interface

The following Function Subclass is used in the Interface Association Descriptor.

- Video Interface Collection

The assigned codes can be found in sections A.2, "Video Interface Subclass Codes" and A.3, "Video Interface Protocol Codes" of this specification. All other subclass codes are unused and reserved except code 0xFF, which is reserved for vendor-specific extensions.

2.3 Video Function Topology

To be able to manipulate the physical properties of a video function, its functionality must be divided into addressable entities. The following two generic entities are identified:

- Units
- Terminals

Units provide the basic building blocks to fully describe most video functions. Video functions are built by connecting together several of these Units. A Unit has one or more Input Pins and a single Output Pin, where each Pin represents a cluster of logical data streams inside the video function.

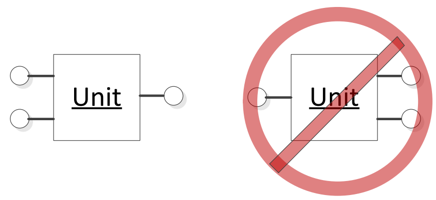


Figure 2-1: One or more Input Pins, single Output Pin

Units are wired together by connecting their I/O Pins according to the required topology. A single Output Pin can be connected to one or more Input Pins (fan-out allowed). However, a single Input Pin can only be connected to one Output Pin (fan-in disallowed). Loops or cycles within the graph topology are not allowed.

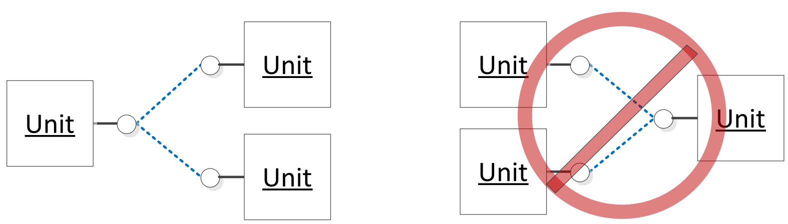


Figure 2-2: Fan-out Allowed / Fan-in Disallowed

In addition, the concept of Terminal is introduced. There are two types of Terminals. An Input Terminal (IT) is an entity that represents a starting point for data streams inside the video function. An Output Terminal (OT) represents an ending point for data streams. From the video function's perspective, a USB endpoint is a typical example of an Input Terminal or Output Terminal. It either provides data streams to the video function (IT) or consumes data streams coming from the video function (OT). Likewise, a Charge Coupled Device (CCD) sensor, built into the video function is represented as an Input Terminal in the video function's model. Connection to a Terminal is made through its single Input Pin or Output Pin.

Input Pins of a Unit are numbered starting from one up to the total number of Input Pins on the Unit. The Output Pin number is always one. Terminals have one Input or Output Pin that is always numbered one.

The information traveling over I/O Pins is not necessarily of a digital nature. It is possible to use the Unit model to describe fully analog or even hybrid video functions. The mere fact that I/O Pins are connected together is a guarantee (by construction) that the protocol and format, used over these connections (analog or digital), is compatible on both ends.

Every Unit in the video function is fully described by its associated Unit Descriptor (UD). The Unit Descriptor contains all necessary fields to identify and describe the Unit. Likewise, there is a Terminal Descriptor (TD) for every Terminal in the video function. In addition, these descriptors provide all necessary information about the topology of the video function. They fully describe how Terminals and Units are interconnected.

The descriptors are further detailed in section 3, "Descriptors" of this document.

This specification describes the following types of standard Units and Terminals that are considered adequate to represent most video functions available today and in the near future:

- Input Terminal
- Output Terminal
- Selector Unit
- Processing Unit
- Encoding Unit
- Extension Unit

Also, there are certain special Terminals that extend the functionality of the basic Input and Output Terminals. These special Terminals support additional Terminal Descriptor fields and Requests that are specific to the extended features these Terminals provide. These include:

- Media Transport Terminal (defined in *USB Device Class Definition for Video Media Transport Terminal* specification)
- Camera Terminal

The types of Units defined in this specification could be extended in future revisions, or via companion specifications. For example, a Tuner Unit could be added as a companion specification to accommodate devices with TV Tuners.

Inside a Unit or Terminal, functionality is further described through Video Controls. A Control typically provides access to a specific video property. Each Control has a set of attributes that can be manipulated or that present additional information about the behavior of the Control. Controls have attributes, which might include:

- Current setting
- Minimum setting
- Maximum setting
- Resolution
- Size
- Default

Consider a Brightness Control inside a Processing Unit. By issuing the appropriate requests, the Host software can obtain values for the Brightness Control's attributes and, for instance, use them to correctly display the Control in a User Interface. Setting the Brightness Control's *current setting* attribute allows the Host software to change the brightness of the video that is being streamed.

The ensemble of Unit Descriptors, Terminal Descriptors and Video Controls provide a full description of the video function to the Host. A generic class driver shall be able to fully control the video function. When functionality is represented by Extension Units, the class driver shall permit access to vendor-specific extensions via a pass-through mechanism. The implementation details of such a class driver are beyond the scope of this specification.

2.3.1 Input Terminal

The Input Terminal (IT) is used as an interface between the video function's "outside world" and other Units inside the video function. It serves as a receptacle for data flowing into the video function. Its function is to represent a source of incoming data after this data has been extracted from the data source. The data may include audio and metadata associated with a video stream. These physical streams are grouped into a cluster of logical streams, leaving the Input Terminal through a single Output Pin.

An Input Terminal can represent inputs to the video function other than USB OUT endpoints. A CCD sensor on a video camera or a composite video input is an example of such a non-USB input. However, if the video stream is entering the video function by means of a USB OUT endpoint, there is a one-to-one relationship between that endpoint and its associated Input Terminal. The class-specific Output Header descriptor contains a field that holds a direct reference to this Input Terminal (see section 3.9.2.2, "Output Header Descriptor"). The Host needs to use both the endpoint descriptors and the Input Terminal descriptor to get a full understanding of the characteristics and capabilities of the Input Terminal. Stream-related parameters are stored in the endpoint descriptors. Control-related parameters are stored in the Terminal descriptor.

The symbol for the Input Terminal is depicted in the following figure.

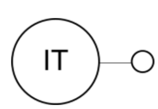


Figure 2-3 Input Terminal Icon

2.3.2 Output Terminal

The Output Terminal (OT) is used as an interface between Units inside the video function and the "outside world". It serves as an outlet for video information, flowing out of the video function. Its function is to represent a sink of outgoing data. The video data stream enters the Output Terminal through a single Input Pin.

An Output Terminal can represent outputs from the video function other than USB IN endpoints. A Liquid Crystal Display (LCD) screen built into a video device or a composite video out connector are examples of such an output. However, if the video stream is leaving the video function by means of a USB IN endpoint, there is a one-to-one relationship between that endpoint and its associated Output Terminal. The class-specific Input Header descriptor contains a field that holds a direct reference to this Output Terminal (see section 3.9.2.1, “Input Header Descriptor”). The Host needs to use both the endpoint descriptors and the Output Terminal descriptor to fully understand the characteristics and capabilities of the Output Terminal. Stream-related parameters are stored in the endpoint descriptors. Control-related parameters are stored in the Terminal descriptor.

The symbol for the Output Terminal is depicted in the following figure.

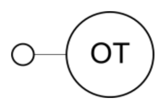


Figure 2-4 Output Terminal Icon

2.3.3 Camera Terminal

The Camera Terminal (CT) controls mechanical (or equivalent digital) features of the device component that transmits the video stream. As such, it is only applicable to video capture devices with controllable lens or sensor characteristics. A Camera Terminal is always represented as an Input Terminal with a single output pin. It provides support for the following features.

- Scanning Mode (Progressive or Interlaced)
- Auto-Exposure Mode
- Auto-Exposure Priority
- Exposure Time
- Focus
- Auto-Focus
- Simple Focus
- Iris
- Zoom

- Pan
- Roll
- Tilt
- Digital Windowing
- Region of Interest

Support for any particular control is optional. The Focus control can optionally provide support for an auto setting (with an on/off state). If the auto setting is supported and set to the on state, the device will provide automatic focus adjustment, and read requests will reflect the automatically set value. Attempts to programmatically set the Focus control when in auto mode shall result in protocol STALL with an error code of **bRequestErrorCode** = "Wrong State". When leaving Auto-Focus mode (entering manual focus mode), the control shall remain at the value that was in effect just before the transition.

2.3.4 Selector Unit

The Selector Unit (SU) selects from n input data streams and routes them unaltered to the single output stream. It represents a source selector, capable of selecting among a number of sources. It has an Input Pin for each source stream and a single Output Pin.

The symbol for the Selector Unit is depicted in the following figure.

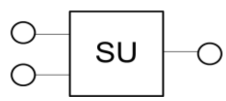


Figure 2-5 Selector Unit Icon (2 input pins)

2.3.5 Processing Unit

The Processing Unit (PU) controls image attributes of the video being streamed through it. It has a single input and output pin. It provides support for the following features:

User Controls
- Brightness
- Hue
- Saturation
- Sharpness
- Gamma
- Digital Multiplier (Zoom)

Auto Controls
- White Balance Temperature
- White Balance Component
- Backlight Compensation

- Contrast

Other
- Gain
- Power Line Frequency
- Analog Video Standard
- Analog Video Lock Status

Support for any particular control is optional. In particular, if the device supports the White Balance function, it shall implement either the White Balance Temperature control or the White Balance Component control, but not both. The User Controls indicate properties that are governed by user preference and not subject to any automatic adjustment by the device. The Auto Controls will provide support for an auto setting (with an on/off state). If the auto setting for a particular control is supported and set to the on state, the device will provide automatic adjustment of the control, and read requests to the related control will reflect the automatically set value. Attempts to programmatically set the Focus control when in auto mode shall result in protocol STALL with an error code of **bRequestErrorCode** = "Wrong State". When leaving an auto mode, the related control shall remain at the value that was in effect just before the transition.

The symbol for the Processing Unit is depicted in the following figure.

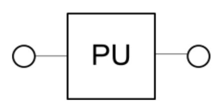


Figure 2-6 Processing Unit Icon

2.3.6 Encoding Unit

The Encoding Unit controls attributes of the encoder that encodes the video being streamed through it. It has a single input and multiple output pins. It provides support for the following features which can be used before or after streaming has started.

- Select Layer
- Video Resolution
- Profile and Toolset
- Minimum Frame Interval
- Slice Mode
- Rate Control Mode

Average Bitrate Control
CPB Size Control

- Peak Bit Rate
- Quantization Parameter
- Synchronization and Long Term Reference Frame

- Long Term Reference Buffers
- Long Term Picture
- Valid Long Term Pictures
- LevelIDC
- SEI Message
- QP Range
- Priority ID
- Start or Stop Layer
- Error Resiliency

Support for the Encoding Unit control is optional and only applicable to devices with onboard video encoders. The Select Layer control also allows control of individual streams for devices that support simulcast transport of more than one stream. Individual payloads may specialize the behavior of each of these controls to align with the feature set defined by the associated encoder, e.g. H.264. This specialized behavior is defined in the associated payload specification.

The symbol for the Encoding Unit is depicted in the following figure.

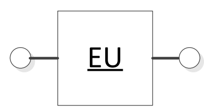


Figure 2-7 Encoding Unit Icon

2.3.7 Extension Unit

The Extension Unit (XU) is the method provided by this specification to add vendor-specific building blocks to the specification. The Extension Unit can have one or more Input Pins and has a single Output Pin.

Although a generic host driver will not be able to determine what functionality is implemented in the Extension Unit, it shall report the presence of these extensions to vendor-supplied client software, and provide a method for sending control requests from the client software to the Unit, and receiving status from the unit.

The symbol for the Extension Unit is depicted in the following figure.

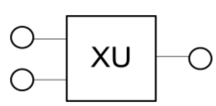


Figure 2-8 Extension Unit Icon

2.4 Operational Model

A device can support multiple configurations. Within each configuration can be multiple interfaces, each possibly having alternate settings. These interfaces can pertain to different functions that co-reside in the same composite device. Several independent video functions can exist in the same device. Interfaces that belong to the same video function are grouped into a Video Interface Collection described by an Interface Association Descriptor. If the device contains multiple independent video functions, there must be multiple Video Interface Collections (and hence multiple Interface Association Descriptors), each providing full access to their associated video function.

As an example of a composite device, consider a desktop camera equipped with a built in microphone. Such a device could be configured to have one interface collection dealing with configuration and control of the audio function, while another interface collection deals with its video aspects. One of those, the VideoControl interface, is used to control the inner workings of the function, whereas the other, the VideoStreaming interface, handles the data traffic received from the camera video subsystem.

Video Interface Collections can be dynamic in devices that support multiple operating modes. Because the VideoControl interface, together with its associated VideoStreaming interface(s), constitutes the 'logical interface' to the video function, they must all come into existence at the same moment in time. Changing the operating mode of a device causes the previous Video Interface Collection to be replaced with a new Video Interface Collection, followed by re-initialization of the host software. This specification does not provide a mechanism for the host to initiate such a mode change, which is typically initiated via a physical switch on the device.

As stated earlier, video functionality is located at the interface level in the device class hierarchy. The following sections describe the Video Interface Collection, containing a single VideoControl interface and optional VideoStreaming interfaces, together with their associated endpoints that are used for video function control and for data stream transfer.

2.4.1 Video Interface Collection

A device must use an Interface Association Descriptor to describe a Video Interface Collection for each device function that requires a VideoControl Interface and one or more VideoStreaming interfaces. The Interface Association Descriptor must always be returned as part of the device's complete configuration descriptor in response to a GetDescriptor (Configuration) request. The Interface Association Descriptor must be located before the VideoControl Interface and its associated VideoStreaming Interfaces (including all alternate settings). All of the interface numbers in the set of associated interfaces must be contiguous.

2.4.2 VideoControl Interface

To control the functional behavior of a particular video function, the Host can manipulate the Units and Terminals inside the video function. To make these objects accessible, the video function must expose a single VideoControl interface. This interface can contain the following endpoints.

- A control endpoint for manipulating Unit and Terminal settings and retrieving the state of the video function. This endpoint is mandatory, and the default endpoint 0 is used for this purpose.
- An interrupt endpoint for status returns. This endpoint is optional, but may be mandatory under certain conditions. See section 2.4.2.2, "Status Interrupt Endpoint" for further information.

The VideoControl interface is the single entry point to access the internals of the video function. All requests that are concerned with the manipulation of certain Video Controls within the video function's Units or Terminals must be directed to the VideoControl interface of the video function. Likewise, all descriptors related to the internals of the video function are part of the class-specific VideoControl interface descriptor.

This specification defines a single alternate setting for the VideoControl interface, the default alternate setting zero.

2.4.2.1 Control Endpoint

The video interface class uses endpoint 0 (the default pipe) as the standard way to control the video function using class-specific requests. These requests are always directed to one of the Units or Terminals that make up the video function. The format and contents of these requests are detailed further in this document.

2.4.2.2 Status Interrupt Endpoint

A USB VideoControl interface can support an optional interrupt endpoint to inform the Host about the status of the different addressable entities (Terminals, Units, interfaces and endpoints) inside the video function. The interrupt endpoint, if present, is used by the entire Video Interface Collection to convey status information to the Host. It is considered part of the VideoControl interface because this is the anchor interface for the Collection.

This interrupt endpoint is mandatory if:

- The device supports hardware triggers for still image capture (see section 2.4.2.3, "Hardware Trigger Interrupts").
- The device implements any *AutoUpdate controls* (controls supporting device initiated changes).
- The device implements any *Asynchronous controls* (see section 2.4.4, "Control Transfer and Request Processing").

The interrupt packet is a variable size data structure depending on the originator of the interrupt status. The **bStatusType** and **bOriginator** fields contain information about the originator of the interrupt. The **bEvent** field contains information about the event triggering the interrupt. If the originator is the Video Control interface, the **bSelector** field reports the Control Selector of the control that issued the interrupt. Any addressable entity inside a video function can be the originator.

The contents of the **bOriginator** field must be interpreted according to the code in D3..0 of the **bStatusType** field. If the originator is the VideoControl interface, the **bOriginator** field contains the Terminal ID or Unit ID of the entity that caused the interrupt to occur. If the **bOriginator** field is set to zero, the *virtual* entity interface is the originator. This can be used to report global VideoControl interface changes to the Host. If the originator is a VideoStreaming interface, the **bOriginator** field contains the interface number of the VideoStreaming interface. This scheme is unambiguous because Units and Terminals are not allowed to have an ID of zero.

If the originator is the VideoControl interface, the **bAttribute** field indicates the type of Control change.

The contents of the **bEvent** field must also be interpreted according to the code in D3..0 of the **bStatusType** field. If the originator is the VideoStreaming interface, there are additional button press events defined as described in the table below.

For all originators, there is a Control Change event defined. Controls that support this event will trigger an interrupt when a host-initiated or externally-initiated control change occurs. The interrupt shall only be sent when the operation corresponding to the control change is completed by the device.

A Control shall support Control Change events if any of the following is true:

- The Control state can be changed independently of host control.
- The Control can take longer than 10ms from the start of the Data stage through the completion of the Status stage when transferring to the device (SET_CUR operations).

If a control is required to support Control Change events, the event shall be sent for all SET_CUR operations, even if the operation can be completed within the 10ms limit. The device indicates support for the Control Change event for any particular control via the GET_INFO attribute (see section 4.1.2, "Get Request"). Section 2.4.4, "Control Transfer and Request Processing" describes in detail the interaction of Control Transfers (Requests) and Control Change events.

The following tables specify the format of the status packet.

Table 2-1 Status Packet Format

Offset	Field	Size	Value	Description
0	**bStatusType**	1	Bitmap/Number	D7..4: Reserved D3..0: Originator 0 = Reserved 1 = VideoControl interface 2 = VideoStreaming interface
1	**bOriginator**	1	Number	ID of the Terminal, Unit or Interface that reports the interrupt

When the originator is a Video Control Interface, the rest of structure is:

Table 2-2 Status Packet Format (VideoControl Interface as the Originator)

Offset	Field	Size	Value	Description
2	**bEvent**	1	Number	0x00: Control Change 0x01 – 0xFF: Reserved
3	**bSelector**	1	Number	Control Change Report the Control Selector of the control that issued the interrupt.
4	**bAttribute**	1	Number	Specify the type of control change: 0x00: Control value change 0x01: Control info change 0x02: Control failure change 0x03: Control min change 0x04: Control max change 0x05 – 0xFF: Reserved
5	**bValue**	n		See control request description in section 4.2 "VideoControl Requests". **bAttribute:** **Description:** 0x00 Equivalent to the result of a GET_CUR request 0x01 Equivalent to the result of a GET_INFO request 0x02 Equivalent to the result of a GET_CUR request on VC_REQUEST_ERROR_ CODE_CONTROL 0x03 Equivalent to the result of a GET_MIN request 0x04 Equivalent to the result of a GET_MAX request

When the originator is a Video Streaming Interface the rest of the structure is:

Table 2-3 Status Packet Format (VideoStreaming Interface as the Originator)

Offset	Field	Size	Value	Description
2	**bEvent**	1	Number	All originators: 0x00 = Button Press 0x01 – 0xFF = Stream Error
3	**bValue**	n	Number	Button Press: (n=1) 0x00: Button released 0x01: Button pressed

2.4.2.3 Hardware Trigger Interrupts

One of the defined usages of the Status Interrupt Endpoint is for hardware triggers to notify host software to initiate still image capture. When the hardware detects a button press, for example, the Status Interrupt Endpoint will issue an interrupt originating from the relevant VideoStreaming interface. The event triggering the interrupt (button press or release) is indicated in the interrupt packet. The default, initial state of the button is the "*release*" state.

The device will have to specify whether it supports hardware triggers, and how the Host software should respond to hardware trigger events. These are specified in the class-specific descriptors within the relevant VideoStreaming interface. See section 3, "Descriptors".

2.4.2.4 Still Image Capture

A common feature of video cameras is the support of still image capture associated with a video stream. This can be initiated either by programmatic software triggers or hardware triggers.

Depending on the method used, the still image frame may have to be the same size as the video frames that are being streamed. There are several supported methods of capturing the still image, and the device will have to specify which method it supports in the class-specific descriptors within the relevant VideoStreaming interface.

Method 1 - The host software will extract the next available video frame from the active video pipe in the relevant VideoStreaming interface upon receiving the hardware trigger event. The hardware does not interrupt or alter the video stream in this case. For this method, the still image frame is always the same size as the video frames being streamed.

Method 2 – If the device supports higher-quality still images, it has the option of streaming still-image-specific packets across the active video pipe. In this case, the host software will temporarily suspend video streaming, select the optimal bandwidth alternate setting based on the still probe/commit negotiation (subject to bandwidth availability), send a VS_STILL_IMAGE_TRIGGER_CONTROL Set request with the "Transmit still image" option (see section 4.3.1.4, "Still Image Trigger Control"), and prepare to receive the still image data. The device transmits the still image data marked as such in the payload header (see section 2.4.3.2.2, "Sample Isochronous Transfers"). Once the complete still image is received, the host software will then revert back to the original alternate setting, and resume video streaming.

Method 3 – This method enables the capture of higher-quality still images from a dedicated bulk still image pipe. By doing so, the active streams would continue uninterrupted. There are two cases covered by this method.

In the first case, the host software initiates the still image capture from the device. It does so by issuing a VS_STILL_IMAGE_TRIGGER_CONTROL *Set* request with the "Transmit still image via dedicated bulk pipe" option (see section 4.3.1.4, "Still Image Trigger Control"). In this case, after issuing the request, the host will start receiving the still image from the bulk still image endpoint of the relevant VideoStreaming interface. The device captures the high-quality still image and transmits the data to the bulk still image endpoint. While transmission is occurring, the **bTrigger** field of the VS_STILL_IMAGE_TRIGGER_CONTROL control shall remain as "Transmit still image via dedicated bulk pipe". After transmission is complete, the device shall reset the control to "Normal operation" and trigger a control change interrupt via the Status Interrupt endpoint.

In the second case, the device initiates the still image transmission after detecting a hardware trigger. When the hardware detects a button press, the Status Interrupt endpoint will issue an interrupt originating from the relevant VideoStreaming interface. If the **bTriggerUsage** field of the selected Format descriptor is set as initiating still image capture, the device shall set the **bTrigger** field of the VS_STILL_IMAGE_TRIGGER_CONTROL control to "Transmit still image via dedicated bulk pipe". The Host software should then begin receiving still image data that was captured by the device after it received the interrupt. After transmission is complete, the device shall reset the **bTrigger** field to "Normal operation". The host software can abort data transmission by issuing a VS_STILL_IMAGE_TRIGGER_CONTROL request with the "Abort still image transmission" option. In either case, the device shall trigger a control change interrupt via the Status Interrupt endpoint

The following table summarizes endpoint usage for the various methods of still image capture.

Table 2-4 Summary of Still Image Capture Methods

	Isochronous video data pipe	**Bulk video data pipe**
Method 1	1 Isochronous (Video)	1 Bulk (Video)
Method 2	1 Isochronous (Video/Still)	1 Bulk (Video/Still)
Method 3	1 Isochronous (Video) 1 Bulk (Still)	1 Bulk (Video) 1 Bulk (Still)

2.4.2.5 Optical Zoom vs Digital Zoom

Users expect to use a single control to traverse the entire range of optical and digital zoom. Further, users expect that digital zoom will not be applied until full optical zoom has been realized. It is recommended that the device enforce this behavior. A solution with details on how this can be achieved is presented in Appendix D.

2.4.3 VideoStreaming Interface

VideoStreaming interfaces are used to interchange digital data streams between the Host and the video function. They are optional. A video function can have zero or more VideoStreaming interfaces associated with it, each possibly carrying data of a different nature and format. Each VideoStreaming interface can have one isochronous or bulk data endpoint for video, and an optional dedicated bulk endpoint for still images related to the video (only for method 3 of still image transfer. See section 2.4.2.4 "Still Image Capture"). This construction guarantees a one-to-one relationship between the VideoStreaming interface and the single data stream related to the endpoint.

A VideoStreaming interface with isochronous endpoints must have alternate settings that can be used to change certain characteristics of the interface and underlying endpoint(s). A typical use of alternate settings is to provide a way to change the bandwidth requirements an active isochronous pipe imposes on the USB. All devices that transfer isochronous video data must incorporate a zero-bandwidth alternate setting for each VideoStreaming interface that has an isochronous video endpoint, and it must be the default alternate setting (alternate setting zero). A device offers to the Host software the option to temporarily relinquish USB bandwidth by switching to this alternate setting. The zero-bandwidth alternate setting does not contain a VideoStreaming isochronous data endpoint descriptor.

A VideoStreaming interface containing a bulk endpoint for streaming shall support only alternate setting zero. Additional alternate settings containing bulk endpoints are not permitted in a device that is compliant with the Video Class specification. This restriction does not prohibit the mix of bulk and isochronous endpoints when the bulk endpoints are used solely for Still Image Transfer Method 3. In that case, each alternate setting will include the descriptors for both an isochronous endpoint and a bulk endpoint.

If a VideoStreaming interface with an isochronous endpoint supports a set of video parameter combinations (including video format, frame size and frame rate) that utilize significantly varying amounts of bandwidth across all combinations, it is recommended that the VideoStreaming interface support a range (greater than two) of alternate interface settings with varying maximum packet sizes. By doing so, the host would be able to select an appropriate alternate setting for the given video parameter combination that makes most efficient use of bus bandwidth.

For device implementers, the process of determining the number of alternate settings to be provided and the maximum packet size for the video data endpoint in each alternate setting is implementation dependent, and would depend on the bandwidth usage across the range of video parameter combinations that the VideoStreaming interface is capable of supporting.

2.4.3.1 Stream Bandwidth Selection

The bandwidth required by a video stream can be satisfied by a USB bandwidth that is equal to or greater than the function stream bandwidth. This can be illustrated as follows.

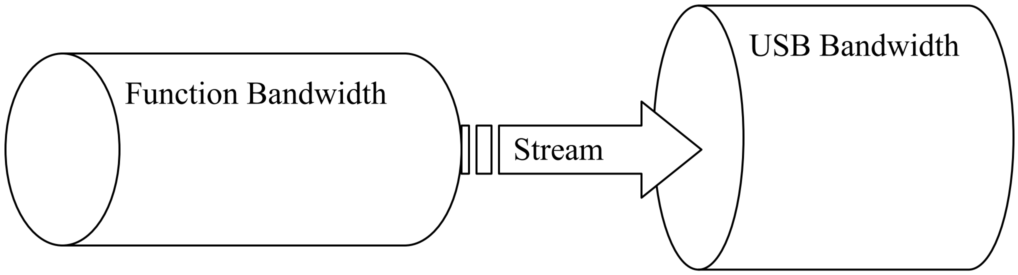


Figure 2-9 Stream Bandwidth Selection

The optimal allocation of the USB bandwidth to match the function's bandwidth requirement is achieved via negotiation between the host and the device.

See section 4.3.1.1, "Video Probe and Commit Control" for a complete description of the negotiation process.

The negotiation process allows the host to provide preferred stream parameters to the device, while the device selects the best combination of streaming parameters and reports the maximum bandwidth usage for those settings. The host will use the bandwidth information to identify the optimal alternate interface. The device is responsible for choosing the live streaming parameters once the bandwidth is allocated. These parameters may be different than originally agreed upon during the negotiation process. However, during the negotiation process, the host provided hints to the device indicating the preferred way to choose the live stream parameters.

Once bandwidth has been allocated and streaming started, further parameter negotiation between the host and the device can be performed without disturbing the current stream. Streaming parameters are set as a group so that the function will have all information available while it attempts to determine a working set.

Still image Method 2 uses a similar mechanism (see section 2.4.2.4, "Still Image Capture").

2.4.3.2 Video and Still Image Samples

A video (or still image) sample refers to an encoded block of video data that the format-specific decoder is able to accept and interpret in a single transmission. A single video sample may or may not correspond to a single decoded video frame, depending on the video format in use. For example, a YUV video stream (which has no inter-frame compression) would have a one to one correspondence between a video sample and video frame. However, a MPEG-2 TS data stream will require many video samples (or TS packets) to form a decoded video frame.

A single video sample may require multiple class-defined Payload Transfers. Conversely, there may be one or more video samples within a single Payload Transfer. In the latter case, there must be an integral number of fixed size samples within each Payload Transfer.

The VideoStreaming endpoint(s) encapsulate data with the class-defined Payload Header. This encapsulation is identical for Payload Transfers on both isochronous and bulk endpoint types, and applies to both the streaming and still image endpoints.

The following block diagram details the protocol layering and abstraction used in Payload Transfers.

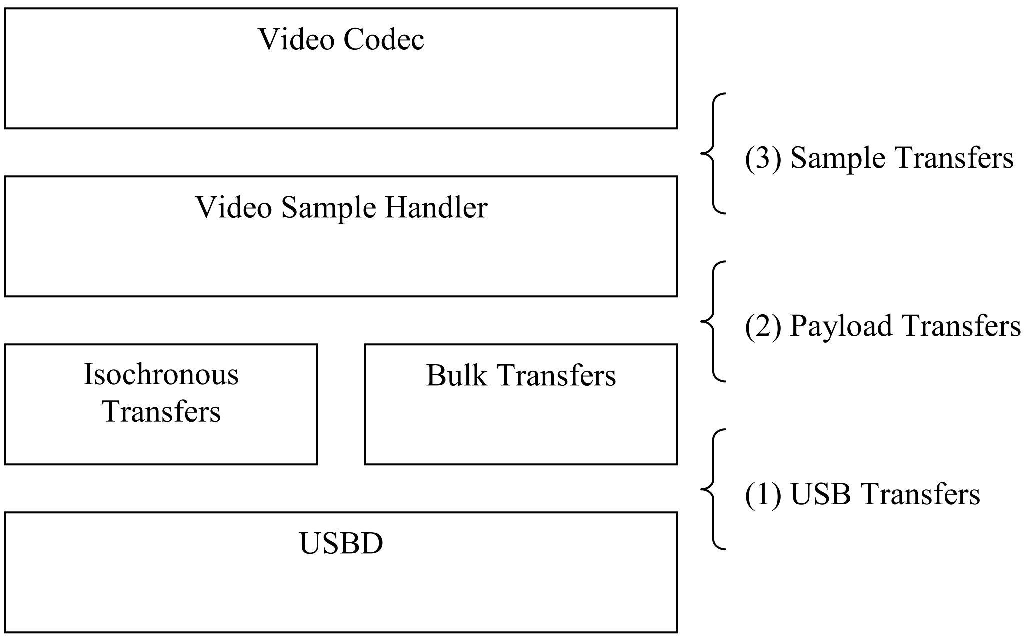


Figure 2-10 Protocol Layering and Abstraction

1. I/O Request Packet (IRP) requests from the client to the USB system software result in USB transfers.
2. In response to IRP completion, the host software forwards the data in the form of payload transfers. The bulk and isochronous handlers hide the transfer type differences from the upper layers of the protocol stack.
3. The video sample handler accumulates the individual payload transfers to form a sample transfer.

A Payload Transfer is composed of the class-defined payload header (see section 2.4.3.3 "Video and Still Image Payload Headers") followed by the format-specific payload data.

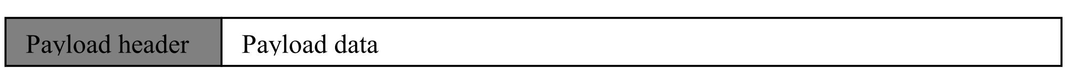


Figure 2-11 A Payload Transfer

2.4.3.2.1 Sample Bulk Transfers

The following examples show the relationship between Video Samples, Payload Transfers and the token and data packets when exchanging bulk transfers with a device. Handshake packets are not shown for the sake of clarity.

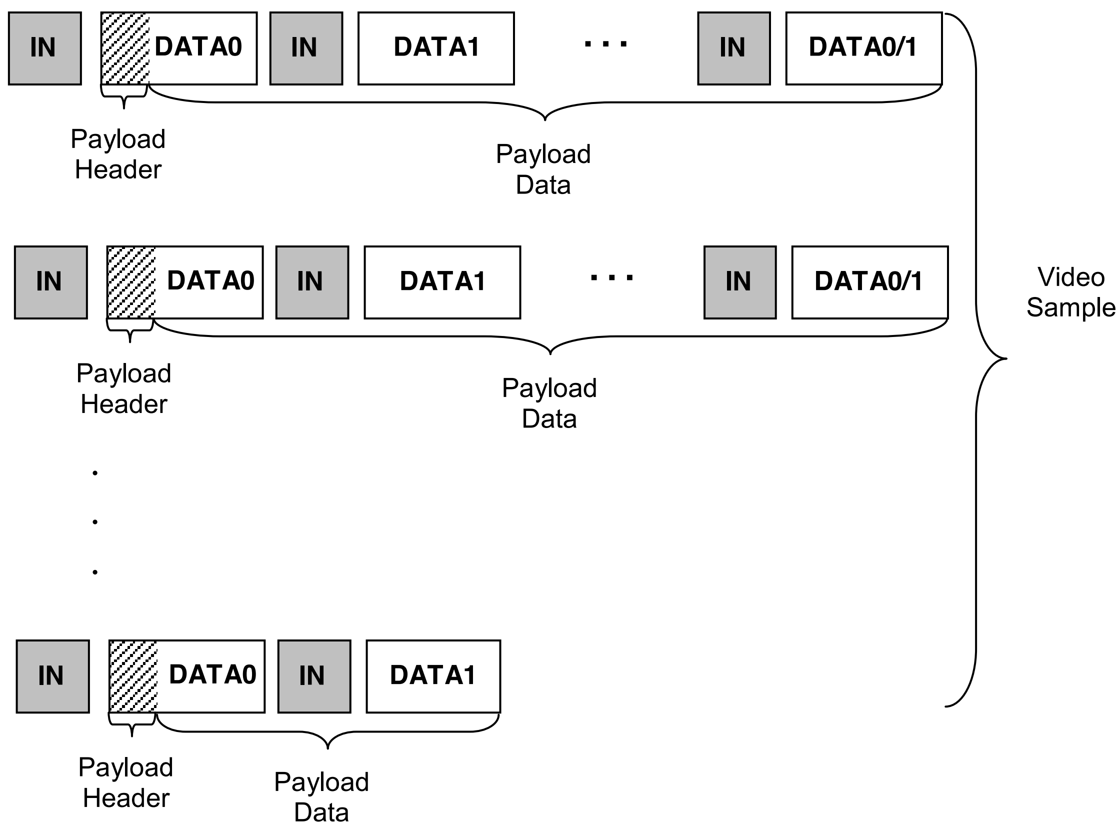


Figure 2-12 Sample Bulk Read (Multiple Transfers per Sample)

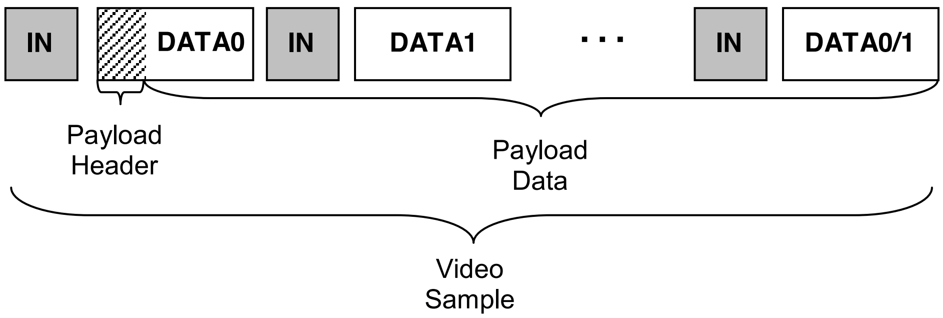


Figure 2-13 Sample Bulk Read (Single Transfer per Sample)

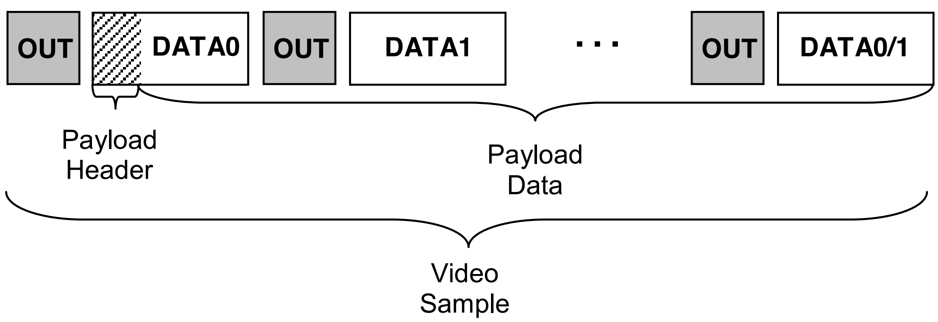


Figure 2-14 Sample Bulk Write (Single Transfer per Sample)

2.4.3.2.2 Sample Isochronous Transfers

The following examples show the relationship between Video Samples, Payload Transfers and the token and data packets when exchanging isochronous transfers with a device. The actual video sample size and bandwidth usage (i.e. number of data transactions and amount of data in the last transaction of each payload) will vary according to the requirements of the device and payload.

Figure 2-15 gives an example of a High Speed/High Bandwidth transfer over an IN endpoint.

