Skype Encoding Camera Specification

Last saved: 2012-10-22	**Written by**: Seungkoo Yang, Simon Wilson, Stephan Lachowsky, Brent Weatherall	**Approved by**: Manrique Brenes
Status: ***Final***	**Version**: 2.2	
Filename: Skype Encoding Camera Specification v2.2.		
Security Classification: Public		

Summary of Revisions

Version	Date	Comments	Valid
1.0	2010-01-22	Initial draft for review	2010-01-22
1.1	2010-02-25	Second revision following external feedback	
1.2	2010-04-30	Third revision following internal review	
1.2.1	2010-06-16	Minor changes	
1.3	2010-09-20		
1.4	2011-05-27	Updated based on workshop proposals	2011-05-27
2.0	2011-08-10	Added new controls to maintain legacy compatibility.	2011-08-10
2.1	2011-09-15	Corrected version control expected output and bmControls(1) value. Added clarifications and corrected spelling and grammar mistakes.	2011-09-15
2.2	2012-10-22	Added control for obtaining audio configuration and added audio profile for Skype cameras.	2012-10-22

Table of Contents

Table of Illustrations

1 Introduction

1.1 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The fields of each control are prefixed with the size of memory they occupy. The prefixes are as follows:

b	Byte
w	Word (two bytes)
dw	Double Word (four bytes)

1.2 Scope

The purpose of this document is to define a method for transmitting compressed video and preview data for real-time streaming applications and to define a profile for acceptable Skype video and audio operations, specifically Skype video conferencing. To this end a Skype Transport Stream Format and a Skype Extension Unit (XU) are specified to accomplish low-latency compressed video transport on top of the existing standardized Universal Serial Bus (USB) Video Class (UVC) infrastructure. In addition to specifying a video transport mechanism and video operational profile, this document also defines an audio operational profile along with a mechanism of communicating information regarding the audio configuration of the camera and level of audio pre-processing for the purpose of improving Skype audio post-processing. For obtaining and sending audio related data that is not supported by the standard USB Audio Class (UAC), audio specific controls are defined in the Skype UVC XU. This specification only addresses cameras communicating over USB to the host. Cameras using Wi-Fi are not considered in this specification.

1.3 Definitions and Abbreviations

Skype Transport Stream

An encapsulation format providing timing and muxing for a number of payload streams.

Skype Transport Stream Packet

The smallest unit of a Skype Transport Stream, it is carried as a UVC frame.

Payload Stream

The reassembled sequence of payloads from a Skype Transport Stream, it contains one type of video codec, or vendor-specific data.

2 Requirements

2.1 Video Requirements of the Device

A device conforming to this specification MUST be able to operate in two video modes: legacy mode and Skype Transport Stream Mode.

2.1.1 Legacy Mode

Legacy mode is the default power-up operating mode of a device. The device MUST function in accordance with the standard UVC specification. The UVC video streaming interface intended for use as the main capture stream of the device MUST advertise MJPEG format support including a minimum resolution of 720p. MJPEG MUST be advertised because the UVC frame in Skype Transport Mode is negotiated as MJPEG even though the contents are custom to the Skype Transport Stream Packet Format. The device MAY support further formats and resolutions in legacy mode; however, they are not required.

2.1.2 Skype Transport Stream Mode

When operating in Skype transport stream mode the device always transfers a Skype transport stream despite the UVC format being negotiated as MJPEG.

When operating in Skype Transport Stream Mode, the device MAY use one video interface or two video interfaces. When using two video interfaces (dual endpoint mode), one of the interfaces MUST be allocated to the main video stream and MUST be capable of an H.264 payload stream supporting 720p constrained baseline profile level 3.1 at minimum. Optionally, the device MAY support other profiles, constraint sets, levels, resolutions and/or entropy coding modes.

The device MUST be capable of simultaneously streaming a secondary YUY2 preview stream at minimum resolutions of 160x120 and 160x90. Higher resolutions are permitted, but not required. The preview stream MAY be carried on a secondary UVC video streaming interface in accordance with the UVC specification. This means when using a secondary video interface for the preview stream in a raw format, the stream need not be packetized in the Skype transport stream format. If a secondary UVC video streaming interface is not available, then the device MUST be capable of including the preview as a YUY2 payload stream inside the Skype transport stream on the single UVC video streaming interface. The device MAY support other preview formats and resolutions in addition to the required YUY2 format; however, it is not required.

If the device supports H.264 encoding on the preview stream and the preview stream is offered on a secondary video interface, then the preview stream MUST send the H.264 video stream within the Skype Transport Stream Mode (which is encapsulated by MJPEG). This means the preview stream will advertise and be capable of streaming MJPEG in legacy mode.

The preview stream MUST have the same aspect ratio, zoom and cropping parameters as the main stream (hence the preview support for 160x120 and 160x90).

The device MUST support at least the following resolutions for the main stream:

- 160x90
- 160x120
- 320x180
- 320x240
- 640x360
- 640x480
- 1280x720

2.2 Audio Requirements of the Device

If a SECS video device includes audio microphones, then a device conforming to this specification MUST operate correctly without the use of the additional audio controls and support the following audio profile. It MUST also support the audio configuration extension unit described in this document.

2.2.1 Legacy Operations

Legacy operations are the default power-up operating mode of a device. The device MUST function in accordance with the standard UAC specification. Depending on the number of microphones and whether or not beam-forming is processed within the device, cameras will enumerate one, two, three or four audio endpoints. The endpoints MUST function normally if no Skype audio based UVC XU controls are issued. When beam-forming is performed by the camera, one endpoint is expected. In the absence of beam-forming the number of audio endpoints will vary. If audio data is passed through directly with no pre-processing, then the number of microphones will match the number of audio endpoints or channels (one, two, three or four). Some devices may perform mixing of internal signals but perform no beam-forming. This produces a left and right audio channel even if three or four microphones are present.

2.2.2 Skype Audio Profile and Extension Unit

A Skype Encoding Camera that includes microphones MUST support the following capabilities available through standard UAC controls where applicable:

- 16 KHz audio
- Analog gain controls (AGC)

Suggested non-mandatory capabilities:

- 24 bit audio
- 24 KHz audio
- 48 KHz audio

In addition to these standard capabilities, data about the camera's microphone spacing and general audio capabilities is needed to facilitate Skype audio quality post processing. If a camera does not perform beam-forming internally, the host should provide this processing in software (already included with the Skype SDK) for enhanced audio quality. To facilitate software beam-forming, data regarding the spacing of the camera's microphones is required. If the camera provides two or three microphones then one spacing value is needed. If the camera provides four microphones then two spacing values are needed. Information regarding the amount of pre-processing performed is also required. To facilitate correct processing, host software must also know if the audio data is down mixed to left and right or if the audio data is simply passed through. If audio is already beam-formed, then the host software does not need to perform any post-processing.

In addition to microphone spacing, Skype audio post-processing for SECS cameras also requires data concerning the presence of voice activity functionality, noise suppression functionality, the gain range of the microphones, the number of steps in the gain range and the recommended default gain step value for a conference call.

The audio configuration XU control details the specifics of how the audio data is expected from the camera.

2.3 Requirements of the Host

The following requirements apply to hosts which will operate a device in Skype transport mode. Other hosts will be able to operate a device in Legacy mode without awareness of this specification.