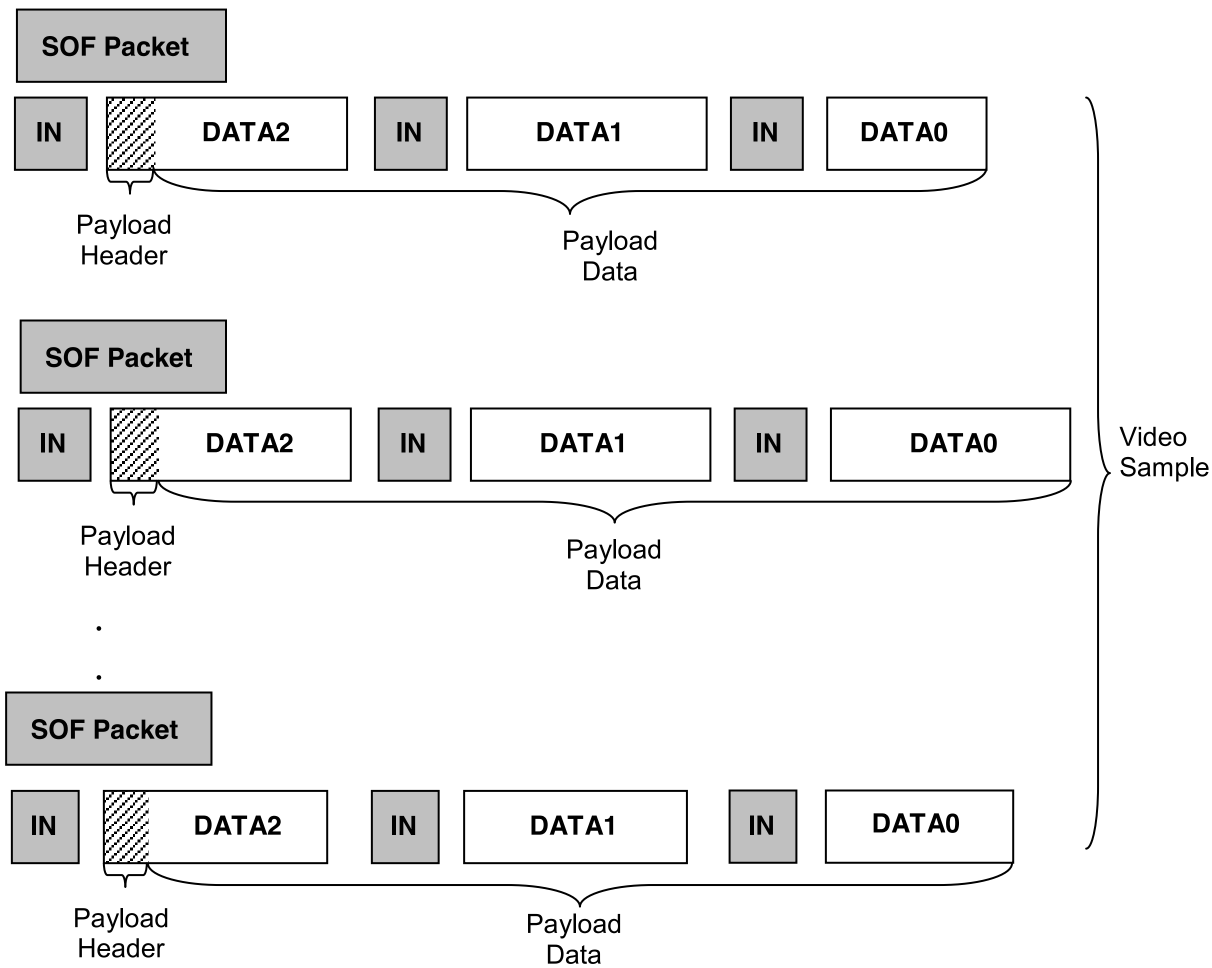


Figure 2-15 Sample Isochronous Transfer, IN endpoint

Figure 2-16 gives an example of a High Speed/High Bandwidth transfer over an OUT endpoint.

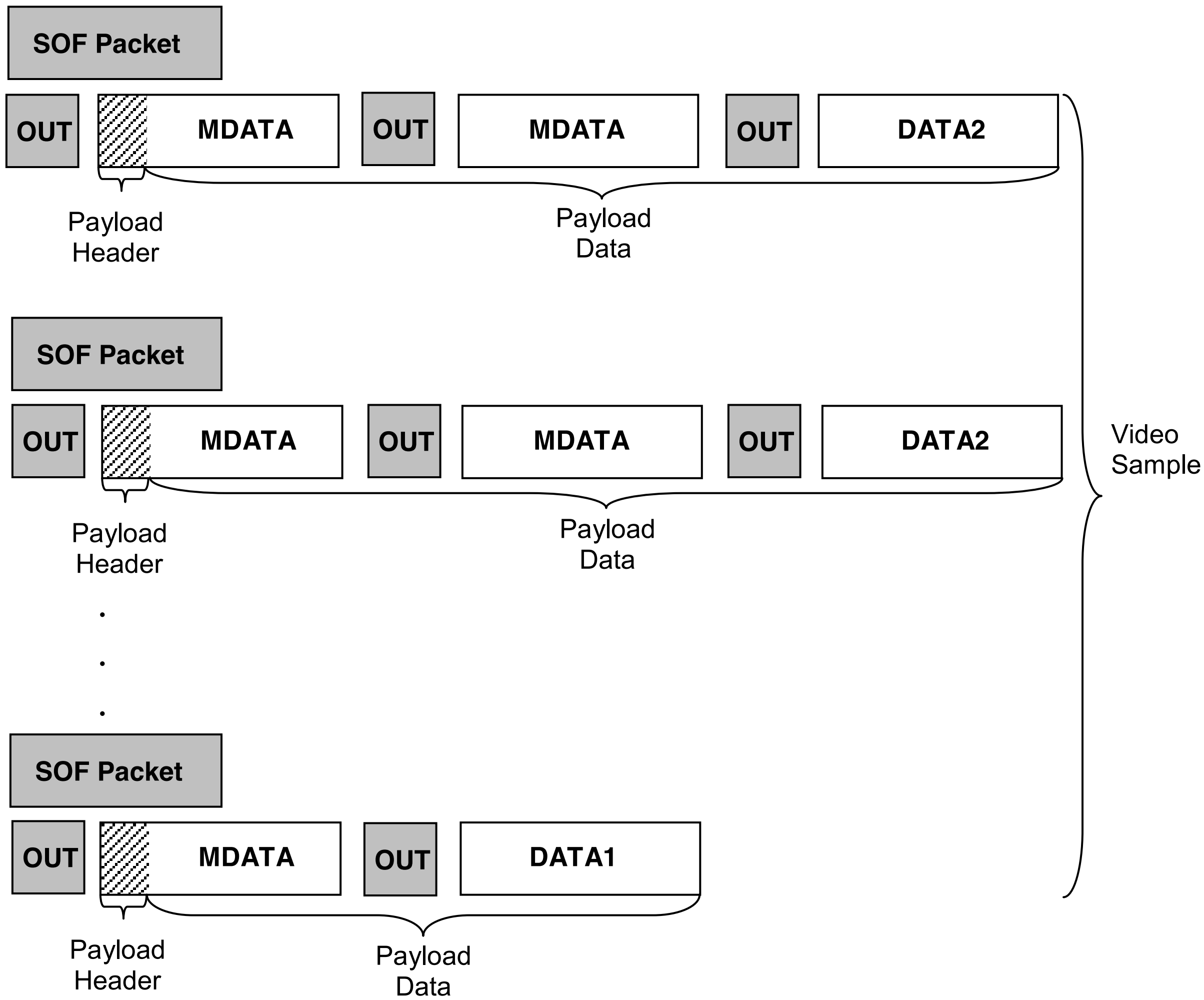


Figure 2-16 Sample Isochronous Transfer, OUT endpoint

Figure 2-17 gives an example of a Full or High Speed transfer over an IN endpoint.

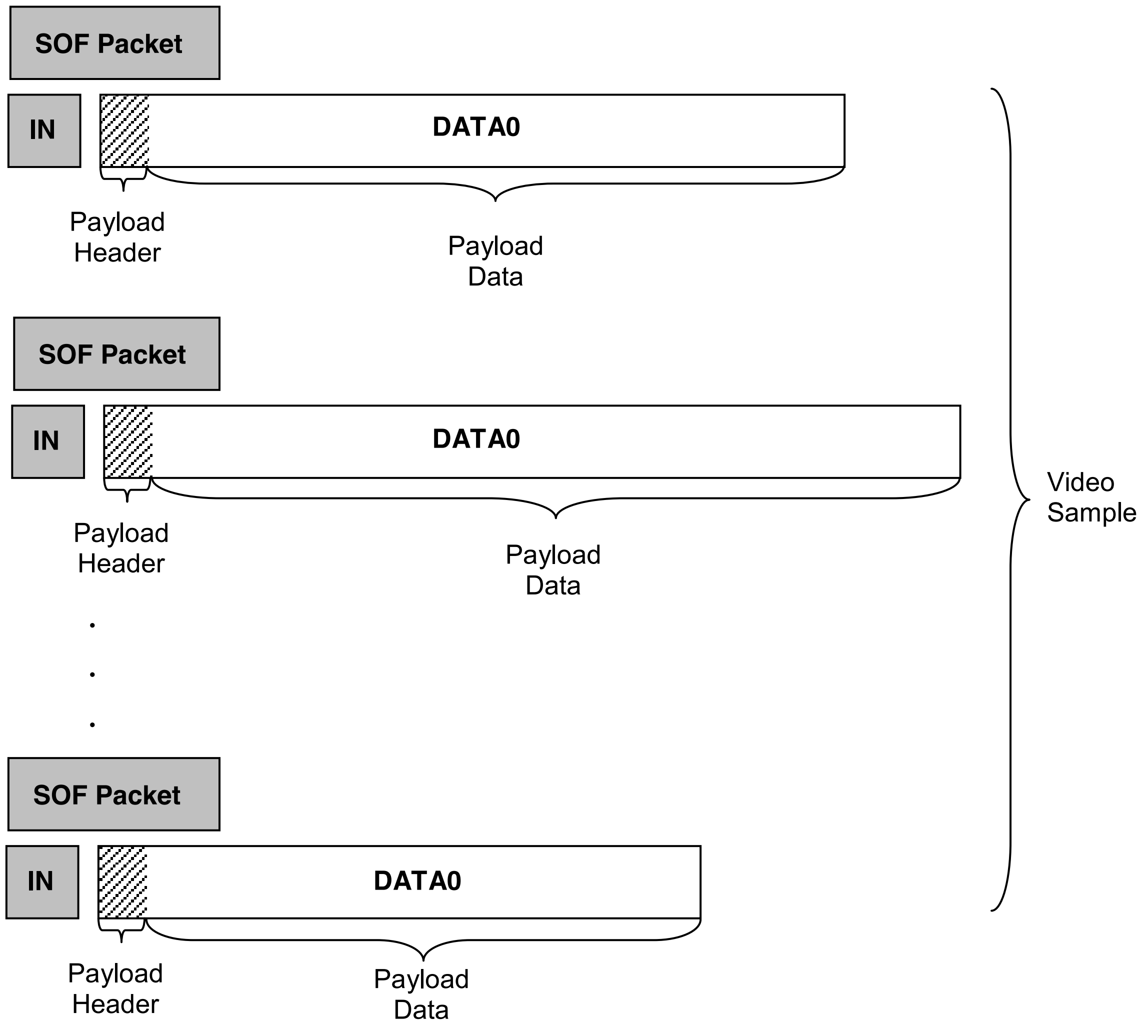


Figure 2-17 Sample Isochronous Transfer, IN endpoint

Figure 2-18 gives an example of a Full or High Speed transfer over an OUT endpoint.

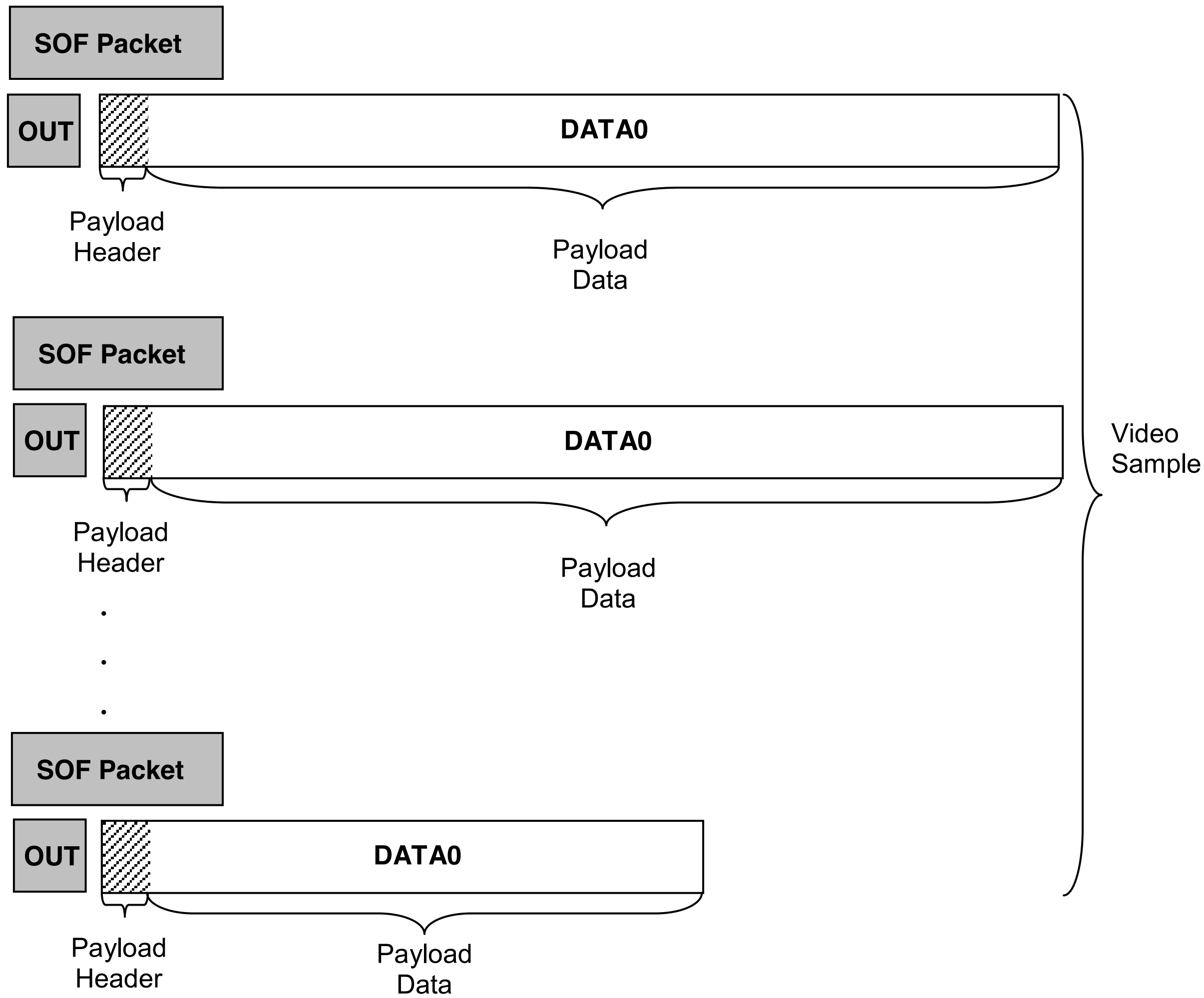


Figure 2-18 Sample Isochronous Transfer, OUT endpoint

2.4.3.3 Video and Still Image Payload Headers

Every Payload Transfer containing video or still-image sample data must start with a Payload Header.

The format of the payload header is defined as follows.

Table 2-5 Format of the Payload Header

Offset	Field	Size	Value	Description
0	**bHeaderLength**	1	Number	Length of the payload header in bytes including this field.
1	**bmHeaderInfo**	1	Bitmap	Provides information on the sample data following the header, as well as the availability of optional header fields in this header. D0: Frame ID – For frame-based formats, this bit toggles between 0 and 1 every time a new video frame begins. For stream-based formats, this bit toggles between 0 and 1 at the start of each new codec-specific segment. This behavior is required for frame-based payload formats (e.g., DV) and is optional for stream-based payload formats (e.g., MPEG-2 TS). For stream-based formats, support for this bit must be indicated via the **bmFramingInfo** field of the Video Probe and Commit controls (see section 4.3.1.1, “Video Probe and Commit Controls”). D1: End of Frame – This bit is set if the following payload data marks the end of the current video or still image frame (for frame-based formats), or to indicate the end of a codec-specific segment (for stream-based formats). This behavior is optional for all payload formats. For stream-based formats, support for this bit must be indicated via the **bmFramingInfo** field of the Video Probe and Commit Controls (see section 4.3.1.1, “Video Probe and Commit Controls”). D2: Presentation Time – This bit is set if the **dwPresentationTime** field is being sent as part of the header.

				D3: Source Clock Reference – This bit is set if the **dwSourceClock** field is being sent as part of the header. D4: Payload specific bit. See individual payload specifications for use. D5: Still Image – This bit is set if the following data is part of a still image frame, and is only used for methods 2 and 3 of still image capture. For temporally encoded formats, this bit indicates the following data is part of an intra-coded frame. D6: Error – This bit is set if there was an error in the video or still image transmission for this payload. The Stream Error Code control would reflect the cause of the error. D7: End of header – This bit is set if this is the last header group in the packet, where the header group refers to this field and any optional fields identified by the bits in this field (Defined for future extension).

The following fields may or may not be included in the header, depending on the bits that were specified in the **bmHeaderInfo** field above.

These fields are in the order in which they are specified in the bitmap header field above, in the order of least significant bit first. Because the header itself might be extended in the future, the offset of **dwPresentationTime** is also variable. The device will indicate if it supports these fields in the Payload Format Descriptor within the class-specific VideoStreaming descriptor. See section 3.9.2.3 "Payload Format Descriptors".

Table 2-6 Extended Fields of the Payload Header

Offset	Field	Size	Value	Description
Variable	**dwPresentationTime**	4	Number	Presentation Time Stamp (PTS). The source clock time in native device clock units when the raw frame capture begins. This field may be repeated for multiple payload transfers comprising a single video frame, with the restriction that the value shall remain the same

				throughout that video frame. The PTS is in the same units as specified in the **dwClockFrequency** field of the Video Probe Control response.
Variable	**scrSourceClock**	6	Number	A two-part Source Clock Reference (SCR) value D31..D0: Source Time Clock in native device clock units D42..D32: 1KHz SOF token counter D47..D43: Reserved, set to zero. The least-significant 32 bits (D31..D0) contain clock values sampled from the System Time Clock (STC) at the source. The clock resolution shall be according to the **dwClockFrequency** field of the Probe and Commit response of the device as defined in Table 4-75 of this specification. This value shall comply with the associated stream payload specification. The times at which the STC is sampled must be correlated with the USB Bus Clock. To that end, the next most-significant 11 bits of the SCR (D42..D32) contain a 1 KHz SOF counter, representing the frame number at the time the STC was sampled. • STC must be captured when the first video data of a video frame is put on the USB bus. • SCR must remain constant for all payload transfers within a single video frame. The most-significant 5 bits (D47..D43) are reserved, and must be set to zero. The maximum interval between Payload Headers containing SCR values is 100ms, or the video frame interval, whichever is greater. Shorter intervals are permitted.

The periodic transmission of the **dwPresentationTime** and **dwSourceClock** fields is mandatory if all of the following conditions are true.

- The device has multiple video and/or audio source functions and is sending audio and video streams to the host.
- The video and/or audio streams are interrelated and therefore need to be kept synchronized.
- The stream format in use does not already contain timestamp and clock reference information (MPEG-2 TS is an example of a format that contains this information).
- The sample is part of a video frame (and not a still image frame).

For temporally encoded payloads, the **dwPresentationTime** and **dwSourceClock** fields may be required for all video frames. See the appropriate payload specification for details.

These time information fields allow the host software to reconstruct the source clock to support high-quality synchronization between separate data pipes (audio, video, etc.) and rate matching between the data source and sink, as discussed in the following section.

2.4.3.4 Stream Synchronization and Rate Matching

To properly synchronize multiple audio and video streams from a media source, the media source must provide (to the media sink) its local stream latency, periodic clock reference information, and a way for the media sink to determine the proper presentation time for samples from each stream (relative to the other streams).

2.4.3.4.1 Latency

The media source is required to report its internal latency (delay from data acquisition to data delivery on the bus). This latency reflects the lag introduced by any buffering, compression, decompression, or processing done by the stream source. Without latency information for each stream, a media sink (or rendering device) cannot properly correlate the presentation times of each stream.

In the case of a video source, this means that the source must guarantee that the portion of a sample fully acquired as of SOF_n (Start Of Frame n) will have been completely sent to the bus as of $SOF_{n+}\delta$. Latency δ is the source's internal delay expressed in number of USB frames (milliseconds). For high-speed endpoints, the resolution increases to 125 microseconds, but the delay will continue to be expressed in number of USB frames. Every VideoStreaming interface must report this latency value. See the description of the **wDelay** parameter in section 4.3.1.1, "Video Probe and Commit Controls". By following these rules, phase jitter is limited to ±1 millisecond. It is up to the video sink to synchronize streams by scheduling the rendering of samples at the correct moment, taking into account the internal delays of all media streams being rendered.

2.4.3.4.2 Clock Reference

Clock reference information is used by a media sink to perform clock rate matching. Rate matching refers to the synchronization of the media sink's rendering clock with the media source's sampling clock. Without clock rate matching, a stream will encounter buffer overrun or underrun errors. This has not been a problem with audio streams due to the relative ease of performing audio sample rate conversion. However, sample rate conversion is significantly more difficult with video, so a method for rate matching is required.

To understand the problem of clocks running at slightly different rates, consider the following example. For simplicity, assume that video buffers can be filled instantaneously, and that there is one buffer available to be filled at any given time within the video frame interval. Also assume that the two crystals governing the source and rendering clocks operate with 100ppm (parts per million) accuracy. The accuracy value is a ratio that can be applied such that for every frame, the clock will drift by a fraction of the frame that is equal to the ratio. In other words, two clocks with accuracy of 100ppm could have a worst case drift relative to each other of 1/5,000th of a frame (two clocks at opposite extremes of their valid operating range for a cumulative error ratio of 2 * 100/1,000,000). Therefore, a frame glitch will occur once every 5,000 frames. At a frame rate of 30 fps, this would equate to a glitch every 166.67 seconds. At a frame rate of 60 fps, it's worse, with one glitch every 83.3 seconds.

Frame glitches can be postponed, but not avoided, by adding additional buffers to hold video frames before they are rendered. If the source clock is running slower than the rendering clock, the buffer underrun could only be postponed by letting the extra buffers fill to a certain threshold before rendering, resulting in unacceptable latency. Once the first glitch occurs, the extra buffers are effectively useless, since the behavior will degrade to the single-buffer case from that point onward.

This specification assumes that in all cases, the media sink has no control over the media source clock, and that the source and sink do not "slave" to a common clock (the bus clock lacking sufficient resolution). Also, due to cost constraints, additional isochronous endpoints to communicate clock rate information will not be used. Therefore, this specification requires that a video stream include clock reference information that can be used to adjust the rendering clock rate. The clock reference information may be encapsulated in a transport stream, or it may be provided via an optional field in each payload header. This field becomes required in the latter case.

2.4.3.4.3 Presentation Time

For fixed rate streams, the presentation time can be derived from the data stream. For a fixed-rate audio stream (e.g., PCM), the media sink can derive the presentation time from the stream offset (typically the count of bytes since start of capture). For variable rate streams, each sample must be accompanied by a presentation timestamp. The media sink is responsible for converting the timestamp to native units and adjusting the timestamp to account for the local clock offset when a stream starts, as well as accounting for source stream latency. Even though video streams might arrive at the media sink at a fixed frame rate, if they are subject to variable rate compression and encoding, they are not considered fixed-rate streams and will require timestamps on the samples.

2.4.3.5 Dynamic Frame Interval Support

In order to adjust to different environmental conditions, such as varying lighting conditions, it may be necessary for a video device (such as a camera) to dynamically change the frame interval and sensor exposure time to maintain acceptable image quality while streaming.

After bus bandwidth for the video data pipe of the corresponding VideoStreaming interface has been allocated and streaming has commenced, the data source may dynamically vary the frame interval (and the corresponding frame rate), as long as the new frame interval does not require greater bus bandwidth than what was originally allocated. The data sink would determine the new frame interval based on the Presentation Time Stamp (PTS) information included in the video payload headers.

2.4.3.6 Device Initiated Dynamic Format Change Support

Certain devices, such as those that contain a tape media transport, are capable of dynamically changing the video format being streamed to the host while streaming is occurring. Since the new video format may have different bus bandwidth requirements from the old format, the host must be notified of the format change and be allowed to perform the reconfiguration and bus bandwidth reallocation necessary to support the new video format.

The device indicates its support for dynamic format change events through the **bmInfo** field of the VideoStreaming Input Header. See section 3.9.2.1 "Input Header Descriptor".

When a dynamic format change event occurs, the following steps take place:

- Device detects dynamic format change (while streaming is occurring).
- Device begins sending empty data payloads to the host with the Error bit set in the video stream payload header.
- Device sets the Stream Error Code Control to "Format Change" (see section 4.3.1.7 "Stream Error Code Control").
- The host queries the new stream state through a VS_PROBE_CONTROL request with the GET_CUR attribute (see 4.3.1.1, “Video Probe and Commit Controls”).
- If the new format is acceptable by the host, it issues a VS_COMMIT_CONTROL request with the SET_CUR attribute and, if necessary, reallocates the USB bandwidth through an alternate interface selection standard request. If the new format is not acceptable, the host will negotiate a new format with the stream PROBE/COMMIT controls.

2.4.3.7 Data Format Classes

For the purposes of host processing of incoming and outgoing data packets, the various video formats supported by the USB Video Class (UVC) can be divided into two broad categories:

- **Frame-based video formats** – These video formats require the frame/sample boundary information to be transmitted out-of-band. Examples of such formats are uncompressed video (formatted in various YUV variants), MJPEG, and DV. For these formats, the FID (and optionally EOF) bits in the UVC payload headers must be supported.
- **Stream-based video formats** – These video formats have the frame/sample boundary information transmitted in-band. Examples of such formats are MPEG-2 TS, MPEG-2 PS and MPEG-1 system streams. For these formats, the FID and EOF bits are optional. If used, the bits allow the sender to identify codec-specific segment boundaries within the stream. The receiver would typically use this information to provide data to a decoder with lower latency than would be possible if buffer fullness alone was used to trigger buffer completion (see section 4.3.1.1, "Video Probe and Commit Controls").
- **Temporally Encoded video formats** – While these video formats have the frame/sample boundary information transmitted in-band, they are often managed as frames or sub-frames by the host. The EOF and EOS bits are required to indicate these boundaries to the host so it may generate time stamps and trigger buffer completion on these boundaries. Examples of temporally encoded video formats are H.264 and VP8.

The following is determined by the format class under which the video format is classified:

- The default Incoming/Outgoing data processing algorithm
- Bit fields supported by default in UVC payload header (BFH[0])

The following is determined by the specific video payload format:

- Format descriptor type
- Frame descriptor type, if needed
- Support for time information fields in UVC payload header

2.4.4 Control Transfer and Request Processing

The Video Class specification's control transfer (or Request) mechanism builds upon sections 5.5, "Control Transfers"; 8.5.3 ,"Control Transfers; 9.2.6, "Request Processing"; and 9.3, "USB Device Requests" of the *Universal Serial Bus Specification, Revision 2.0* (the USB 2.0 spec). Those sections describe the timing and error handling of control transfers, but do not prescribe a method for control transfer completion using interrupt pipes. The following paragraphs describe Control Transfer operations in the context of the Video Class, including the use of the Status Interrupt pipe to provide notification of state changes within the device.

Control transfers minimally have two transaction stages: Setup and Status. A control transfer may optionally contain a Data stage between the Setup and Status stages. The Setup stage contains all information necessary to address a particular entity, specify the desired operation, and prepare for an optional Data stage. A Data stage can be host-to-device (OUT transactions), or device-to-host (IN transactions), depending on the direction and operation specified in the Setup stage via the **bmRequestType** and **bRequest** fields.

In the context of the Video Class specification, SET_CUR requests will always involve a Data stage from host to device, and GET_* requests will always involve a Data stage from device to host. Although none are defined currently, an exception to this rule would be a SET_CUR request where the **bRequest** field contains all information necessary to place the device into a known state. However, “toggle” requests without a Data stage are explicitly disallowed.

The device shall use *protocol STALL* (not function stall) during the Data or Status stages if the device is unable to complete the Control transfer (see section 8.5.3.4 of the *USB Specification Revision 2.0*). Reasons for protocol STALL include unsupported operations, invalid target entity, invalid control selector, unexpected Data length, or invalid Data content. The device shall update the value of Request Error Code Control, and the host may use that control to determine the reason for the protocol STALL (see section 4.2.1.2 "Request Error Code Control"). The device must not NAK or STALL the SETUP transaction.

Typically, the host will serialize Control Transfers, meaning that the next Setup stage will not begin until the previous Status stage has completed. However, in situations where the bus has experienced errors, a Setup transaction may be sent before the completion of a previous control transfer. The device must abandon the previous control transfer.

Due to this command serialization, it is important that the duration of control transfers (from Setup stage through Status stage) be kept as short as possible. For this reason, as well as the desire to avoid polling for device status, this specification defines an interrupt status mechanism to convey status changes independently of the control transfers that caused the state change. This mechanism is described in section 2.4.2.2, "Status Interrupt Endpoint". Any control that requires more than 10ms to respond to a SET_CUR request (asynchronous control), or that can change independently of any external SET_CUR request (Autoupdate control), must send a Control Change status interrupt. These characteristics will be reflected in the GET_INFO response for that control (see 4.1.2, “Get Request”).

If a SET_CUR request is issued to an Asynchronous Control with unsupported operations, invalid target entity, unexpected data Length or invalid data content, the device shall use *protocol STALL* since the device is unable to complete the Control Transfer (see section 8.5.3.4 of the *USB Specification Revision 2.0*). The device shall update the value of the Request Error Code Control (see section 4.2.1.2 "Request Error Code Control").

In the case of a SET_CUR request with valid parameters to an Asynchronous Control, the Control Transfer operation shall enter the Status stage immediately after receiving the data transferred during the Data stage. Once the Status stage has successfully completed, the device shall eventually send a Control Change Interrupt that will reflect the outcome of the request:

- If the request succeeded, the Control Change Interrupt will advertise the new value (see section 2.4.2.2 “Status Interrupt Endpoint”).
- If the request could not be executed, the device shall send a Control Change Interrupt using the Control Failure Change mechanism to describe the reason for the failure (see

Table 2-1 in section 2.4.2.2 “Status Interrupt Endpoint” and Figure 2-23 in section 2.4.4 “Control Transfer and Request Processing”).

The amount of time between the end of a successful Status stage and the Control Change interrupt is implementation specific. For instance, a tape transport might take 3-5 seconds to completely change state, so the Control Change interrupt would be sent within 3-5 seconds.

The following flow diagrams show the Setup, Data and Status stages of SET_CUR Control Transfers for controls supporting one of the two legal bit combinations with the D1 (SET) bit enabled. These are described because they show the relationship between a SET_CUR request and the resulting state change.

SET/GET Supported

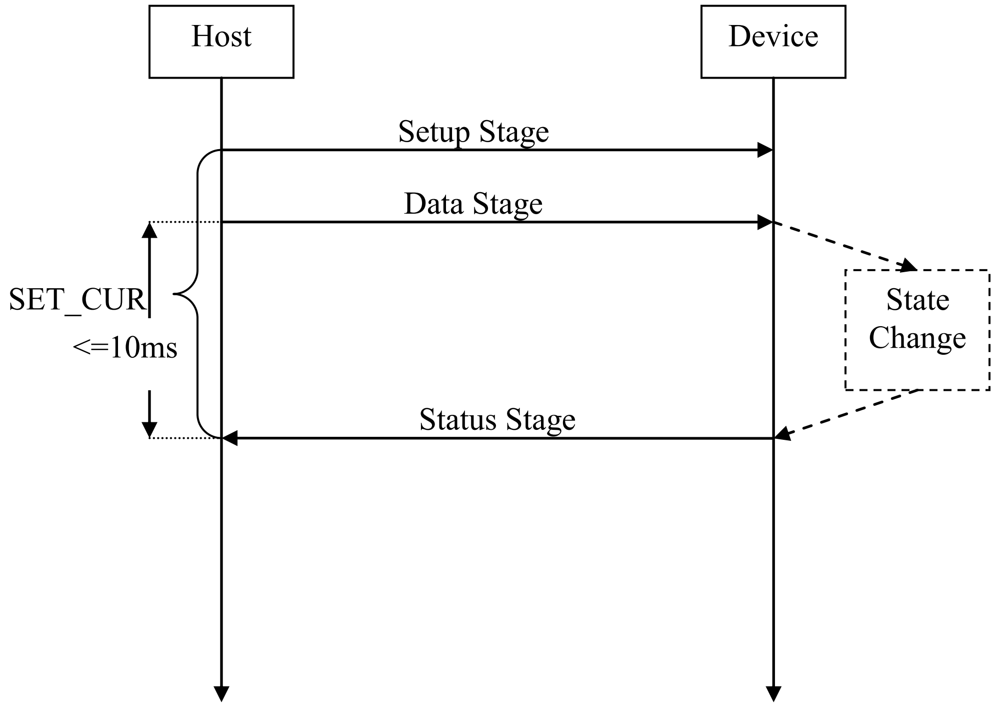


Figure 2-19 Control Transfer Example (Case 1)

SET/GET/Interrupt Supported

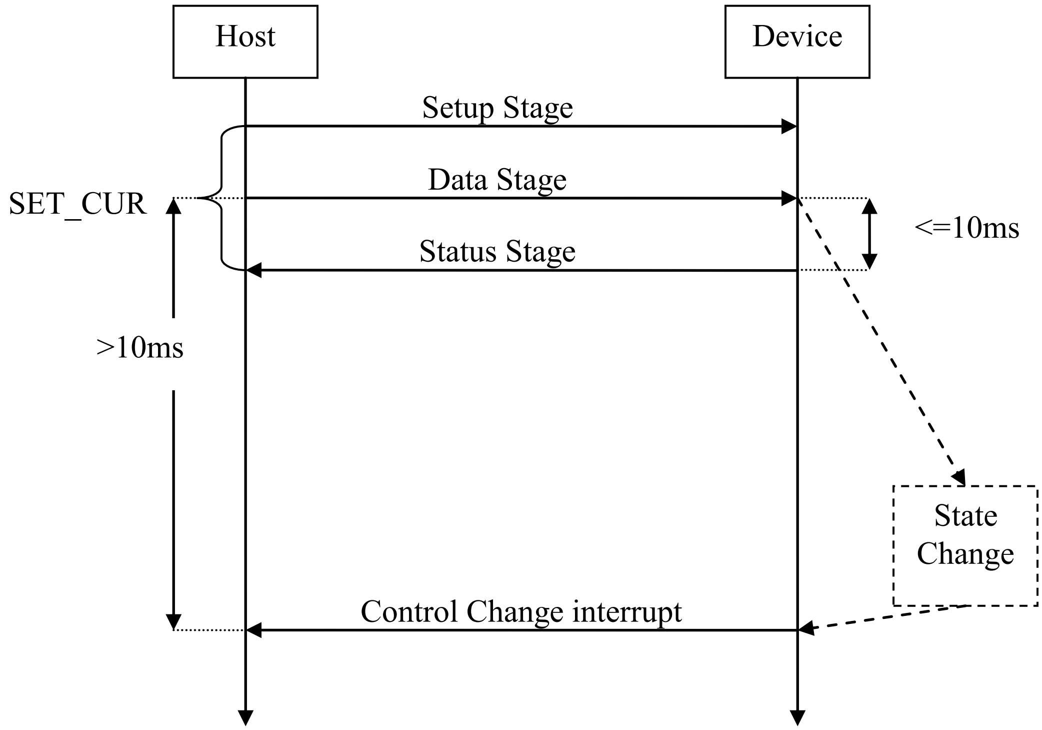


Figure 2-20 Control Transfer Example (Case 2)

SET/GET/Interrupt Supported (with *error scenarios*)

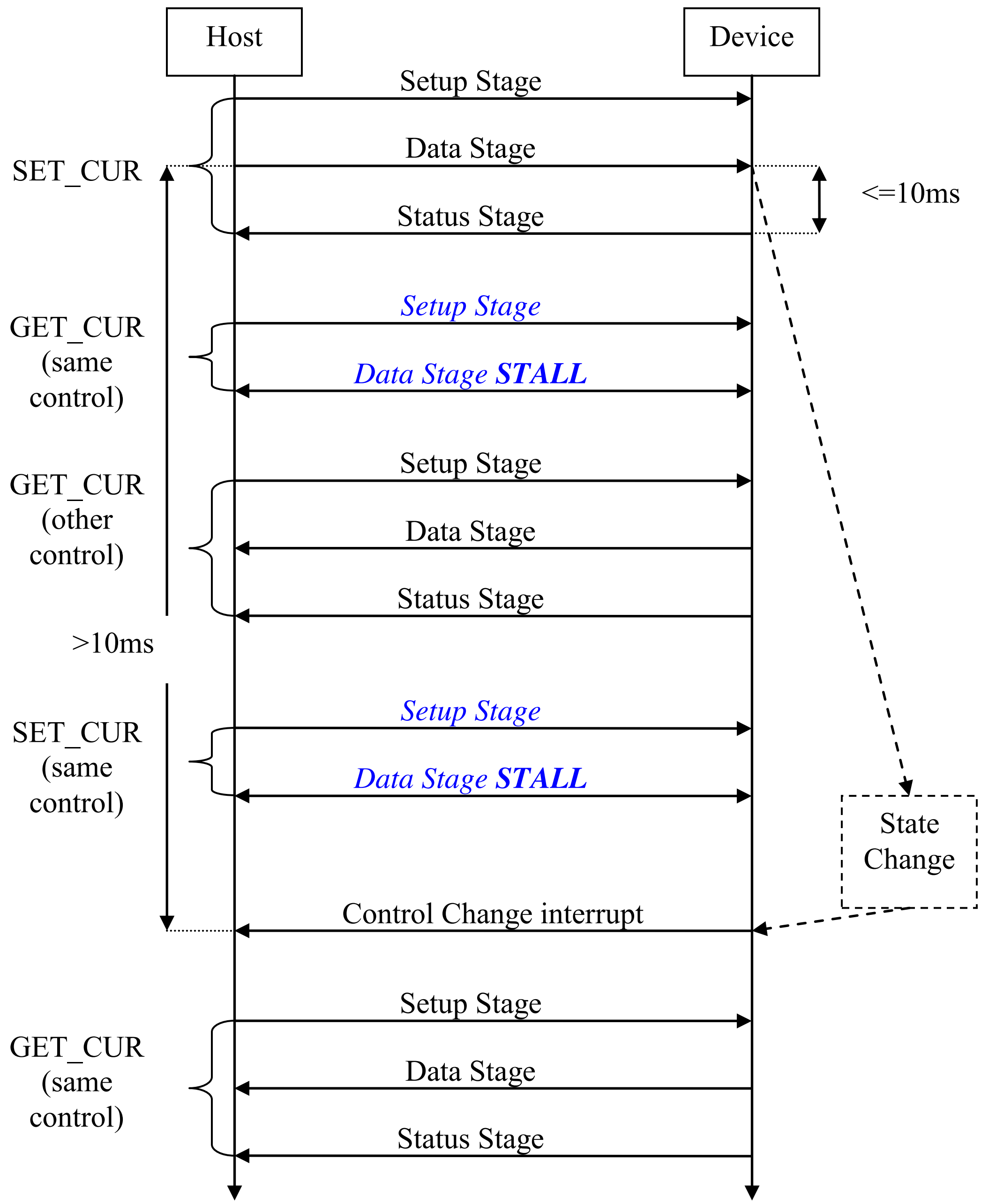


Figure 2-21 Control Transfer Example (Case 3)

SET/GET/Interrupt Supported (*Device busy before first SET request*)

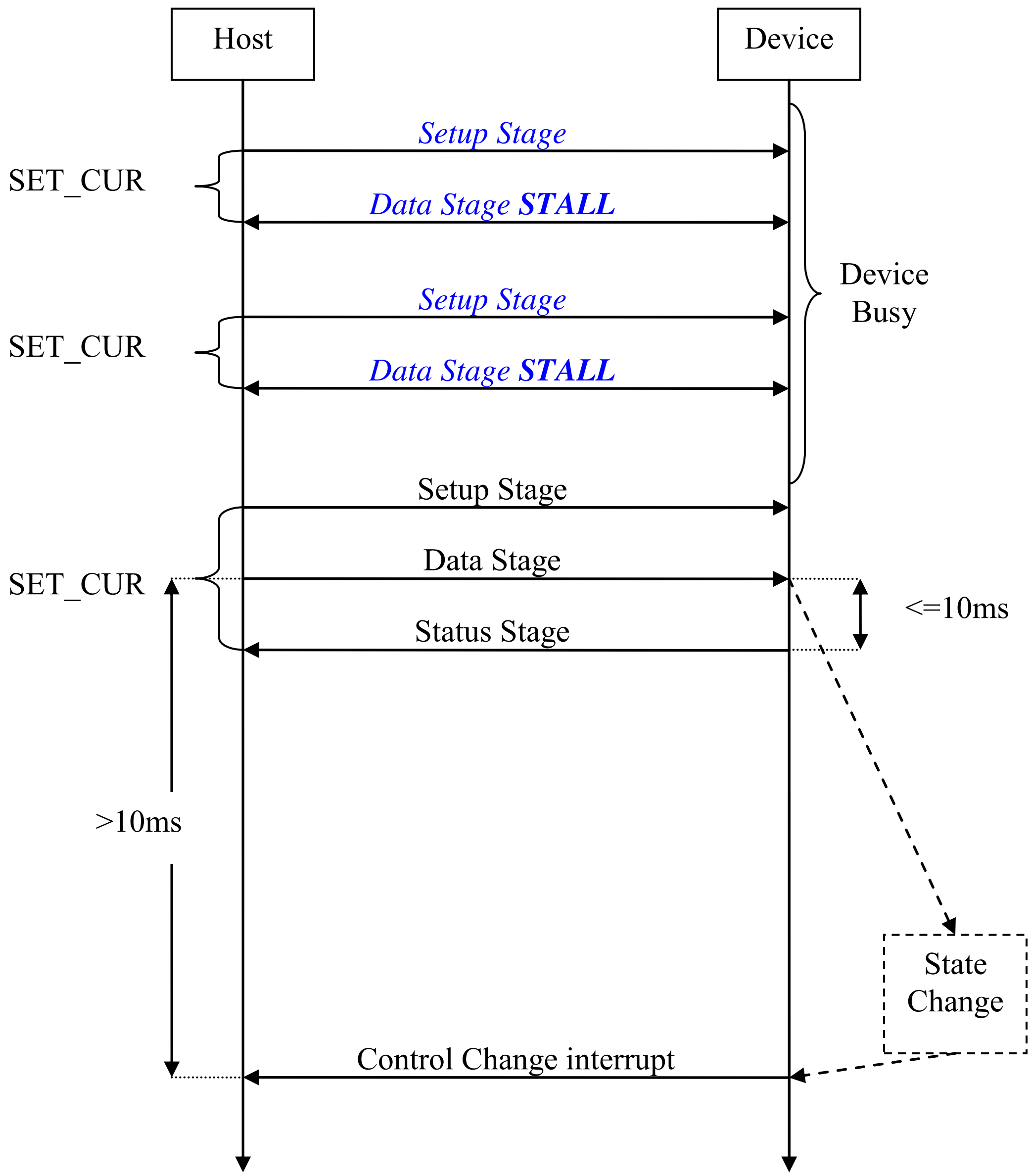


Figure 2-22 Control Transfer Example (Case 4)

SET/GET/Interrupt Supported/State Change Failure

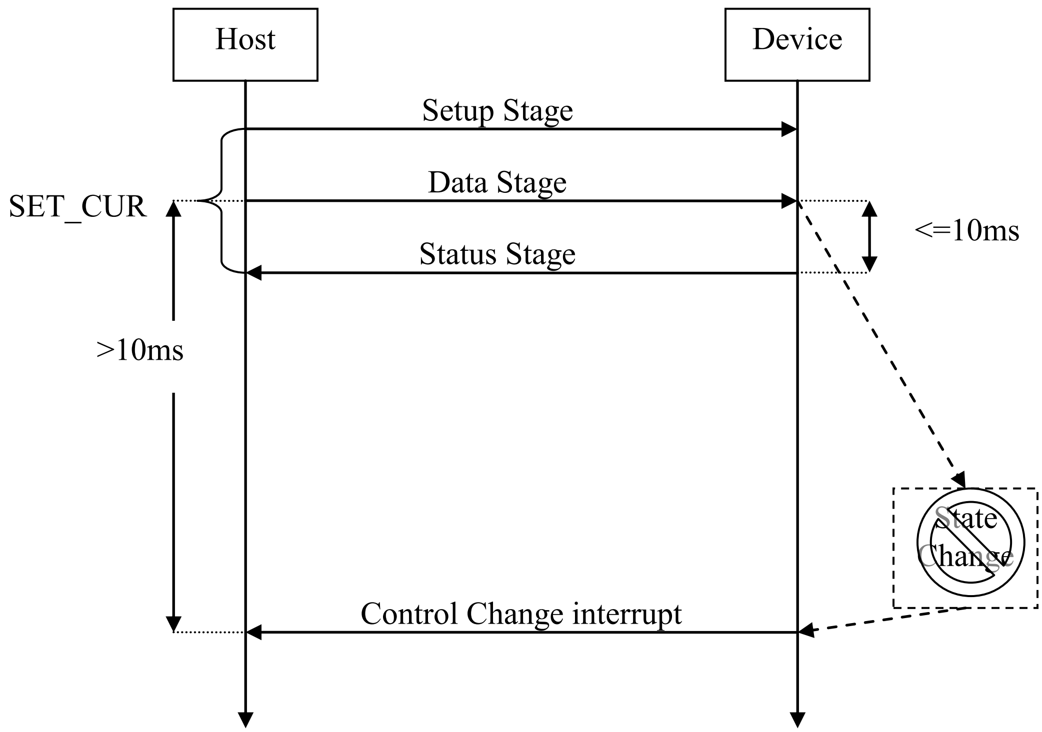


Figure 2-23 Control Transfer Example (Case 5)

3 Descriptors

Descriptors are used by USB devices to report their attributes. A descriptor is a data structure with a defined format. For information, see section 9.5 Descriptors of *USB Specification Revision 2.0*.

The following sections describe the standard and class-specific USB descriptors for the Video Interface Class.

3.1 Descriptor Layout Overview

The following diagram illustrates the descriptor layout for an entire device. The example used in this case is for a desktop video camera device with a single isochronous video pipe and a dedicated bulk still image pipe. If the video pipe supported temporally encoded video formats, the descriptor layout would include an Encoding Unit.

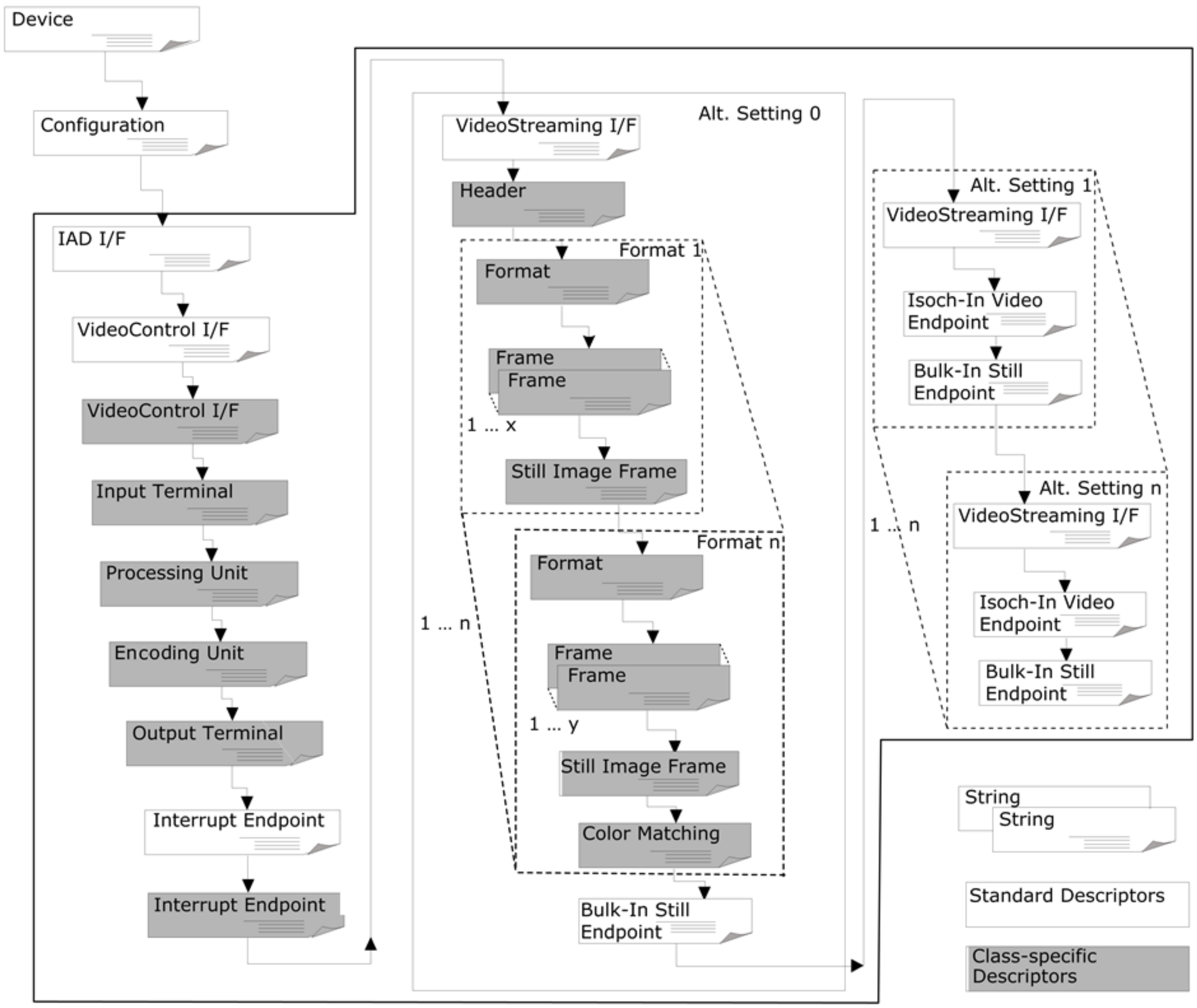


Figure 3-1 Video Camera Descriptor Layout Example

3.2 Device Descriptor

Because video functionality is always considered to reside at the interface level, this class specification does not define a specific video device descriptor.

For devices that contain a video function that only exposes a VideoControl Interface, the device descriptor must indicate that class information is to be found at the interface level. Therefore, the **bDeviceClass** field of the device descriptor must contain zero so that enumeration software looks down at the interface level to determine the Interface Class. The **bDeviceSubClass** and **bDeviceProtocol** fields must be set to zero.

Devices that expose one or more Video Interface Collections also indicate that class information is to be found at the interface level. However, since the device uses an Interface Association Descriptor in order to describe the Video Interface Collection, it must set the **bDeviceClass**, **bDeviceSubClass** and **bDeviceProtocol** fields 0xEF, 0x02 and 0x01 respectively. This set of class codes is defined as the Multi-interface Function Class codes.

All other fields of the device descriptor must comply with the definitions in section 9.6.1 "Device" of the appropriate USB specification (*USB Specification Revision 2.0* or *USB Specification Revision 3.0).* There is no class-specific device descriptor.

3.3 Device_Qualifier Descriptor (deprecated)

3.4 Configuration Descriptor

The configuration descriptor for a device containing a video function is identical to the standard Configuration descriptor defined in section 9.6.3 "Configuration" of *USB Specification Revision 2.0 or USB Specification Revision 3.0*. There is no class-specific configuration descriptor.

3.4.1 Backwards Compatibility

Separate configurations should be used when a device wishes to support more than one version of the USB Video Class specification. Configurations shall be listed by increasing version, starting from index 0.

3.5 Other_Speed_Configuration Descriptor (deprecated)

3.6 Interface Association Descriptor

A device must use an Interface Association Descriptor to describe a Video Interface Collection for each device function that requires a VideoControl Interface and one or more VideoStreaming interfaces. The standard VIC Interface Association Descriptor is identical to the standard Interface Association Descriptor defined in the *Interface Association Descriptor ECN*, except that some fields now have dedicated values.

If the VideoControl interface is part of a Video Interface Collection, the **iFunction** field in the IAD and the **iInterface** field in the Standard VC interface descriptor for this Video Interface Collection must be equal.

Table 3-1 Standard Video Interface Collection IAD

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 8
1	**bDescriptorType**	1	Constant	INTERFACE ASSOCIATION Descriptor.
2	**bFirstInterface**	1	Number	Interface number of the first VideoControl interface that is associated with this function.
3	**bInterfaceCount**	1	Number	Number of contiguous VideoStreaming interfaces that are associated with this function. The count includes the first VideoControl interface and all its associated VideoStreaming interfaces.
4	**bFunctionClass**	1	Class	CC_VIDEO. Video Interface Class code (assigned by the USB). See section A.1, "Video Interface Class Code".
5	**bFunctionSubClass**	1	SubClass	SC_VIDEO_INTERFACE_COLLECTION. Video Interface Subclass code. Assigned by this specification. See section A.2, "Video Interface Subclass Codes".
6	**bFunctionProtocol**	1	Protocol	Not used. Must be set to PC_PROTOCOL_UNDEFINED.
7	**iFunction**	1	Index	Index of a string descriptor that describes this interface. This must be used for the device (function) name and be implemented in US English (LANGID = 0x0409) at the minimum.

3.7 VideoControl Interface Descriptors

The VideoControl (VC) interface descriptors contain all relevant information to fully characterize the corresponding video function. The standard interface descriptor characterizes the interface itself, whereas the class-specific interface descriptor provides pertinent information concerning the internals of the video function. It specifies revision level information and lists the capabilities of each Unit and Terminal.

3.7.1 Standard VC Interface Descriptor

The standard VC interface descriptor is identical to the standard interface descriptor defined in section 9.6.5 "Interface" of *USB Specification Revision 2.0*, except that some fields have now dedicated values.

Table 3-2 Standard VC Interface Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 9
1	**bDescriptorType**	1	Constant	INTERFACE descriptor type
2	**bInterfaceNumber**	1	Number	Number of interface. A zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	**bAlternateSetting**	1	Number	Value used to select an alternate setting for the interface identified in the prior field.
4	**bNumEndpoints**	1	Number	Number of endpoints used by this interface (excluding endpoint 0). This number is 0 or 1 depending on whether the optional status interrupt endpoint is present.
5	**bInterfaceClass**	1	Class	CC_VIDEO. Video Interface Class code (assigned by the USB). See section A.1, "Video Interface Class Code".
6	**bInterfaceSubClass**	1	Subclass	SC_VIDEOCONTROL. Video Interface Subclass code. Assigned by this specification. See section A.2, "Video Interface Subclass Codes".
7	**bInterfaceProtocol**	1	Protocol	Must be set to PC_PROTOCOL_15.
8	**iInterface**	1	Index	Index of a string descriptor that describes this interface. This must be used for the device (function) name and be implemented in US English (LANGID = 0x0409) at the minimum.

3.7.2 Class-Specific VC Interface Descriptor

The class-specific VC interface descriptor is a concatenation of all the descriptors that are used to fully describe the video function, i.e., all Unit Descriptors (UDs) and Terminal Descriptors (TDs).

The total length of the class-specific VC interface descriptor depends on the number of Units and Terminals in the video function. Therefore, the descriptor starts with a header that reflects the total length in bytes of the entire class-specific VC interface descriptor in the **wTotalLength** field. The **bcdUVC** field identifies the release of the Video Device Class Specification with which this video function and its descriptors are compliant. The **bInCollection** field indicates how many VideoStreaming interfaces there are in the Video Interface Collection to which this VideoControl interface belongs. The **baInterfaceNr()** array contains the interface numbers of all the VideoStreaming interfaces in the Collection. The **bInCollection** and **baInterfaceNr()** fields together provide all necessary information to determine which interfaces together constitute the entire USB interface to the video function, i.e., describe the Video Interface Collection.

The order in which the Unit and Terminal descriptors are reported is not important, because every descriptor can be identified through its **bDescriptorType** and **bDescriptorSubtype** fields. The **bDescriptorType** field identifies the descriptor as being a class-specific interface descriptor. The **bDescriptorSubtype** field further qualifies the exact nature of the descriptor.

The following table defines the class-specific VC interface header descriptor.

Table 3-3 Class-specific VC Interface Header Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 12+n
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubType**	1	Constant	VC_HEADER descriptor subtype
3	**bcdUVC**	2	BCD	Video Device Class Specification release number in binary-coded decimal. (i.e. 2.10 is 210H and 1.50 is 150H)
5	**wTotalLength**	2	Number	Total number of bytes returned for the class-specific VideoControl interface descriptor. Includes the combined length of this descriptor header and all Unit and Terminal descriptors.
7	**dwClockFrequency**	4	Number	Use of this field has been deprecated. The device clock frequency in Hz. This will specify the units used for the time information fields in the Video Payload Headers of the primary data stream and format. The **dwClockFrequency** field of the Video Probe and Commit control replaces this descriptor field. A value for this field shall be chosen such that the primary or default function of the device will be available to host software that implements Version 1.0 of this specification.
11	**bInCollection**	1	Number	The number of VideoStreaming interfaces in the Video Interface Collection to which this VideoControl interface belongs: n
12	**baInterfaceNr(1)**	1	Number	Interface number of the first VideoStreaming interface in the Collection
…	**…**	…	…	…

12+(n-1)	**baInterfaceNr(n)**	1	Number	Interface number of the last VideoStreaming interface in the Collection

This header is followed by one or more Unit and/or Terminal Descriptors. The layout of the descriptors depends on the type of Unit or Terminal they represent. There is a descriptor type for each Unit and Terminal described in section 2.3, "Video Function Topology". They are summarized in the following sections. The first four fields are common for all Unit and Terminal Descriptors. They contain the Descriptor Length, Descriptor Type, Descriptor Subtype, and Unit or Terminal ID.

Each Unit and Terminal within the video function is assigned a unique identification number, the Unit ID (UID) or Terminal ID (TID), contained in the **bUnitID** or **bTerminalID** field of the descriptor. The value 0x00 is reserved for undefined ID, effectively restricting the total number of addressable entities in the video function (both Units and Terminals) to 255.

Besides uniquely identifying all addressable entities in a video function, the IDs also serve to describe the topology of the video function; i.e., the **bSourceID** field of a Unit or Terminal descriptor indicates to which other Unit or Terminal this Unit or Terminal is connected.

3.7.2.1 Input Terminal Descriptor

The Input Terminal descriptor (ITD) provides information to the Host that is related to the functional aspects of the Input Terminal.

The Input Terminal is uniquely identified by the value in the **bTerminalID** field. No other Unit or Terminal within the same video function may have the same ID. This value must be passed in the **bTerminalID** field of each request that is directed to the Terminal.

The **wTerminalType** field provides pertinent information about the physical entity that the Input Terminal represents. This could be a USB OUT endpoint, an external Composite Video In connection, a camera sensor, etc. A complete list of Terminal Type codes is provided in section B.2, "Input Terminal Types".

The **bAssocTerminal** field is used to associate an Output Terminal to this Input Terminal, effectively implementing a bi-directional Terminal pair. An example of this would be a tape unit on a camcorder, which would have Input and Output Terminals to sink and source video respectively. If the **bAssocTerminal** field is used, both associated Terminals must belong to the bi-directional Terminal Type group. If no association exists, the **bAssocTerminal** field must be set to zero.

The Host software can treat the associated Terminals as being physically related. In many cases, one Terminal can not exist without the other. An index to a string descriptor is provided to further describe the Input Terminal.

The following table presents an outline of the Input Terminal descriptor.

Table 3-4 Input Terminal Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 8 (+ x)
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_INPUT_TERMINAL descriptor subtype
3	**bTerminalID**	1	Constant	A non-zero constant that uniquely identifies the Terminal within the video function. This value is used in all requests to address this Terminal.
4	**wTerminalType**	2	Constant	Constant that characterizes the type of Terminal. See Appendix B, "Terminal Types".
6	**bAssocTerminal**	1	Constant	ID of the Output Terminal to which this Input Terminal is associated, or zero (0) if no such association exists.
7	**iTerminal**	1	Index	Index of a string descriptor, describing the Input Terminal.
…	…	…	…	Depending on the Terminal type, certain Input Terminal descriptors have additional fields. The descriptors for these special Terminal types are described in separate sections specific to those Terminals, and in accompanying documents.

3.7.2.2 Output Terminal Descriptor

The Output Terminal descriptor (OTD) provides information to the Host that is related to the functional aspects of the Output Terminal.

The Output Terminal is uniquely identified by the value in the **bTerminalID** field. No other Unit or Terminal within the same video function may have the same ID. This value must be passed in the **bTerminalID** field of each request that is directed to the Terminal.

The **wTerminalType** field provides pertinent information about the physical entity the Output Terminal represents. This could be a USB IN endpoint, an external Composite Video Out connection, a LCD display, etc. A complete list of Terminal Type codes is provided in section B.3, "Output Terminal Types".

The **bAssocTerminal** field is used to associate an Input Terminal to this Output Terminal, effectively implementing a bi-directional Terminal pair. If the **bAssocTerminal** field is used, both associated Terminals must belong to the bi-directional Terminal Type group. If no association exists, the **bAssocTerminal** field must be set to zero.

The **bSourceID** field is used to describe the connectivity for this Terminal. It contains the ID of the Unit or Terminal to which this Output Terminal is connected via its Input Pin.
An index to a string descriptor is provided to further describe the Output Terminal.

The following table presents an outline of the Output Terminal descriptor.

Table 3-5 Output Terminal Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 9 (+ x)
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_OUTPUT_TERMINAL descriptor subtype
3	**bTerminalID**	1	Constant	A non-zero constant that uniquely identifies the Terminal within the video function. This value is used in all requests to address this Terminal.
4	**wTerminalType**	2	Constant	Constant that characterizes the type of Terminal. See Appendix B, "Terminal Types".
6	**bAssocTerminal**	1	Constant	Constant, identifying the Input Terminal to which this Output Terminal is associated, or zero (0) if no such association exists.
7	**bSourceID**	1	Constant	ID of the Unit or Terminal to which this Terminal is connected.
8	**iTerminal**	1	Index	Index of a string descriptor, describing the Output Terminal.
…	…	…	…	Depending on the Terminal type, certain Output Terminal descriptors have additional fields. The descriptors for these special Terminal types are described in accompanying documents.

3.7.2.3 Camera Terminal Descriptor

The Camera Terminal is uniquely identified by the value in the **bTerminalID** field. No other Unit or Terminal within the same video function may have the same ID. This value must be passed in the **bTerminalID** field of each request that is directed to the Terminal.

The **wTerminalType** field provides pertinent information about the physical entity that the Input Terminal represents. For the Camera Terminal, this field shall be set to ITT_CAMERA (see section B.2, “Input Terminal Types”).

The **bAssocTerminal** field is used to associate an Output Terminal to this Input Terminal, effectively implementing a bi-directional Terminal pair. An index to a string descriptor is provided to further describe the Camera Terminal.

The **bmControls** field is a bitmap, indicating the availability of certain camera controls for the video stream.

The layout of the Camera Terminal descriptor is detailed in the following table.

Table 3-6 Camera Terminal Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 18
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_INPUT_TERMINAL descriptor subtype
3	**bTerminalID**	1	Constant	A non-zero constant that uniquely identifies the Terminal within the video function. This value is used in all requests to address this Terminal.
4	**wTerminalType**	2	Constant	Constant that characterizes the type of Terminal. This is set to the ITT_CAMERA value.
6	**bAssocTerminal**	1	Constant	ID of the Output Terminal to which this Input Terminal is associated.
7	**iTerminal**	1	Index	Index of a string descriptor that describes the Camera Terminal.
8	**wObjectiveFocalLengthMin**	2	Number	The value of L_{min} If Optical Zoom is not supported; this field shall be set to 0.
10	**wObjectiveFocalLengthMax**	2	Number	The value of L_{max} If Optical Zoom is not supported; this field shall be set to 0.
12	**wOcularFocalLength**	2	Number	The value of L_{ocular} If Optical Zoom is not supported; this field shall be set to 0.
14	**bControlSize**	1	Number	Size in bytes of the **bmControls** field: 3
15	**bmControls**	3	Bitmap	A bit set to 1 indicates that the mentioned Control is supported for the video stream. D0: Scanning Mode D1: Auto-Exposure Mode

				D2: Auto-Exposure Priority D3: Exposure Time (Absolute) D4: Exposure Time (Relative) D5: Focus (Absolute) D6 : Focus (Relative) D7: Iris (Absolute) D8 : Iris (Relative) D9: Zoom (Absolute) D10: Zoom (Relative) D11: PanTilt (Absolute) D12: PanTilt (Relative) D13: Roll (Absolute) D14: Roll (Relative) D15: Reserved D16: Reserved D17: Focus, Auto D18: Privacy D19: Focus, Simple D20: Window D21: Region of Interest D22 – D23: Reserved, set to zero

3.7.2.4 Selector Unit Descriptor

The Selector Unit is uniquely identified by the value in the **bUnitID** field of the Selector Unit descriptor (SUD). No other Unit or Terminal within the same video function may have the same ID. This value must be passed with each request that is directed to the Selector Unit.

The **bNrInPins** field contains the number of Input Pins *(*p) of the Selector Unit. The connectivity of the Input Pins is described via the **baSourceID()** array that contains *p* elements. The index *i* into the array is one-based and directly related to the Input Pin numbers. **baSourceID(i)** contains the ID of the Unit or Terminal to which Input Pin *i* is connected.

An index to a string descriptor is provided to further describe the Selector Unit.

The following table details the structure of the Selector Unit descriptor.

Table 3-7 Selector Unit Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 6+p
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_SELECTOR_UNIT descriptor subtype
3	**bUnitID**	1	Number	A non-zero constant that uniquely identifies the Unit within the video function. This

				value is used in all requests to address this Unit.
4	**bNrInPins**	1	Number	Number of Input Pins of this Unit: p
5	**baSourceID(1)**	1	Number	ID of the Unit or Terminal to which the first Input Pin of this Selector Unit is connected.
…	**…**	…	…	…
5+(p-1)	**baSourceID(p)**	1	Number	ID of the Unit or Terminal to which the last Input Pin of this Selector Unit is connected.
5+p	**iSelector**	1	Index	Index of a string descriptor, describing the Selector Unit.

3.7.2.5 Processing Unit Descriptor

The Processing Unit is uniquely identified by the value in the **bUnitID** field of the Processing Unit descriptor (PUD). No other Unit or Terminal within the same video function may have the same ID. This value must be passed with each request that is directed to the Processing Unit.

The **bSourceID** field is used to describe the connectivity for this Processing Unit. It contains the ID of the Unit or Terminal to which this Processing Unit is connected via its Input Pin. **bSourceID** must refer to a Unit or Terminal in the same video function. The **bmControls field** is a bit-map, indicating the availability of certain processing Controls for the video stream.

An index to a string descriptor is provided to further describe the Processing Unit.

The layout of the Processing Unit descriptor is detailed in the following table.

Table 3-8 Processing Unit Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 13
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_PROCESSING_UNIT descriptor subtype
3	**bUnitID**	1	Number	A non-zero constant that uniquely identifies the Unit within the video function. This value is used in all requests to address this Unit.
4	**bSourceID**	1	Constant	ID of the Unit or Terminal to which this Unit is connected.
5	**wMaxMultiplier**	2	Number	If the Digital Multiplier control is supported, this field indicates the maximum digital magnification, multiplied by 100. For example, for a device that supports 1-4.5X digital zoom (a multiplier of 4.5), this field would be set to 450. If the Digital Multiplier

				control is not supported, this field shall be set to 0.
7	**bControlSize**	1	Number	Size of the **bmControls** field, in bytes: 3
8	**bmControls**	3	Bitmap	A bit set to 1 indicates that the mentioned Control is supported for the video stream. D0: Brightness D1: Contrast D2: Hue D3: Saturation D4: Sharpness D5: Gamma D6: White Balance Temperature D7: White Balance Component D8: Backlight Compensation D9: Gain D10: Power Line Frequency D11: Hue, Auto D12: White Balance Temperature, Auto D13: White Balance Component, Auto D14: Digital Multiplier D15: Digital Multiplier Limit D16: Analog Video Standard D17: Analog Video Lock Status D18: Contrast, Auto D19 – D23: Reserved. Set to zero.
11	**iProcessing**	1	Index	Index of a string descriptor that describes this processing unit.
12	**bmVideoStandards**	1	Bitmap	A bitmap of all analog video standards supported by the Processing Unit. A value of zero indicates that this bitmap should be ignored. D0: None D1: NTSC – 525/60 D2: PAL – 625/50 D3: SECAM – 625/50 D4: NTSC – 625/50 D5: PAL – 525/60 D6-D7: Reserved. Set to zero.

3.7.2.6 Encoding Unit Descriptor

The Encoding Unit is uniquely identified by the value in the **bUnitID** field of the Encoding Unit descriptor (EUD). No other Unit or Terminal within the same video function may have the same ID. This value must be passed with each request that is directed to the Encoding Unit.

The **bSourceID** field is used to describe the connectivity for this Encoding Unit. It contains the ID of the Unit or Terminal to which this Encoding Unit is connected via its Input Pin. **bSourceID** must refer to a Unit or Terminal in the same video function. The **bmControls** field is a bit-map, indicating the availability of certain encoding Controls for the video stream.

An index to a string descriptor is provided to further describe the Encoding Unit.

The layout of the Encoding Unit descriptor is detailed in the following table.

Table 3-9 Encoding Unit Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes. The value must be 13.
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type.
2	**bDescriptorSubtype**	1	Constant	VC_ENCODING_UNIT descriptor subtype.
3	**bUnitID**	1	Number	A non-zero constant that uniquely identifies the Unit within the video function. This value is used in all requests to address this Unit.
4	**bSourceID**	1	Constant	ID of the Unit or Terminal to which this Unit is connected.
5	**iEncoding**	1	Index	Index of a string descriptor that describes this encoding unit.
6	**bControlSize**	1	Number	Size, in bytes, of the bmControls and bmControlsRuntime fields: The value must be 3.
7	**bmControls**	3	Bitmap	A bit set to 1 indicates that the specified control is supported for initialization: D0: Select Layer D1: Profile and Toolset D2: Video Resolution D3: Minimum Frame Interval D4: Slice Mode D5: Rate Control Mode D6: Average Bit Rate D7: CPB Size D8: Peak Bit Rate D9: Quantization Parameter

				D10: Synchronization and Long-Term Reference Frame D11: Long-Term Buffer D12: Picture Long-Term Reference D13: LTR Validation D14: Level IDC D15: SEI Message D16: QP Range D17: Priority ID D18: Start or Stop Layer/View D19: Error Resiliency D20 - 23: Reserved; set to zero
10	**bmControlsRuntime**	3	Bitmap	A bit set to 1 indicates that the mentioned control is supported during runtime: D0: Select Layer D1: Profile and Toolset. D2: Video Resolution D3: Minimum Frame Interval D4: Slice Mode. D5: Rate Control Mode D6: Average Bit Rate D7: CPB Size D8: Peak Bit Rate. D9: Quantization Parameter D10: Synchronization and Long-Term Reference Frame D11: Long-Term Buffer D12: Picture Long-Term Reference D13: LTRValidation D14: Level IDC D15: SEI Message D16: QP Range D17: Priority ID. D18: Start or Stop Layer/View D19: Error Resiliency D20 – D23: Reserved, set to zero.

The Encoding Unit Descriptor supports separate lists for **bmControls** and **bmRuntimeControls**. The use here of two lists reflects the expectation that many UVC devices will not be able to support the same features while streaming video as during initialization. This is partially due to the inherent asynchronous nature of encoder control offered over USB.

3.7.2.7 Extension Unit Descriptor

The Extension Unit is uniquely identified by the value in the **bUnitID** field of the Extension Unit descriptor (XUD). No other Unit or Terminal within the same video function may have the same ID. This value must be passed with each request that is directed to the Extension Unit.

The Extension Unit Descriptor allows the hardware designer to define any arbitrary set of controls such that a class driver can act as an intermediary between vendor-supplied host software and functionality of the device.

The **guidExtensionCode** field contains a vendor-specific code that further identifies the Extension Unit.

The **bNrInPins** field contains the number of Input Pins (p) of the Extension Unit. The connectivity of the Input Pins is described via the **baSourceID()** array that contains *p* elements. The index *i* into the array is one-based and directly related to the Input Pin numbers. **baSourceID(i)** contains the ID of the Unit or Terminal to which Input Pin *i* is connected.

The **bmControls** field is a bitmap, indicating the availability of certain video Controls in the Extension Unit. For future expandability, the number of bytes occupied by the **bmControls** field is indicated in the **bControlSize** field. All Controls are optional.

An index to a string descriptor is provided to further describe the Extension Unit.

The following table outlines the Extension Unit descriptor.

Table 3-10 Extension Unit Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 24+p+n
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VC_EXTENSION_UNIT descriptor subtype
3	**bUnitID**	1	Number	A non-zero constant that uniquely identifies the Unit within the video function. This value is used in all requests to address this Unit.
4	**guidExtensionCode**	16	GUID	Vendor-specific code identifying the Extension Unit
20	**bNumControls**	1	Number	Number of controls in this extension unit
21	**bNrInPins**	1	Number	Number of Input Pins of this Unit: p
22	**baSourceID(1)**	1	Number	ID of the Unit or Terminal to which the first Input Pin of this Extension Unit is connected.
…	**…**	…	…	…

22+(p-1)	**baSourceID(p)**	1	Number	ID of the Unit or Terminal to which the last Input Pin of this Extension Unit is connected.
22+p	**bControlSize**	1	Number	Size of the **bmControls** field, in bytes: n
23+p	**bmControls**	n	Bitmap	A bit set to 1 indicates that the mentioned Control is supported: D(n*8-1)..0: Vendor-specific
23+p+n	**iExtension**	1	Index	Index of a string descriptor that describes this extension unit.

3.8 VideoControl Endpoint Descriptors

The following sections describe all possible endpoint-related descriptors for the VideoControl interface.

3.8.1 VC Control Endpoint Descriptors

3.8.1.1 Standard VC Control Endpoint Descriptor

Because endpoint 0 is used as the VideoControl control endpoint, there is no dedicated standard control endpoint descriptor.

3.8.1.2 Class-Specific VC Control Endpoint Descriptor

There is no dedicated class-specific control endpoint descriptor.

3.8.2 VC Interrupt Endpoint Descriptors

The standard and class-specific Interrupt Endpoint descriptors provide all information about the device interrupt usage.

3.8.2.1 Standard VC Interrupt Endpoint Descriptor

The interrupt endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.6.6 "Endpoint" of *USB Specification Revision 2.0* or *USB Specification Revision 3.0*. Its fields are set to reflect the interrupt type of the endpoint. This endpoint is optional.