The host MUST be able to transition the device from legacy mode into Skype transport stream mode, through the manipulation of Skype Extension Unit controls. This transition happens at the behest of the host, and the host MUST be aware that the contents of the UVC stream will change from MJPEG to Skype transport. Configuring all supported streams with the XU controls transitions the interface from legacy mode to the Skype transport stream mode. The device is NOT guaranteed to transition to Skype transport stream mode until at least the main and preview stream IDs are configured using Skype Extension Unit controls. Use of the audio configuration control does not affect the audio data from the device. The data extrapolated from the audio configuration control only affects how the host (the Skype SDK in most cases) performs audio post processing. The host is NOT required to use the data provided by the audio control.

The host MUST support retrieving the H.264 payload stream on the main UVC video streaming interface.

The host MUST support the ability to:

- retrieve the preview stream from either a secondary UVC endpoint in raw format or in the Skype Transport Stream format
- de-mux the preview stream from the main endpoint when the preview stream is sent along with the main stream in single endpoint mode

3 Skype Transport Stream

The Skype transport stream is defined to provide timing information and multiplexing for a variety of encapsulated payload streams.

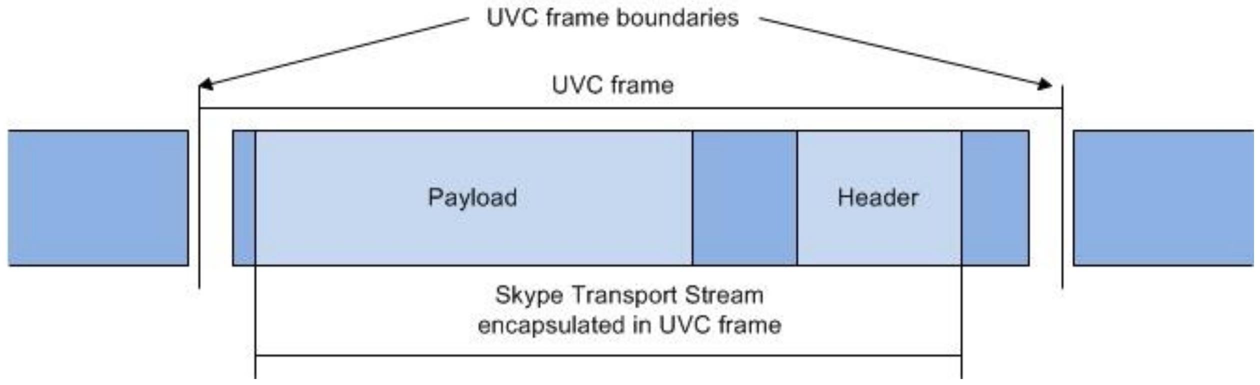

Illustration 1: Skype Transport Stream

The Skype transport stream is itself encapsulated as a UVC payload format, as defined in the following documents:

[2] USB_Video_Payload_Frame_Based_1.1
[3] USB_Video_Payload_Stream_Based_1.1
[4] USB_Video_Payload_MJPEG_1.1

Each Skype transport stream packet described in this section is a UVC frame as defined in [2],[3] and [4] and as such the total length of the packet is informed by the underlying UVC framing. The device MUST advertise a maximum frame rate large enough to support all possible payload frames without fragmentation.

3.1 Skype Transport Stream Packet Format

In this section all reference to individual values and fields are understood to be in big-endian format. The terms 'start', beginning' and 'earlier' refer to bits with smaller address values. For example, beginning refers to byte 0 which appears first on the USB wire. In illustrations, the smaller bit address values are on the left, and increase from left to right.

A Skype transport stream packet consists of three sections which are always found in the following order:

- A data section of arbitrary length containing the payload data of the encapsulated stream(s), with optional undefined padding between payloads.
- A header section containing
  - An arbitrary number of 160 bit stream headers (denoted as HDR0 … HDRn-1 in the diagram)
  - 32 bit value “N” denoting the number of stream headers

    - 32 bit constant magic number 0x534B5950 (SKYP)
- An optional undefined trailing bytes section, guaranteed to not contain the magic number.

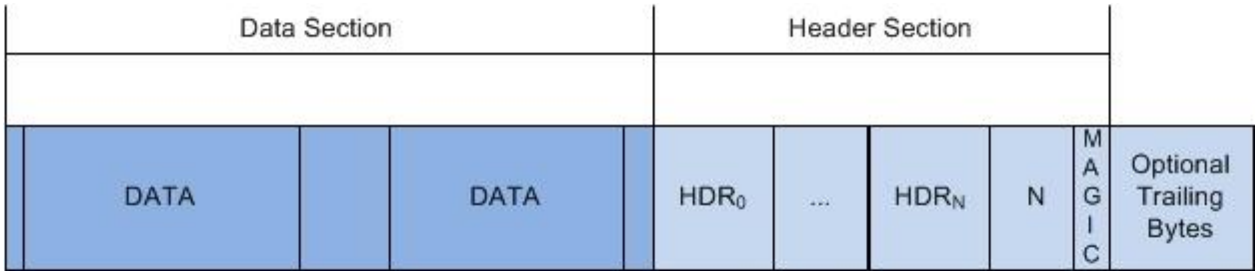

Illustration 2: Skype Transport Stream Packet

The host WILL NOT attempt to interpret data in the areas denoted as "undefined". This allows the device to optionally align payloads and headers or conform to additional external bit stream specifications. The host MUST NOT rely on any specific alignment, padding or private data; the host MAY, however, detect the presence of such to enable processing using more optimal methods.

3.1.1 Stream Header Format

The stream header defines the location, type and timing attributes of a single payload frame within a Skype transport stream packet. Stream headers are only found in the header section of the Skype transport stream packet.

The stream header has the following format:

Offset	Name	Bits	Description
0	PTS	64	Based on 90 kHz clock
64	StreamID	8	Identifier distinguishing between multiple streams
72	StreamType	8	Denote the type of the payload stream
80	Continuity/Sequence	16	Allows the host to detect lost payloads
96	PayloadOffset	32	Delimits the beginning of a payload
128	PayloadSize	32	Delimits the length of a payload

Table 3.1: Skype XU Payload Header Format

Fields in the stream header have the following meaning:

PTS – The Presentation Time Stamp MUST be derived from the video capture clock and is measured in 90 kHz ticks.

StreamID – Identifier denoting which stream a given payload being referenced belongs to.

- 0 – Refers to the main capture stream.
- 1 – Refers to the preview capture stream.
- 2-127 – Refer to video payloads that are not explicitly the capture or the preview stream.
- 128-255 – Refer to streams which are reserved for vendor use. The host should not interpret contents.

Note that if operating in dual endpoint mode and the preview interface is using the Skype transport stream packet format, the StreamIDs returned by the main and preview video endpoints in the same UVC frame MUST be mutually exclusive.

In the dual endpoint case if both interfaces use the Skype transport stream format, then the full range of supported stream IDs MAY appear on either interface.

StreamType – Identifies the format of the payload being referenced.

- 0 – YUY2 (see Video References)
- 1 – NV12 (see see Video References)
- 2 – MJPEG (see see Video References)
- 3– H.264 (see see Video References)
- 4-127 – Reserved by Skype for future use.
- 128-255 – Vendor specific types. The host should not interpret contents.