The following table outlines the standard VC Interrupt Endpoint descriptor.

Table 3-11 Standard VC Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 7
1	**bDescriptorType**	1	Constant	ENDPOINT descriptor type
2	**bEndpointAddress**	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: D7: Direction. 1 = IN endpoint

				D6..4: Reserved, reset to zero. D3..0: The endpoint number, determined by the designer.
3	**bmAttributes**	1	Bitmap	D3..2: Synchronization type. Must be set to 00 (None) D1..0: Transfer type. Must be set to 11 (Interrupt). All other bits are reserved, and must be set to zero.
4	**wMaxPacketSize**	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	**bInterval**	1	Number	Interval for polling endpoint for data transfers. For full-speed interrupt endpoints, this value is expressed in frames, and must range from 1 to 255. For high-speed interrupt endpoints, the **bInterval** value is used as the exponent for a $2^{\mathbf{bInterval}-1}$ value; e.g., a **bInterval** of 4 means a period of 8 (2^3). This value must be from 1 to 16.

3.8.2.2 Class-specific VC Interrupt Endpoint Descriptor

The class-specific interrupt endpoint descriptor provides information about the maximum interrupt structure size that the device is capable of sending. The host driver will use this value to allocate a buffer of sufficient size to receive the maximum interrupt structure size. This descriptor is mandatory if the standard interrupt endpoint descriptor is defined.

Table 3-12 Class-specific VC Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 5
1	**bDescriptorType**	1	Constant	CS_ENDPOINT descriptor type
2	**bDescriptorSubType**	1	Constant	EP_INTERRUPT descriptor type
3	**wMaxTransferSize**	2	Number	Maximum interrupt structure size this endpoint is capable of sending.

3.9 VideoStreaming Interface Descriptors

The VideoStreaming (VS) interface descriptors contain all relevant information to characterize the VideoStreaming interface in full.

3.9.1 Standard VS Interface Descriptor

The standard VS interface descriptor is identical to the standard interface descriptor defined in section 9.6.5 "Interface" of *USB Specification Revision 2.0* or *USB Specification Revision 3.0*, except that some fields now have dedicated values.

Table 3-13 Standard VS Interface Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 9
1	**bDescriptorType**	1	Constant	INTERFACE descriptor type
2	**bInterfaceNumber**	1	Number	Number of the interface. A zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	**bAlternateSetting**	1	Number	Value used to select this alternate setting for the interface identified in the prior field.
4	**bNumEndpoints**	1	Number	Number of endpoints used by this interface (excluding endpoint 0).
5	**bInterfaceClass**	1	Class	CC_VIDEO. Video Interface Class code (assigned by the USB). See section A.1, "Video Interface Class Code".
6	**bInterfaceSubClass**	1	subclass	SC_VIDEOSTREAMING. Video interface subclass code (assigned by this specification). See section A.2, "Video Interface Subclass Codes".
7	**bInterfaceProtocol**	1	Protocol	Must be set to PC_PROTOCOL_15.
8	**iInterface**	1	Index	Index of a string descriptor that describes this interface.

3.9.2 Class-Specific VS Interface Descriptors

The class-specific VS interface descriptors consist of Input Header, Output Header, Format and Frame descriptors.

There is a single Input or Output Header descriptor for each VS interface, and a separate Format descriptor for each supported video stream format and a separate list of Frame descriptors for each Format descriptor (if the Format requires Frame descriptors). Header, Format and Frame descriptors are only defined in alternate setting 0 of the relevant interface. They are not repeated within subsequent alternate settings of the same interface.

3.9.2.1 Input Header Descriptor

The Input Header descriptor is used for VS interfaces that contain an IN endpoint for streaming video data. It provides information on the number of different Format descriptors that will follow it, as well as the total size of all class-specific descriptors in alternate setting zero of this interface.

The following table defines the class-specific VS interface Input Header descriptor.

Table 3-14 Class-specific VS Interface Input Header Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 13+(p*n).
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VS_INPUT_HEADER descriptor subtype
3	**bNumFormats**	1	Number	Number of video payload Format descriptors following for this interface (excluding video Frame descriptors): p
4	**wTotalLength**	2	Number	Total number of bytes returned for the class-specific VideoStreaming interface descriptors including this header descriptor.
6	**bEndpointAddress**	1	Endpoint	The address of the isochronous or bulk endpoint used for video data. The address is encoded as follows: D7: Direction 1 = IN endpoint D6..4: Reserved, set to zero. D3..0: The endpoint number, determined by the designer.
7	**bmInfo**	1	Bitmap	Indicates the capabilities of this VideoStreaming interface: D0: Dynamic Format Change supported D7..1: Reserved, set to zero.
8	**bTerminalLink**	1	Constant	The terminal ID of the Output Terminal to which the video endpoint of this interface is connected.
9	**bStillCaptureMethod**	1	Number	Method of still image capture supported as described in section 2.4.2.4, "Still Image Capture": 0: None (Host software will not support any form of still image capture) 1: Method 1 2: Method 2 3: Method 3
10	**bTriggerSupport**	1	Number	Specifies if hardware triggering is supported through this interface 0: Not supported 1: Supported
11	**bTriggerUsage**	1	Number	Specifies how the host software shall respond to a hardware trigger interrupt event from this interface. This is ignored if

				the **bTriggerSupport** field is zero. 0: Initiate still image capture 1: General purpose button event. Host driver will notify client application of button press and button release events
12	**bControlSize**	1	Number	Size of each **bmaControls(x)** field, in bytes: n
13	**bmaControls(1)**	n	Bitmap	For bits D3..0, a bit set to 1 indicates that the named *field* is supported by the Video Probe and Commit Control when **bFormatIndex** is 1: D0: **wKeyFrameRate** D1: **wPFrameRate** D2: **wCompQuality** D3: **wCompWindowSize** For bits D5..4, a bit set to 1 indicates that the named *control* is supported by the device when **bFormatIndex** is 1: D4: Generate Key Frame D5: Update Frame Segment D6..(n*8-1): Reserved, set to zero *Note* going forward from version 1.5 the proper means to detect whether a field is supported by Probe & Commit is to issue a VS_PROBE_CONTROL(GET_CUR).
…	**…**	…	…	…
13+(p*n-n)	**bmaControls(p)**	n	Bitmap	For bits D3..0, a bit set to 1 indicates that the named *field* is supported by the Video Probe and Commit Control when **bFormatIndex** is p: D0: **wKeyFrameRate** D1: **wPFrameRate** D2: **wCompQuality** D3: **wCompWindowSize** For bits D5..4, a bit set to 1 indicates that the named *control* is supported by the device when **bFormatIndex** is p: D4: Generate Key Frame D5: Update Frame Segment D6..(n*8-1): Reserved, set to zero

				Note D0-D3 are deprecated. Going forward from version 1.5 the proper means to detect whether a field is supported by Probe & Commit is to issue a VS_PROBE_CONTROL(GET_CUR).

3.9.2.2 Output Header Descriptor

The Output Header descriptor is used for VS interfaces that contain an OUT endpoint for streaming video data. It provides information on the number of different Format descriptors that will follow it, as well as the total size of all class-specific descriptors in alternate setting zero of this interface.

The following table defines the class-specific VS interface output header descriptor:

Table 3-15 Class-specific VS Interface Output Header Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 9+(p*n)
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VS_OUTPUT_HEADER descriptor subtype
3	**bNumFormats**	1	Number	Number of video payload Format descriptors following for this interface (excluding video Frame descriptors): p
4	**wTotalLength**	2	Number	Total number of bytes returned for the class-specific VideoStreaming interface descriptors including this header descriptor.
6	**bEndpointAddress**	1	Endpoint	The address of the isochronous or bulk endpoint used for video data. The address is encoded as follows: D7: Direction 0 = OUT endpoint D6..4: Reserved, set to zero D3..0: The endpoint number, determined by the designer.
7	**bTerminalLink**	1	Constant	The terminal ID of the Input Terminal to which the video endpoint of this interface is connected.
8	**bControlSize**	1	Number	Size of each **bmaControls(x)** field, in bytes: n
9	**bmaControls(1)**	n	Bitmap	For bits D3..0, a bit set to 1 indicates that the named *field* is supported by the Video Probe and Commit Control when

				bFormatIndex is 1: D0: **wKeyFrameRate** D1: **wPFrameRate** D2: **wCompQuality** D3: **wCompWindowSize** D4..(n*8-1): Reserved, set to zero *Note* D0-D3 are deprecated. Going forward from version 1.5, the proper means to detect whether a field is supported by Probe & Commit is to issue a VS_PROBE_CONTROL(GET_CUR).
…	**…**	…	…	…
9+(p*n-n)	**bmaControls(p)**	n	Bitmap	For bits D3..0, a bit set to 1 indicates that the named *field* is supported by the Video Probe and Commit Control when **bFormatIndex** is p: D0: **wKeyFrameRate** D1: **wPFrameRate** D2: **wCompQuality** D3: **wCompWindowSize** D4..(n*8-1): Reserved, set to zero

3.9.2.3 Payload Format Descriptors

A Payload Format descriptor defines the characteristics of a video stream with its specific format. The following descriptors are defined in the separate Payload Specification documents that accompany this document. For more information about a specific descriptor, see the corresponding document.

Table 3-16 Payload Format Descriptor

Payload Format Descriptor	Document
Uncompressed Video	USB_Video_Payload_Uncompressed
MJPEG Video	USB_Video_Payload_MJPEG
MPEG1-SS	USB_Video_Payload_Stream_Based
MPEG2-PS	USB_Video_Payload_Stream_Based
MPEG-2 TS	USB_Video_Payload_MPEG2-TS
H.264	USB_Video_Payload_H264
VP8	USB_Video_Payload_VP8
SMTPE VC1	USB_Video_Payload_MPEG2-TS
MPEG-4 SL	USB_Video_Payload_MPEG2-TS

DV	USB_Video_Payload_DV
Vendor Defined	USB_Video_Payload_Stream_Based or USB_Video_Payload_Frame_Based

3.9.2.4 Video Frame Descriptor

A Video Frame descriptor (or Frame descriptor for short) is used to describe the decoded video and still image frame dimensions and other frame-specific characteristics supported by Frame-based formats. Frame descriptors (if required) immediately follow the associated Format descriptor.
The following Video Frame descriptors are defined in the separate Payload Specification documents that accompany this document. For more information about a specific Frame descriptor, see the corresponding document.

Table 3-17 Defined Video Frame Descriptor Resources

Video Frame Descriptor	Document
Uncompressed	USB_Video_Payload_Uncompressed
MJPEG	USB_Video_Payload_MJPEG
Generic Frame-Based	USB_Video_Payload_Frame_Based
H.264	USB_Video_Payload_H.264
VP8	USB_Video_Payload_VP8

3.9.2.5 Still Image Frame Descriptor

The Still Image Frame descriptor is only applicable for a VS interface that supports method 2 or 3 of still image capture in conjunction with frame-based Payload formats (e.g., MJPEG, uncompressed, etc.). The Still Image Frame descriptor defines the characteristics of the still image capture for these frame-based formats. A single still Image Frame descriptor follows the Frame descriptor(s) for each Format descriptor group. If the Input Header descriptor's **bStillCaptureMethod** field is set to method 2 or 3, this Still Image Frame descriptor shall be defined (see section 3.9.2.1, "Input Header Descriptor").

The Still Image Frame descriptor contains the range of image sizes available from the device, which comprise the list of possible still image formats. To select a particular still image format, host software sends control requests to the corresponding interface (see section 4.3.1.2, "Video Still Probe Control and Still Commit Control").

The Still Image Frame descriptor is shown in Table 3-18 Still Image Frame Descriptor below.

The **bEndpointAddress** field contains the bulk endpoint address within the related VS interface that is used for still image capture. The endpoint always functions as an IN-Endpoint.

The **wWidth(x)** and **wHeight(x)** fields form an array of image sizes supported by the device, measured in pixels of an uncompressed image.

The **bNumImageSizePattern**s represents the number of **wWidth** and **wHeight** pairs in the array.

The **bCompression** field represents the image quality that would be generated by the device. The range of compression values is from 0 to 255. A small value indicates a low compression ratio and high quality image. The default setting of this value depends on device implementation. The **bCompression(x)** fields form an array of compression ratios supported by the device for all image sizes. The **bNumCompressionPatterns** field represents the number of **bCompression** fields in this array**.**

A Still Image Frame descriptor identifies the following:

Table 3-18 Still Image Frame Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 10+(4*n)-4+m
1	**bDescriptorType**	1	Constant	CS_INTERFACE descriptor type
2	**bDescriptorSubtype**	1	Constant	VS_STILL_IMAGE_FRAME descriptor subtype
3	**bEndpointAddress**	1	Endpoint	If method 3 of still image capture is used, this contains the address of the bulk endpoint used for still image capture. The address is encoded as follows: D7: Direction. (set to 1 = IN endpoint) D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer If method 2 of still image capture is used, this field shall be set to zero.
4	**bNumImageSizePatterns**	1	Number	Number of Image Size patterns of this format: n
5	**wWidth(1)**	2	Number	Width of the still image in pattern 1
7	**wHeight(1)**	2	Number	Height of the still image in pattern 1
…	**…**	…	…	…
…	**…**	…	…	…
5+4*n-4	**wWidth(n)**	2	Number	Width of the still image in pattern n
7+4*n-4	**wHeight(n)**	2	Number	Height of the still image in pattern n
9+4*n-4	**bNumCompressionPattern**	1	Number	Number of Compression pattern of this format: m
10+4*n-4	**bCompression(1)**	1	Number	Compression of the still image in pattern 1

…	**…**	…	…	…
10+4*n -4+m-1	**bCompression(m)**	1	Number	Compression of the still image in pattern m

3.9.2.6 Color Matching Descriptor

The Color Matching descriptor is an optional descriptor used to describe the color profile of the video data in an unambiguous way. Only one instance is allowed for a given format and if present, the Color Matching descriptor shall be placed following the Video and Still Image Frame descriptors for that format.

For example, this descriptor would be used with Uncompressed Video, MJPEG and MPEG-1 formats. It would not be used in the case MPEG-2, DV or MPEG-4 because the information is already available implicitly (DV) or explicitly (MPEG-2, MPEG-4). If a format requires this descriptor, the corresponding payload specification must enforce this requirement.
In the absence of this descriptor, or in the case of "Unspecified" values within the descriptor, color matching defaults will be assumed. The color matching defaults are compliant with sRGB since the BT.709 transfer function and the sRGB transfer function are very similar.

The viewing conditions and monitor setup are implicitly based on sRGB and the device should compensate for them (D50 ambient white, dim viewing or 64 lux ambient illuminance, 2.2 gamma reference CRT, etc).

Table 3-19 Color Matching Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Constant	6
1	**bDescriptorType**	1	Number	CS_INTERFACE type
2	**bDescriptorSubtype**	1	Number	VS_COLORFORMAT
3	**bColorPrimaries**	1	Number	This defines the color primaries and the reference white. 0: Unspecified (Image characteristics unknown) 1: BT.709, sRGB (default) 2: BT.470-2 (M) 3: BT.470-2 (B, G) 4: SMPTE 170M 5: SMPTE 240M 6-255: Reserved
4	**bTransferCharacteristics**	1	Number	This field defines the opto-electronic transfer characteristic of the source picture also called the gamma function. 0: Unspecified (Image characteristics unknown)

				1: BT.709 (default) 2: BT.470-2 M 3: BT.470-2 B, G 4: SMPTE 170M 5: SMPTE 240M 6: Linear (V = Lc) 7: sRGB (very similar to BT.709) 8-255: Reserved
5	**bMatrixCoefficients**	1	Number	Matrix used to compute luma and chroma values from the color primaries. 0: Unspecified (Image characteristics unknown) 1: BT. 709 2: FCC 3: BT.470-2 B, G 4: SMPTE 170M (BT.601, default) 5: SMPTE 240M 6-255: Reserved

3.10 VideoStreaming Endpoint Descriptors

The following sections describe all possible endpoint-related descriptors for the VideoStreaming interface.

3.10.1 VS Video Data Endpoint Descriptors

The video data endpoint can be implemented as either an isochronous or bulk endpoint. The standard isochronous or bulk endpoint descriptor provides pertinent information about how video data streams are communicated to the video function. In addition, specific endpoint capabilities and properties are reported.

3.10.1.1 Standard VS Isochronous Video Data Endpoint Descriptor

The standard VS isochronous video data endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.6.6 "Endpoint" of *USB Specification Revision 2.0*. D7 of the **bEndpointAddress** field indicates whether the endpoint is a video source (D7 = 1) or a video sink (D7 = 0). The **bmAttributes** field bits are set to reflect the isochronous type of the endpoint. The synchronization type is indicated by D3..2 and must be set to Asynchronous. For further details, refer to section 5.12.4.1 "Synchronization Type," of *USB Specification Revision 2.0*.

Table 3-20 Standard VS Isochronous Video Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 7
1	**bDescriptorType**	1	Constant	ENDPOINT descriptor type
2	**bEndpointAddress**	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: D7: Direction 0 = OUT endpoint 1 = IN endpoint D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer
3	**bmAttributes**	1	Bitmap	D3..2: Synchronization type 01 = Asynchronous D1..0: Transfer type 01 = Isochronous All other bits are reserved.
4	**wMaxPacketSize**	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected. This is determined by the video bandwidth constraints of the endpoint.
6	**bInterval**	1	Number	Interval for polling endpoint for data transfers. This value is expressed as a period of frames or microframes depending on device speed, and must range from 1 to 16. The bInterval value is used as the exponent for a $2^{bInterval-1}$ period.

3.10.1.2 Standard VS Bulk Video Data Endpoint Descriptor

The standard VS Bulk video data endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.6.6 "Endpoint" of *USB Specification Revision 2.0*. D7 of the **bEndpointAddress** field indicates that this endpoint is a data source (D7 = 1) or a video sink (D7 = 0). The **bmAttributes** field bits are set to reflect the bulk type of the endpoint.

Table 3-21 Standard VS Bulk Video Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 7
1	**bDescriptorType**	1	Constant	ENDPOINT descriptor type
2	**bEndpointAddress**	1	Endpoint	The address of the endpoint on the USB

				device described by this descriptor. The address is encoded as follows: D7: Direction 0 = OUT endpoint 1 = IN endpoint D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer
3	**bmAttributes**	1	Bitmap	D1..0: Transfer type (set to 10 = Bulk) All other bits are reserved.
4	**wMaxPacketSize**	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	**bInterval**	1	Number	Interval for polling endpoint for data transfers. For high-speed bulk OUT endpoints, the **bInterval** must specify the maximum NAK rate of the endpoint. A value of 0 indicates the endpoint never NAKs. Other values indicate at most 1 NAK each **bInterval** number of microframes. This value must be in the range from 0 to 255.

3.10.2 VS Bulk Still Image Data Endpoint Descriptors

The standard bulk still image data endpoint descriptor provides pertinent information on how still image data are communicated to the video function. In addition, specific endpoint capabilities and properties are reported.

3.10.2.1 Standard VS Bulk Still Image Data Endpoint Descriptor

The standard VS Bulk still image data endpoint descriptor is identical to the standard endpoint descriptor defined in section 9.6.6 "Endpoint" of *USB Specification Revision 2.0*. D7 of the **bEndpointAddress** field indicates that this endpoint is a data source (D7 = 1). The **bmAttributes** field bits are set to reflect the bulk type of the endpoint.

This optional endpoint is only implemented by the device if it supports method 3 of still image capture. If implemented, it should always follow the Video Data endpoint (where available) in descriptor ordering and endpoint addressing.

Table 3-22 Standard VS Bulk Still Image Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	**bLength**	1	Number	Size of this descriptor, in bytes: 7

1	**bDescriptorType**	1	Constant	ENDPOINT descriptor type
2	**bEndpointAddress**	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: D7: Direction (set to 1 = IN endpoint) D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer
3	**bmAttributes**	1	Bitmap	D1..0: Transfer type (set to 10 = Bulk) All other bits are reserved.
4	**wMaxPacketSize**	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	**bInterval**	1	Number	Not used, set to zero.

3.11 String Descriptors

The baseline requirement for devices in this class is for the device implementation to provide a function name string descriptor in US English (LANGID = 0x0409). This will be referenced in the **iInterface** field in the standard VideoControl interface descriptor. See section 3.7.1, "Standard VC Interface Descriptor".

If the VideoControl interface is part of a Video Interface Collection, the **iFunction** field in the IAD and the **iInterface** field in the Standard VC interface descriptor for this Video Interface Collection must be equal. See section 3.5.

All other string descriptors are optional.

Since the device must implement the device name string descriptor, it must also support String Descriptor Zero which contains the list of LANGID codes supported by the device. This descriptor, as well as the layout of a standard UNICODE String Descriptor, is defined in section 9.6.7 "String" of the *USB Specification Revision 2.0*.

4 Class-Specific Requests

Most class-specific requests are used to set and get video related Controls. These Controls fall into two main groups: those that manipulate controls related to the video function, such as brightness, exposure, selector position, etc. and those that influence data transfer over a video data endpoint, such as the current frame rate.

- VideoControl Requests. Control of a video function is performed through the manipulation of the attributes of individual Controls that are embedded in the Units and Terminals of the video function. The class-specific VideoControl interface descriptor contains a collection of Unit and Terminal descriptors, each indicating which Controls are present in each entity. VideoControl requests are always directed to the single VideoControl interface of the video function. The request contains enough information (Unit ID, Control Selector) for the video function to route a specific request correctly.
- VideoStreaming Requests. Control of the class-specific behavior of a VideoStreaming interface is performed through manipulation of Interface Controls. VideoStreaming requests are directed to the interface where the Control resides.

Requests may be mandatory or optional and listed as such for every control. Where SET_CUR is optional, its presence is determined via GET_INFO. If a video function does not support a certain request, it must indicate this by stalling the control pipe when that request is issued to the function.

4.1 Request Layout

The following paragraphs describe the general structure of the Set and Get requests. Subsequent paragraphs detail the use of the Set/Get requests for the different request types.

4.1.1 Set Request

This request is used to set an attribute of a Control inside an entity of the video function.

Table 4-1 Set Request

<table>
<tr><th>bmRequestType</th><th>bRequest</th><th>wValue</th><th>wIndex</th><th>wLength</th><th>Data</th></tr>
<tr><td>00100001</td><td rowspan="2">SET_CUR</td><td rowspan="2">See following paragraphs.</td><td>Entity ID and Interface.</td><td rowspan="2">Length of parameter block.</td><td rowspan="2">Parameter block.</td></tr>
<tr><td>00100010</td><td>Endpoint.</td></tr>
</table>

The **bmRequestType** field specifies that this is a SET request (D7=0). It is a class-specific request (D6..5=01), directed to either the VideoControl interface, or a VideoStreaming interface of the video function (D4..0=00001), or the video data endpoint of a VideoStreaming interface (D4..0=00010).

The **bRequest** field contains a constant that identifies which attribute of the addressed Control is to be modified. Possible attributes for a Control are:

- Current setting attribute (SET_CUR)

If the addressed Control or entity does not support modification of a certain attribute, the control pipe must indicate a stall when an attempt is made to modify that attribute. Only the CUR attribute is supported for the Set request. For the list of Request constants, refer to section A.8, "Video Class-Specific Request Codes"

The **wValue** field interpretation is qualified by the value in the **wIndex** field. Depending on what entity is addressed, the layout of the **wValue** field changes. The following paragraphs describe the contents of the **wValue** field for each entity separately. In most cases, the **wValue** field contains the Control Selector (CS) in the high byte. It is used to address a particular Control within entities that can contain multiple Controls. If the entity only contains a single Control, there is no need to specify a Control Selector and the **wValue** field can be used to pass additional parameters.

The **wIndex** field specifies the interface or endpoint to be addressed in the low byte, and the entity ID or zero in the high byte. In case an interface is addressed, the virtual entity "interface" can be addressed by specifying zero in the high byte. The values in **wIndex** must be appropriate to the recipient. Only existing entities in the video function can be addressed, and only appropriate interface or endpoint numbers may be used. If the request specifies an unknown or non-entity ID or an unknown interface or endpoint number, the control pipe must indicate a stall.

The actual parameter(s) for the Set request are passed in the data stage of the control transfer. The length of the parameter block is indicated in the **wLength** field of the request. The layout of the parameter block is qualified by both the **bRequest** and **wIndex** fields. Refer to the following sections for a detailed description of the parameter block layout for all possible entities.

4.1.2 Get Request

This request returns the attribute setting of a specific Control inside an entity of the video function.

Table 4-2 Get Request

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001	GET_CUR GET_MIN GET_MAX GET_RES GET_LEN GET_INFO GET_DEF	See following paragraphs.	Entity ID and Interface	Length of parameter block	Parameter block
10100010			Endpoint.		

The **bmRequestType** field specifies that this is a GET request (D7=1). It is a class-specific request (D6..5=01), directed to either the VideoControl interface or a VideoStreaming interface of the video function (D4..0=00001), or the video data endpoint of a VideoStreaming interface (D4..0=00010).

The **bRequest** field contains a constant that identifies which attribute of the addressed Control or entity is to be returned. Possible attributes for a Control are:

- Current setting attribute (GET_CUR)
- Minimum setting attribute (GET_MIN)
- Maximum setting attribute (GET_MAX)
- Default setting attribute (GET_DEF)
- Resolution attribute (GET_RES)
- Data length attribute (GET_LEN)
- Information attribute (GET_INFO)

The GET_INFO request queries the capabilities and status of the specified control. When issuing this request, the **wLength** field shall always be set to a value of 1 byte. The result returned is a bit mask reporting the capabilities of the control. The bits are defined as:

Table 4-3 Defined Bits Containing Capabilities of the Control

Bit field	Description	Bit State
D0	1=Supports GET value requests	Capability
D1	1=Supports SET value requests	Capability
D2	1=Disabled due to automatic mode (under device control)	State
D3	1= Autoupdate Control (see section 2.4.2.2 "Status Interrupt Endpoint")	Capability
D4	1= Asynchronous Control (see sections 2.4.2.2 "Status Interrupt Endpoint" and 2.4.4, “Control Transfer and Request Processing”)	Capability
D5	1= Disabled due to incompatibility with Commit state.	State
D7..D6	Reserved (Set to 0)	--

The two bits in GET_INFO that reflect the state of the control are D2 (Disabled due to Automatic Mode) and D5 (Disabled due to incompatibility with Commit state). The other bits are capability bits. Capability bits should not change when state bits change. For example, when a control is set in Automatic Mode (D2 set), the bit D1 must not be updated in GET_INFO.
If a control is implemented such that D2 can be set, the device needs to have the capability of sending Control Change Interrupts, thus D3 (Autoupdate Control) must be set. If a control is implemented such that D5 can be set, the device should to have the capability of sending Control Change Interrupts.

If an Encoding Unit control is implemented such that the device may initiate a change in the minimum and/or maximum setting attribute for that control, then the device should have the capability of sending Control Change Interrupts to notify the host of the new GET_MIN and/or GET_MAX settings, thus the D3 (Autoupdate Control) must be set.

The device indicates hardware default values for Unit, Terminal and Interface Controls through their GET_DEF values. These values may be used by the host to restore a control to its default setting.

If the addressed Control or entity does not support readout of a certain attribute, the control pipe must indicate a stall when an attempt is made to read that attribute. For the list of Request constants, refer to section A.8, "Video Class-Specific Request Codes".

The **wValue** field interpretation is qualified by the value in the **wIndex** field. Depending on what entity is addressed, the layout of the **wValue** field changes. The following paragraphs describe the contents of the **wValue** field for each entity separately. In most cases, the **wValue** field contains the Control Selector (CS) in the high byte. It is used to address a particular Control within entities that can contain multiple Controls. If the entity only contains a single Control, there is no need to specify a Control Selector and the **wValue** field can be used to pass additional parameters.

The **wIndex** field specifies the interface or endpoint to be addressed in the low byte, and the entity ID or zero in the high byte. In case an interface is addressed, the virtual entity "interface" can be addressed by specifying zero in the high byte. The values in **wIndex** must be appropriate to the recipient. Only existing entities in the video function can be addressed, and only appropriate interface or endpoint numbers may be used. If the request specifies an unknown or non-entity ID, or an unknown interface or endpoint number, the control pipe must indicate a stall.

The actual parameter(s) for the Get request are returned in the data stage of the control transfer. The length of the parameter block to return is indicated in the **wLength** field of the request. If the parameter block is longer than is indicated in the **wLength** field, only the initial bytes of the parameter block are returned. If the parameter block is shorter than is indicated in the **wLength** field, the device indicates the end of the control transfer by sending a short packet when further data is requested. The layout of the parameter block is qualified by both the **bRequest** and **wIndex** fields. Refer to the following sections for a detailed description of the parameter block layout for all possible entities.

4.2 VideoControl Requests

The following sections describe the possible requests that can be used to manipulate the video Controls that a video function exposes through its VideoControl interface and Units contained within it. The same layout of the parameter blocks is used for both the Set and Get requests.

Each of the following control definitions specifies whether requests are mandatory or optional for that control. Any implemented request must comply with the definition for that control. The device manufacturer is free to implement any other requests, but the definition of those unspecified requests shall be ignored by host implementations, with the exception of the GET_LEN request. If the GET_LEN request is implemented, the host software will use the result to determine the correct buffer length for Set and Get requests.

4.2.1 Interface Control Requests

These requests are used to set or read an attribute of an interface Control inside the VideoControl interface of the video function.

Table 4-4 Interface Control Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_CUR	CS	Zero and Interface.	Length of parameter block.	Parameter block.
10100001	GET_CUR GET_MIN GET_MAX GET_RES GET_INFO				

The **bRequest** field indicates which attribute the request is manipulating. The MIN, MAX, and RES attributes are not supported for the Set request.

The **wValue** field specifies the Control Selector (CS) in the high byte, and the low byte must be set to zero. The Control Selector indicates which type of Control this request is manipulating. If the request specifies an unknown CS to that endpoint, the control pipe must indicate a stall.

4.2.1.1 Power Mode Control

This control sets the device power mode. Power modes are defined in the following table.

Table 4-5 Power Mode Control

Device power mode	Description
Full power mode	Device operates at full functionality in this mode. For example, the device can stream video data via USB, and can execute all requests that are supported by the device. This mode is mandatory, even if the device doesn't support VIDEO POWER MODE CONTROL.

Vendor-dependent power mode	Device operates in low power mode. In this mode, the device continues to operate, although not at full functionality. For example, as the result of setting the device to this power mode, the device will stop the Zoom function. To avoid confusing the user, the device should issue an interrupt (GET_INFO) to notify the user that the Zoom function is disabled. In this mode, the device can stream video data, the functionality of USB is not affected, and the device can execute all requests that it supports. This mode is optional.

The power mode that is supported by the device must be passed to the host, as well as the power source, since if the device is working with battery power, the host can change the device power mode to “vendor-dependent power mode” to reduce power consumption.
Information regarding power modes and power sources is communicated through the following bit fields. D7..D5 indicates which power source is currently used in the device. The D4 indicates that the device supports “vendor-dependent power mode”. Bits D7..D4 are set by the device and are read-only. The host can change the device power mode by setting a combination of D3..D0.

The host can update the power mode during video streaming.

The D3..D0 value of 0000B indicates that the device is in, or should transition to, full power mode. The D3..D0 value of 0001B indicates that the device is in, or should transition to, vendor-dependent power mode.
The host must specify D3..D0 only when the power mode is required to switch, and the other fields must be set to 0.

Table 4-6 Device Power Mode

<table>
<tr><td colspan="2">Control selector</td><td colspan="6">VC_VIDEO_POWER_MODE_CONTROL</td></tr>
<tr><td colspan="2">Mandatory Requests</td><td colspan="6">SET_CUR, GET_CUR, GET_INFO</td></tr>
<tr><td colspan="2">wLength</td><td colspan="6">1</td></tr>
<tr><td>Offset</td><td>Field</td><td>Size</td><td>Value</td><td colspan="4">Description</td></tr>
<tr><td rowspan="4">0</td><td rowspan="4">bDevicePowerMode</td><td rowspan="4">1</td><td rowspan="4">Bitmap</td><td colspan="4"></td></tr>
<tr><td>Bit</td><td>Description</td><td>R</td><td>W</td></tr>
<tr><td>D3..0</td><td>Power Mode setting
0000B:Full power mode
0001B:device dependent power mode (opt.)
All other bits are reserved.</td><td>o</td><td>o</td></tr>
<tr><td>D4</td><td>Device dependent power mode supported.</td><td>o</td><td>x</td></tr>
</table>

				D5	Device uses power supplied by USB.	o	x
				D6	Device uses power supplied by Battery.	o	x
				D7	Device uses power supplied by A.C.	o	x

4.2.1.2 Request Error Code Control

This read-only control indicates the status of each host-initiated request to a Terminal, Unit, interface or endpoint of the video function. If the device is unable to fulfill the request, it will indicate a stall on the control pipe and update this control with the appropriate code to indicate the cause. This control will be reset to 0 (No error) upon the successful completion of any control request (including requests to this control). The table below specifies the **bRequestErrorCode** error codes that the device must return from a VC_REQUEST_ERROR_CODE_CONTROL request. Asynchronous control requests are a special case, where the initial request will update this control, but the final result is delivered via the Status Interrupt Endpoint (see sections 2.4.2.2, "Status Interrupt Endpoint" and 2.4.4, "Control Transfer and Request Processing").

Table 4-7 Request Error Code Control

Control Selector		VC_REQUEST_ERROR_CODE_CONTROL		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bRequestErrorCode**	1	Number	0x00: No error 0x01: Not ready 0x02: Wrong state 0x03: Power 0x04: Out of range 0x05: Invalid unit 0x06: Invalid control 0x07: Invalid Request 0x08: Invalid value within range 0x09-0xFE: Reserved for future use 0xFF: Unknown

No error: The request succeeded.

Not ready: The device has not completed a previous operation. The device will recover from this state as soon as the previous operation has completed.

Wrong State: The device is in a state that disallows the specific request. The device will remain in this state until a specific action from the host or the user is completed.

Power: The actual Power Mode of the device is not sufficient to complete the Request.

Out of Range: Result of a SET_CUR Request when attempting to set a value outside of the MIN and MAX range, or a value that does not satisfy the constraint on resolution (see section 4.2.2, “Unit and Terminal Control Requests”).

Invalid Unit: The Unit ID addressed in this Request is not assigned.

Invalid Control: The Control addressed by this Request is not supported.

Invalid Request: This Request is not supported by the Control.

Invalid value with range: Results of a SET_CUR Request when attempting to set a value that is inside the MIN and MAX range but is not supported.

4.2.2 Unit and Terminal Control Requests

The requests below are used to set or read an attribute of a Control inside of a Unit or Terminal of the video function.

Table 4-8 Unit and Terminal Control Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_CUR	CS	Unit or Terminal ID and Interface	Length of parameter block	Parameter block
10100001	GET_CUR GET_MIN GET_MAX GET_RES GET_INFO GET_DEF				
00100001	SET_CUR_ALL	Set to 0	Unit or Terminal ID		
10100001	GET_CUR_ALL GET_MIN_ALL GET_MAX_ALL GET_RES_ALL GET_DEF_ALL				

The **bRequest** field indicates which attribute the request is manipulating. The MIN, MAX and RES attributes are not supported for the Set request.

The **wValue** field specifies the Control Selector (CS) in the high byte, and zero in the low byte. The Control Selector indicates which type of Control this request is manipulating. When processing all Controls as part of a batch request (GET_###_ALL), **wValue** is not needed and must be set to 0. If the request specifies an unknown or unsupported CS to that Unit or Terminal, the control pipe must indicate a protocol STALL.

The value of **wLength** must be calculated as follows. Use wIndex to determine the Unit or Terminal of interest. For that Unit or Terminal, establish which Controls are supported using the **bmControls** field of the associated Unit or Terminal Descriptor. **wLength** is the sum of the length of all supported Controls for the target Unit or Terminal. The **Data** must be ordered according to the order of the Controls listed in the **bmControls** field of the target Unit or Terminal descriptor. If the Unit or Terminal supports batch requests, then each Control in the Unit or Terminal must contribute to the **Data** field, even if it does not support the associated single operation request.

If a Control supports GET_MIN, GET_MAX and GET_RES requests, the values of MAX, MIN and RES shall be constrained such that *(MAX-MIN)/RES* is an integral number. Furthermore, the CUR value (returned by GET_CUR, or set via SET_CUR) shall be constrained such that *(CUR-MIN)/RES* is an integral number. The device shall indicate protocol STALL and update the Request Error Code Control with 0x04 “Out of Range” if an invalid CUR value is provided in a SET_CUR operation (see section 2.4.4, “Control Transfer and Request Processing”).

There are special Terminal types (such as the Camera Terminal and Media Transport Terminal) that have type-specific Terminal Controls defined. The controls for the Media Transport Terminal are defined in a companion specification (see the *USB Device Class Definition for Video Media Transport Terminal* specification). The controls for the Camera Terminal are defined in the following sections.

As this specification evolves, new controls in the Camera Terminal, Processing Unit, and Encoding Unit are added to the list of associated Control Selectors at the end (Tables A-12 through A-14). However, in the sections below, the description of the functionality is placed next to controls with associated functionality.

4.2.2.1 Camera Terminal Control Requests

The following paragraphs present a detailed description of all possible Controls a Camera Terminal can incorporate. For each Control, the layout of the parameter block together with the appropriate Control Selector is listed for all forms of the Get/Set Camera Terminal Control request. All values are interpreted as absolute (fixed-origin), and not relative unless otherwise specified. They are also assumed to be unsigned unless otherwise specified.

4.2.2.1.1 Scanning Mode Control

The Scanning Mode Control setting is used to control the scanning mode of the camera sensor. A value of 0 indicates that the interlace mode is enabled, and a value of 1 indicates that the progressive or the non-interlace mode is enabled.

Table 4-9 Scanning Mode Control

Control Selector		CT_SCANNING_MODE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bScanningMode**	1	Boolean	The setting for the attribute of the addressed Scanning Mode Control: 0: Interlaced 1: Progressive

4.2.2.1.2 Auto-Exposure Mode Control

The Auto-Exposure Mode Control determines whether the device will provide automatic adjustment of the Exposure Time and Iris controls. Attempts to programmatically set the auto-adjusted controls shall result in a protocol STALL and an error code of **bRequestErrorCode** = "Wrong state". A GET_RES request issued to this control will return a bitmap of the modes supported by this control. A valid request to this control would have only one bit set (a single mode selected). This control must accept the GET_DEF request and return its default value.

Table 4-10 Auto-Exposure Mode Control

Control Selector		CT_AE_MODE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_RES, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bAutoExposureMode**	1	Bitmap	The setting for the attribute of the addressed Auto-Exposure Mode Control: D0: Manual Mode – manual Exposure Time, manual Iris D1: Auto Mode – auto Exposure Time, auto Iris D2: Shutter Priority Mode – manual Exposure Time, auto Iris D3: Aperture Priority Mode – auto Exposure Time, manual Iris D4..D7: Reserved, set to zero.

4.2.2.1.3 Auto-Exposure Priority Control

The Auto-Exposure Priority Control is used to specify constraints on the Exposure Time Control when the Auto-Exposure Mode Control is set to Auto Mode or Shutter Priority Mode. A value of zero indicates that the frame rate must remain constant. A value of 1 indicates that the frame rate may be dynamically varied by the device. The default value is zero (0).

Table 4-11 Auto-Exposure Priority Control

Control Selector		CT_AE_PRIORITY_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bAutoExposurePriority**	1	Number	The setting for the attribute of the addressed AutoExposure Priority control.

4.2.2.1.4 Exposure Time (Absolute) Control

The Exposure Time (Absolute) Control is used to specify the length of exposure. This value is expressed in 100µs units, where 1 is 1/10,000th of a second, 10,000 is 1 second, and 100,000 is 10 seconds. A value of zero (0) is undefined. Note that the manual exposure control is further limited by the frame interval, which always has higher precedence. If the frame interval is changed to a value below the current value of the Exposure Control, the Exposure Control value will automatically be changed. The default Exposure Control value will be the current frame interval until an explicit exposure value is chosen. When the Auto-Exposure Mode control is in Auto mode or Aperture Priority mode attempts to programmatically set this control shall result in a protocol STALL and an error code of **bRequestErrorCode** = “Wrong state”. This control must accept the GET_DEF request and return its default value.

Table 4-12 Exposure Time (Absolute) Control

Control Selector		CT_EXPOSURE_TIME_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwExposureTimeAbsolute**	4	Number	The setting for the attribute of the addressed Exposure Time (Absolute) Control: 0: Reserved 1: 0.0001 sec … 100000: 10 sec …

4.2.2.1.5 Exposure Time (Relative) Control

The Exposure Time (Relative) Control is used to specify the electronic shutter speed. This value is expressed in number of steps of exposure time that is incremented or decremented. A value of one (1) indicates that the exposure time is incremented one step further, and a value 0xFF indicates that the exposure time is decremented one step further. This step is implementation specific. A value of zero (0) indicates that the exposure time is set to the default value for implementation. The default values are implementation specific. . When the Auto-Exposure Mode control is in Auto mode or Aperture Priority mode attempts to programmatically set this control shall result in a protocol STALL and an error code of **bRequestErrorCode** = "Wrong state".

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control (see section 2.4.2.2, "Status Interrupt Endpoint").

Table 4-13 Exposure Time (Relative) Control

Control Selector		CT_EXPOSURE_TIME_RELATIVE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bExposureTimeRelative**	1	Signed Number	The setting for the attribute of the addressed Exposure Time (Relative) Control: 0: default 1: incremented by 1 step 0xFF: decremented by 1 step

4.2.2.1.6 Focus (Absolute) Control

The Focus (Absolute) Control is used to specify the distance to the optimally focused target. This value is expressed in millimeters. The default value is implementation-specific. This control must accept the GET_DEF request and return its default value. When the Auto-Focus Mode control is enabled, attempts to programmatically set this control shall result in a protocol STALL and an error code of **bRequestErrorCode** = "Wrong state".

Table 4-14 Focus (Absolute) Control

Control Selector		CT_FOCUS_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description
0	**wFocusAbsolute**	2	Number	The setting for the attribute of the addressed Focus (Absolute) Control.

4.2.2.1.7 Focus (Relative) Control

The Focus (Relative) Control is used to move the focus lens group to specify the distance to the optimally focused target.

The **bFocusRelative** field indicates whether the focus lens group is stopped or is moving for near or for infinity direction. A value of 1 indicates that the focus lens group is moved for near direction. A value of 0 indicates that the focus lens group is stopped. And a value of 0xFF indicates that the lens group is moved for infinity direction. The GET_MIN, GET_MAX, GET_RES and GET_DEF requests will return zero for this field.

The **bSpeed** field indicates the speed of the lens group movement. A low number indicates a slow speed and a high number indicates a high speed. The GET_MIN, GET_MAX and GET_RES requests are used to retrieve the range and resolution for this field. The GET_DEF request is used to retrieve the default value for this field. If the control does not support speed control, it will return the value 1 in this field for all these requests.

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control at the end of the movement (see section 2.4.2.2, "Status Interrupt Endpoint"). The end of movement can be due to physical device limits, or due to an explicit request by the host to stop the movement. If the end of movement is due to physical device limits (such as a limit in range of motion), a Control Change interrupt shall be generated for this Relative Control. If there is no limit in range of motion, a Control Change interrupt is not required.
When the Auto-Focus Mode control is enabled, attempts to programmatically set this control shall result in a protocol STALL and an error code of **bRequestErrorCode** = "Wrong state".

Table 4-15 Focus (Relative) Control

Control Selector		CT_FOCUS_RELATIVE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF, GET_MIN, GET_MAX, GET_RES		
wLength		2		
Offset	Field	Size	Value	Description
0	**bFocusRelative**	1	Signed number	The setting for the attribute of the addressed Focus (Relative) Control: 0: Stop 1: Focus Near direction 0xFF: Focus Infinite direction
1	**bSpeed**	1	Number	Speed for the control change

4.2.2.1.8 Focus, Simple Range

The Focus, Simple Control setting determines the absolute focus of the lens at a very granular level: Macro, People, and Scene. This control may only be used when the camera is in manual or auto focus mode. This control must accept the GET_DEF request and return its default value. When the Auto-Focus Mode control is enabled, attempts to programmatically set this control shall result in a protocol STALL and an error code of **bRequestErrorCode** = "Wrong state".

Table 4-16 Focus, Simple Range Control

Control Selector		CT_FOCUS_SIMPLE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bFocus**	1	Number	0x00 – full range 0x01 – macro. Less than 0.3 meters. 0x02 – people. 0.3m to 3m 0x03 – scene. 3m to infinity 0x04 to 0xFF - reserved

4.2.2.1.9 Focus, Auto Control

The Focus, Auto Control setting determines whether the device will provide automatic adjustment of the Focus Absolute and/or Relative Controls. A value of 1 indicates that automatic adjustment is enabled. Attempts to programmatically set the related controls shall result in a protocol STALL and an error code **bRequestErrorCode** = "Wrong state". This control must accept the GET_DEF request and return its default value.

Table 4-17 Focus, Auto Control

Control Selector		CT_FOCUS_AUTO_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bFocusAuto**	1	Boolean	The setting for the attribute of the addressed Focus Auto control.

4.2.2.1.10 Iris (Absolute) Control

The Iris (Absolute) Control is used to specify the camera's aperture setting. This value is expressed in units of f_{stop} * 100. The default value is implementation-specific.
This control will not accept SET requests when the Auto-Exposure Mode control is in Auto mode or Shutter Priority mode, and the control pipe shall indicate a stall in this case. This control must accept the GET_DEF request and return its default value.

Table 4-18 Iris (Absolute) Control

Control Selector		CT_IRIS_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description
0	**wIrisAbsolute**	2	Number	The setting for the attribute of the

				addressed Iris (Absolute) Control.

4.2.2.1.11 Iris (Relative) Control

The Iris (Relative) Control is used to specify the camera's aperture setting. This value is a signed integer and indicates the number of steps to open or close the iris. A value of 1 indicates that the iris is opened 1 step further. A value of 0xFF indicates that the iris is closed 1 step further. This step of iris is implementation specific. A value of zero (0) indicates that the iris is set to the default value for the implementation. The default value is implementation specific. This control will not accept SET requests when the Auto-Exposure Mode control is in Auto mode or Shutter Priority mode, and the control pipe shall indicate a stall in this case.

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control (see section 2.4.2.2, “Status Interrupt Endpoint”).

Table 4-19 Iris (Relative) Control

Control Selector		CT_IRIS_RELATIVE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bIrisRelative**	1	Number	The setting for the attribute of the addressed Iris (Relative) Control: 0: Default 1: Iris is opened by 1 step. 0xFF: Iris is closed by 1 step.

4.2.2.1.12 Zoom (Absolute) Control

The Zoom (Absolute) Control is used to specify or determine the Objective lens focal length. This control is used in combination with the **wObjectiveFocalLengthMin** and **wObjectiveFocalLengthMax** fields in the Camera Terminal descriptor to describe and control the Objective lens focal length of the device. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent. This control must accept the GET_DEF request and return its default value.

Table 4-20 Zoom (Absolute) Control

Control Selector		CT_ZOOM_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description

0	**wObjectiveFocalLength**	2	Number	The value of Z_{cur}

4.2.2.1.13 Zoom (Relative) Control

The Zoom (Relative) Control is used to specify the zoom focal length relatively as powered zoom.

The **bZoom** field indicates whether the zoom lens group is stopped or the direction of the zoom lens. A value of 1 indicates that the zoom lens is moved towards the telephoto direction. A value of zero indicates that the zoom lens is stopped, and a value of 0xFF indicates that the zoom lens is moved towards the wide-angle direction. The GET_MIN, GET_MAX, GET_RES and GET_DEF requests will return zero for this field.

The **bDigitalZoom** field specifies whether digital zoom is enabled or disabled. If the device only supports digital zoom, this field would be ignored. The GET_DEF request will return the default value for this field. The GET_MIN, GET_MAX and GET_RES requests will return zero for this field.

The **bSpeed** field indicates the speed of the control change. A low number indicates a slow speed and a high number indicates a higher speed. The GET_MIN, GET_MAX and GET_RES requests are used to retrieve the range and resolution for this field. The GET_DEF request is used to retrieve the default value for this field. If the control does not support speed control, it will return the value 1 in this field for all these requests.

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control at the end of the movement (see section 2.4.2.2, “Status Interrupt Endpoint”). The end of movement can be due to physical device limits, or due to an explicit request by the host to stop the movement. If the end of movement is due to physical device limits (such as a limit in range of motion), a Control Change interrupt shall be generated for this Relative Control.

Table 4-21 Zoom (Relative) Control

Control Selector		CT_ZOOM_RELATIVE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF, GET_MIN, GET_MAX, GET_RES		
wLength		3		
Offset	Field	Size	Value	Description
0	**bZoom**	1	Signed number	The setting for the attribute of the addressed Zoom Control: 0: Stop 1: moving to telephoto direction 0xFF: moving to wide-angle direction
1	**bDigitalZoom**	1	Boolean	0: Digital Zoom OFF 1: Digital Zoom On

2	**bSpeed**	1	Number	Speed for the control change

4.2.2.1.14 PanTilt (Absolute) Control

The PanTilt (Absolute) Control is used to specify the pan and tilt settings.
The d**wPanAbsolute** is used to specify the pan setting in arc second units. 1 arc second is 1/3600 of a degree. Values range from –180*3600 arc second to +180*3600 arc second, or a subset thereof, with the default set to zero. Positive values are clockwise from the origin (the camera rotates clockwise when viewed from above), and negative values are counterclockwise from the origin. This control must accept the GET_DEF request and return its default value.
The **dwTiltAbsolute** Control is used to specify the tilt setting in arc second units. 1 arc second is 1/3600 of a degree. Values range from –180*3600 arc second to +180*3600 arc second, or a subset thereof, with the default set to zero. Positive values point the imaging plane up, and negative values point the imaging plane down. This control must accept the GET_DEF request and return its default value.

Table 4-22 PanTilt (Absolute) Control

Control Selector		CT_PANTILT_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		8		
Offset	Field	Size	Value	Description
0	**dwPanAbsolute**	4	Signed number	The setting for the attribute of the addressed Pan (Absolute) Control.
4	**dwTiltAbsolute**	4	Signed number	The setting for the attribute of the addressed Tilt (Absolute) Control.

4.2.2.1.15 PanTilt (Relative) Control

The PanTilt (Relative) Control is used to specify the pan and tilt direction to move.

The **bPanRelative** field is used to specify the pan direction to move. A value of 0 indicates to stop the pan, a value of 1 indicates to start moving clockwise direction, and a value of 0xFF indicates to start moving counterclockwise direction. The GET_DEF, GET_MIN, GET_MAX and GET_RES requests will return zero for this field.

The **bPanSpeed** field is used to specify the speed of the movement for the Pan direction. A low number indicates a slow speed and a high number indicates a higher speed. The GET_MIN, GET_MAX and GET_RES requests are used to retrieve the range and resolution for this field. The GET_DEF request is used to retrieve the default value for this field. If the control does not support speed control for the Pan control, it will return the value 1 in this field for all these requests.

The **bTiltRelative** field is used to specify the tilt direction to move. A value of zero indicates to stop the tilt, a value of 1 indicates that the camera point the imaging plane up, and a value of 0xFF indicates that the camera point the imaging plane down. The GET_DEF, GET_MIN, GET_MAX and GET_RES requests will return zero for this field.

The **bTiltSpeed** field is used to specify the speed of the movement for the Tilt direction. A low number indicates a slow speed and a high number indicates a higher speed. The GET_MIN, GET_MAX and GET_RES requests are used to retrieve the range and resolution for this field. The GET_DEF request is used to retrieve the default value for this field. If the control does not support speed control for the Tilt control, it will return the value 1 in this field for all these requests.

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control at the end of the movement (see section 2.4.2.2, "Status Interrupt Endpoint"). The end of movement can be due to physical device limits, or due to an explicit request by the host to stop the movement. If the end of movement is due to physical device limits (such as a limit in range of motion), a Control Change interrupt shall be generated for this Relative Control. If there is no limit in range of motion, a Control Change interrupt is not required.

Table 4-23 PanTilt (Relative) Control

Control Selector		CT_PANTILT_RELATIVE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF, GET_MIN, GET_MAX, GET_RES		
wLength		4		
Offset	Field	Size	Value	Description
0	**bPanRelative**	1	Signed number	The setting for the attribute of the addressed Pan (Relative) Control: 0: Stop 1: moving to clockwise direction 0xFF: moving to counter clockwise direction
1	**bPanSpeed**	1	Number	Speed of the Pan movement
2	**bTiltRelative**	1	Signed number	The setting for the attribute of the addressed Tilt (Relative) Control: 0: Stop 1: point the imaging plane up 0xFF: point the imaging plane down
3	**bTiltSpeed**	1	Number	Speed for the Tilt movement

4.2.2.1.16 Roll (Absolute) Control

The Roll (Absolute) Control is used to specify the roll setting in degrees. Values range from –180 to +180, or a subset thereof, with the default being set to zero. Positive values cause a clockwise rotation of the camera along the image viewing axis, and negative values cause a counterclockwise rotation of the camera. This control must accept the GET_DEF request and return its default value.

Table 4-24 Roll (Absolute) Control

Control Selector		CT_ROLL_ABSOLUTE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description
0	**wRollAbsolute**	2	Signed number	The setting for the attribute of the addressed Roll (Absolute) Control.

4.2.2.1.17 Roll (Relative) Control

The Roll (Relative) Control is used to specify the roll direction to move.

The **bRollRelative** field is used to specify the roll direction to move. A value of 0 indicates to stop the roll, a value of 1 indicates to start moving in a clockwise rotation of the camera along the image viewing axis, and a value of 0xFF indicates to start moving in a counterclockwise direction. The GET_DEF, GET_MIN, GET_MAX and GET_RES requests will return zero for this field.

The **bSpeed** is used to specify the speed of the roll movement. A low number indicates a slow speed and a high number indicates a higher speed. The GET_MIN, GET_MAX and GET_RES requests are used to retrieve the range and resolution for this field. The GET_DEF request is used to retrieve the default value for this field. If the control does not support speed control, it will return the value 1 in this field for all these requests.

If both Relative and Absolute Controls are supported, a SET_CUR to the Relative Control with a value other than 0x00 shall result in a Control Change interrupt for the Absolute Control at the end of the movement (see section 2.4.2.2, “Status Interrupt Endpoint”). The end of movement can be due to physical device limits, or due to an explicit request by the host to stop the movement. If the end of movement is due to physical device limits (such as a limit in range of motion), a Control Change interrupt shall be generated for this Relative Control. If there is no limit in range of motion, a Control Change interrupt is not required.

Table 4-25 Roll (Relative) Control

Control Selector	CT_ROLL_RELATIVE_CONTROL
Mandatory Requests	SET_CUR, GET_CUR, GET_INFO, GET_DEF, GET_MIN,

		GET_MAX, GET_RES		
wLength		2		
Offset	Field	Size	Value	Description
0	**bRollRelative**	1	Signed number	The setting for the attribute of the addressed Roll (Relative) Control: 0: Stop 1: moving clockwise rotation 0xFF: moving counter clockwise rotation
1	**bSpeed**	1	Number	Speed for the Roll movement

4.2.2.1.18 Privacy Control

The Privacy Control setting is used to prevent video from being acquired by the camera sensor. A value of 0 indicates that the camera sensor is able to capture video images, and a value of 1 indicates that the camera sensor is prevented from capturing video images.
This control shall be reported as an AutoUpdate control.

Table 4-26 Privacy Shutter Control

Control Selector		CT_PRIVACY_CONTROL		
Mandatory Requests		GET_CUR, GET_INFO		
Optional Requests		SET_CUR		
wLength		1		
Offset	Field	Size	Value	Description
0	**bPrivacy**	1	Boolean	The setting for the attribute of the addressed Privacy Control: 0: Open 1: Close

4.2.2.1.19 Digital Window Control

The windowing API is based on “pixel” coordinates where each row and column of pixels on the sensor can be referred to by integers between zero - (height-1) and zero- (width-1). The point at 0,0 is the top, left of the coordinate system and (height-1), (width-1) is the bottom, right of the coordinates system.

Table 4- 27 Digital Window Control

Control Selector		CT_DIGITAL_WINDOW_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF		
wLength		12		
Offset	Field	Size	Value	Description
0	**wWindow_Top**	2	Number	Top coordinate In global sensor coordinates (pixels)
2	**wWindow_Left**	2	Number	Left coordinate In global sensor coordinates (pixels)

4	**wWindow_Bottom**	2	Number	Bottom coordinate In global sensor coordinates (pixels)
6	**wWindow_Right**	2	Number	Right coordinate In global sensor coordinates (pixels)
8	**wNumSteps**	2	Number	Number of steps to move from current window to window specified by rectangle above.
10	**bmNumStepsUnits**	2	Bitmap	Units of number of steps specified above. D0: video frames D1: milliseconds D2-D15: Reserved

CT_DIGITAL_WINDOW_CONTROL is used to specify a target window for viewing, and the number of steps to take in moving from the current window to the new window the control specifies. To prevent invalid windows being specified:

wWindow_Bottom ≥ wWindow_Top

and

wWindow_Right ≥ wWindow_Left

GET_MAX should return the sensor size as well as maximum number of supported steps in the units indicated by **bmNumStepsUnits**. If the **bmNumStepsUnits** has not been set, the default value should be used. GET_CUR returns the current coordinates of the digitial window used for capture. If the device is moving between settings (e.g. wNumSteps > 1), GET_CUR references the digital window of the current step.

4.2.2.1.20 Digital Region of Interest (ROI) Control

The rectangle specified by CT_REGION_OF_INTEREST_CONTROL will be in global sensor coordinates. The units are in Pixels and independent of the field of view. They are not impacted by any cropping or scaling that is currently being used. The ROI must be within the current Digital Window as specified by the CT_WINDOW control.

The **bmAutoControls** bitmask determines which, if any, on board features should track to the region of interest. To detect if a device supports a particular Auto Control, use GET_MAX which returns a mask indicating all supported Auto Controls.

Table 4- 28 Region of Interest Control

Control Selector		CT_REGION_OF_INTEREST_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF		
wLength		10		
Offset	Field	Size	Value	Description
0	**wROI_Top**	2	Number	Top coordinate In global sensor coordinates (pixels)
2	**wROI_Left**	2	Number	Left coordinate

				In global sensor coordinates (pixels)
4	**wROI_Bottom**	2	Number	Bottom coordinate In global sensor coordinates (pixels)
6	**wROI_Right**	2	Number	Right coordinate In global sensor coordinates (pixels)
8	**bmAutoControls**	2	Bitmap	D0: Auto Exposure D1: Auto Iris D2: Auto White Balance D3: Auto Focus D4: Auto Face Detect D5: Auto Detect and Track D6: Image Stabilization D7: Higher Quality D8 – D15: Reserved, set to zero

GET_MAX shall return the current Window as specified by CT_DIGITAL_WINDOW_CONTROL.

GET_CUR returns the current Region of Interest (RoI) being employed by the device. This RoI should be the same as specified in most recent SET_CUR except in the case where the 'Auto Detect and Track' and/or 'Image Stabilization' bit have been set.

4.2.2.2 Selector Unit Control Requests

These requests are used to set or read an attribute of a Selector Control inside a Selector Unit of the video function.

A Selector Unit represents a video stream source selector. The valid range for the CUR, MIN, and MAX attributes is from one up to the number of Input Pins of the Selector Unit. This value can be found in the **bNrInPins** field of the Selector Unit descriptor. The RES attribute can only have a value of one.

Table 4-29 Selector Unit Control Requests

Control Selector		SU_INPUT_SELECT_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX,GET_RES, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bSelector**	1	Number	The setting for the attribute of the Selector Control.

4.2.2.3 Processing Unit Control Requests

These requests are used to set or read an attribute of a video Control inside a Processing Unit of the video function.

The following paragraphs present a detailed description of all possible Controls a Processing Unit can incorporate. For each Control, the layout of the parameter block together with the appropriate Control Selector is listed for all forms of the Get/Set Processing Unit Control request. All values are interpreted as unsigned unless otherwise specified.

4.2.2.3.1 Backlight Compensation Control

The Backlight Compensation Control is used to specify the backlight compensation. A value of zero indicates that the backlight compensation is disabled. A non-zero value indicates that the backlight compensation is enabled. The device may support a range of values, or simply a binary switch. If a range is supported, a low number indicates the least amount of backlight compensation. The default value is implementation-specific, but enabling backlight compensation is recommended. This control must accept the GET_DEF request and return its default value.

Table 4-30 Backlight Compensation Control

Control Selector		PU_BACKLIGHT_COMPENSATION_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wBacklightCompensation**	2	Number	The setting for the attribute of the addressed Backlight Compensation control.

4.2.2.3.2 Brightness Control

This is used to specify the brightness. This is a relative value where increasing values indicate increasing brightness. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent. This control must accept the GET_DEF request and return its default value.

Table 4-31 Brightness Control

Control Selector		PU_BRIGHTNESS_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wBrightness**	2	Signed number	The setting for the attribute of the addressed Brightness control.

4.2.2.3.3 Contrast Control

This is used to specify the contrast value. This is a relative value where increasing values indicate increasing contrast. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent. This control must accept the GET_DEF request and return its default value.

Table 4-32 Contrast Control

Control Selector	PU_CONTRAST_CONTROL

Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wContrast**	2	Number	The setting for the attribute of the addressed Contrast control.

4.2.2.3.4 Contrast, Auto Control

The Contrast Auto Control setting determines whether the device will provide automatic adjustment of the related control. A value of 1 indicates that automatic adjustment is enabled. Attempts to set the related Contrast control shall result in a protocol STALL and an error code of bRequestErrorCode = 'Wrong state'. This control must accept the GET_DEF request and return its default value.

Table 4-33 Contrast, Auto Control

Control Selector		PU_CONTRAST_AUTO_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bContrastAuto**	1	Number	The setting for the attribute of the addressed Contrast, Auto control.

4.2.2.3.5 Gain Control

This is used to specify the gain setting. This is a relative value where increasing values indicate increasing gain. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent. This control must accept the GET_DEF request and return its default value.

Table 4-34 Gain Control

Control Selector		PU_GAIN_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wGain**	2	Number	The setting for the attribute of the addressed Gain control.

4.2.2.3.6 Power Line Frequency Control

This control allows the host software to specify the local power line frequency, in order for the device to properly implement anti-flicker processing, if supported. The default is implementation-specific. This control must accept the GET_DEF request and return its default value.

Table 4-35 Power Line Frequency Control

Control Selector		PU_POWER_LINE_FREQUENCY_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bPowerLineFrequency**	1	Number	The setting for the attribute of the addressed Power Line Frequency control: 0: Disabled 1: 50 Hz 2: 60 Hz 3: Auto

4.2.2.3.7 Hue Control

This is used to specify the hue setting. The value of the hue setting is expressed in degrees multiplied by 100. The required range must be a subset of -18000 to 18000 (-180 to +180 degrees). The default value must be zero. This control must accept the GET_DEF request and return its default value.

Table 4-36 Hue Control

Control Selector		PU_HUE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description
0	**wHue**	2	Signed number	The setting for the attribute of the addressed Hue control.

4.2.2.3.8 Hue, Auto Control

The Hue Auto Control setting determines whether the device will provide automatic adjustment of the related control. A value of 1 indicates that automatic adjustment is enabled. Attempts to programmatically set the related control shall result in a protocol STALL with an error code of **bRequestErrorCode** = “Wrong state”. This control must accept the GET_DEF request and return its default value.

Table 4-37 Hue, Auto Control

Control Selector		PU_HUE_AUTO_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bHueAuto**	1	Number	The setting for the attribute of the addressed Hue, Auto control.

4.2.2.3.9 Saturation Control

This is used to specify the saturation setting. This is a relative value where increasing values indicate increasing saturation. A Saturation value of 0 indicates grayscale. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation-dependent. This control must accept the GET_DEF request and return its default value.

Table 4-38 Saturation Control

Control Selector		PU_SATURATION_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wSaturation**	2	Number	The setting for the attribute of the addressed Saturation control.

4.2.2.3.10 Sharpness Control

This is used to specify the sharpness setting. This is a relative value where increasing values indicate increasing sharpness, and the MIN value always implies "no sharpness processing", where the device will not process the video image to sharpen edges. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation-dependent. This control must accept the GET_DEF request and return its default value.

Table 4-39 Sharpness Control

Control Selector		PU_SHARPNESS_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wSharpness**	2	Number	The setting for the attribute of the addressed Sharpness control.

4.2.2.3.11 Gamma Control

This is used to specify the gamma setting. The value of the gamma setting is expressed in gamma multiplied by 100. The required range must be a subset of 1 to 500, and the default values are typically 100 (gamma = 1) or 220 (gamma = 2.2). This control must accept the GET_DEF request and return its default value.

Table 4-40 Gamma Control

Control Selector		PU_GAMMA_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wGamma**	2	Number	The setting for the attribute of the addressed Gamma control.

4.2.2.3.12 White Balance Temperature Control

This is used to specify the white balance setting as a color temperature in degrees Kelvin. This is offered as an alternative to the White Balance Component control. Minimum range should be 2800 (incandescent) to 6500 (daylight) for webcams and dual-mode cameras. The supported range and default value for white balance temperature is implementation-dependent. This control must accept the GET_DEF request and return its default value.

Table 4-41 White Balance Temperature Control

Control Selector		PU_WHITE_BALANCE_TEMPERATURE_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		2		
Offset	Field	Size	Value	Description
0	**wWhiteBalanceTemperature**	2	Number	The setting for the attribute of the addressed White Balance Temperature control.

4.2.2.3.13 White Balance Temperature, Auto Control

The White Balance Temperature Auto Control setting determines whether the device will provide automatic adjustment of the related control. A value of 1 indicates that automatic adjustment is enabled. Attempts to programmatically set the related control shall result in a protocol STALL with an error code of **bRequestErrorCode** = “Wrong state”. This control must accept the GET_DEF request and return its default value.

Table 4-42 White Balance Temperature, Auto Control

Control Selector	PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL
Mandatory Requests	SET_CUR, GET_CUR, GET_INFO, GET_DEF
wLength	1

Offset	Field	Size	Value	Description
0	**bWhiteBalanceTemperatureAuto**	1	Number	The setting for the attribute of the addressed White Balance Temperature, Auto control.

4.2.2.3.14 White Balance Component Control

This is used to specify the white balance setting as Blue and Red values for video formats. This is offered as an alternative to the White Balance Temperature control. The supported range and default value for white balance components is implementation-dependent. The device shall interpret the controls as blue and red pairs. This control must accept the GET_DEF request and return its default value.

Table 4-43 White Balance Component Control

Control Selector		PU_WHITE_BALANCE_COMPONENT_CONTROL		
Mandatory Requests		GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
Optional Requests		SET_CUR		
wLength		4		
Offset	Field	Size	Value	Description
0	**wWhiteBalanceBlue**	2	Number	The setting for the blue component of the addressed White Balance Component control.
1	**wWhiteBalanceRed**	2	Number	The setting for the red component of the addressed White Balance Component control.

4.2.2.3.15 White Balance Component, Auto Control

The White Balance Component Auto Control setting determines whether the device will provide automatic adjustment of the related control. A value of 1 indicates that automatic adjustment is enabled. Attempts to programmatically set the related control shall result in a protocol STALL with an error code of **bRequestErrorCode** = "Wrong state". This control must accept the GET_DEF request and return its default value.

Table 4-44 White Balance Component, Auto Control

Control Selector		PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bWhiteBalanceComponentAuto**	1	Number	The setting for the attribute of the addressed White Balance Component, Auto control.

4.2.2.3.16 Digital Multiplier Control

Use of this control has been deprecated and will be removed in the next revision of this specification.

This is used to specify the amount of Digital Zoom applied to the optical image. This is the position within the range of possible values of multiplier *m*, allowing the multiplier resolution to be described by the device implementation. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent. If the Digital Multiplier Limit Control is supported, the MIN and MAX values shall match the MIN and MAX values of the Digital Multiplier Control. The Digital Multiplier Limit Control allows either the Device or the Host to establish a temporary upper limit for the Z'_{cur} value, thus reducing dynamically the range of the Digital Multiplier Control. If Digital Multiplier Limit is used to decrease the Limit below the current Z'_{cur} value, the Z'_{cur} value will be adjusted to match the new limit and the Digital Multiplier Control shall send a Control Change Event to notify the host of the adjustment.

Table 4-45 Digital Multiplier Control

Control Selector		PU_DIGITAL_MULTIPLIER_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wMultiplierStep**	2	Number	The value Z'_{cur}

4.2.2.3.17 Digital Multiplier Limit Control

This is used to specify an upper limit for the amount of Digital Zoom applied to the optical image. This is the maximum position within the range of possible values of multiplier m. The MIN and MAX values are sufficient to imply the resolution, so the RES value must always be 1. The MIN, MAX and default values are implementation dependent.

Table 4-46 Digital Multiplier Limit Control

Control Selector		PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wMultiplierLimit**	2	Number	A value specifying the upper bound for Z'_{cur}

4.2.2.3.18 Analog Video Standard Control

This is used to report the current Video Standard of the stream captured by the Processing Unit.

Table 4-47 Analog Video Standard Control

Control Selector		PU_ANALOG_VIDEO_STANDARD_CONTROL		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bVideoStandard**	1	Number	The Analog Video Standard of the input video signal. 0: None 1: NTSC – 525/60 2: PAL – 625/50 3: SECAM – 625/50 4: NTSC – 625/50 5: PAL – 525/60 6-255: Reserved. Do not use.

4.2.2.3.19 Analog Video Lock Status Control

This is used to report whether the video decoder has achieved horizontal lock of the analog input signal. If the decoder is locked, it is assumed that a valid video stream is being generated. This control is to be supported only for analog video decoder functionality.

Table 4-48 Analog Video Lock Status Control

Control Selector	PU_ANALOG_LOCK_STATUS_CONTROL
Mandatory Requests	GET_CUR, GET_INFO
wLength	1

Offset	Field	Size	Value	Description
0	**bStatus**	1	Number	0: Video Decoder is locked. 1: Video Decoder is not locked. 2-255: Reserved. Do not use.

4.2.2.4 Encoding Units

Encoding Unit requests are used to set or read the attributes of a video Control inside an Encoding Unit of the video function. The following paragraphs present a detailed description of all possible Controls an Encoding Unit can incorporate. For each Control, the Control data structure together with the appropriate Control Selector is listed for all forms of the Get/Set Encoding Unit Control requests. All values are interpreted as unsigned unless otherwise specified.

Encoding Units, when considered as a whole, define the Encoding Unit State of the device. The Encoding Unit State is per Encoding Unit and is established with a successful VS_COMMIT_CONTROL(SET_CUR) to the associated VSInterface. The Encoding Unit State establishes values for all fields in all supported Encoding Units.

Encoding Units can be called before streaming starts as well as during streaming. When called before streaming, Encoding Units must not be called until a successful VS_COMMIT_CONTROL(SET_CUR) request has been accepted by the device. Encoding Units should use the Commit state, as returned by VS_COMMIT_CONTROL(GET_CUR), to understand the boundaries of what is currently possible. A request issued to an Encoding Unit applies to the layers/views specified by **wLayerOrViewID**. When the stream consists of a single layer, the Encoding Unit is applied to the stream given by the stream_id subfield in **wLayerOrViewID**, all other subfields in **wLayerOrViewID** shall be ignored by the device.

4.2.2.4.1 Encoding Units Operational Model

Device support of Encoding Unit controls is optional. Encoding Units can be used to configure the codec before streaming or while streaming. All requests to Encoding Unit controls before a successful VS_COMMIT_CONTROL(SET_CUR), except GET_LEN, shall result in protocol STALL. Use VS_COMMIT_CONTROL(GET_CUR) to get the Commit state at any time. The device shall use the Commit state together with the current Encoding Unit state to validate any subsequent Encoding Unit control request. When a GET_CUR request is issued to the EU control before any SET_CUR, the GET_CUR state shall be the same as the GET_DEF state. Upon a successful VS_COMMIT_CONTROL(SET_CUR), the device must have a valid GET_DEF state for each Encoding Unit control supported. Upon a successful VS_COMMIT_CONTROL(SET_CUR), there may be fields in the Encoding Unit controls data structures that have not yet been initialized such as parameters for rate control or resolutions for a simulcast payload. For these unspecified fields, the device must establish default values such that the device can successfully stream with no further configuration from the host. The rules the device must follow to establish defaults are defined in Table 4-49 . After a successful VS_COMMIT_CONTROL(SET_CUR) request, the Encoding Unit state can only be changed by either issuing a successful SET_CUR request to an Encoding Unit Control, or by setting the device to a new Commit state that is different from the previous Commit state. When the device enters a Commit state that is different from the previous Commit state, the device must update the Encoding Unit state using the rules defined in Table 4-49 together with the new Commit data structure.

Table 4-49 Default Encoding Unit State after VS_COMMIT_CONTROL(SET_CUR) Request.

EU Control	Method of Configuration	
	First stream (stream_id = 0)	**Additional streams in simulcast payload (stream_id = 1…n)**
wLayerOrViewID	0	0
Width, height	COMMIT (bFrameIndex)	Device determined. Must be less than or equal to resolution of first stream.
Minimum frame interval	COMMIT (dwFrameInterval)	Device determined.Must be greater than or equal to minimum frame interval of first stream.
Average bit rate	Device determined	Device determined
CPB size	Device determined	Device determined
Rate control mode	COMMIT (bmRateControlModes)	COMMIT (bmRateControlModes)
Quantization parameter	Device determined	Device determined
Priority ID	COMMIT (bUsage & bmLayoutPerStream)	COMMIT (bUsage & bmLayoutPerStream)
Slice mode	Device determined	Device determined
Profile	COMMIT (bFrameIndex)	Same as first stream
bmSettings	COMMIT (bFrameIndex)	Same as first stream.
Peak bit rate	Peak bit rate in Rate Control Parameter EU.	Peak bit rate in Rate Control Parameter EU.
Level IDC	COMMIT (bFrameIndex)	Device determined. Must be less than or equal to LevelIDC of first stream.
bNumHostControlLTRBuffers	Device determined	Device determined
bLTRMode	Device determined	Device determined
bmValidLTRs	Device determined	Device determined
bmErrorResiliencyFeatures	Device determined	Device determined

- **COMMIT**(*x*): Indicates that the parameter is given by the VS_COMMIT_CONTROL(GET_CUR) structure field indicated within parentheses.

In this specification, all Encoding Unit Controls except EU_SELECT_LAYER_CONTROL- and EU_START_OR_STOP_LAYER_CONTROL are encoder configuration controls. The host may issue SET_CUR requests to encoder configuration controls to configure encoder attributes for one or multiple layers.
Once a Commit state has been established, the host may issue one or more Encoder Configuration Control requests before or during streaming. From the perspective of this specification, each Encoder Configuration Control operation is atomic.