Continuity/Sequence – An unsigned counter used by the host to detect lost frames, this MUST increment for every new payload belonging to the given StreamID. The counter will wrap back to zero after reaching its maximum representable value.

PayloadOffset – The beginning of the payload as the number of bytes from the start of the data section.

PayloadSize – The length of the payload in bytes.

3.1.2 YUY2 Payload Format

When utilizing the Skype transport stream packet format (section 3.1), a YUY2 payload (StreamType 0) consists of a 32 bit header comprised of the YUY2 frame's width (16 bits), followed by the YUY2 frame's height (16 bits) , followed by one frame of YUY2 data. Width is constrained to be an even number.

This yields a required PayloadSize of 2 + 2 + 2*(Width*Height) bytes. The YUY2 frame data format follows:

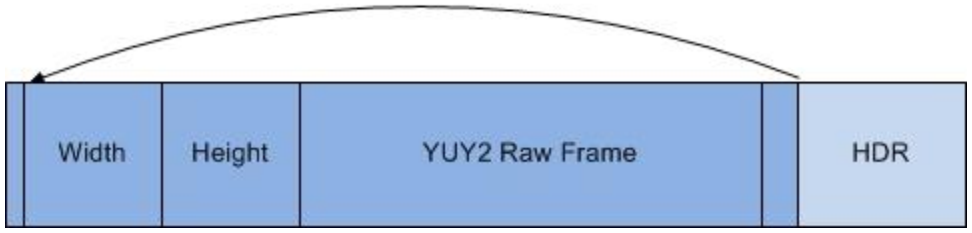

Illustration 3: YUY2 Payload Format

3.1.3 NV12 Payload Format

When utilizing the Skype transport stream packet format (section 3.1), an NV12 payload (StreamType 0) consists of a 32 bit header comprised of the NV12 frame's width (16 bits), followed by the NV12 frame's height (16 bits) , followed by one frame of NV12 data. Width and height are constrained to be even numbers.

This gives a required PayloadSize of 2 + 2 + 3*(Width*Height)/2 bytes. The NV12 frame data format follows:

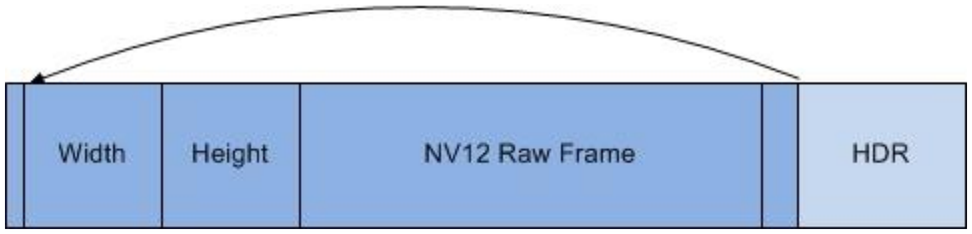

Illustration 4: NV12 Payload Format

3.1.4 MJPEG Payload Format

For a Skype transport stream packet sending a MJPEG payload (StreamType 2) the payload consists of an entire frame of motion jpeg data. The PayloadSize is variable. The MJPEG frame data format is as follows:

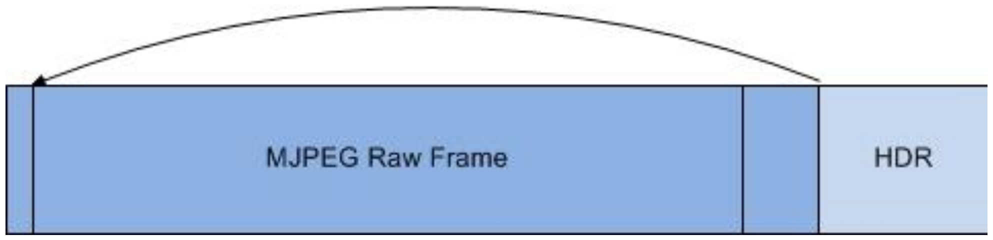

Illustration 5: MJPEG Payload Format

3.1.5 H.264 Payload Format

In both the single and dual endpoint cases, sending H.264 payloads (StreamType 3) requires use of the Skype Transport Stream Packet Format (section 3.1). For example, to configure the preview stream as H.264 using dual endpoints, the preview stream MUST send the H.264 payload within the Skype transport stream packet container. An H.264 payload MAY consist of multiple NAL units or a single NAL unit. If an H.264 payload consists of multiple NAL units, all NAL units are considered a single payload. This means all NAL units in a single frame have one header. The PayloadSize is considered the cumulative sum of the lengths of each individual NAL unit. If there are multiple NAL units in a single payload, each individual NAL unit should NOT have multiple headers. See the examples below:

Example of proper H.264 payload within a Skype UVC XU frame containing multiple NAL units:

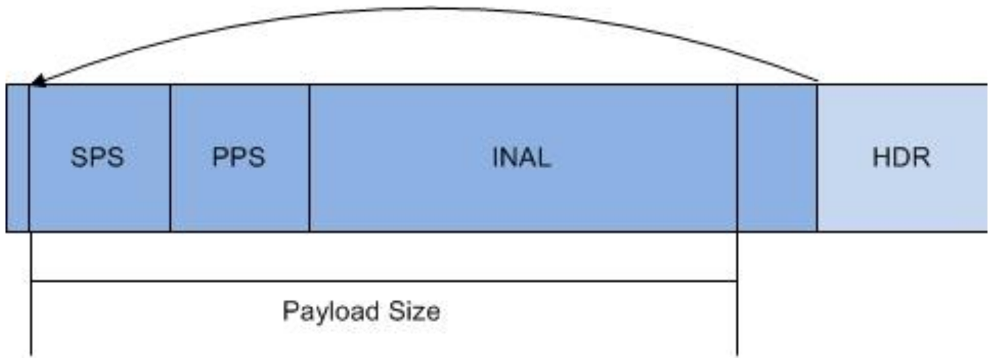

Illustration 6: Correct UVC Frame with One Header

Example of an incorrect H.264 payload within a Skype UVC XU frame containing multiple NAL units:

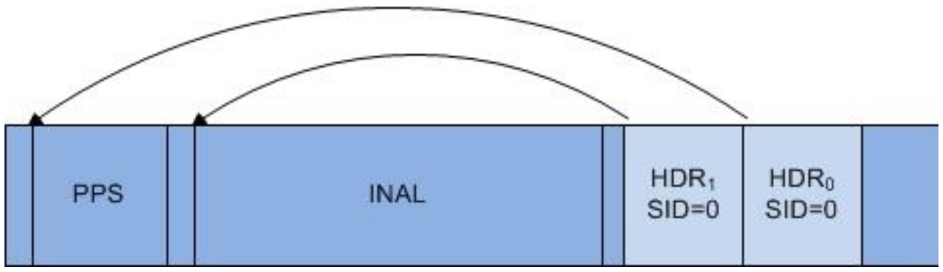

Illustration 7: Incorrect UVC Frame with Multiple Headers

All H.264 streams are assumed to use the constrained baseline profile. Explicitly setting the H.264 stream to any other profile is out of the scope of this XU.