4.2.2.4.1.1 Device States for Probe, Commit, and Encoding Units

Figure 4-1 illustrates three high level device states and the USB requests that can trigger transitions between these device states when the requests succeed. USB requests not shown in the Figure below, such as GET_CUR, GET_MIN, GET_MAX, GET_DEF among others, do not trigger a state transition. State 0 represents the device in the USB Configured State (see Section 9.1.1.5 "Configured" of *USB Specification Revision 2.0* and *USB Specification Revision 3.0*). At this point, the Encoding Unit State in undefined. State 1 represents the active device state after a successful VS_COMMIT_CONTROL(SET_CUR). Additional configuration of the encoder through Encoding Units may occur before or during streaming. When in state 1 or 2, a VS_COMMIT_CONTROL(SET_CUR) equal to the current VS_COMMIT_CONTROL(GET_CUR) will have no effect on the device, that means that the Encoding Unit state will also remain the same. When in state 1 or 2, a VS_COMMIT_CONTROL(SET_CUR) different than the current VS_COMMIT_CONTROL(GET_CUR) will update both the Commit and the Encoding Unit State according to the new Commit state and the rules defined in Table 4-49; this transition is not shown in the state diagram.
Table 4-50 describes how Encoding Units interact with each these states. This description includes possible errors that may be logged by the device in response to USB Requests issued to Encoding Unit controls. Errors may be retrieved by the host using the Request Error Code Control (see section 4.2.1.2 "Request Error Code Control").

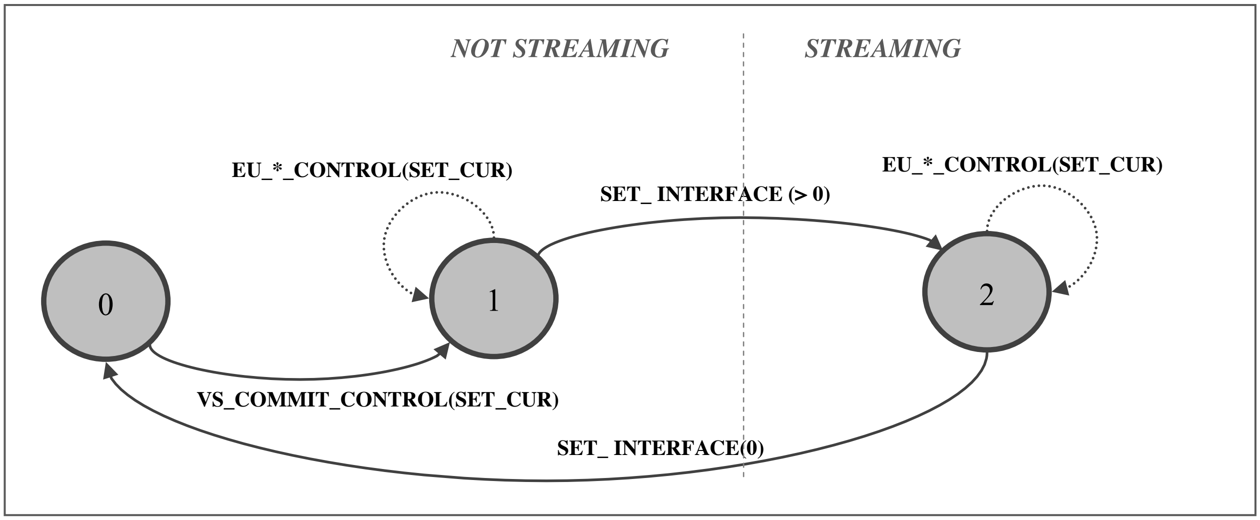


Figure 4-1 Device State Transitions Diagram

When in state 2, streaming, the only valid parameter value in a SET_INTERFACE request is alternate setting 0.

Table 4-50 Encoding Units, Devices States and Error Code Control Responses

State	Streaming State	**Encoding Unit State**	**Error Code Control Response to USB Requests issued to Encoding Unit Controls**
0	not streaming	Undefined, except GET_LEN	Protocol STALL with error code: • “Invalid Control” if EU Control is not supported. Else, • “Invalid Request” if USB request is not supported for this Control. Else, • “Wrong State” if EU Control is supported after initial VS_COMMIT_CONTROL(SET_CUR)
1	not streaming	defined by the default values as given in Table 4-49 together with the values set in a successful SET_CUR request to the Encoding Unit	Protocol STALL with error code: • “Invalid Control” if EU Control is not supported. Else, • “Invalid Request” if USB request is not supported for this Control. Else, • “Wrong State” if EU is supported only while streaming or if the active wLayerOrView is not valid. Else, • “Out of Range” if any of the input

			arguments to the EU is invalid
2	streaming	defined by the default values as given in Table 4-49 together with the values set in a successful SET_CUR request to the Encoding Unit	Protocol STALL with error code: • "Invalid Control" if EU Control is not supported. Else, • "Invalid Request" if USB request is not supported for this Control. Else, • "Wrong State" if EU Control is supported only before streaming or if the active wLayerOrView is not valid. Else, • "Out of Range" if any of the input arguments to the EU is invalid

4.2.2.4.1.2 Encoding Unit Requests

Table 4-51describes Encoding Unit request attributes. Each Encoding Unit supports one or more of these attributes as specified in sections 4.2.2.4.2 through 4.2.2.4.19.

Table 4-51 Encoding Unit Requests

Attribute	Description
GET_CUR	Returns the current values of all fields. Prior to the initial VS_COMMIT_CONTROL(SET_CUR), the GET_CUR state for all Encoding Units is undefined. After the the initial VS_COMMIT_CONTROL(SET_CUR) and prior to the initial SET_CUR request to the Encoding Unit control, the GET_CUR state must be the same as the GET_DEF state.
GET_MIN	Returns the minimum value for all fields.
GET_MAX	Returns the maximum value for all fields.
GET_RES	Returns the resolution value for all fields.
GET_DEF	Returns the default value for all fields.
GET_INFO	Returns the capabilities and status of the specified control
GET_LEN	Returns the data length of the specified control
SET_CUR	Sets the current Encoding Unit state. For Encoder Configuration Controls, SET_CUR is used for layer configuration and sets all the fields. This request shall stall in case an unsupported state or field is specified. See the Request Error Code Control section for the definition of specific request error codes.

4.2.2.4.1.3 Stream Negotiation Examples

The example below shows a successful configuration and streaming of a temporally encoded video stream.

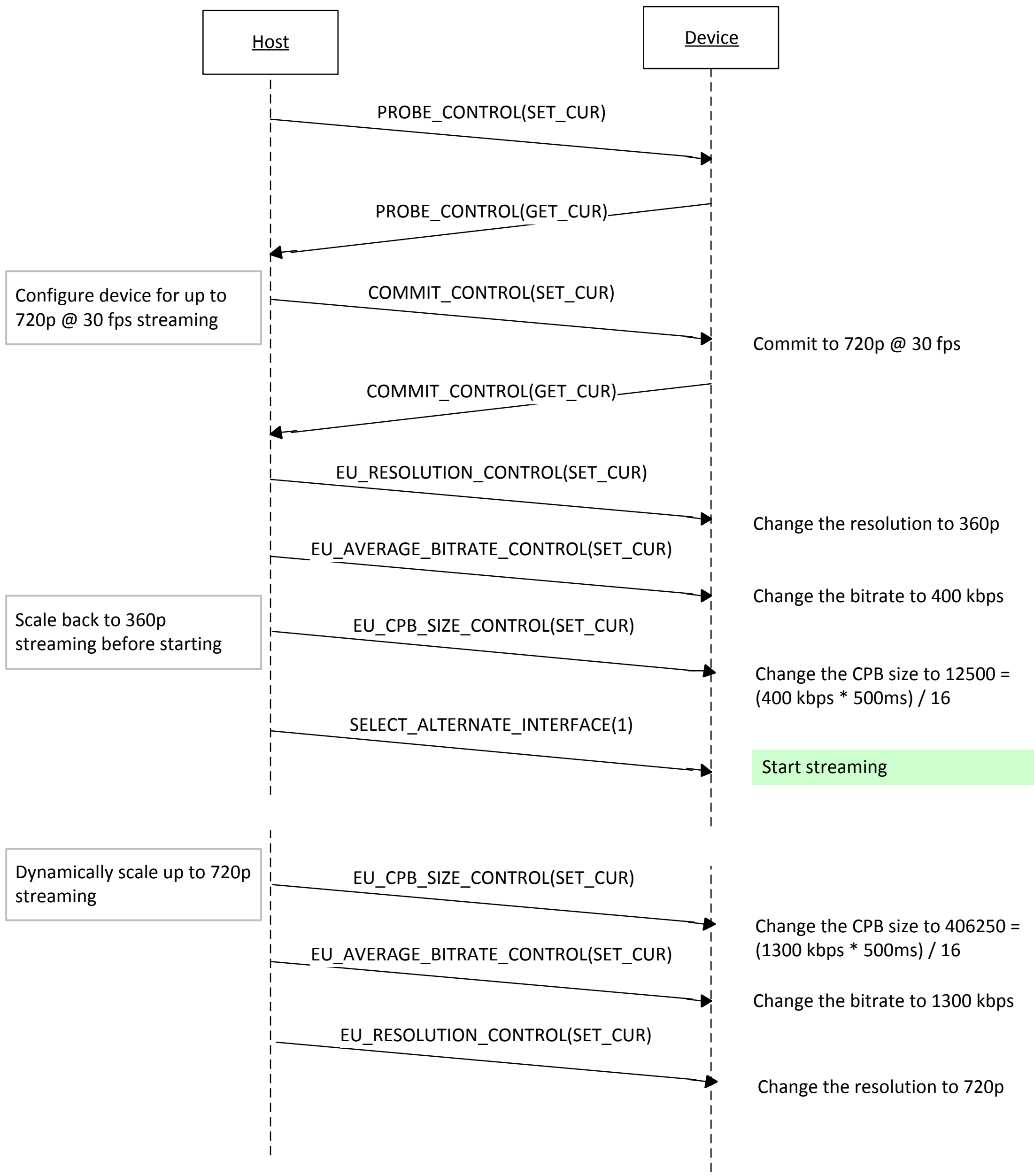


Figure 4-2 Successful USB Isochronous Negotiation Example including Dynamic Resolution Changes

Figure 4-2 illustrates stream configuration and successful USB isochronous bandwidth negotiation in a single stream encoded video scenario. In this example, during Probe/Commit the host negotiates a video stream to allow a resolution and frame rate up to 720p at 30 fps. Then, before streaming, the host configures the device to stream 360p at 30 fps, including the appropriate bitrate and CPB size. At this point the host starts the stream selecting alternate interface number 1. At a later time, while still streaming, the host increases the resolution to 720p. To do this, the host first increases the CPB size and then increases the bit rate. It is important to keep within the limits of all bitrate and buffer size limits when making these changes. Finally, the host changes the resolution to 720p.

The example below shows a failed Encoding Unit request to set the frame rate to a higher value than was negotiated.

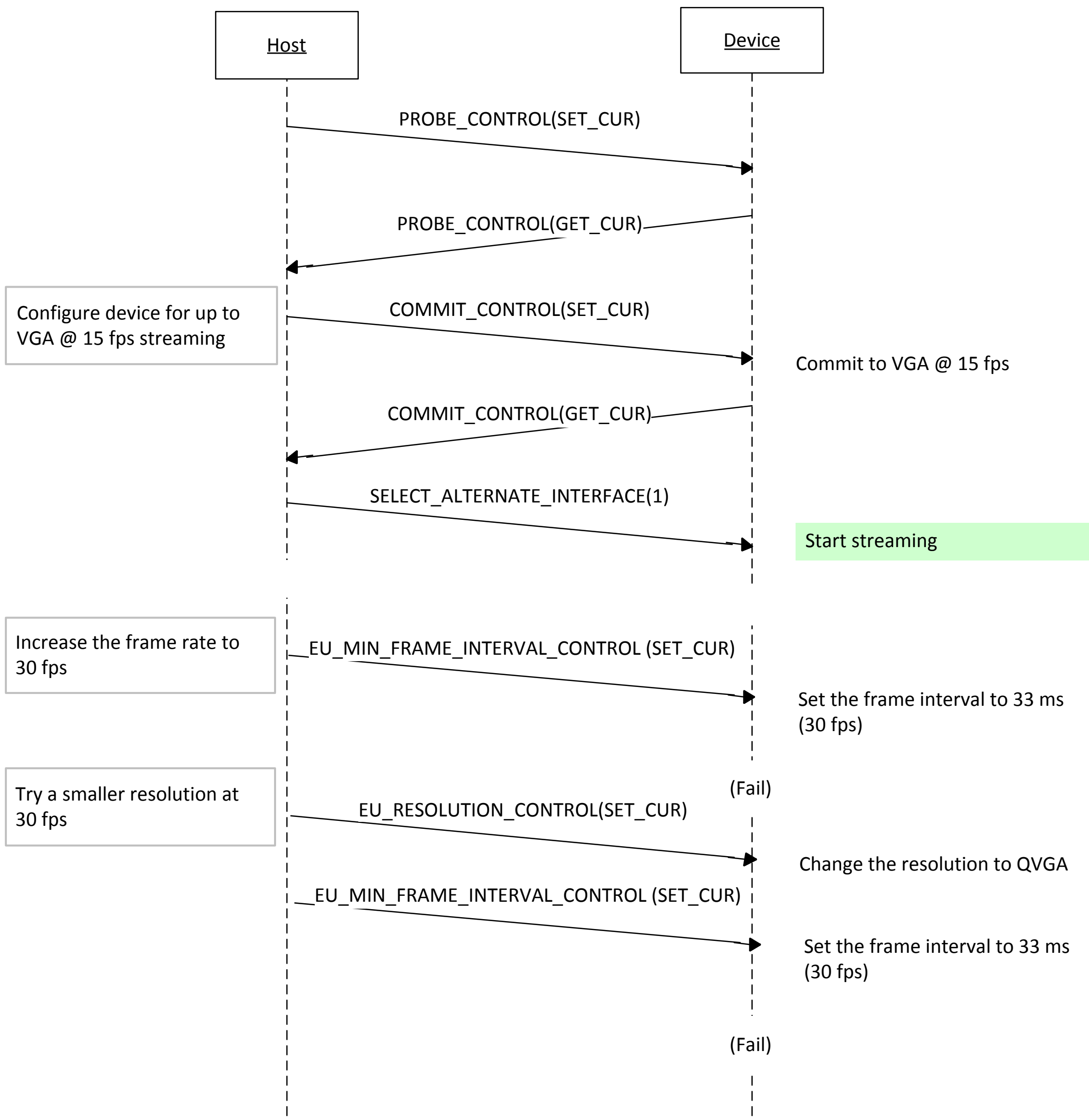


Figure 4-3 Failed Request to an Encoding Unit Control Example

Figure 4-3 demonstrates a failed Encoding Unit request. In this example, the host attempts to set the frame interval to a shorter duration than negotiated during Probe and Commit. After failure, the host attempts to negotiate a shorter frame interval by reducing the resolution first. This attempt also fails since the limits on frame interval are independent of the resolution or any other stream parameter. This example illustrates the importance of negotiating the expected maximum for each parameter (in this case minimum frame interval) as part of Probe and Commit.

4.2.2.4.1.4 Setting Resolution and Frame Interval

When changing resolution and / or minimum frame interval, whether during initialization or streaming, the following restrictions apply.

- The resolution must be equal to or less than the value established during Probe and Commit
- The frame interval must be equal to or greater than the frame interval established during Probe and Commit
- Care should be taken to avoid exceeding the negotiated bandwidth and buffer size limits when changing resolution, bitrate, or frame interval.

4.2.2.4.1.5 Wildcard Masks

Wildcard masks may be used in association with the **wLayerOrViewID** field of the Select Layer control. Use of a wildcard mask selects all layers whose type is determined by the associated sub-field(s), for example, using a wildcard mask in the temporal_id subfield results in the selection of all temporal layers. Wildcard masks must address one or more subfield. Any Encoding Unit called subsequent to a Select Layer request using wildcard masks must respect the resulting layer selection unless otherwise noted in the control. Note that using wildcards implies a transaction where the Request must succeed or fail completely, not partially. Except for the Select Layer control, GET_MIN, GET_MAX, GET_DEF, GET_RES and GET_CUR requests shall protocol STALL if **wLayerOrViewID** is SET to a wildcard mask. The example table below shows which bits are set in **wLayerOrViewID** to create a wildcard mask for the four subfields.

Table 4-52 Bit Layout of wLayerOrViewID for SVC Wildcard Masks

wLayerOrViewID bits	**reserved**			**stream_id**			**temporal_id**			**quality_id**				**dependency_id**		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
All Dependency Layers														1	1	1
All Quality Layers										1	1	1	1			
All Temporal Layers							1	1	1							
All Streams				1	1	1										

4.2.2.4.2 Select Layer Control

The Select Layer control is used to specify the active layer/view for subsequent EU control operations. This includes the stream_id for simulcast streams. The default value for **wLayerOrViewID** is 0. This value selects the base layer of the first stream. Multiple layers/views can be specified in **wLayerOrViewID** by using a wildcard mask for SET_CUR requests. For the Select Layer control wildcard masks are also allowed for GET_CUR requests.

Table 4-53 Select Layer Control

Control Selector		EU_SELECT_LAYER_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_LEN		
wLength		2		
Offset	Field	Size	Value	Description
0	**wLayerOrViewID**	2	Number	For multi-layer streams, a combination of dependency_id, quality_id, temporal_id and stream_id. Bits: 0-2: dependency_id 3-6: quality_id 7-9: temporal_id 10-12: stream_id 13-15: Reserved, set to 0 For multi-view streams, this value contains a combination of view_id, temporal_id, stream_id, and interface number. Bits: 0-6: view_id 7-9: temporal_id 10-12: stream_id 13-15: Reserved, set to 0 When the stream does not support layering or multiples view, bits 0-9 are zero.

The dependency_id, quality_id, temporal_id, and view_id of each layer of a stream are determined by the scaling capability mode negotiated in Probe and Commit. The stream_id parameter is used to differentiate between streams when simulcast of two or more streams is enabled. In the case of a single stream, stream_id is always zero. Each additional stream is given a unique stream_id by incrementing the stream_id by 1. If the stream supports simulcast transport and the stream_id does not exist, the device should protocol STALL with error code “Out of Range”. If **wLayerOrViewID** specifies a layer or view that is not defined by the **bmLayoutPerStream** established during VS_COMMIT the device shall protocol STALL with error code “Out of Range”. If the stream does not support simulcast transport and the stream_id is not zero, the device should ignore the stream_id and proceed with the Select Layer request for the first stream.

4.2.2.4.2.1 Sub-bitstream Definition

Several Encoding Units apply to sub-bitstreams instead of individual layers. The sub-bitstream is determined as follows. Let **wLayerOrViewID** be the result from a GET_CUR request issued to the EU_SELECT_LAYER_CONTROL control. The sub-bitstream is given by all the layers that meet all the following conditions:

- stream_id is equal to the stream_id indicated by **wLayerOrViewID**.
- temporal_id is less or equal than temporal_id indicated by **wLayerOrViewID**.
- dependency_id is less than dependency_id indicated by **wLayerOrViewID**, or dependency_id is equal to dependency_id indicated by **wLayerOrViewID** and quality_id is less or equal than quality_id indicated by **wLayerOrViewID**.

4.2.2.4.2.2 Multicast

This specification supports multicast, e.g. multiple encoded video streams from a single video function. The device must offer a separate Encoding Unit for each Video Streaming interface that delivers encoded video.

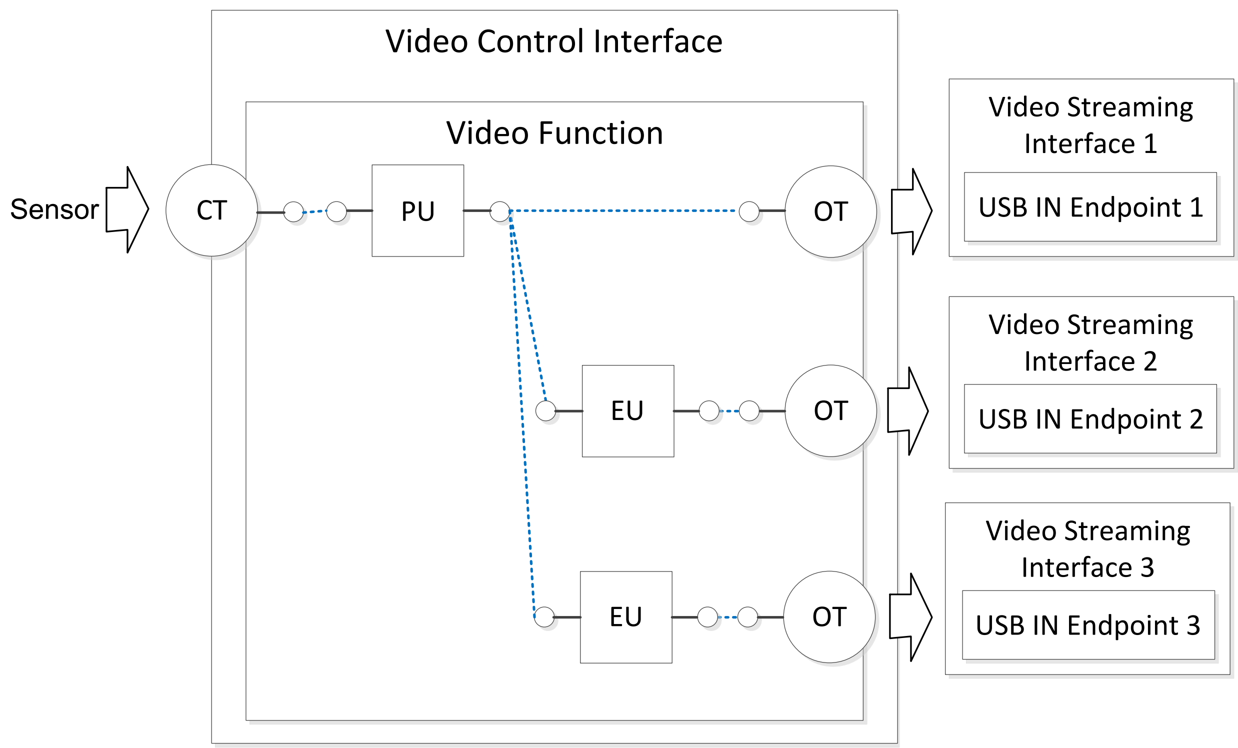


Figure 4-4 Example of Multicast Topology

By using the **bSourceID** of the Encoding Units, Output Terminals, and VS Interfaces, the device topology above can be understood by the host.

The device must support a complete Probe and Commit operation for each streaming interface, one at a time. The expected behavior is that Probe and Commit is complete on one interface before beginning with another. Additionally, if a device supports multicast, GET_MAX requests to the

Probe control must accurately reflect the capabilities of the device, including what is already been negotiated using VS_COMMIT_CONTROL on other streaming interfaces.

4.2.2.4.3 Video Resolution Control

The Video Resolution control is used to independently initialize the resolution of each layer in a multi-layer stream. The **wWidth** and **wHeight** fields are used to set the width and height of the decoded bitmap frame of a stream.
When using this control to change video resolution, the new resolution is restricted to ratios as described in the **bResolutionScaling** field in the video Format descriptor.
The bit rate limits described by **dwMinBitRate** and **dwMaxBitRate** may change when video resolution is changed. The application should use the GET_MIN and GET_MAX request on the Rate Control Parameter control to determine the new values for minimum bit rate and maximum bit rate.
If the stream does not support spatial scalability, then this control applies to the stream_id of the active **wLayerOrViewID** and the device shall ignore all other subfields in **wLayerOrViewID**.GET_RES applies to encoders that support arbitrary resolutions and shall reflect any limitations on the step size between resolutions.

Table 4-54 Video Resolution Control

Control Selector		EU_VIDEO_RESOLUTION_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF, GET_INFO, GET_LEN		
wLength		4		
Offset	Field	Size	Value	Description
0	**wWidth**	2	Number	The width, in pixels, of pictures output from the decoding process.
2	**wHeight**	2	Number	The height, in pixels, of pictures output from the decoding process.

4.2.2.4.4 Profile and Toolset Control

The Profile and Toolset control is used to specify the profile_idc and constraint flags for the current layer(s) and to further constrain features within the profile. There are restrictions on using this control to change profile. Specifically, this control cannot be used to increase the number of layers beyond the number negotiated in Probe & Commit. Also, this control cannot be used to move the codec between different **bUsage** values. The **wConstrainedToolset** field contains reserved bits for future extensibility.

Table 4-55 Profile Toolset Control

Control Selector		EU_PROFILE_TOOLSET_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_INFO, GET_LEN		
wLength		5		
Offset	Field	Size	Value	Description
0	**wProfile**	2	Number	**wProfile** indicates the profile and

				applicable constraints for the current format. This field is used with video formats that support multiple profiles. For example, for H.264: 0x4240: Constrained Baseline Profile 0x4200: Baseline Profile 0x4D00: Main Profile 0x640C: Constrained High Profile 0x6400: High Profile 0x5304: Scalable Constrained Baseline Profile 0x5300: Scalable Baseline Profile 0x5604: Scalable Constrained High Profile 0x5600: Scalable High Profile 0x7600: Multiview High Profile 0x8000: Stereo High Profile
2	**wConstrainedToolset**	2	Number	Reserved. Set to 0.
4	**bmSettings**	1	Bitmap	This bitmap enables features reported by the **bmCapabilities** field of the Video Format Descriptor. Refer to payload specification for details.

4.2.2.4.5 Minimum Frame Interval Control

The Minimum Frame Interval control is used to specify the minimum frame interval of the stream associated with the current **wLayerOrViewID**. This value reflects the maximum frame rate when combining the base layer and all temporal enhancement layers. The new **dwFrameInterval** must be a frame interval that is advertised in the currently negotiated Frame descriptor and greater than or equal to the frame interval negotiated in probe and commit. The minimum frame interval is given per stream where the stream is indicated by the stream_id subfield in **wLayerOrViewID**. All other subfields in **wLayerOrViewID** shall be ignored by the device.

For SET_CUR requests, this control can be used to configure multiple streams simultaneously by using a wildcard mask-e.g. stream_id=7 (‘111’). For streams with multiple temporal layers each layer shall have a frame interval that is **dwFrameInterval** * $2^{(N\text{-}1\text{-layer\_number})}$, where *N* is the number of temporal layers and *layer_number* is an incremental identifier for each layer starting at 0 for the base layer and incrementing by 1 for each successive temporal layer. For example, a stream with three temporal layers may have aggregate frame intervals of 333333, 666666, and 1333332.

Table 4-56 Minimum Frame Interval Control

Control Selector	EU_MIN_FRAME_INTERVAL_CONTROL
Mandatory Requests	SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF, GET_INFO, GET_LEN
wLength	4

Offset	Field	Size	Value	Description
0	**dwFrameInterval**	4	Number	Frame interval in 100-ns units. Cannot go below the value of **dwFrameInterval** set in Probe/Commit. The value must be an interval advertised in the associated Frame Descriptor that has the same resolution, profile, and toolset of the current stream. This control is applied per stream and ignores any layer information in **wLayerOrViewID**.

Note that changing the frame interval using this control may also change the bit rate limits described by **dwMinBitRate** and **dwMaxBitRate** in the negotiated video Frame descriptor. Use GET_MIN and GET_MAX with the Average Bitrate control to determine the new values for minimum bit rate and maximum bit rate.

4.2.2.4.6 Slice Mode Control

The Slice Mode control is used to specify the slice mode for the current **wLayerOrViewID**. This control is only supported if the codec supports using more than one slice per frame. If the device supports this control it must support one or more of the slice modes described below.

Table 4-57 Slice Mode Control

Control Selector		EU_SLICE_MODE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_MIN, GET_MAX, GET_INFO, GET_LEN		
wLength		4		
Offset	Field	Size	Value	Description
0	**wSliceMode**	2	Number	Most significant byte: **Mode**. 0: Maximum number of MBs per slice mode 1: Target compressed size per slice mode 2: Number of slices per frame mode 3: Number of Macroblock rows per slice mode 4-255: Reserved
2	**wSliceConfig Setting**		Number	The meaning of this field depends on **wSliceMode** mode: Mode 0: Maximum number of MBs per slice. Mode 1: Target size for each slice NALU in bytes. Mode 2: Number of slices per frame. Mode 3: Number of macroblock rows per slice.

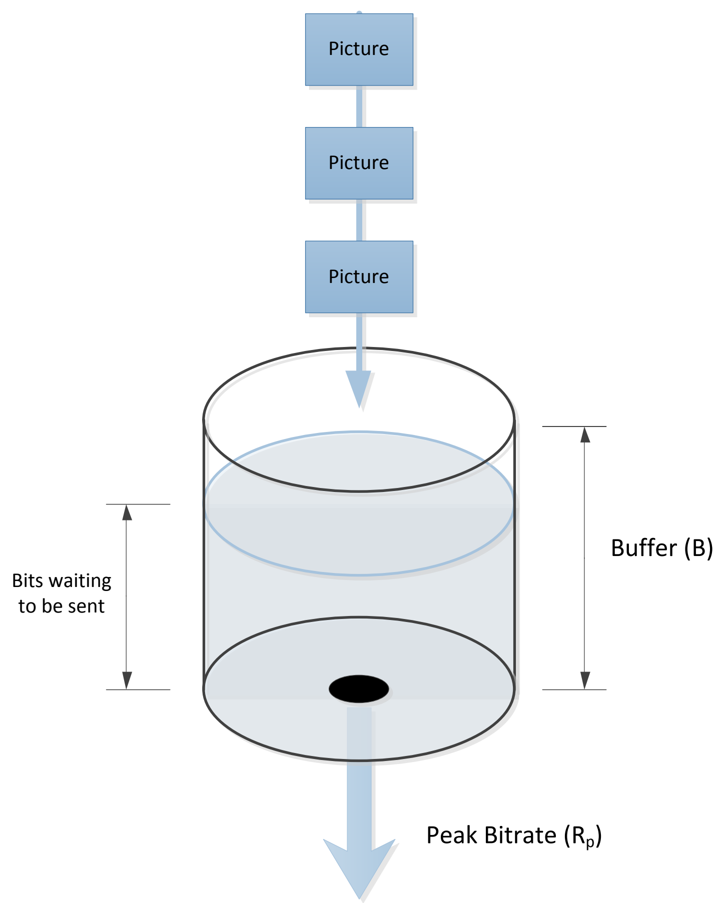
GET_MIN, GET_MAX, and GET_RES return values for **wSliceConfigSetting** based on the current **wSliceMode**

4.2.2.4.7 Rate Control Mode Control

The Rate Control Mode control is used to specify the rate control mode of the current layer. Once the mode is established, rate control parameters are set using the Peak Bit Rate Control, Average Rate Control Parameter Control, or Quantization Control.

For variable and constant bit rate buffer modeling, this document specifies rate control operation in terms of a leaky bucket model. The bits used to encode each picture are analogous to cups of water being dumped into the top of the bucket when each picture is encoded (after re-ordering the pictures as necessary for bitstream orders that differ from display order); the level of water in the bucket indicates the number of bits waiting to be sent to the decoder; and the water leaking out of a hole in the bottom of the bucket corresponds to bits flowing into the decoder through a transmission channel. The leaky bucket is a traffic meter that contains two parameters:

- R_P = dwPeakBitRate $\times$ 64 (bits per second), which is the peak bit rate at which bit can flow out from the bottom of the bucket
- B = dwCPBsize $\times$ 16 (bits), which is the coded picture buffer (CPB) capacity.



The buffer serves to smooth out local bit rate fluctuations while limiting the total bit usage that is possible over longer durations and limiting the buffering capacity necessary for a decoder to be able to decode the video content.
The leaky bucket model at the encoder has a corresponding mirror-image model that operates from the decoder perspective. As bits leak out of the encoder buffer, they conceptually enter into

a corresponding decoder input buffer, which continues to fill up until the decoding time of a picture arrives – at which time the bits for that picture are removed from the decoder's CPB.
If too many bits are dumped into the bucket too quickly, the buffer capacity B would be exceeded before enough bits have time to drain out of the hole in the bottom of the bucket, and the buffer is said to "overflow" from the encoder perspective. From the decoder perspective, an overflow could occur if the removal of pictures from the decoder CPB at the decoding times of those pictures is not fast enough to keep up with the amount of bits that have been flowing into it from the encoder.
The encoder shall ensure that the leaky bucket never overflows.

Table 4-58 Rate Control Mode Control

Control Selector		EU_RATE_CONTROL_MODE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_INFO, GET_LEN		
wLength		1		
Offset	Field	Size	Value	Description
0	**bRateControlMode**	1	Number	0: Reserved 1: Variable Bit Rate low delay (VBR) 2: Constant bit rate (CBR) 3: Constant QP 4: Global VBR low delay (GVBR) 5: Variable bit rate non-low delay (VBRN) 6: Global VBR non-low delay (GVBRN) 7-255: Reserved

Note that **bRateControlMode** selected must be marked as supported by **bmSupportedRateControlModes** in the Video Format Descriptor.

4.2.2.4.7.1 Variable Bit Rate (VBR)

The VBR control mode limits the long-term maximum bit rate of transmission, but allows the bit rate to vary substantially on a short-term basis. In particular, in the VBR mode the encoder is not required to continue to produce a significant bit rate in cases where there is little or no motion activity in the coded video scene content.

In the variable bit rate (VBR) mode of bit rate control operation, it is allowed for the encoder's leaky bucket to sometimes "run dry" – i.e., for all bits to leave the encoder CPB and for there to be some period of time during which no bits are flowing. The presence of such periods of time during which no bits are flowing implies that the average bit rate over a long duration of video content can be less than the peak bit rate R_P.

The average bit rate parameter **dwAverageBitRate** serves only as a guideline, as follows:

> The average bit rate parameter **dwAverageBitRate** represents only a "target" or "guideline", indicating the average bit rate expected to be produced by the video encoder when operating under normal lighting conditions with a normal degree of video scene activity. To calculate **dwAvergeBitRate** the encoder should sample over the period T = CPB size / Peak bit rate.

It is allowed for the bit rate produced by the video encoder to exceed **dwAverageBitRate** (e.g., when there is an exceptionally high degree of activity in the video scene).
It is also allowed for the bit rate produced by the video encoder to be less than **dwAverageBitRate** (e.g., when there is very little activity in the video scene or when lighting conditions are poor).

In contrast, the peak bit rate R_P = **dwPeakBitRate** $\times 64$ bps and the total buffer capacity B, which are the operating parameters of the leaky bucket model, correspond to a mandatory maximum not to be exceeded by the encoder on a long-term basis (i.e., the leaky bucket model shall not overflow).
The average bit rate parameter **dwAverageBitRate** shall be set to a value less than or equal to peak bit rate = dwPeakBitRate x 64.

For multi-layer bitstreams, the VBR-control model applies to the currently selected sub-bitstream as defined in section 4.2.2.4.2.1 For single layer bitstreams, this rate control model applies to the entire bitstream, because there is only one layer in the bitstream.

4.2.2.4.7.2 Constant Bit Rate (CBR)

CBR control mode is similar to the VBR control mode, except that dummy bits are generated when necessary in order to assure an exactly constant bit rate flowing from the encoder leaky bucket (there are always bits flowing to the decoder). For CBR operation the average bit rate is equal to the peak bit rate, as there is no variability in the bit rate in this case. In the CBR mode, it is required that the average bit rate parameter **dwAverageBitRate** shall be set equal to the peak bit rate = **dwPeakBitRate** x 64.

For multi-layer bitstreams, this rate-control model applies to the currently selected sub-bitstream as defined in section 4.2.2.4.2.1. For single layer bitstreams, this rate-control model applies to the entire bitstream, because there is only one layer in the bitstream.
The encoder shall ensure that the leaky bucket never overflows nor underflows in the CBR case.

4.2.2.4.7.3 Constant QP Mode

In Constant QP mode, the encoder shall use the **dwQpPrime** fields to derive a constant QP for I, P, and B slices (or EI, EP, and EB slices if quality or spatial scalability is employed).

4.2.2.4.7.4 Global VBR

In Global VBR mode, the rate-control model applies to the entire bitstream. For single-layer bitstreams, this mode is identical to VBR mode. For multi-layer bitstreams, this mode implies that no explicit rate control is required for the sub-bitstreams, as long as the entire bitstream obeys the rate-control model specified in the Rate Control Mode EU.

4.2.2.4.7.5 Low Delay and Non-Low Delay Modes

See individual payload specifications for details on how these modes are supported.

4.2.2.4.8 Average Bit Rate Control

The Average bit rate control is used to specify the average bit rate of the current selected substream. This control is used in all **bRateControlMode** modes except Constant QP. SET_CUR shall protocol STALL with an error code of **bRequestErrorCode** = "Wrong state" if the specified stream is not set to CBR, VBR, VBRN, Global VBR or Global VBRN.
For single bitstreams or when in Global VBR rate-control mode, this EU applies to the entire bitstream. For multi-layer streams, **dwAverageBitRate** is set per sub-bitstream, as defined in section 4.2.2.4.2.1.
This EU has no effect when the device operates in Constant QP rate-control mode.

The **dwAverageBitRate** value returned by the device upon a GET_MIN request specifies the minimum average bit rate for the sub-bitstream at the sub-bitstream frame interval. For GET_MAX, the device returns the maximum capability for the overall stream at the current frame interval for that stream, summing up bit rates across all layers in the sub-bitstream. These values are for the active resolution as specified by EU_VIDEO_RESOLUTION_CONTROL(GET_CUR).

Table 4-59: Average Bitrate Control

Control Selector		EU_AVERAGE_BITRATE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF, GET_INFO, GET_LEN, GET_RES		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwAverageBitRate**	4	Number	Average bit rate, in bits per second. Must be less than or equal to **dwPeakBitRate** x 64. Applies for all rate control modes except Constant QP.

This control should be reported as an AutoUpdate control. When this control is reported as AutoUpdate, any changes to the encoder that alter the values of GET_MIN and GET_MAX must be reported via AutoUpdate.

4.2.2.4.9 CPB Size Control

The CPB Size control is used to specify the CPB size of the current selected substream. This control is used in all **bRateControlMode** modes except Constant QP. SET_CUR shall protocol STALL with an error code of **bRequestErrorCode** = "Wrong state" if the specified stream is not set to CBR, VBR, VBRN, Global VBR or Global VBRN.
For single bitstreams or when in Global VBR rate-control mode, this EU applies to the entire bitstream. For multi-layer streams, **dwCPBsize** is set per sub-bitstream, as defined in section 9.3.4. EU has no effect when the device operates in Constant QP rate-control mode.

The **dwCPBsize** value returned by the device upon a GET_MIN request specifies the minimum CPB size for use with the average bit rate for the current sub-bitstream. For GET_MAX the device shall return the maximum CPB size for the current sub-bitstream supported by the device for the given **bLevelIDC** specified in the Frame descriptor. The host should set the current **dwCPBsize** appropriately for the **dwPeakBitRate**.

Table 4-60: CPB Size Control

Control Selector		EU_CPB_SIZE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF, GET_INFO, GET_LEN, GET_RES		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwCPBsize**	4	Number	CPB size, in units of 16 bits, corresponding to the CPB size. Must be a size supported by the profile and level combination, if that applies. This control applies for all rate control modes except Constant QP.

This control should be reported as an AutoUpdate control. When this control is reported as AutoUpdate, any changes to the encoder that alter the values of GET_MIN and GET_MAX must be reported via AutoUpdate.

4.2.2.4.10 Peak Bit Rate Control

The Peak Bit Rate control is used to specify the peak bit rate of the current layer. This control only applies when **bRateControlMode** is set to *VBR*, *VBRN*, *Global VBR or Global VBRN*. SET_CUR shall protocol STALL with an error code of **bRequestErrorCode** = “Wrong state” if the specified stream is not set to VBR, VBRN, Global VBR or Global VBRN.

Table 4-61: Peak Bit Rate Control

Control Selector		EU_PEAK_BIT_RATE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF, GET_INFO, GET_LEN, GET_RES		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwPeakBitRate**	4	Number	Peak bit rate, in units of 64 bits per second. Must be a rate supported by the profile and level combination. Used only for VBR, VBRN, Global VBR and Global VBRN mode.

This control should be reported as an AutoUpdate control. When this control is reported as AutoUpdate, any changes to the encoder that alter the values of GET_MIN and GET_MAX must be reported via AutoUpdate.

.

4.2.2.4.11 Quantization Parameter Control

The Quantization Parameter control is used to specify quantization parameters for the current layer. This control only applies when **bRateControlMode** is set to *Constant QP*. SET_CUR shall stall if the **bRateControlMode** mode for the active layer is not set to Constant QP. The value of the quantization parameters set by this control are valid until modified by this control, or until the rate control mode changes to something other than Constant QP. GET_MIN, GET_MAX, and GET_RES shall return values for each field.

Table 4-62: Quantization Parameter Control

Control Selector		EU_QUANTIZATION_PARAMS_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MAX, GET_MIN, GET_DEF, GET_INFO, GET_LEN, GET_RES		
wLength		6		
Offset	Field	Size	Value	Description
0	**wQpPrime_I**	2	Number	Only applicable in constant QP rate-control mode. Use this parameter to set/get QP for I frames. The details on how to interpret this number is located in the appropriate payload specification.
2	**wQpPrime_P**	2	Number	Only applicable in constant QP rate-control mode. Use this parameter to set/get QP for P frames. The details on how to interpret this number is located in the appropriate payload specification.
4	**wQpPrime_B**	2	Number	Only applicable in constant QP rate control mode. Use this parameter to set/get QP for B frames. The details on how to interpret this number is located in the appropriate payload specification.

If a slice type (I, P, B) is not supported by the format, device or if it is not supported for the active **wLayerOrViewID**, then

- For a SET_CUR request to the EU_QUANTIZATION_PARAMS_CONTROL the device will ignore the **wQpPrime** field for that slice type.

- For a GET_CUR request to EU_QUANTIZATION_PARAMS_CONTROL the device will return 0xFFFF for the **wQpPrime** field for that slice type.

4.2.2.4.12 Quantization Parameter Range Control

The QP Range control is used to specify the allowed quantization parameters when encoding to target bitrate. This control can be used in all **bRateControlMode** modes except Constant QP. SET_CUR shall protocol STALL with an error code of **bRequestErrorCode** = "Wrong state" if **bRateControlMode** is Constant QP.

For single layer bitstreams, this EU applies to the entire bitstream. For multi-layer streams, **bMinQP** and **bMaxQP** are set per sub-bitstream, as defined in section 9.3.4.

Table 4-63: Quantization Parameter Range Control

Control Selector	EU_QP_RANGE_CONTROL
Mandatory Requests	SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES GET_INFO, GET_DEF, GET_LEN

wLength		2		
Offset	Field	Size	Value	Description
0	**bMinQp**	1	Number	Minimum quantization parameter to use for the frame (both luma and chroma).
1	**bMaxQp**	1	Number	Maximum quantization parameter to use for the frame (both luma and chroma).

This control can be used to narrow the range of possible quantization parameter values (for both luma and chroma) to use in the encoding process. GET_MIN returns the highest **bMinQp** and lowest **bMaxQp** for the given rate control mode and negotiated bit rate. GET_MAX returns the lowest **bMinQp** and highest **bMaxQp** for the given rate control mode and negotiated bit rate. The **bMinQp** and **bMaxQp** values reflect the global minimum and maximum Qp, including all offsets.
This control should be reported as an AutoUpdate control. When this control is reported as AutoUpdate, any changes to the encoder that alter the values of GET_MIN and GET_MAX must be reported via AutoUpdate.

4.2.2.4.13 Synchronization and Long Term Reference Frame Control

This control is used to manage insertion of synchronization frames and long-term reference frames into the current layer. For options that insert an IDR or GDR, this control applies to the entire stream. When the host requests the generation of a sync frame, the encoder shall insert the specified **bSyncFrameType** into all the dependency layers associated with the current stream as identified by the current **wLayerOrViewID**. This control can be used to generate long term reference frames only if bit D6 in **bmCapabilities** in the negotiated Frame Descriptor is set to 1.

Table 4-64: Synchronization and Long Term Reference Frame Control

Control Selector		EU_SYNC_REF_FRAME_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_INFO, GET_LEN		
wLength		4		
Offset	Field	Size	Value	Description
0	**bSyncFrameType**	1	Number	0: Reset. Allow the encoder to determine the timing and type of synchronization frames. 1: Generate an IDR frame for all of the dependency layers of the current stream, if all of the layers of the current stream have quality_id equal to 0. Otherwise, generate key frames for all the dependency layers. 2: Generate an IDR frame that is a long-term reference frame for all of the dependency layers of the current stream, if all of the layers of the current stream have quality_id equal to 0. Otherwise, generate a key frame for the associated dependency layers of the current

				wLayerOrViewID. 3: Generate a non-IDR random-access I frame for the associated dependency layers of the current **wLayerOrViewID**. 4: Generate a non-IDR random-access I frame that is a long-term reference frame for the associated dependency layers of the current **wLayerOrViewID**. 5: Generate a P frame that is a-long term reference frame for the associated dependency layers of the current **wLayerOrViewID**. 6: Gradual Decoder Refresh (GDR) 7-255: Reserved.
1	**wSyncFrameInterval**	2	Number	In milliseconds. This field indicates the periodic recurrences of the selected **bSyncFrameType**. A value of **wSyncFrameInterval** = 0 indicates a single **bSyncFrameType** with no requirement for periodic recurrence.
3	**bGradualDecoderRefresh**	1	Number	Indicates a count of frames over which the gradual decoder refresh occurs. Only valid when **bSyncFrameType** = 6 (GDR). When **bSyncFrameType** is not 6, this field must be 0. From a recovery point of view, **bGradualDecoderRefresh** + 1 represents the number of frames required to completely refresh the picture. Bits: 0-6: recovery_frame_cnt 7: Reserved Use **wSyncFrameInterval** to establish the interval between Gradual Decoder Refresh (GDR) periods. When using GDR, the encoder should use recovery point SEI messages to indicate gradual decoder refresh access points.

GET_MIN and GET_MAX can be used to determine the minimum and maximum recovery_frame_cnt over which the encoder can implement GDR. GET_MIN and GET_MAX can also be used to determine the minimum and maximum **wSynchFrameInterval** of the current **bSyncFrameType**.

GET_MIN and GET_MAX may be used to determine if the device supports changes to **wSyncFrameInterval**. If GET_MIN and GET_MAX return the same **wSyncFrameInterval** value as GET_CUR, then the device does not support changes to this value.

4.2.2.4.14 Long-Term Buffer Control

The EU_LTR_BUFFER_CONTROL provides control of encoder's long term reference buffers. When combined with the EU_LTR_PICTURE_CONTROL, this control enables outside management over one or more of the encoder long term reference buffers. The host should check the device's long term buffer availability using this control before using the EU_LTR_PICTURE_CONTROL.

The Long Term Reference controls defined in this specification support two different trust models:

Trust Until (only notify when something has gone wrong)
Don't Trust Until (notify all results, success or failure)

The host can decide which Trust Model to use by setting **bTrustMode** to the desired mode. To implement a "Don't Trust Until" solution, set **bTrustMode** to 0. If an LTR has been validated by the host, then the associated position in the list of LTRs can be set to 1 using the EU_LTR_VALIDATION_CONTROL. To implement a "Trust Until" model, set **bTrustMode** to 1. If an LTR is confirmed as no longer valid by the host, then the associated position in the list of LTRs can be set to 0 using the EU_LTR_VALIDATION_CONTROL.

Table 4-65: Long-Term Buffer Control

Control Selector		EU_LTR_BUFFER_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_INFO, GET_LEN, GET_MAX		
wLength		2		
Offset	Field	Size	Value	Description
0	**bNumHostControlLTRBuffers**	1	Number	Number of Long-Term Reference Frames the host can control.
1	**bTrustMode**	1	Number	Trust mode for the LTR feature. 0 – For each inserted LTR, device sets associated bit in bmValidLTRs to 0 (Don't Trust Until) 1 – For each inserted LTR, device sets associated bit in bmValidLTRs to 1 (Trust Until)

The EU_LTR_BUFFER_CONTROL allows for discovery and allocation of long term reference (LTR) frames on the device. It also allows the host to set the device behavior when inserting new LTR frames using EU_LTR_PICTURE_CONTROL. If the encoder on the device does not have enough memory to enable long term reference frames at the current resolution, then the GET_MAX shall return **bNumHostControlLTRBuffers** equal to 0. If the encoder allows the host to manage the LTR buffers, it shall assign continuous index space starting from index 1.

GET_MAX returns the maximum number of LTR buffers available for host control.

4.2.2.4.15 Long-Term Reference Picture Control

The EU_LTR_PICTURE_CONTROL tells the encoder to generate an LTR and place it at a specific index in the LTR buffer.
If **bTrustMode** is 0, the device shall consider the LTR inserted at **bPutAtPositionInLTRBuffer** as invalid until validated using EU_LTR_VALIDATION_CONTROL. This means that index **bPutAtPositionInLTRBuffer** in **bmValidLTRs** must be 0 as soon as the new LTR is available.
If **bTrustMode** is 1, the device shall consider the LTR inserted at **bPutAtPositionInLTRBuffer** as valid. This means that index **bPutAtPositionInLTRBuffer** in **bmValidLTRs** must be 1 as soon as the new LTR is available.

Table 4-66: Long-Term Reference Picture Control

Control Selector		EU_LTR_PICTURE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_INFO, GET_LEN		
wLength		2		
Offset	Field	Size	Value	Description
0	**bPutAtPositionInLTRBuffer**	1	Number	Next frame should be put at certain position in Long Term Reference Buffer (LTRB). This frame must be in the temporal base layer. It must not be one of the scheduled sync frames schedule by the EU_SYNC_REF_FRAME_CONTROL control. 0 - Encoder is free to choose where to save the frame inside the encoder controlled buffers. It cannot be saved in any of the host controlled LTR Buffers. 1 : index 1 in the host controlled buffers 2 : index 2 in the host controlled buffers … M : position M (maximum) in the host controlled buffers Where M = bNumAppControlLTRBuffers
1	**bLTRMode**	1	Number	Determines which frames can be used as references when creating the new LTR. 0 - Encoder is restricted to valid host managed LTR frames 1 - Encoder may use any valid host or encoder controlled LTR frames.

				2 – Encoder may use any valid reference in the DPB

When **bLTRMode** is 0, the new LTR generated by this control must only reference host controlled LTRs that have been validated. When **bLTRMode** is 1, the new LTR generated by this control must only reference valid host or encoder controlled LTRs. When **bLTRMode** is 2, the encoder is free to use any long or short term references it wishes when creating the new LTR.. When a request for a new LTR frame is still pending, the device shall protocol STALL any new requests to this control for new LTR frames with **bRequestErrorCode** = "Not Ready",

4.2.2.4.16 Long-Term Reference Validation Control

The EU_LTR_VALIDATION_CONTROL instructs the encoder to update the list of valid LTRs which may be used for subsequent LTR frames. For each bit in **bValidLTRs**, a value of 1 means the LTR is valid and a value of 0 means the LTR is invalid. There are two different lists of LTR buffers, those managed by the host and those managed by the encoder. This control validates or invalidates both. The **bmValidLTRs** bitmask in this control can be used to indicate which host controlled LTRs can be used by EU_LTR_PICTURE_CONTROL. The **bmValidLTRs** bitmask may also be used to limit which LTRs should be used as references.
bmValidLTRs should be initialized to all zeros.

Table 4-67: Long-Term Reference Validation Control

Control Selector		EU_LTR_VALIDATION_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_INFO, GET_LEN		
wLength		2		
Offset	Field	Size	Value	Description
0	**bmValidLTRs**	2	Bitmap	A list of reference LTRs that may be used to generate subsequent frames. D0: LTR frame with index = 0 D1 - LTR frame with index = 1 D2 - LTR frame with index = 2 D3 - LTR frame with index = 3 Etc.

4.2.2.4.17 SEI Messages Control

The control allows discovery and control of the H.264 SEI_payloads, as specified in Annex D of the H.264 specification.

Table 4-68: SEI Message Control

Control Selector		EU_SEI_PAYLOADTYPE_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF, GET_MAX, GET_MIN		
wLength		8		
Offset	Field	Size	Value	Description
0	**bmSEIMessages**	8	Bitmap	A bit set indicates the associated SEI type is enabled. D0: buffering_period D1: pic_timing D2: pan_scan_rect D3: filler_payload D4: user_data_registered_itu_t_t35 D5: user_data_unregistered D6: recovery_point D7: dec_ref_pic_marking_repetition D8: spare_pic D9: scene_info D10: sub_seq_info D11: sub_seq_layer_characteristics D12: sub_seq_characteristics D13: full_frame_freeze D14: full_frame_freeze_release D15: full_frame_snapshot D16: progressive_refinement_segment_start D17: progressive_refinement_segment_end D18: motion_constrained_slice_group_set D19: film_grain_characteristics D20: deblocking_filter_display_preference D21: stereo_video_info D22: post_filter_hint D23: tone_mapping_info D24: scalability_info D25: sub_pic_scalable_layer D26: non_required_layer_rep D27: priority_layer_info D28: layers_not_present D29: layer_dependency_change D30: scalable_nesting D31: base_layer_temporal_hrd D32: quality_layer_integrity_check D33: redundant_pic_property D34: tl0_dep_rep_index D35: tl_switching_point

				D36: parallel_decoding_info D37: mvc_scalable_nesting D38: view_scalability_info D39: multiview_scene_info D40: multiview_acquisition_info D41: non_required_view_component D42: view_dependency_change D43: operation_points_not_present D44: base_view_temporal_hrd D45: frame_packing_arrangement D63..D46: Reserved. Set to 0

Each bit in bmSEIMessages represents a different SEI message and when the associated bit is 1 the SEI message is enabled. Multiple types of SEI messages can be enabled/disabled simultaneously with this control. Bits set in the GET_CUR response will indicate the SEI messages that are currently enabled. Bits set in the GET_MAX response will indicate the SEI messages that the device supports. Bits set in the GET_MIN response will indicate which SEI messages are enabled and cannot be disabled by the host.

4.2.2.4.18 Priority Control

This control can be used to set priority for video payload layers.

Table 4-69: Priority Control

Control Selector		EU_PRIORITY_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_LEN		
wLength		1		
Offset	Field	Size	Value	Description
0	**bPriority**	1	Number	The value of priority for the scalable layer specified by the current **wLayerOrViewID**.

On a GET_DEF request, the device shall return values based on **bUsage**.

4.2.2.4.19 Start or Stop Layer Control

This control is used to start or stop streaming of the current layer(s). By default all layers will be streamed. The host may stop individual layers (and their dependents) from streaming by using this control. Once stopped, this control may be used to restart them.

Table 4-70: Start or Stop Layer Control

Control Selector		EU_START_OR_STOP_LAYER_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_LEN		
wLength		1		
Offset	Field	Size	Value	Description
0	**bUpdate**	1	Number	0: Stop streaming the current layer and all layers

				depending on it. 1: Start streaming the current layer (and all layers it depends on) once the device starts streaming. If this control is not issued before streaming is enabled (before SET_INTERFACE), the encoder shall stream every layer when streaming is enabled GET_CUR returns the current state of the layer (0 if stopped, or 1 if started).

4.2.2.4.20 Level IDC Control

This control is used to set the level_idc of the video stream. The default value of **bLevelIDC** in this control is initialized to the value of **bLevelIDC** from the video Frame descriptor selected as part of probe and commit. The host can issue SET_CUR requests to this control to reduce the maximum level_idc the encoder shall use, i.e. to reduce the minimum level_idc that the decoder will need to support.

Table 4-71: Level IDC Control

Control Selector		EU_LEVEL_IDC_LIMIT_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bLevelIDC**	1	Number	The level, as specified by the level_idc flag (9, 10, 11, 12, 13, 20, 21, 22, 30, 31, 32, 40, 41, 42, etc). For example: 0x1F: Level 3.1. 0x28: Level 4.0. Note that this should indicate the minimum level that supports the resolution, maximum bit rate and CPB size set for the stream. For a multi-layer stream this refers to the highest enhancement layer. For multi-view streams this refers to all views.

Note that the host must adjust the current average bit rate and current CBP size prior to changing the level_idc to guarantee those do not violate the new level_idc .
When this value changes, the device must adjust the values returned by EU_AVERAGE_BITRATE_CONTROL(GET_MAX) and EU_CPB_SIZE_CONTROL(GET_MAX) to satisfy the new level_idc.

4.2.2.4.21 Error Resiliency Control

The Error Resiliency control is used to set error resiliency features on the video encoder. Because error resiliency features are specific to each encoder, the only field in this control is a 16 bit bitmap that allows up to 16 different features to be turned on or off per video payload.

Table 4-72: Error Resiliency Control

Control Selector		EU_ERROR_RESILIENCY_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_DEF, GET_RES		
wLength		2		
Offset	Field	Size	Value	Description
0	**bmErrorResiliencyFeatures**	2	Bitmap	Payload format specific bitmap of available error resiliency features.

In response to a GET_RES response, device shall set the bits for supported error resiliency features to 1. All other bits should be set to 0. In response to a GET_DEF request device shall set the bits to 1 for the tools that are enabled in the device default configuration. All other bits shall be set to 0. While the exact meaning of **bmErrorResiliencyFeatures** is established in each payload specification, several possible examples are given below.

For VP8, **bmErrorResiliencyFeatures** could have the following assignment:

D0: Random Macroblock Intra Refresh.
D1: Frame-level error resiliency.
D2: Partition-level error resiliency.
D15-D3: Reserved. Set to 0.

For H.264, **bmErrorResiliencyFeatures** could have the following assignment:

D0: Random Macroblock Intra Refresh.
D1: Flexible Macroblock Ordering.
D2: Arbitrary Slice Ordering.
D3: Redundant Slices.
D4: Data Partitioning.
D15-D5: Reserved. Set to 0.

4.2.2.5 Extension Unit Control Requests

These requests are used to set or read a video Control within an Extension Unit.

Table 4-73 Extension Unit Control Requests

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_CUR	CS	Extension Unit ID and Interface	Length of parameter block	Parameter block
10100001	GET_CUR GET_MIN GET_MAX GET_RES GET_DEF GET_LEN GET_INFO				

The **bRequest** field indicates which attribute the request is manipulating. The MIN, MAX, and RES attributes are not supported for the Set request.

The **wValue** field specifies the Control Selector (CS) in the high byte and zero in the low byte. The Control Selector indicates which vendor-defined control within the Extension Unit that this request is manipulating. If the request specifies an unknown or unsupported CS to that Unit, the control pipe must indicate a stall. However, if the request specifies an available control, the request should succeed.

The range of CS values supported by the Extension Unit is dictated by the number of controls specified by the **bNumControls** field in the Extension Unit descriptor. See section 3.7.2.7, "Extension Unit Descriptor". The range shall be [1..bNumControls].

The GET_LEN request queries for the length of the parameter block of the specified control. When issuing the GET_LEN request, the **wLength** field shall always be set to a value of 2 bytes. The result returned shall be the length specified for all other requests on the same control.

All controls supported by the Extension Unit must support the following requests:
GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF, GET_LEN.

The following request(s) are optional, depending on the control usage and behavior:
SET_CUR

All Extension Unit controls are vendor-defined. The vendor must provide the relevant host software to program these controls. The generic host driver will not have knowledge of the control semantics, but acts as a control transport between the vendor-provided host software and the device.

However, by using the GET_LEN request, the host driver would be able to query the length and raw data stored in the vendor-defined controls. While it would not be able to interpret this data, it would be capable of saving and restoring these control settings if required.

4.3 VideoStreaming Requests

VideoStreaming requests can be directed either to the VideoStreaming interface or to the associated video-data endpoint, depending on the location of the Control to be manipulated.

4.3.1 Interface Control Requests

These requests are used to set or read an attribute of an interface Control inside a particular VideoStreaming interface of the video function.

Table 4-74 Interface Control Requests inside a Particular VideoStreaming Interface

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001	SET_CUR	CS	Zero and Interface	Length of parameter block	Parameter block
10100001	GET_CUR GET_MIN GET_MAX GET_RES GET_DEF GET_LEN GET_INFO				

The **bRequest** field indicates which attribute the request is manipulating.

The **wValue** field specifies the Control Selector (CS) in the high byte, and the low byte must be set to zero. The CS indicates the type of Control that this request is manipulating. If the request specifies an unknown CS to that endpoint, the control pipe must indicate a stall.

The VideoStreaming interface controls allow the host software to query and set parameters related to the video stream format and the video stream encoder. These parameters include the format, frame size and frame rate of the video stream, as well as the format and frame size of still images captured by the device that are associated with the video stream. For devices that support host-adjustable video stream encoder parameters, controls allowing the adjustment of the key frame rate and compression quality, among other parameters, are also supported. Only Stream Error Code Control supports interrupt with VideoStreaming interface.

4.3.1.1 Video Probe and Commit Controls

The streaming parameters selection process is based on a shared negotiation model between the host and the video streaming interface, taking into account the following features:

- shared nature of the USB
- interdependency of streaming parameters
- payload independence
- modification of streaming parameters during streaming

This negotiation model is supported by the Video Probe and Commit controls. The Probe control allows retrieval and negotiation of streaming parameters. When an acceptable combination of streaming parameters has been obtained, the Commit control is used to configure the hardware with the negotiated parameters from the Probe control.

Additional Encoding Units may used to finalize the configuration of the video streaming interface after Probe and Commit but before streaming starts. This hybrid model of Descriptor plus Encoding Unit was chosen as the best model to navigate the complex space of encoder configuration.

Table 4-75 Video Probe and Commit Controls

Control Selector		VS_PROBE_CONTROL VS_COMMIT_CONTROL		
Mandatory Requests		See tables below		
wLength		48		
Offset	Field	Size	Value	Description
0	**bmHint**	2	Bitmap	Bitfield control indicating to the function what fields shall be kept fixed (indicative only): D0: **dwFrameInterval** D1: **wKeyFrameRate** D2: **wPFrameRate** D3: **wCompQuality** D4: **wCompWindowSize** D15..5: Reserved (0) The hint bitmap indicates to the video streaming interface which fields shall be kept constant during stream parameter negotiation. For example, if the selection wants to favor frame rate over quality, the **dwFrameInterval** bit will be set (1). This field is set by the host, and is read-only for the video streaming interface.
2	**bFormatIndex**	1	Number	Video format index from a Format descriptor for this video interface. Select a specific video stream format by setting this field to the one-based index of the associated Format descriptor. To select the first format defined by a device, a value one (1) is written to this field. This field must be supported even if only one video format is supported by the device. This field is set by the host.

3	**bFrameIndex**	1	Number	Video frame index from a Frame descriptor. This field selects the video frame resolution from the array of resolutions supported by the selected stream. The index value ranges from 1 to the number of Frame descriptors following a particular Format descriptor. For frame-based formats, this field must be supported even if only one video frame index is supported by the device. For video payloads with no defined Frame descriptor, this field shall be set to zero (0). This field is set by the host.
4	**dwFrameInterval**	4	Number	Frame interval in 100 ns units. This field sets the desired video frame interval for the selected video stream and frame index. The frame interval value is specified in 100 ns units. The device shall support the setting of all frame intervals reported in the Frame Descriptor corresponding to the selected Video Frame Index. For frame-based formats, this field must be implemented even if only one video frame interval is supported by the device. When used in conjunction with an IN endpoint, the host shall indicate its preference during the Probe phase. The value must be from the range of values supported by the device. When used in conjunction with an OUT endpoint, the host shall accept the value indicated by the device.
8	**wKeyFrameRate**	2	Number	Key frame rate in key-frame per video-frame units.

				This field is only applicable to sources (and formats) capable of streaming video with adjustable compression parameters. Use of this control is at the discretion of the device, and is indicated in the VS Input or Output Header descriptor. The Key Frame Rate field is used to specify the compressor's key frame rate. For example, if one of every ten encoded frames in a video stream sequence is a key frame, this control would report a value of 10. A value of 0 indicates that only the first frame is a key frame. When used in conjunction with an IN endpoint, the host shall indicate its preference during the Probe phase. The value must be from the range of values supported by the device. When used in conjunction with an OUT endpoint, the host shall accept the value indicated by the device.
10	**wPFrameRate**	2	Number	PFrame rate in PFrame/key frame units. This field is only applicable to sources (and formats) capable of streaming video with adjustable compression parameters. Use of this control is at the discretion of the device, and is indicated in the VS Input or Output Header descriptor. The P Frame Rate Control is used to specify the number of P frames per key frame. As an example of the relationship between the types of encoded frames, suppose a key frame occurs once in every 10 frames, and there are 3 P frames per key frame. The P frames will be spaced evenly between the key frames. The other 6 frames, which occur between the key frames and the P frames, will be bi-directional (B) frames.

				When used in conjunction with an IN endpoint, the host shall indicate its preference during the Probe phase. The value must be from the range of values supported by the device. When used in conjunction with an OUT endpoint, the host shall accept the value indicated by the device.
12	**wCompQuality**	2	Number	Compression quality control in abstract units 1 (lowest) to 10000 (highest). This field is only applicable to sources (and formats) capable of streaming video with adjustable compression parameters. Use of this field is at the discretion of the device, and is indicated in the VS Input or Output Header descriptor. This field is used to specify the quality of the video compression. Values for this property range from 1 to 10000 (1 indicates the lowest quality, 10000 the highest). The resolution reported by this control will determine the number of discrete quality settings that it can support. When used in conjunction with an IN endpoint, the host shall indicate its preference during the Probe phase. The value must be from the range of values supported by the device. When used in conjunction with an OUT endpoint, the host shall accept the value indicated by the device.
14	**wCompWindowSize**	2	Number	Window size for average bit rate control. This field is only applicable to sources (and formats) capable of streaming video with adjustable compression parameters. Use of this control is at the discretion of

				the device, and is indicated in the VS Input or Output Header descriptor. The Compression Window Size Control is used to specify the number of encoded video frames over which the average size cannot exceed the specified data rate. For a window of size *n*, the average frame size of any consecutive *n* frames will not exceed the stream's specified data rate. Individual frames can be larger or smaller. For example, if the data rate has been set to 100 kilobytes per second (KBps) on a 10 frames per second (fps) movie with a compression window size of 10, the individual frames can be any size, as long as the average size of a frame in any 10-frame sequence is less than or equal to 10 kilobytes. When used in conjunction with an IN endpoint, the host shall indicate its preference during the Probe phase. The value must be from the range of values supported by the device. When used in conjunction with an OUT endpoint, the host shall accept the value indicated by the device.
16	**wDelay**	2	Number	Internal video streaming interface latency in ms from video data capture to presentation on the USB. When used in conjunction with an IN endpoint, this field is set by the device and read only from the host. When used in conjunction with an OUT endpoint, this field is set by the host and read only from the device.
18	**dwMaxVideoFrameSize**	4	Number	Maximum video frame or codec-specific

				segment size in bytes. For frame-based formats, this field indicates the maximum size of a single video frame. When streaming simulcast this number reflects the maximum video frame size of the negotiated Frame descriptor. For frame-based formats, this field must be supported. For stream-based formats, and when this behavior is enabled via the **bmFramingInfo** field (below), this field indicates the maximum size of a single codec-specific segment. The sender is required to indicate a segment boundary via the FID bit in the payload header. This field is ignored (for stream-based formats) if the **bmFramingInfo** bits are not enabled. When used in conjunction with an IN endpoint, this field is set by the device and read only from the host. When used in conjunction with an OUT endpoint, this field is set by the host and read only from the device.
22	**dwMaxPayloadTransfer Size**	4	Number	Specifies the maximum number of bytes that the device can transmit or receive in a single payload transfer. This field must be supported. This field is set by the device and read only from the host. Some host implementations restrict the maximum value permitted for this field. The host shall avoid overshoot of single payload transfer size by reconfiguring the device. (e.g. by updating bitrates, resolutions etc.)
26	**dwClockFrequency**	4	Number	The device clock frequency in Hz for the specified format. This will specify the units used for the time information fields

				in the Video Payload Headers in the data stream. This parameter is set by the device and read only from the host.
30	**bmFramingInfo**	1	Bitmap	Bitfield control supporting the following values: D0: If set to 1, the Frame ID (FID) field is required in the Payload Header (see description of D0 in section 2.4.3.3, "Video and Still Image Payload Headers"). The sender is required to toggle the Frame ID at least every dwMaxVideoFrameSize bytes (see above). D1: If set to 1, indicates that the End of Frame (EOF) field may be present in the Payload Header (see description of D1 in section 2.4.3.3, "Video and Still Image Payload Headers"). It is an error to specify this bit without also specifying D0. D2: If set to 1, indicates that the End of Slice (EOS) field may be present in the Payload Header. It is an error to specify this bit without also specifying D0. D7..3: Reserved (0) This control indicates to the function whether payload transfers will contain out-of-band framing information in the Video Payload Header (see section 2.4.3.3, "Video and Still Image Payload Headers"). For known frame-based formats (e.g., MJPEG, Uncompressed, DV), this field is ignored. For known stream-based formats, this

				field allows the sender to indicate that it will identify segment boundaries in the stream, enabling low-latency buffer handling by the receiver without the overhead of parsing the stream itself. When used in conjunction with an IN endpoint, this control is set by the device, and is read-only from the host. When used in conjunction with an OUT endpoint, this parameter is set by the host, and is read-only from the device.
31	**bPreferedVersion**	1	Number	The preferred payload format version supported by the host or device for the specified **bFormatIndex** value. This parameter allows the host and device to negotiate a mutually agreed version of the payload format associated with the **bFormatIndex** field. The host initializes this and the following **bMinVersion** and **bMaxVersion** fields to zero on the first Probe Set. Upon Probe Get, the device shall return its preferred version, plus the minimum and maximum versions supported by the device (see **bMinVersion** and **bMaxVersion** below). The host may issue a subsequent Probe Set/Get sequence to specify its preferred version (within the ranges returned in **bMinVersion** and **bMaxVersion** from the initial Probe Set/Get sequence). The host is not permitted to alter the **bMinVersion** and **bMaxVersion** values. This field will support up to 256 (1-255) versions of a single payload format. The version number is drawn from the minor version of the Payload Format specification. For example, version 1.2 of a Payload Format specification would result in a value of 2 for this parameter.

32	**bMinVersion**	1	Number	The minimum payload format version supported by the device for the specified **bFormatIndex** value. This value is initialized to zero by the host and reset to a value in the range of 1 to 255 by the device. The host is not permitted to modify this value (other than to restart the negotiation by setting **bPreferredVersion**, **bMinVersion** and **bMaxVersion** to zero).
33	**bMaxVersion**	1	Number	The maximum payload format version supported by the device for the specified **bFormatIndex** value. This value is initialized to zero by the host and reset to a value in the range of 1 to 255 by the device. The host is not permitted to modify this value (other than to restart the negotiation by setting **bPreferredVersion**, **bMinVersion** and **bMaxVersion** to zero).
34	**bUsage**	1	Number	Current **bUsage**: 1-8: Real-time modes 9-16: Broadcast modes 17-24: File storage modes 25 – 31: Multiview modes 32-255: Reserved This bitmap enables features reported by the **bmUsages** field of the Video Frame Descriptor. For temporally encoded video formats, this field must be supported, even if the device only supports a single value for bUsage.
35	**bBitDepthLuma**	1	Number	Represents bit_depth_luma_minus8 + 8, which must be the same as bit_depth_chroma_minus8 + 8.
36	**bmSettings**	1	Bitmap	A bitmap of flags that is used to discover and control specific features of a temporally encoded video stream. When it is supported, it is defined in the

				associated Payload specification. This bitmap enables features reported by the **bmCapabilities** field of the Video Frame Descriptor. For temporally encoded video formats, this field must be supported.
37	**bMaxNumberOfRefFramesPlus1**	1	Number	Host indicates the maximum number of frames stored for use as references.
38	**bmRateControlModes**	2	Number	This field contains 4 subfields, each of which is a 4 bit number. It enables features reported by the **bmSupportedRateControlModes** field of the Video Format Descriptor. Each 4 bit number indicates the rate-control mode for a stream of encoded video. If the video payload does not support rate control, this entire field should be set to 0. **bmRateControlModes** supports up to four simulcast streams. For simulcast transport the number of streams is inferred from the **bmLayoutPerStream** field. Otherwise, the number of streams is 1. D3-D0: Rate-control mode for the first simulcast stream (with stream_id=0.) D7-D4: Rate-control mode for the second simulcast stream (with stream_id=1). D11-D8: Rate control mode for the third simulcast stream (with stream_id=2). D15-D12: Rate control mode for the fourth simulcast stream (with stream_id=3.) When **bmRateControlModes** is non-zero, each 4-bit subfield can take one of the following values:

				0: Not applicable, because this stream is non-existent. 1: VBR with underflow allowed 2: CBR 3: Constant QP 4: Global VBR, underflow allowed 5: VBR without underflow 6: Global VBR without underflow 7-15: Reserved For temporally encoded video formats, this field must be supported, even if the device only supports a single value for **bmRateControlModes**.
40	**bmLayoutPerStream**	8	Number	This field contains 4 subfields, each of which is a 2 byte number. For simulcast transport, this field indicates the specific layering structure for each stream, up to four simulcast streams. For a single, multi-layer stream, only the first two bytes are used. For a single stream with no enhancement layers, this field shall be set to 0. See individual payload specification for how to interpret each 2 byte sub-field. For temporally encoded video formats, this field must be supported.

4.3.1.1.1 Probe and Commit Operational Model

Unsupported fields shall be set to zero by the device. Fields left for streaming parameters negotiation shall be set to zero by the host. For example, after a SET_CUR request initializing the FormatIndex and FrameIndex, the device will return the new negotiated field values for the supported fields when retrieving the Probe control GET_CUR attribute.
In order to avoid negotiation loops, the device shall always return streaming parameters with decreasing data rate requirements. Unsupported streaming parameters shall be reset by the streaming interface to supported values according to the negotiation loop avoidance rules. This convention allows the host to cycle through supported values of a field.

During Probe and Commit, the following fields, if supported, shall be negotiated in order of decreasing priority:

- **bFormatIndex**

- **bFrameIndex**
- **dwMaxPayloadTransferSize**
- **bUsage**
- **bmLayoutPerStream**
- Fields set to zero by the host with their associated **bmHint** bit set to 1
- All the remaining fields set to zero by the host

For simplicity when streaming temporally encoded video, the required bandwidth for each streaming interface shall be estimated using the maximum bit rate for the selected profile/resolution and the number of simulcast streams. The USB bandwidth reserved shall be the calculated by the host as the advertised **dwMaxBitRate** from the selected Frame Descriptor multiplied times the number of simulcast streams as defined in the **bmLayoutPerStream** field. The interface descriptor for the video function should have multiple alternate settings that support the required bandwidths calculated in the manner above.

The following table describes VS_PROBE_CONTROL request attributes.