3.2 Skype Transport Stream Decoding Process

The host will decode each Skype transport stream packet by executing the following algorithm:

1. Start at the end of the Skype transport stream packet and scan backwards until the magic number is found. The magic number delimits the end of the header section and the beginning of the optional trailing bytes section. If the magic number is not found, no further processing is performed and the packet is discarded.
2. Read the 32-bit value "N" immediately preceding the magic number. "N" is the number of headers present in the header section, which implicitly determines the end of the data section and the beginning of the header section. If "N" defines a header section length such that it does not fit within the Skype transport stream packet, no further processing is performed and the packet is discarded.
3. For each header in the header section, starting from the beginning:
    a. Read the PayloadOffset and PayloadSize, denoting the byte offset and byte length respectively within the data section. If the payload delimited by PayloadOffset and PayloadSize does not completely reside within the data section of the packet, no further processing is performed and the packet is discarded.
    b. The delimited payload data, along with the remaining header information is given to the [next level] for processing.

Any implementation of this algorithm is valid; provided that error conditions leading to the packet being discarded, and the order of header processing is maintained.

Note: The order in which payloads are processed is determined by the order of the headers in the header section. This may not necessarily correspond to the order that the payloads are laid out in the data section.

Note: A Skype transport stream packet can vary widely in complexity; it MAY contain multiple payloads per stream for multiple payload streams, or it MAY contain a single payload from a single stream. Host implementations should take care to ensure that they are able to handle all streams, as no logical upper limit on packet size or number of payloads is defined. The number of payloads defined is physically limited by the bit size of the StreamID field inasmuch as this field is 8 bits and can only support 256 different stream IDs.

4 Skype UVC XU Descriptor

To detect the presence of a camera supporting the Skype XU, the camera MUST advertise the following XU Descriptor:

Name	Value	Hex Value	Size	Description
bLength	29	0x1D	1	Length of the descriptor itself.
bDescriptorType	36	0x24	1	CS_INTERFACE – Identifies this descriptor as a class specific interface.
bDescriptorSubtype	6	0x06	1	VC_EXTENSION_UNIT – Identifies the subtype of this descriptor as an extension unit.
bUnitID	7	0x07	1	Unique identifier to reference the extension unit when sending and receiving XU control data.
guidExtensionCode	BD5321B4-D635-CA45-B203-4E0149B301BC	N/A	16	Globally unique identifier of the extension unit in the context of the UVC driver. This field guarantees the device supports the Skype XU. This value is also sent to the UVC driver to add the XU and its associated controls the UVC driver. Note that some platforms require byte swapping of various fields when sending the GUID to the UVC driver. Care should be taken to ensure the proper byte format for the target platform.
bNumControls	32	0x20	1	Number of controls allocated to the XU.

Name	Value	Hex Value	Size	Description
bNrInPins	1	0x01	1	Number of input pins of this Unit.
baSourceID(0)	3	0x03	1	ID of the Terminal or Unit connected to the first input pin. There is one entry in this array for each input pin specified by bNrInPins.
bmControlSize	4	0x04	1	Size of the active controls bitmap specified by the bmControls array.
bmControls(0)	0xBF	0xBF	1	Bitmap specifying which controls are active. The position of each bit correlates to an XU control selector.
bmControls(1)	0x3F	0x3F	1	Bitmap specifying which controls are active. The position of each bit correlates to an XU control selector.
bmControls(2)	0x80	0x80	1	Bitmap specifying which controls are active. The position of each bit correlates to an XU control selector.
bmControls(3)	0x03	0x03	1	Bitmap specifying which controls are active. The position of each bit correlates to an XU control selector.
iExtension	0	0x00	1	Index of a string descriptor that describes this extension unit.

5 Skype UVC XU Controls

The controls and parameters described in this section will be in little-endian format, in keeping with USB conventions. The XU controls provide a common interface not available in the standardized UVC driver allowing for specialized camera interfacing specific to using the Skype application. All XU control requests are initiated on the first interface whether that interface is the preview or main stream.

Name	Select	Description
Version	1	Specification version the device is conforming to
LastError	2	Holds the most recent error
FirmwareDays	3	Firmware build date, as days since the millennium
StreamID	4	Control selecting the stream to configure/control
EndpointSetting	5	Control distinguishing between the main and preview streams when operating in dual endpoint mode.
AudioConfiguration	6	Returns the microphone spacing and audio processing characteristics of the device.
	7	Reserved for future Global Class Controls
StreamFormatProbe	8	Query Supported Stream Formats
StreamFormatCommit	9	Set Supported Stream Formats
StreamFormatProbeType	10	Query minimum and maximum supported stream types
StreamFormatProbeWidth	11	Query minimum and maximum supported frame width based on the last queried bStreamType.
StreamFormatProbeHeight	12	Query maximum and minimum supported frame height based on the last queried bStreamType
StreamFormatProbeFrameInterval	13	Query maximum and minimum supported frame intervals based on the last queried value of bStreamType
StreamFormatProbeBitrate	14	Query maximum and minimum supported bit rates based on the last queried value of bStreamType

Name	Select	Description
	15-17	Reserved for future Stream Dependent Configuration Controls
	18-23	Reserved for future Stream Type Dependent Configuration Controls
BitrateControl	24	Adjust Bitrates Dynamically
FrameIntervalControl	25	Adjust Frame Intervals Dynamically
GenerateKeyFrameControl	26	Force Key Frames Dynamically
	27-31	Reserved for future dynamic controls
	32-X	Vendor may specify private controls in this range

5.1 Global Class Controls

Controls in this class reflect status which is of a device wide nature, or settings which change device wide state and behavior. In addition to the mandatory requests, all controls MUST conform to the requirements for XU controls from the standard UVC specification.

5.1.1 Version Control

This control queries the specification version implemented by the device.

Control Selector		**Version**		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bVersion**	1	Number	The version of this specification being implemented 0x21: Version 2.1

Calling this control will always set current error returned by LastError (section 5.1.3) to '0'.

5.1.2 Firmware Days Control

This control queries the number of days since Jan 1, 2000 until the release of the firmware.

Control Selector		**FirmwareDays**		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		2		
Offset	Field	Size	Value	Description
0	**wFirmwareDays**	2	Number	The firmware build date, as the number of days since January 1, 2000.

Calling this control will always set current error returned by LastError (section 5.1.3) to '0'.

5.1.3 Last Error Control

This control queries the most recent error detected when setting XU controls.

Control Selector		**LastError**		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bLastError**	1	Number	Numeric code indicating the error condition. 0 – NoError 1 – NotAllowed 2 – InvalidArgument 3 – NotSupported

Calling this control will always set current error returned by LastError (section 5.1.3) to '0'. This means, calling the LastError control effectively returns the last detected error and resets the last detected error. Calling this control two or more times consecutively will always return '0'.

In the event a control sets multiple errors, only one error code is returned. The error codes are ordered by priority. NotAllowed ('1') is the highest priority error. The error priority decreases as the error number increases.

5.1.4 StreamID Control

This control queries supported the stream IDs of the device and sets the current stream ID for all stream specific controls. The stream identifier is used by all subsequent controls to indirectly control multiple streams. The GET_MIN request MUST always return zero, the GET_MAX request MUST return the maximum video stream id supported. The SET_CUR request sets **bStreamID** to the desired stream. **bStreamID** indicates which stream all XU stream specific controls are manipulating.

Control Selector		**StreamID**		
Mandatory Requests		GET_MIN, GET_MAX, SET_CUR		
wLength		1		
Offset	Field	Size	Value	Description
0	**bStreamID**	1	Number	StreamID being configured/controlled.

The StreamID control is global for both endpoints. Note that in dual endpoint mode using GET_MAX returns the maximum supported StreamID between both endpoints. If the control returns a GET_MAX value higher than 1 (preview stream) and the preview stream does NOT support the Skype transport stream it (i.e. it only operates in legacy mode), then it is implied StreamID 1 is not included in the range reported by this control.

Calling SET_CUR on this control with an unsupported stream ID will set the current error returned by LastError (section 5.1.3) to InvalidArgument ('2').

5.1.5 EndpointSetting Control

The EndpointSetting control informs the host software of the endpoint configuration of the video streaming device. If there is only one endpoint, the control returns the value 0. If there are two endpoints, either the main endpoint or the preview endpoint will be first in the endpoint enumeration. If the main endpoint is first, the control returns 1. If the preview endpoint is first, the control returns 2. Note, in either configuration the first endpoint is ALWAYS the video control interface to which all XU commands are issued.

Control Selector		**InterfaceType**		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		1		
Offset	Field	Size	Value	Description
0	**bInterfaceType**	1	Number	Numeric code enumerating the configuration of the video interfaces. 0 – Single Endpoint 1 – Dual Endpoint, main first and preview second 2 – Dual Endpoint, preview first and main second

Calling this control will always set current error returned by LastError (section 5.1.3) to '0'.

5.1.6 AudioConfiguration Control

The AudioConfiguration control informs the host software of the microphone spacing characteristics, level of audio pre-processing, gain range in decibels, number of supported gain range steps and the recommended default gain step value of the device for a conference call. This control returns a structure containing data that communicates the number of microphones, level of pre-processing, microphone spacing values measured in millimeters, flag values for the presence of voice activity and noise suppression availability, gain range in decibels, number of gain range steps and the recommended default gain step value to start a video call. Only the GET_CUR requested is supported.

With respect to microphone spacing, if a device uses three microphones, it is assumed these microphones are configured with one middle microphone and then two evenly spaced flanking microphones. The microphone processing field of the structure is defined as the following enumeration:

- 0 – Beam-forming performed internally (implies one channel or endpoint)
- 1 – Four microphones down-mixed (implies two channels or endpoints)
- 2 – Three microphones down-mixed (implies two channels or endpoints)
- 3 – Four microphones pass through (implies four channels or endpoints)
- 4 – Three microphones pass through (implies three channels or endpoints)
- 5 – Two microphones pass through (implies two channels or endpoints)

If the device performs internal beam-forming then both microphone spacing values are returned as -1. If the device does not perform internal beam-forming and uses two microphones, the first value in the returned structure is the spacing in millimeters between the two microphones (by default considered the outer microphones) and the second value is -1. If the device does not perform internal beam-forming and uses three microphones, the first value in the in the returned structure is the spacing in millimeters between the two outer flanking microphones as the third microphone is assumed to be centered. If the device does not perform internal beam-forming and uses four microphones, the first value in the returned structure is the spacing in millimeters between the outer microphones and the second value is the spacing in millimeters between the inner microphones. The inner microphones are assumed to be evenly spaced with respect to the outer microhpones.

The flag register contains flags indicating if the device's audio processing includes voice activity and noise suppression. A '1' in the corresponding bit indicates the feature is present in the device and a '0' indicates the feature is not present in the device. The flag register is laid out as depicted below:

Bits 2-15: Not Used	Bit 1: Noise Suppresion	Bit 0: Voice Activity

Gain range is returned as an integer. The number of gain steps value is also returned as an integer. Each gain step MUST be an equal amount of decibels. The host will determine the decibel amount of each step by dividing the gain range into an equal number of slices based on the number of steps. The number of gain steps value MUST NOT include 0. For example, if the gain range is 30 and each step is 2 decibels, then the number of gain step should be 15, NOT 16. A step of zero will indicate zero decibels.

The recommended default step value is returned as an integer. This value indicates the device's recommended default gain step to begin a video call.

Control Selector		**InterfaceType**		
Mandatory Requests		GET_CUR, GET_INFO		
wLength		12		
Offset	Field	Size	Value	Description

0	**wMicConfiguration**	2	Number	Numeric code enumerating the microphone configuration and indicating the level of audio pre-processing: 0 – Beam-formed 1 – Four mics down-mixed 2 – Three mics down-mixed 3 – Four mics pass through 4 – Three mics pass though 5 – Two mics pass through
2	**wOuterSpacing**	2	Number	Spacing in millimeters between the outer microphones. -1 – Indicates beam-forming is performed internally.
4	**wInnerSpacing**	2	Number	Spacing in millimeters between the inner microphones. -1 – Indicates the device only has two or three microphones. Note, when wOuterSpacing is returned as -1, then wInnerSpacing has no meaning and MUST be -1.
6	**wGainRangeDB**	2	Number	The maximum gain range of the device's microphones in decibels. This value represents the highest possible decibel gain possible from the microphone array. This value MUST not be less than zero.
8	**bStepCount**	1	Number	Number of evenly spaced gain steps supported by the device. This value MUST not be less than zero. This value MUST not include zero.
9	**bDefaultStep**	1	Number	The recommended default gain step value for making a video call. This value MUST not be less than zero.
10	**wFeatureSet**	2	Flag Register	Available audio features of the device. '1' indicates feature is available and '0' indicates the feature is not available. Bit 0 – Voice Activity Bit 1 – Noise Suppresion Bits 2 – 15 – Unused

The device MUST respond to this control any time it is requested, even if streaming is currently active. Calling this control will always set the current error returned by LastError (section 5.1.3) to '0'.

Stream Dependent Configuration Class Controls

Controls in this class are dependent on the current value of **bStreamID** (set by the StreamID control) to decide which payload stream they are configuring.

5.1.7 Stream Format Probe and Commit Controls

These controls have an identical layout.

The device SHOULD return GET_MIN and GET_MAX values such that for all supported configurations, each field value is within the range returned. These controls operate on the current stream set by **bStreamID** from the StreamID control.

If using a legacy system, it is possible using the StreamFormatProbe command with GET_MIN and GET_MAX will not operate correctly. In the event this occurs, the controls in sections 5.2.2 through 5.2.6 can be utilized for probing the maximum and minimum supported values for each stream type on field at a time.

Control Selector		**StreamFormatProbe & StreamFormatCommit**		
Mandatory Requests		GET_CUR, GET_MAX, GET_MIN, SET_CUR		
wLength		13		
Offset	Field	Size	Value	Description
0	**bStreamType**	1	Number	As per StreamFormat listing.
1	**wWidth**	2	Number	Frame width in pixels.
3	**wHeight**	2	Number	Frame height in pixels.
5	**dwFrameInterval**	4	Number	Frame interval in 100 ns units
9	**dwBitrate**	4	Number	Bitrates in bits per second

If the values submitted to the StreamFormatProbe control are not supported or the stream ID specified by bStreamID is not supported, the error returned by LastError (section 5.1.3) is set to NotSupported.

If the values submitted to the StreamFormatCommit control are not supported by the device or the stream ID specified by bStreamID is not supported, the error returned by LastError (section 5.1.3) is set to NotAllowed.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

5.1.8 StreamFormatProbeType Control

This control is intended to allow support for legacy system drivers that cannot support returning minimum and maximum values for complex data sets. The intended use of this control is to discover the maximum and minimum support stream types as defined in section 3.1.1.