Table 4-76 VS_PROBE_CONTROL Requests

Attribute	Description
GET_CUR	Returns the current state of the streaming interface. All supported fields set to zero will be returned with an acceptable negotiated value. Prior to the initial SET_CUR operation, the GET_CUR state is undefined. This request shall stall in case of negotiation failure.
GET_MIN	Returns the minimum value for negotiated fields.
GET_MAX	Returns the maximum value for negotiated fields.
GET_RES	Return the resolution of each supported field in the Probe/Commit data structure.
GET_DEF	Returns the default value for the negotiated fields.
GET_LEN	Returns the length of the Probe data structure.
GET_INFO	Queries the capabilities and status of the Control. The value returned for this request shall have bits D0 and D1 each set to one (1), and the remaining bits set to zero (0) (see section 4.1.2, “Get Request”).
SET_CUR	Sets the streaming interface Probe state. This is the attribute used for stream parameter negotiation. This request shall protocol STALL in the case where the device would be place into an unsupported state or the case where value for a negotiated field is out of range. For exact errors to register, see section 4.2.1.2 "Request Error Code Control".

The following table describes VS_COMMIT_CONTROL request attributes.

Table 4-77 VS_COMMIT_CONTROL Requests

Attribute	Description
GET_CUR	Returns the current state of the streaming interface. Prior to the initial SET_CUR operation, the GET_CUR state is undefined.
GET_MIN	Not specified.
GET_MAX	Not specified.
GET_RES	Not specified.
GET_DEF	Not specified.
GET_LEN	Returns the length of the Commit data structure.
GET_INFO	Queries the capabilities and status of the Control. The value returned for this request shall have bits D0 and D1 each set to one (1), and the remaining bits set to zero (0) (see section 4.1.2, “Get Request”).
SET_CUR	Sets the device state. This sets the active device state. The field values must be the result of a successful VS_PROBE_CONTROL(GET_CUR) request. This request shall protocol STALL in case an unsupported state is specified. This request shall protocol STALL in the case where the device would be place into an unsupported state or the case where value for a negotiated field is out of range. For exact errors to register, see section 4.2.1.2 "Request Error Code Control".

4.3.1.1.2 Stream Negotiation Examples

Successful USB isochronous bandwidth negotiation

Host issues a VS_PROBE_CONTROL(SET_CUR) which is accepted and populated by the device in a VS_PROBE_CONTROL(GET_CUR). After this single round of negotiation the host sends a VS_COMMIT_CONTROL(SET_CUR) using the field values of VS _PROBE_CONTROL returned by the device in VS_PROBE_CONTROL(GET_CUR). After the successful Commit, the host starts streaming by selecting the first alternate interface.

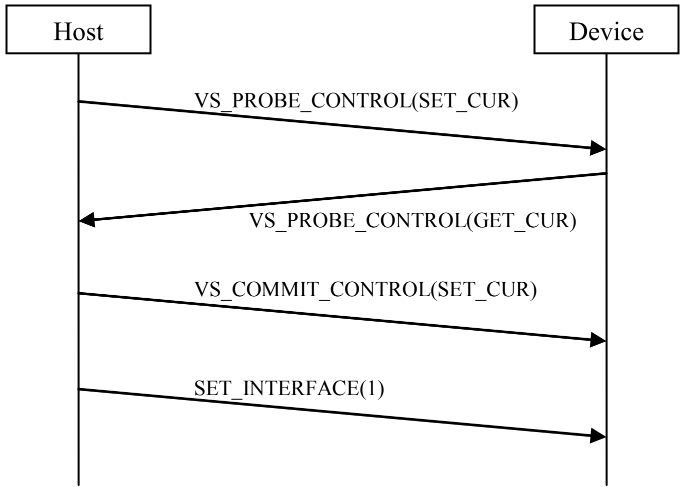


Figure 4-5 Successful USB Isochronous Bandwidth Negotiation

4.3.1.1.2.1 USB isochronous bandwidth negotiation failure

In this case the host requests video settings and an alternate interface that are incompatible. This could happen if the alternate interface does not support the data throughput required by the video settings. This request shall protocol STALL with "Wrong State". The host tries again with a set of features that demand less USB bandwidth, and this time succeeds.

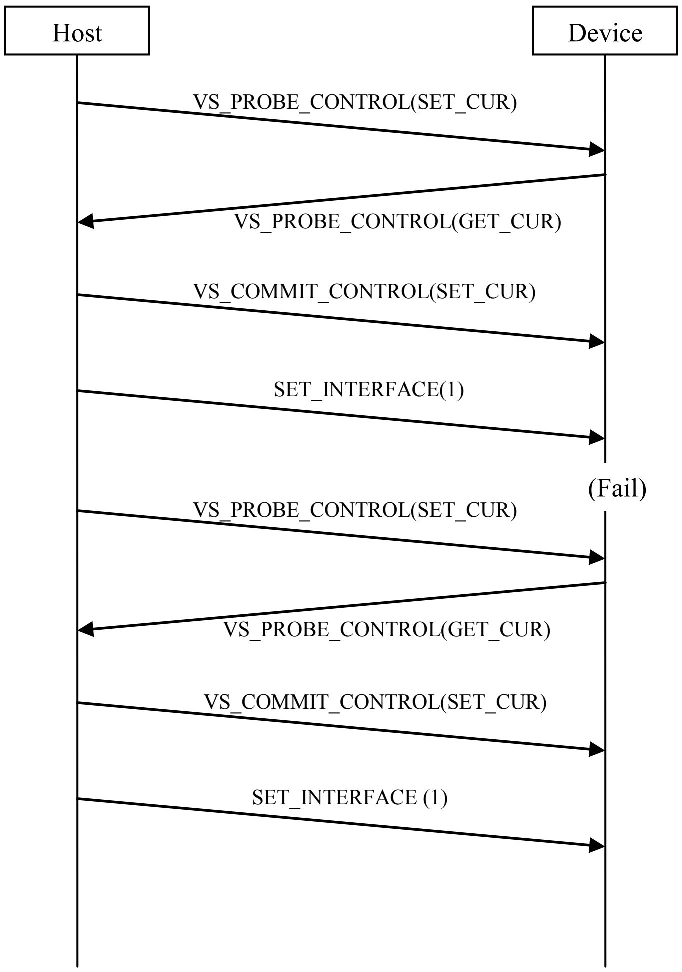


Figure 4-6 Failed USB Isochronous Bandwidth Negotiation

Dynamic stream settings *modification* while streaming
In the example below, the host changes the streaming parameters **bCompQuality** and **bCompWindowSize** after streaming has started by issuing an updated VS_COMMIT_CONTROL(SET_CUR). This operation was successful because the new settings did not exceed the negotiated USB bandwidth.

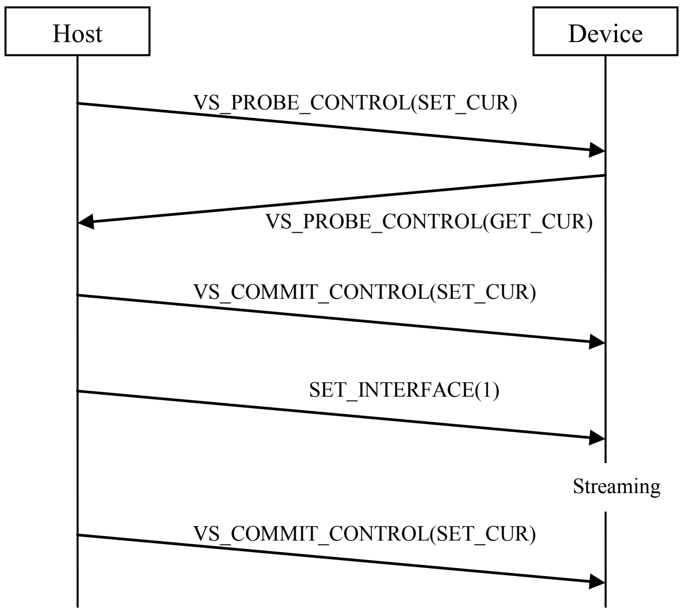


Figure 4-7 Dynamic Stream Settings Modification while Streaming

4.3.1.2 Video Still Probe Control and Still Commit Control

These still-image controls are required for video functions supporting method 2 or 3 for still-image retrieval.

Table 4-78 Video Still Probe Control and Still Commit Control

Control Selector		VS_STILL_PROBE_CONTROL VS_STILL_COMMIT_CONTROL		
Mandatory Requests		See tables below		
wLength		11		
Offset	Field	Size	Value	Description
0	**bFormatIndex**	1	Number	Video format index from a Format descriptor. A specific still-image format is selected by setting this field to the one-based index for the associated Format descriptor. To select the first format defined by a device, a value of 1 is written to this control.
1	**bFrameIndex**	1	Number	Video frame index from a Frame descriptor. This field selects the still-image frame resolution from the array of resolutions supported by the selected still-image format. The index value ranges from one to the number of Still Image Size Patterns reported by the selected Still Image Frame descriptor.
2	**bCompressionIndex**	1	Number	Compression index from a Frame descriptor. This field selects the still-image frame compression from the array of compression patterns supported by the selected still-image format. The index value ranges from one to the number of Still Image Compression Patterns reported by the selected Still Image Frame descriptor.
3	**dwMaxVideoFrameSize**	4	Number	Maximum still image size in bytes. This field indicates the maximum size of a single still image. This parameter is set by the device

				and read only from the host.
7	**dwMaxPayloadTransferSize**	4	Number	Specifies the maximum number of bytes that the device can transmit or receive in a single payload transfer.

The following table describes VS_STILL_PROBE_CONTROL request attributes:

Table 4-79 VS_STILL_PROBE_CONTROL Requests

Attribute	Description
GET_CUR	Returns the current state of the device. Prior to the initial SET_CUR operation, the GET_CUR state is undefined.
GET_MIN	Returns the minimum value for negotiated fields.
GET_MAX	Returns the maximum value for negotiated fields.
GET_RES	Not specified.
GET_DEF	Returns the default value for negotiated fields.
GET_LEN	Returns the length of the Probe data structure.
GET_INFO	Queries the capabilities and status of the Control. The value returned for this request shall have bits D0 and D1 each set to one (1), and the remaining bits set to zero (0) (see section 4.1.2, “Get Request”).
SET_CUR	Sets the streaming interface state. This is the state used for stream parameters negotiations.

The following table describes VS_STILL_COMMIT_CONTROL request attributes:

Table 4-80 VS_STILL_COMMIT_CONTROL Requests

Attribute	Description
GET_CUR	Returns the current state of the device. After device configuration, this state is undefined.
GET_MIN	Not specified.
GET_MAX	Not specified.
GET_RES	Not specified.
GET_DEF	Not specified.
GET_LEN	Returns the length of the Commit data structure.
GET_INFO	Queries the capabilities and status of the Control. The value returned for this request shall have bits D0 and D1 each set to one (1), and the remaining bits set to zero (0) (see section 4.1.2, “Get Request”).
SET_CUR	Sets the device state. This sets the active device state; the field values must be the result of a VS_STILL_PROBE_CONTROL(GET_CUR) request. When the associated still-image pipe is active, this attribute cannot be used and the control pipe shall indicate a stall for this request.

4.3.1.3 Synch Delay Control

The purpose for Synch Delay Control is to dynamically synchronize multiple video streams from a device to host, or from multiple devices to the host to compensate for differing latencies among multiple streams. Latency is the internal delay of the source from acquisition to data delivery on the bus.

Only those devices that are capable of video streaming with adjustable delay latency parameters support this control.

The Control is used to notify the video application buffer memory manager on the device to control an internal latency by controlling output timing of the video data to its endpoint.

It is the responsibility of the host (video sink) to synchronize streams by scheduling the rendering of samples at the correct moment, taking into account the internal delays of all media streams being rendered.

Table 4-81 Synch Delay Control

Control Selector		VS_SYNCH_DELAY_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**wDelay**	2	Number	Delay from the time that the packet should be sent. wDelay is expressed in microsecond units.

4.3.1.4 Still Image Trigger Control

This control notifies the device to begin sending still-image data over the relevant isochronous or bulk pipe. A dedicated still-image bulk pipe is only used for method 3 of still image capture. This control shall only be set while streaming is occurring, and the hardware shall reset it to the “Normal Operation” mode after the still image has been sent. This control is only required if the device supports method 2 or method 3 of still-image retrieval. See section 2.4.2.4 “Still Image Capture”.

Table 4-82 Still Image Trigger Control

Control Selector		VS_STILL_IMAGE_TRIGGER_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bTrigger**	1	Number	The setting for the Still Image Trigger Control: 0: Normal operation. 1: Transmit still image.

				2: Transmit still image via dedicated bulk pipe. 3: Abort still image transmission.

4.3.1.5 Generate Key Frame Control

This control is only supported by devices capable of streaming video with adjustable compression parameters, and support for this control is indicated in the VideoStreaming Header descriptor.

The Generate Key Frame Control is used to notify the video encoder on the device to generate a key frame in the device stream at its earliest opportunity. After the key frame has been generated, the device shall reset the control to the “Normal Operation” mode. This control is only applicable to video formats that support temporal compression (such as MPEG-2 Video), and while streaming is occurring. In all other cases, the device shall respond to requests by indicating a stall on the control pipe.

Table 4-83 Generate Key Frame Control

Control Selector		VS_GENERATE_KEY_FRAME_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bGenerateKeyFrame**	1	Number	The setting for the attribute of the addressed Generate Key Frame control: 0: Normal operation 1: Generate Key Frame

4.3.1.6 Update Frame Segment Control

This control is only supported by devices capable of streaming video with adjustable compression parameters, and support for this control is indicated in the VideoStreaming Header descriptor.

The Update Frame Segment Control is used to notify the video encoder on the device to encode the specified range of video frame segments with intra coding (no dependency on surrounding frames) at its earliest opportunity. A video frame segment corresponds to a group of macroblocks that can be decoded independently, such as a slice in MPEG Video, or a Group of Blocks in H.26x Video. This control is only applicable to video formats that support the concept of a video frame segment, and while streaming is occurring. In all other cases, the device shall respond to requests by indicating a stall on the control pipe.

The device will indicate the number of frame segments that it supports through the GET_MAX request, for which the device will indicate the maximum frame segment index supported in both the **bStartFrameSegment** and **bEndFrameSegment** fields. The minimum value for these fields shall always be zero. The resolution for this control shall always be set to 1.

Table 4-84 Update Frame Segment Control

Control Selector		VS_UPDATE_FRAME_SEGMENT_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_MIN, GET_MAX, GET_RES, GET_INFO, GET_DEF		
wLength		2		
Offset	Field	Size	Value	Description
0	**bStartFrameSegment**	1	Number	The zero-based index of the first frame segment in the range to update
1	**bEndFrameSegment**	1	Number	The zero-based index of the last frame segment in the range to update

4.3.1.7 Stream Error Code Control

This read-only control indicates the cause of a stream error that may arise during video or still-image transfer. In such cases, the device will update this control with the appropriate code to indicate the cause of the error.

The host software should send a GET_CUR request to this control to determine the error when one of the following events occurs:

- The Error bit in the video or still image payload header is set by the device (see section 2.4.3.2.2, "Sample Isochronous Transfers").
- The device issues a "Stream Error" interrupt to the host, with the source being the Stream Error Code Control (see section 2.4.2.2, "Status Interrupt Endpoint").
- A bulk video endpoint returns a STALL packet to the host in the data or handshake stage of the transaction.

For scenarios where the host is transmitting video data to the device, the host can not use the Error bit in the payload header to detect a device error. Therefore, in order to determine when a streaming error occurs, the host must rely on either a Control Change interrupt from the device or a bulk endpoint stall.

Table 4-85 Stream Error Code Control

Control Selector		VS_STREAM_ERROR_CODE_CONTROL		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bStreamErrorCode**	1	Number	0: No Error. 1: Protected content – This situation occurs if the data source device detects that the video or still-image data is

				protected and cannot be transmitted. In this case, empty packets containing only headers will be sent for the duration of the protected content. 2: Input buffer underrun – If the data source device is not able to supply data at the requested rate, it will transmit empty packets containing only headers for the duration of the buffer underrun. 3. Data discontinuity – Indicates a data discontinuity (arising from bad media, encoder errors, etc.) preceding the data payload in the current transfer. 4: Output buffer underrun – The data sink device is not being supplied with data at a sufficient rate. 5: Output buffer overrun – The data sink device is being supplied with data at a rate exceeding its buffering capabilities. 6: Format change – A dynamic format change event occurred. See section 2.4.3.6, "Device Initiated Dynamic Format Change Support". 7: Still image capture error - An error occurred during still-image capture.

Appendix A. Video Device Class Codes

A.1. Video Interface Class Code

Table A- 1 Video Interface Class Code

Video Interface Class Code	Value
CC_VIDEO	0x0E

A.2. Video Interface Subclass Codes

Table A- 2 Video Interface Subclass Codes

Video Subclass Code	Value
SC_UNDEFINED	0x00
SC_VIDEOCONTROL	0x01
SC_VIDEOSTREAMING	0x02
SC_VIDEO_INTERFACE_COLLECTION	0x03

A.3. Video Interface Protocol Codes

Table A- 3 Video Interface Protocol Codes

Video Protocol Code	Value
PC_PROTOCOL_UNDEFINED	0x00
PC_PROTOCOL_15	0x01

A.4. Video Class-Specific Descriptor Types

Table A- 4 Video Class-Specific Descriptor Types

Descriptor Type	Value
CS_UNDEFINED	0x20
CS_DEVICE	0x21
CS_CONFIGURATION	0x22
CS_STRING	0x23
CS_INTERFACE	0x24
CS_ENDPOINT	0x25

A.5. Video Class-Specific VC Interface Descriptor Subtypes

Table A- 5 Video Class-Specific VC Interface Descriptor Subtypes

Descriptor Subtype	Value
VC_DESCRIPTOR_UNDEFINED	0x00
VC_HEADER	0x01
VC_INPUT_TERMINAL	0x02
VC_OUTPUT_TERMINAL	0x03
VC_SELECTOR_UNIT	0x04
VC_PROCESSING_UNIT	0x05
VC_EXTENSION_UNIT	0x06
VC_ENCODING_UNIT	0x07

A.6. Video Class-Specific VS Interface Descriptor Subtypes

Table A- 6 Video Class-Specific VS Interface Descriptor Subtypes

Descriptor Subtype	Value
VS_ UNDEFINED	0x00
VS_INPUT_HEADER	0x01
VS_OUTPUT_HEADER	0x02
VS_STILL_IMAGE_FRAME	0x03
VS_FORMAT_UNCOMPRESSED	0x04
VS_FRAME_UNCOMPRESSED	0x05
VS_FORMAT_MJPEG	0x06
VS_FRAME_MJPEG	0x07
Reserved	0x08
Reserved	0x09
VS_FORMAT_MPEG2TS	0x0A
Reserved	0x0B
VS_FORMAT_DV	0x0C
VS_COLORFORMAT	0x0D
Reserved	0x0E
Reserved	0x0F
VS_FORMAT_FRAME_BASED	0x10
VS_FRAME_FRAME_BASED	0x11
VS_FORMAT_STREAM_BASED	0x12
VS_FORMAT_H264	0x13
VS_FRAME_H264	0x14
VS_FORMAT_H264_SIMULCAST	0x15

VS_FORMAT_VP8	0x16
VS_FRAME_VP8	0x17
VS_FORMAT_VP8_SIMULCAST	0x18

A.7. Video Class-Specific Endpoint Descriptor Subtypes

Table A- 7 Video Class-Specific Endpoint Descriptor Subtypes

Descriptor Subtype	Value
EP_UNDEFINED	0x00
EP_GENERAL	0x01
EP_ENDPOINT	0x02
EP_INTERRUPT	0x03

A.8. Video Class-Specific Request Codes

Table A- 8 Video Class-Specific Request Codes

Class-Specific Request Code	Value
RC_UNDEFINED	0x00
SET_CUR	0x01
SET_CUR_ALL	0x11
GET_CUR	0x81
GET_MIN	0x82
GET_MAX	0x83
GET_RES	0x84
GET_LEN	0x85
GET_INFO	0x86
GET_DEF	0x87
GET_CUR_ALL	0x91
GET_MIN_ALL	0x92
GET_MAX_ALL	0x93
GET_RES_ALL	0x94
GET_DEF_ALL	0x97

A.9. Control Selector Codes

A.9.1. VideoControl Interface Control Selectors

Table A- 9 VideoControl Interface Control Selectors

Control Selector	Value
VC _CONTROL_ UNDEFINED	0x00
VC_VIDEO_POWER_MODE_CONTROL	0x01
VC_REQUEST_ERROR_CODE_CONTROL	0x02
Reserved	0x03

A.9.2. Terminal Control Selectors

Table A- 10 Terminal Control Selectors

Control Selector	Value
TE_CONTROL_UNDEFINED	0x00

A.9.3. Selector Unit Control Selectors

Table A- 11 Selector Unit Control Selectors

Control Selector	Value
SU_CONTROL_UNDEFINED	0x00
SU_INPUT_SELECT_CONTROL	0x01

A.9.4. Camera Terminal Control Selectors

Table A- 12 Camera Terminal Control Selectors

Control Selector	Value
CT_CONTROL_UNDEFINED	0x00
CT_SCANNING_MODE_CONTROL	0x01
CT_AE_MODE_CONTROL	0x02
CT_AE_PRIORITY_CONTROL	0x03
CT_EXPOSURE_TIME_ABSOLUTE_CONTROL	0x04
CT_EXPOSURE_TIME_RELATIVE_CONTROL	0x05
CT_FOCUS_ABSOLUTE_CONTROL	0x06
CT_FOCUS_RELATIVE_CONTROL	0x07
CT_FOCUS_AUTO_CONTROL	0x08
CT_IRIS_ABSOLUTE_CONTROL	0x09
CT_IRIS_RELATIVE_CONTROL	0x0A
CT_ZOOM_ABSOLUTE_CONTROL	0x0B
CT_ZOOM_RELATIVE_CONTROL	0x0C
CT_PANTILT_ABSOLUTE_CONTROL	0x0D
CT_PANTILT_RELATIVE_CONTROL	0x0E
CT_ROLL_ABSOLUTE_CONTROL	0x0F

CT_ROLL_RELATIVE_CONTROL	0x10
CT_PRIVACY_CONTROL	0x11
CT_FOCUS_SIMPLE_CONTROL	0x12
CT_WINDOW_CONTROL	0x13
CT_REGION_OF_INTEREST_CONTROL	0x14

A.9.5. Processing Unit Control Selectors

Table A- 13 Processing Unit Control Selectors

Control Selector	Value
PU_CONTROL_UNDEFINED	0x00
PU_BACKLIGHT_COMPENSATION_CONTROL	0x01
PU_BRIGHTNESS_CONTROL	0x02
PU_CONTRAST_CONTROL	0x03
PU_GAIN_CONTROL	0x04
PU_POWER_LINE_FREQUENCY_CONTROL	0x05
PU_HUE_CONTROL	0x06
PU_SATURATION_CONTROL	0x07
PU_SHARPNESS_CONTROL	0x08
PU_GAMMA_CONTROL	0x09
PU_WHITE_BALANCE_TEMPERATURE_CONTROL	0x0A
PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL	0x0B
PU_WHITE_BALANCE_COMPONENT_CONTROL	0x0C
PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL	0x0D
PU_DIGITAL_MULTIPLIER_CONTROL	0x0E
PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL	0x0F
PU_HUE_AUTO_CONTROL	0x10
PU_ANALOG_VIDEO_STANDARD_CONTROL	0x11
PU_ANALOG_LOCK_STATUS_CONTROL	0x12
PU_CONTRAST_AUTO_CONTROL	0x13

A.9.6. Encoding Unit Control Selectors

Table A- 14 Encoding Unit Control Selectors

Control Selector	VALUE
EU_CONTROL_UNDEFINED	0x00
EU_SELECT_LAYER_CONTROL	0x01
EU_PROFILE_TOOLSET_CONTROL	0x02
EU_VIDEO_RESOLUTION_CONTROL	0x03
EU_ MIN_FRAME_INTERVAL_CONTROL	0x04
EU_ SLICE_MODE_CONTROL	0x05
EU_RATE_CONTROL_MODE_CONTROL	0x06

EU_AVERAGE_BITRATE_CONTROL	0x07
EU_CPB_SIZE_CONTROL	0x08
EU_PEAK_BIT_RATE_CONTROL	0x09
EU_QUANTIZATION_PARAMS_CONTROL	0x0A
EU_SYNC_REF_FRAME_CONTROL	0x0B
EU_LTR_BUFFER_ CONTROL	0x0C
EU_LTR_PICTURE_CONTROL	0x0D
EU_LTR_VALIDATION_CONTROL	0x0E
EU_LEVEL_IDC_LIMIT_CONTROL	0x0F
EU_SEI_PAYLOADTYPE_CONTROL	0x10
EU_QP_RANGE_CONTROL	0x11
EU_PRIORITY_CONTROL	0x12
EU_START_OR_STOP_LAYER_CONTROL	0x13
EU_ERROR_RESILIENCY_CONTROL	0x14

A.9.7. Extension Unit Control Selectors

Table A- 15 Extension Unit Control Selectors

Control Selector	Value
XU_CONTROL_UNDEFINED	0x00

A.9.8. VideoStreaming Interface Control Selectors

Table A- 16 VideoStreaming Interface Control Selectors

Control Selector	Value
VS_CONTROL_UNDEFINED	0x00
VS_PROBE_CONTROL	0x01
VS_COMMIT_CONTROL	0x02
VS_STILL_PROBE_CONTROL	0x03
VS_STILL_COMMIT_CONTROL	0x04
VS_STILL_IMAGE_TRIGGER_CONTROL	0x05
VS_STREAM_ERROR_CODE_CONTROL	0x06
VS_GENERATE_KEY_FRAME_CONTROL	0x07
VS_UPDATE_FRAME_SEGMENT_CONTROL	0x08
VS_SYNCH_DELAY_CONTROL	0x09

Appendix B. Terminal Types

The following is a list of possible Terminal types. This list is non-exhaustive and could be expanded in the future.

B.1. USB Terminal Types

These Terminal types describe Terminals that handle signals carried over the USB, through isochronous or bulk pipes. These Terminal types are valid for both Input and Output Terminals.

Table B- 1 USB Terminal Types

Terminal Type	Code	I/O	Description
TT_VENDOR_SPECIFIC	0x0100	I/O	A Terminal dealing with a signal carried over a vendor-specific interface. The vendor-specific interface descriptor must contain a field that references the Terminal.
TT_STREAMING	0x0101	I/O	A Terminal dealing with a signal carried over an endpoint in a VideoStreaming interface. The VideoStreaming interface descriptor points to the associated Terminal through the **bTerminalLink** field.

B.2. Input Terminal Types

These Terminal Types describe Terminals that are designed to capture video. They either are physically part of the video function or can be assumed to be connected to it in normal operation. These Terminal Types are valid only for Input Terminals.

Table B- 2 Input Terminal Types

Terminal Type	Code	I/O	Description
ITT_ VENDOR_SPECIFIC	0x0200	I	Vendor-Specific Input Terminal.
ITT_CAMERA	0x0201	I	Camera sensor. To be used only in Camera Terminal descriptors.
ITT_MEDIA_TRANSPORT_INPUT	0x0202	I	Sequential media. To be

			used only in Media Transport Terminal Descriptors.

B.3. Output Terminal Types

These Terminal types describe Terminals that are designed to render video. They are either physically part of the video function or can be assumed to be connected to it in normal operation. These Terminal types are only valid for Output Terminals.

Table B- 3 Output Terminal Types

Terminal Type	Code	I/O	Description
OTT_ VENDOR_SPECIFIC	0x0300	O	Vendor-Specific Output Terminal.
OTT_DISPLAY	0x0301	O	Generic display (LCD, CRT, etc.).
OTT_MEDIA_TRANSPORT_OUTPUT	0x0302	O	Sequential media . To be used only in Media Transport Terminal Descriptors.

B.4. External Terminal Types

These Terminal types describe external resources and connections that do not fit under the categories of Input or Output Terminals because they do not necessarily translate video signals to or from the user of the computer. Most of them may be either Input or Output Terminals.

Table B- 4 External Terminal Types

Terminal type	Code	I/O	Description
EXTERNAL_ VENDOR_SPECIFIC	0x0400	I/O	Vendor-Specific External Terminal.
COMPOSITE_CONNECTOR	0x0401	I/O	Composite video connector.
SVIDEO_CONNECTOR	0x0402	I/O	S-video connector.
COMPONENT_CONNECTOR	0x0403	I/O	Component video connector.

Appendix C. Video and Still Image Formats

C.1. Supported video and still image formats

This specification is designed to be format-agnostic, and will support any present or future video or still image format. The video and still image formats supported by the device are reported to the host software via Format descriptors (see section 3.9.2.3, "Payload Format Descriptors").

C.2. Proprietary video formats

New or proprietary video and still-image formats must be defined outside of this specification via Payload Format Specifications. . The host software will require a matching video encoder or decoder module.

Appendix D. Optical and Digital Zoom

Optical and digital zoom are functionally independent, so each will be discussed separately in the following sections. Although functionally independent, users will expect a single zoom control that integrates both.

D.1. Optical Zoom

Although lens groups can be quite sophisticated, this specification describes a simple two-lens system, which is sufficient to model optical zoom. Given objective and ocular lens focal lengths ($L_{objective}$ and L_{ocular}), magnification (M) can be calculated as follows:

$$M = \frac{L_{objective}}{L_{ocular}}$$

The objective lens is the one nearest the subject, while the ocular lens is the one nearest the viewer, or in our case, the camera sensor. A zoom lens varies the objective focal length.

Since magnification is a ratio of the objective and ocular focal lengths, the Units used to specify these focal lengths can be of any resolution supported by the device. In other words, these Units do not need to be specified in real physical units (millimeters or fractions of inches). The only requirement is that the two focal lengths are specified in the same units.

Note that when $L_{objective} < L_{ocular}$, the lenses are at a wide-angle setting. The subject will appear smaller than life, and the field of view will be wider.

L_{ocular} will be a device-specific constant value for each camera implementation, so it will be specified within the static Camera Terminal descriptor. If a camera implements an optical zoom function, $L_{objective}$ can vary within a specified range. In order to properly represent the range of magnification, $L_{objective}$ will be specified as a range L_{min} to L_{max}, which will also be specified within the static Camera Descriptor.

Finally, the variable position within the range of possible $L_{objective}$ values will be specified via a dynamic Camera Zoom Control, as integral values Z_{min}, Z_{max}, Z_{step}, and Z_{cur}. See sections 4.2.2.1.12, "Zoom (Absolute) Control" and 4.2.2.1.13, "Zoom (Relative) Control". This allows the Units of the objective lens focal length to be de-coupled from the Units used to control zoom. For simplicity, Z_{step} will be constrained to equal one (1). Values of L_{min} and L_{max} are constrained to be non-zero integral numbers; however, for the purpose of the following calculations, L_{cur} will be a real number.

Note: A typical choice for L_{ocular} would be half the length of a diagonal line of the imager (CCD, etc.), however there is no requirement for this value to be a direct physical measurement.

Given a known Z_{cur}, the current objective focal length (L_{cur}) can be calculated as follows:

$$L_{cur} = \frac{(Z_{cur} - Z_{min}) \quad * \quad (L_{max} - L_{min})}{(Z_{max} - Z_{min})} + L_{min}$$

From this, the relative magnification can be calculated as follows:

$$M = \frac{L_{cur}}{L_{ocular}}$$

Working from the opposite direction, given a known magnification (M), L_{cur} can be calculated as follows:

$$L_{cur} = M * L_{ocular}$$

From this, the current Zoom control value (Z_{cur}) can be calculated as follows:

$$Z_{cur} = \left\lfloor \frac{(L_{cur} - L_{min}) \quad * \quad (Z_{max} - Z_{min})}{(L_{max} - L_{min})} \right\rfloor + Z_{min}$$

To further simplify the calculations, Z_{min} can be constrained to be zero (0). The camera designer will choose the values and ranges of the remaining variables according to the capabilities of the device.

As an example, substituting some plausible values for each of these variables:

$L_{min} = 800$
$L_{max} = 10000$
$Z_{min} = 0$
$Z_{max} = 255$

The current Objective focal length (L_{cur}) can be calculated as follows:

$$L_{cur} = \frac{Z_{cur} \quad * \quad 9200}{255} + 800$$

The current Zoom control value (Z_{cur}) can be calculated as follows:

$$Z_{cur} = \left\lfloor \frac{(L_{cur} - 800) * 255}{9200} \right\rfloor$$

When choosing a camera sensor to match a lens system, the camera designer may need to consider a *multiplier* effect caused by a sensor that is smaller than the exit pupil of the ocular lens. This multiplier will not be represented explicitly in the USB Video Class specification, since its effect can be represented via adjustments to the $L_{objective}$ values.

Note The Z_{cur} value can be mapped to the physical lens position sensor control/status register.

D.2. Digital Zoom

Digital zoom is applied after the image has been captured from the sensor. Thus, digital zoom is independent of optical zoom, and is a function of either the Processing Unit or host post-processing. Although digital zoom is independent of optical zoom, users have come to expect that camera implementations will not apply digital zoom until full optical zoom has been realized. This will be enforced by the host software. There is no requirement for the device to enforce this, but it is recommended.

Digital zoom is represented as a *multiplier* of the current optical magnification of the captured image. In order to change the amount of digital zoom, the multiplier is changed through a range from 1 to some maximum value m_{max}, and m_{max} will be specified in the Processing Unit Descriptor. The position within the range of possible values of multiplier m will be expressed via a Processing Unit Digital Multiplier Control, as Z'_{min}, Z'_{max}, Z'_{step}, and Z'_{cur}. See section 4.2.2.3.16, "Digital Multiplier Control". This allows the multiplier resolution to be described by the device implementation. Z'_{step} will be constrained to equal one (1).

Given a known Z'_{cur}, the current multiplier m_{cur} can be calculated as follows:

$$m_{cur} = \frac{(Z'_{cur} - Z'_{min}) * (m_{max} - 1)}{(Z'_{max} - Z'_{min})} + 1$$

From this, and referring to the optical zoom values of L_{max} and L_{ocular} described in the previous section, the total magnification M' can be calculated as follows:

$$M' = \frac{L_{max}}{L_{ocular}} * m_{cur}$$

Working from the opposite direction, given a known magnification *M*, the multiplier m_{cur} can be calculated as follows:

$$m_{cur} = M' * \frac{L_{ocular}}{L_{max}}$$

From this, the current Digital Multiplier Control value (Z'_{cur}) can be calculated as follows:

$$Z'_{cur} = \left\lfloor \frac{(m_{cur} - 1) * (Z'_{max} - Z'_{min})}{(m_{max} - 1)} \right\rfloor + Z'_{min}$$

For simplicity, Z'_{min} can be constrained to be zero (0). The camera designer will choose the values and ranges of the remaining variables according to the capabilities of the device.

As an example, substituting some plausible values for each of these variables:

$m_{max} = 40$
$Z'_{min} = 0$
$Z'_{max} = 255$

The current multiplier (m_{cur}) can be calculated as follows:

$$m_{cur} = \frac{Z'_{cur} * 39}{255} + 1$$

The current Digital Zoom control value (Z'_{cur}) can be calculated as follows:

$$Z'_{cur} = \left\lfloor \frac{(m_{cur} - 1) * 255}{39} \right\rfloor$$

In addition to the Digital Multiplier Control, devices may optionally support a Digital Multiplier Limit control, allowing either the camera or the host to establish a temporary upper limit for the Z'_{cur} value. This control may be read-only if the limit can only be established via physical camera configuration. If this control is used to decrease the limit below the current Z'_{cur} value, the Z'_{cur} value will be adjusted to match the new limit.

D.3. Relationship between Optical and Digital Zoom

As mentioned in the preceding sections, users expect to use a single control on the device (or from within an application on the host) to traverse the entire range of optical and digital zoom. Further, users expect that digital zoom will not be active except at full optical zoom.

The following diagram illustrates the relationship between optical and digital zoom, and the constraints on the zoom control variables:

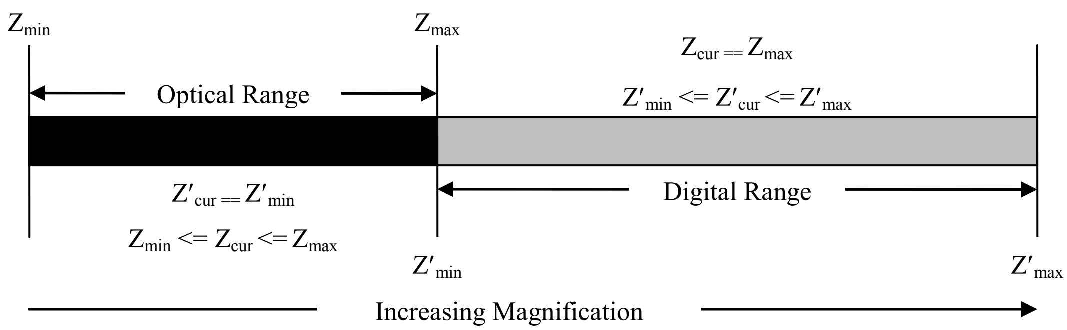


Figure 4-8 Relationship between Optical and Digital Zoom

D.4. Absolute vs. Relative Zoom

The equations and examples given in the previous sections describe independent, *absolute* optical and digital zoom controls. However, based on users' expectations that devices provide a single relative zoom control allowing them to move across the entire zoom range (from wide to telephoto and back again), many cameras will implement a *relative* zoom control that supports increasing and decreasing the zoom parameters without actually specifying the parameter values. Devices that allow only *relative* zoom control should still report the optical focal lengths and maximum digital multiplier in their respective descriptors, as well as maintain read-only *absolute* optical and digital zoom controls. This way, the host software will always be able to determine the current state of the zoom values.