This control is used in concert with remaining singleton stream format probe controls. Using this control with SET_CUR, sets the stream type that applies to the remaining probe controls. For example, sending stream type 3 (H.264) with SET_CUR and then calling the width and height probe controls with GET_MAX will return the maximum supported width and height for the steam type of 3 (H.264).

Control Selector		**StreamFormatProbeType**		
Mandatory Requests		GET_MAX, GET_MIN, SET_CUR, GET_CUR		
wLength		1		
Offset	Field	Size	Value	Description
0	**bStreamType**	1	Number	As per StreamFormat listing.

If the values submitted to the StreamFormatProbeType control are not supported or the stream ID specified by bStreamID is not supported, the error returned by LastError (section 5.1.3) is set to NotSupported.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

Note, that mixing and matching values obtained using GET_MIN/MAX for stream type settings are NOT guaranteed to result in a valid configuration. The intent of obtaining maximum and minimum values is to give the host a sense of the capability range of the camera.

5.1.9 StreamFormatProbeWidth Control

This control is intended to allow support for legacy system drivers that cannot support returning minimum and maximum values for complex data sets. The intended use of this control is to discover the maximum and minimum supported width per stream type.

This control is used in concert with StreamFormatProbeType control. Whichever stream type is set first with the SteamFormatProbeType control determines which stream type the returned width applies to.

Control Selector		**StreamFormatProbeWidth**		
Mandatory Requests		GET_MAX, GET_MIN		
wLength		1		
Offset	Field	Size	Value	Description
0	**wWdith**	2	Number	Frame width in pixels.

This control is a read only function and there are no anticipated error conditions generated by the function.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

5.1.10 StreamFormatProbeHeight Control

This control is intended to allow support for legacy system drivers that cannot support returning minimum and maximum values for complex data sets. The intended use of this control is to discover the maximum and minimum supported height per stream type.

This control is used in concert with StreamFormatProbeType control. Whichever stream type is set first with the SteamFormatProbeType control determines which stream type the returned height applies to.

Control Selector		**StreamFormatProbeHeight**		
Mandatory Requests		GET_MAX, GET_MIN		
wLength		1		
Offset	Field	Size	Value	Description
0	**wHeight**	2	Number	Frame height in pixels.

This control is a read only function and there are no anticipated error conditions generated by the function.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

5.1.11 StreamFormatProbeFrameInterval

This control is intended to allow support for legacy system drivers that cannot support returning minimum and maximum values for complex data sets. The intended use of this control is to discover the maximum and minimum supported frame interval per stream type. Note, the returned min/max frame interval is related to the min/max resolution (width and height) respectively.

This control is used in concert with StreamFormatProbeType control. Whichever stream type is set first with the SteamFormatProbeType control determines which stream type the returned frame interval applies to.

Control Selector		**StreamFormatProbeFrameInterval**		
Mandatory Requests		GET_MAX, GET_MIN		
wLength		1		
Offset	Field	Size	Value	Description
0	**dwFrameInterval**	4	Number	Frame interval in 100 ns units.

This control is a read only function and there are no anticipated error conditions generated by the function.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

5.1.12 StreamFormatProbeBitrate

This control is intended to allow support for legacy system drivers that cannot support returning minimum and maximum values for complex data sets. The intended use of this control is to discover the maximum and minimum supported bit rate per stream type. Note, the returned min/max bitrate is related to the min/max resolution (width and height) respectively.

This control is used in concert with StreamFormatProbeType control. Whichever stream type is set first with the SteamFormatProbeType control determines which stream type the returned bit rate applies to.

Control Selector		**StreamFormatProbeBitrate**		
Mandatory Requests		GET_MAX, GET_MIN		
wLength		1		
Offset	Field	Size	Value	Description
0	**dwBitrate**	4	Number	Bit rate in bits per second.

This control is a read only function and there are no anticipated error conditions generated by the function.

Upon success, the value returned by the LastError control (section 5.1.3) is set to NoError.

Stream Dependent Dynamic Class Controls

Controls in this class are dependent on the current value of **bStreamID** (set by the StreamID control) to decide which payload stream they are configuring. Controls in this class are REQUIRED to work while the device is actively streaming data. The payload stream format MUST already be negotiated prior to setting these controls.

5.1.13 Dynamic BitrateControl Control

The BitrateControl Control sets the desired bitrate of the active stream (designated by **bStreamID** set by the StreamID control).

Control Selector		**BitrateControl**		
Mandatory Requests		SET_CUR		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwBitrate**	4	Number	The bitrates in bits per second.

The behavior of the value returned by the LastError control (section 5.1.3) based on the input value, **dwBitrate,** is as follows:

- **dwBitrate** is out of range of the camera's physical capabilities: NotSupported.
- **dwBitrate** is within a valid range; however, the camera is unable to change to the requested bit rate for any other reason: NotAllowed.
- Success: NoError.

5.1.14 Dynamic FrameIntervalControl Control

The FrameIntervalControl Control determines the time interval in 100 nanosecond chunks of the active stream (designated by **bStreamID** set by the StreamID control).

Control Selector		**FrameIntervalControl**		
Mandatory Requests		SET_CUR		
wLength		4		
Offset	Field	Size	Value	Description
0	**dwFrameInterval**	4	Number	The frame interval in 100 ns units.

The behavior of the value returned by the LastError control (section 5.1.3) based on the input value, **dwFrameInterval** is as follows:

- If **dwFrameInterval** is set to a value the camera cannot or will not support: NotAllowed.
- Success: NoError.

5.1.15 GenerateKeyFrame Control

The GenerateKeyFrame control forces the device to generate a key frame or IDR frame for the active stream (designated by **bStreamID** set by the StreamID control). The device MUST generate a key frame when this control is sent with a value of '1'. The host should send this command one time for each forced key frame desired. Calling the control with any other input value has no effect.

Control Selector		**GenerateKeyFrameControl**		
Mandatory Requests		SET_CUR		
wLength		1		
Offset	Field	Size	Value	Description
0	**bGenerateKeyFrame**	1	Number	Request the device to generate a Key Frame: 1 - Generate key frame

If **bGenerateKeyFrame** is set to any value other than '1', the value returned by the LastError control (section 5.1.3) is set to InvalidArgument.

If the device is unable to generate a key frame for any reason, the value returned by the LastError control (section 5.1.3) is set to NotAllowed.

Otherwise, the value is set to NoError.

6 Video Operational Model

The device will be configured according to the following model and assumes the intent is to use the Skype XU:

Legacy Mode

1. Using standard UVC controls, the host probes for video devices supporting the Skype XU by looking for the XU Descriptor GUID or attempting to use any of the XU controls on the desired video interface with a successful result.
2. Host selects an available device based on a host specific priority scheme.
3. Host will operate the device in a UVC compliant manner
4. Global class controls MAY be queried by the host in preparation for transitioning to Skype transport stream mode (e.g. Query version control, read FirmwareDays/VID/PID to lookup known bugs/quirks)

Stream Format Configuration(s)

1. Host adds the XU to the UVC driver
2. If a device with two video endpoints is chosen, the host calls the EndpointSetting control to determine which endpoint corresponds to the main capture stream and which endpoint corresponds to the preview stream
3. Host ensures the main endpoint is configured as MJPEG with a minimum 720p resolution (1280x720)
4. Host calls the StreamID control to determine the range of supported stream IDs
5. Host MAY now use Stream Dependent Configuration Class Controls
6. Host sets the StreamID control to the desired stream to run a probe/commit phase with.
7. Host goes through a stream configuration probe & commit phase for all possible StreamIDs
8. The stream format is now considered set for each StreamID.

Dynamic Configuration

1. Host MAY now set StreamID dynamically to any StreamID configured in (Stage 2.)
2. Host MAY now use Stream Dependent Dynamic Class controls to adjust parameter or generate key frames during streaming

Stream Format Configuration (Stage 2.) will occur for each payload stream supported by the device. The host will configure StreamIDs incrementally starting from StreamID zero.

Note that in dual endpoint mode, the preview stream MAY not use the Skype transport stream format. If the preview stream is separate and does not advertise MJPEG then it is assumed to operate in legacy mode. If the preview stream is separate and advertises MJPEG, the StreamID control MUST be used to determine if the preview stream supports the Skype transport stream format. If setting the StreamID control **bStreamID** to 1 causes an error (checked with the LastError control) then the preview stream will only operate in legacy mode. Note that operating the preview endpoint (in dual endpoint mode) in Skype transport mode will generally not occur. Situations (in dual endpoint mode) where the preview endpoint send multiple streams via the Skype transport format are not likely to occur and will normally be supported by vendor specific host software. Skype generally utilizes only a YUY2 preview stream and H.264 main stream.

The following functional stages are also illustrated below:

- Device is ready to begin legacy streaming during (Stage 1.)
- Device is ready to begin Skype transport streaming at any point after (Stage 2.) Note that each endpoint MUST be configured using the StreamFormatCommit control before it will send a Skype transport stream packet frame. If the commit control is not used on a streamID, that StreamID will not appear in the packet. For example, if StreamID 0 is NOT configured using the StreamFormatCommit, the main endpoint will send standard video data in its default format instead of a Skype transport stream packet. If at least the preview and main streams are NOT configured, the device MAY not transition to Skype transport mode.
- Device can perform dynamic configuration before/during/after streaming (Stage 3.)

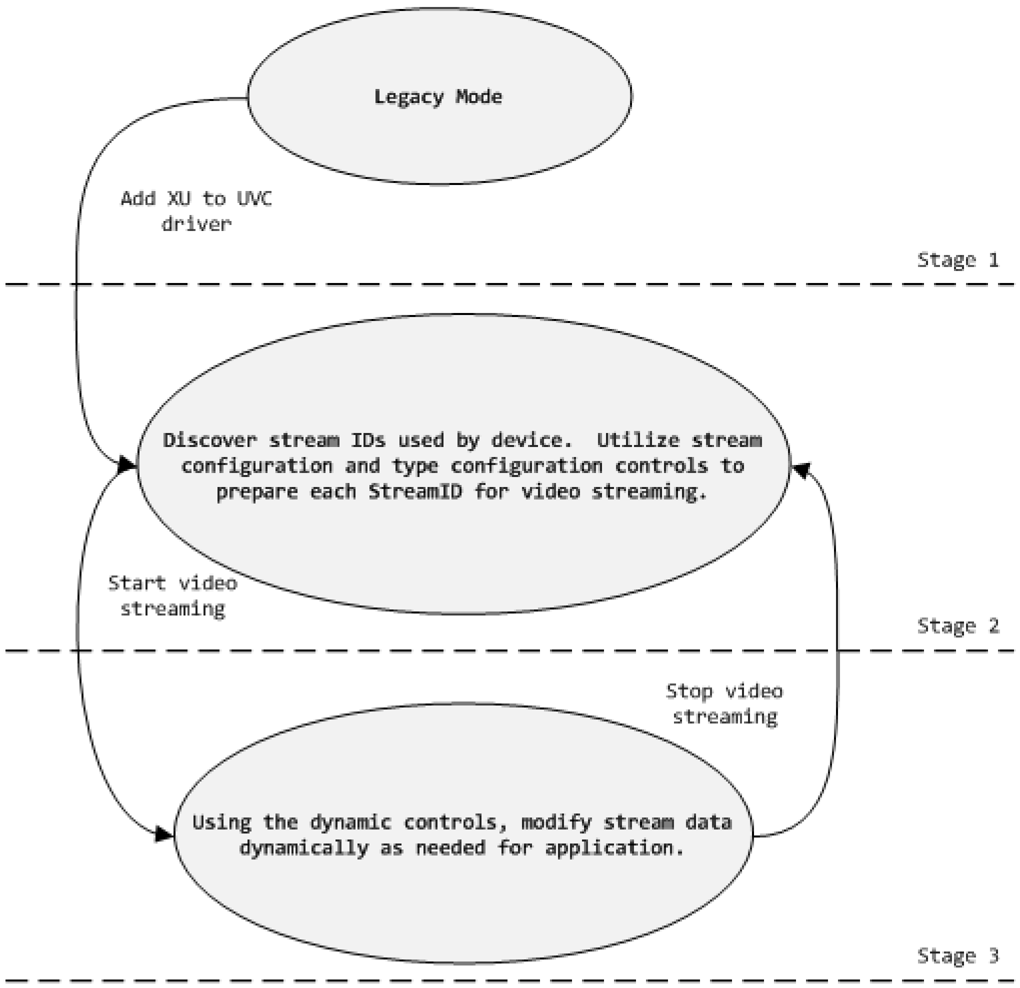
Legacy Mode
Add XU to UVC driver
Stage 1
Discover stream IDs used by device. Utilize stream configuration and type configuration controls to prepare each StreamID for video streaming.
Start video streaming
Stage 2
Stop video streaming
Using the dynamic controls, modify stream data dynamically as needed for application.
Stage 3

Illustration 8: Operational State Transitions

6.1 Probe and Commit Phases

Many controls come in pairs with suffixes Probe and Commit (e.g. StreamFormatProbe/Commit). The probe controls are used by the host to query the device for a supported configuration and the commit controls are used to activate a supported configuration.

This works as follows:

1. The host sets the desired stream to configure by using the StreamID control with the SET_CUR command.
2. The host performs a SET_CUR on the probe control with the configuration values it wants to query for device support.
3. The device may not exactly support the configuration values. If the exact values are not supported the device sets the current value of the probe control to the closest value supported by the device. The device always supports at least one configuration, so a value is always available. If the device supports the configuration value exactly, it sets the current value of the probe control to the unmodified configuration value.
4. The host performs a GET_CUR on the probe control to get the device modified value.
5. The host MAY choose to accept the modified value, or it MAY continue querying the device (that is, Jump to step 1.)
6. The host performs a SET_CUR on the commit control with one of the values it retrieved in step 4.

The device now considers this configuration to be active for the purpose of streaming.

Any value returned from a probe control by a GET_CUR request MUST be settable to the corresponding commit control without error. The result of setting a commit control to a value not returned from a probe control GET_CUR request is defined in the control description.

If using a legacy system that requires the single value stream probe controls to find the maximum and minimum values of the stream information, the host must first call StreamFormatProbeType with GET_MIN and GET_MAX. Then the host must iterate through the individual probe controls calling GET_MIN/MAX to obtain the minimum and maximum supported values for that stream. Each iteration, the host must call StreamFormatProbeType with SET_CUR and the call each of the remaining StreamFormatProbe* controls (Width, Height, FrameInterval and Bitrate). The values returned by the other probe controls are dependent on the value set by the StreamFormatProbeType control.

The following diagram illustrates a typical probe and commit scenario:

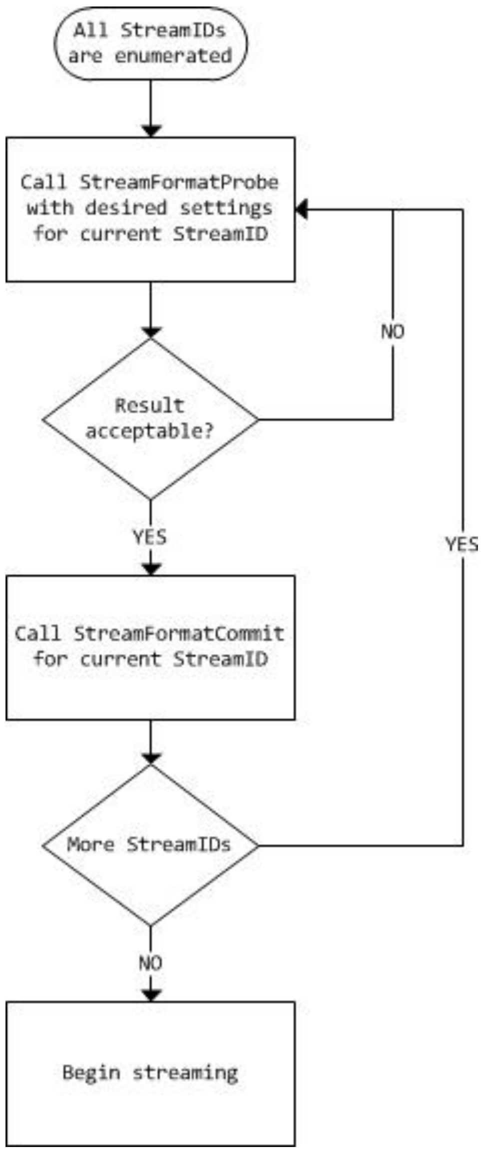
All StreamIDs are enumerated
Call StreamFormatProbe with desired settings for current StreamID
NO
Result acceptable?
YES
YES
Call StreamFormatCommit for current StreamID
More StreamIDs
NO
Begin streaming

Illustration 9: Probe and Commit Control Flow Example

It is the responsibility of the device to ensure that all values returned from GET_MIN and GET_MAX calls are supported.

7 Video Examples

Example 1 – Configure a single endpoint as the main capture stream using H.264 video and use the preview endpoint in legacy mode:

This example assumes the dual endpoint case; the main endpoint is already configured as MJPEG 720p and starts assuming the desired endpoints are already known.

1. Add the XU to the UVC driver.
2. Call the Version control to ensure the correct XU version.
3. Call the EndpointSetting control to determine the main and preview endpoint configuration.
4. Call the StreamID control with GET_MAX and GET_MIN to discover how many payloads each frame MAY contain. In this example, GET_MIN and GET_MAX return 0 (the main Stream ID). Activate the main stream to probe and commit by setting **bStreamID** to 0 with the SET_CUR command of the StreamID control.
5. Call the StreamFormatProbe control using SET_CUR with the desired configuration.
6. Call the StreamFormatProbe control with GET_CUR to check the closest match found by the device.
7. Repeat 5 and 6 until a suitable configuration is discovered.
8. Call the StreamFormatCommit control using SET_CUR with the discovered settings.
9. Use standard UVC discovery and commit methods to configure the preview stream.
10. Initiate video streaming on the main interface and the preview interface using standard UVC methods.
11. Process the raw video frames from the preview interface using standard methods.
12. Parse UVC frame buffer from the main interface formatted as seen below:

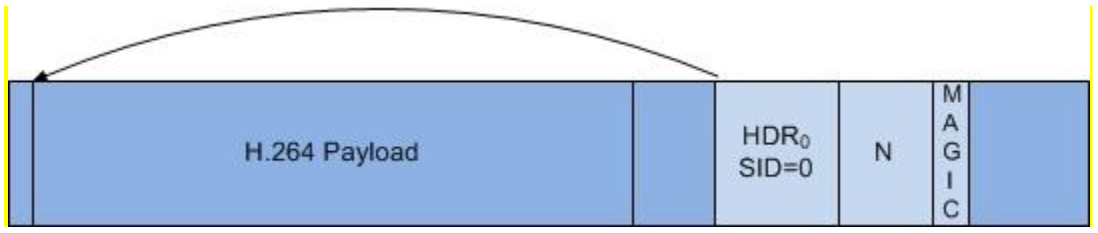

Illustration 10: UVC Frame Containing Main Stream Only

Example 2 – Configures a single endpoint as the main and preview stream using H.264 main and YUY2 preview

This example assumes single endpoint mode with only the main and preview stream type supported. It is also assumed the single endpoint is already known and configured as MJPEG at 720p resolution.

1. Add the XU to the UVC driver.
2. Call the Version control to ensure the correct XU version.

3. Call the StreamID control with GET_MAX and GET_MIN to discover how many payloads each frame MAY contain. In this example, GET_MIN is 0 and GET_MAX is 1. Call the StreamID with SET_CUR for each stream before doing the probe/commit sequence for that stream.
4. Call the StreamFormatProbe control using SET_CUR with the desired configuration for both StreamID 0 and 1.
5. Call the StreamFormatProbe control with GET_CUR to check the closest match found by the device for both StreamID 0 and 1.
6. Repeat 4 and 5 until a suitable configuration is discovered.
7. Call the StreamFormatCommit control using SET_CUR with the discovered settings for both StreamID 0 and 1.
8. Initiate video streaming using standard UVC methods.
9. Parse UVC frame buffer formatted as seen below:

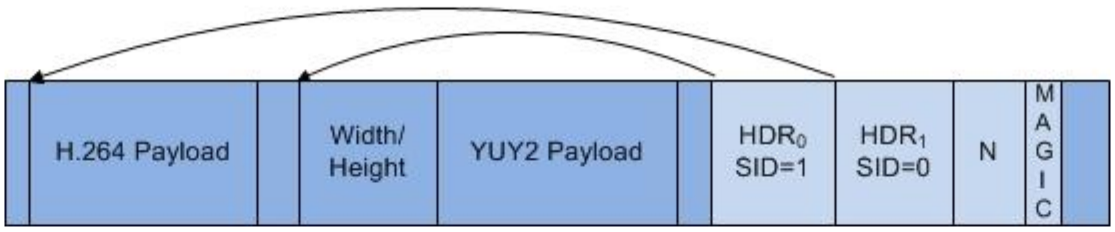

Illustration 11: UVC Frame Containing Main and Capture Streams

8 Audio Operational Model

At any time during operation of the device, the MicrophoneConfiguration control can be called with GET_CUR to probe the configuration of the device. This data can then be used by the host to provide higher quality audio.

9 Video References

YUY2 and NV12 Formats:

http://www.fourcc.org

MJPEG:

http://www.usb.org/developers/devclass_docs/USB_Video_Class_1_1.zip

H.264:

http://www.itu.int/rec/T-REC-H.264