

the high performance USB stack

Supplement to USBX USB Host Video Class

Express Logic, Inc.

858.613.6640

Toll Free 888.THREADX

FAX 858.521.4259

<http://www.expresslogic.com>

**©1999-2019 by Express Logic, Inc.**

All rights reserved. This document and the associated USBX software are the sole property of Express Logic, Inc. Each contains proprietary information of Express Logic, Inc. Reproduction or duplication by any means of any portion of this document without the prior written consent of Express Logic, Inc. is expressly forbidden.

Express Logic, Inc. reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of USBX. The information in this document has been carefully checked for accuracy; however, Express Logic, Inc. makes no warranty pertaining to the correctness of this document.

**Trademarks**

FileX, and ThreadX are registered trademarks of Express Logic, Inc., and USBX, NetX, *picokernel, preemption-threshold,* and *event-chaining* are trademarks of Express Logic, Inc. All other product and company names are trademarks or registered trademarks of their respective holders.

**Warranty Limitations**

Express Logic, Inc. makes no warranty of any kind that the USBX products will meet the USER’s requirements, or will operate in the manner specified by the USER, or that the operation of the USBX products will operate uninterrupted or error-free, or that any defects that may exist in the USBX products will be corrected after the warranty period. Express Logic, Inc. makes no warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose, with respect to the USBX products. No oral or written information or advice given by Express Logic, Inc., its dealers, distributors, agents, or employees shall create any other warranty or in any way increase the scope of this warranty and licensee may not rely on any such information or advice.

Part Number: 000-1010

Revision 5.9

# Chapter 1: Introduction to USBX UVC

USBX is a full-featured USB stack for deeply embedded applications. This chapter introduces USBX, describing its applications and benefits.

## UVC Configuration Options

The following symbols are defined in ux\_host\_class\_video.h. User may modify these values to better suit the application.

**UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST\_COUNT**

This symbol defines the maximum number of transfer buffers an application may post to video device. The default value is 4.

# Chapter 2: USBX UVC Operation

Using USBX UVC services to operate a USB camera is easy. Application needs to provide the following information:

**Video Format:**

USBX UVC defines the following video formats:

***UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_UNCOMPRESSED***

***UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_MJPEG***

***UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_MPEG2TS***

***UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_DV***

Application needs to be aware that the camera may not support all formats mentioned above.

**Resolution**:

Application shall specify the video resolution from the camera. The video resolution is represented in number of pixels in the video frame width and height. Typical screen resolutions are 320 by 240, 640 by 480, 1280 by 720. Application needs to make sure the camera supports the desired resolution.

**Inter-frame time:**

Application shall specify the time between each video frame, in unites of 100ns. For example, a video stream at 30 frames-per-second has inter-frame time of 33,333,333ns.

**Memory space**:

Application needs to allocate memory space for video device to store incoming video data. The memory space required to store video data depends on the video format and resolution. After configuring video format and resolution, application can use the service ***ux\_host\_class\_video\_max\_payload\_get*** to find the maximum payload size. The size of memory buffer passed into the video class needs to be at least this value.

To start a video service, application needs to obtain an instance of the video class. Refer to USBX Host Stack User Guide on how to register USBX Video Class, and how to obtain the instance once the video device is enumerated.

Once the application obtains an instance to the video device, the application needs to specify the video parameters by calling ***ux\_host\_class\_video\_frame\_parameters\_set()***. Application shall also use the service ***ux\_host\_class\_video\_max\_payload\_get()*** to find the maximum memory requirement for the given video configuration. Memory buffer can be passed to the video device by the API ***ux\_host\_class\_video\_transfre\_buffer\_add.*** Before enabling the video stream, application needs to register a video transfer done call back function by using the API ***ux\_host\_class\_video\_transfer\_callback\_set***. This callback function is called by the USB host thread when it finishes transferring a video frame. Application shall use this callback function as a notification that the video buffer previous passed to the video class is ready to be processed. Note that to keep the video streaming, application shall send another memory buffer while processing the data. This way, the video device always has memory to work with.

After the video device is configured, memory buffer provided, application starts the video stream by calling ***ux\_host\_class\_video\_start()***, and stop the video stream by calling ***ux\_host\_class\_video\_stop()***.

The following example outlines a typical video application. Note that proper error checking has been omitted to focus on the video class operation.

/\* Assume free\_memory points to a block of free available memory. \*/

extern UCHAR \*free\_memory;

/\* This semaphore is used for the callback function to signal application thread

that video data is received and can be processed. \*/

TX\_SEMAPHORE data\_received\_semaphore;

/\* Define the number of buffers used in this demo. \*/

#define MAX\_NUM\_BUFFERS (UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST\_COUNT-1)

/\* Video data received callback function. \*/

VOID video\_transfer\_done (UX\_TRANSFER \* transfer\_request)

{

/\* This is the callback function invoked by UVC class after a packet of data is received. \*/

/\* The actual number of bytes being received into the data buffer is recorded in tranfer\_request -> ux\_transfer\_request\_actual\_length. \*/

/\* Since this callback function executes in the USB host controller thread, a semaphore is released so the application can pick up the video data in application thread. \*/

tx\_semaphore\_put(&data\_received\_semaphore);

}

/\* Assume the caller passes in video\_ptr that points to ta valid

video instance. \*/

void video\_application(UX\_HOST\_CLASS\_VIDEO \*video\_ptr)

{

/\* This demo uses several buffers. One buffer is used by video device while the application consumes data in the other buffer. \*/

UCHAR \*buffer\_ptr[UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST\_COUNT-1];

/\* Index variable keeping track of the current buffer being used by

the video device. \*/

UINT buffer\_index;

/\* Maximum buffer requirement reported by the video device. \*/

INT max\_buffer\_size;

/\* Assume video\_ptr points to a valid video instance. \*/

/\* Create the semaphore for signaling video data received. \*/

tx\_semaphore\_create(&data\_received\_semaphore, "payload semaphore", 0);

/\* Set video parameters to MJPEG, 640x480 resolution, 30fps. \*/

**ux\_host\_class\_video\_frame\_parameters\_set**(video\_ptr, UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_MJPEG, 640, 480, 333333);

/\* Set transfer callback. \*/

**ux\_host\_class\_video\_transfer\_callback\_set**(video\_ptr,

video\_transfer\_done);

/\* Find out the maximum memory buffer size for the video configuration

set above. \*/

max\_buffer\_size = **ux\_host\_class\_video\_max\_payload\_get**(video\_ptr);

/\* Allocate space for video buffer. \*/

for(buffer\_index = 0; buffe\_index < MAX\_NUM\_BUFFERS; buffer\_index++)

{

buffer\_ptr[buffer\_index] = free\_memory + max\_buffer\_size \* buffer\_index;

}

/\* Start video transfer. \*/

ux\_host\_class\_video\_start(video\_ptr);

buffer\_index = 0;

/\* Add buffers list to the video device for video streaming data. \*/

**ux\_host\_class\_video\_transfer\_buffers\_add**(video\_ptr,

buffer\_ptr, MAX\_NUM\_BUFFERS);

while (1)

{

/\* Suspend here until a transfer callback is called. \*/

tx\_semaphore\_get(&data\_received\_semaphore, TX\_WAIT\_FOREVER);

/\* Received data. The callback function needs to obtain the actual number of bytes received, so the application routine can read the correct amount of data from the buffer. \*/

/\* Application can now consume video data while the video device stores the data into the other buffer. \*/

/\* Add the buffer back for video transfer. \*/

**ux\_host\_class\_video\_transfer\_buffer\_add**(video\_ptr,

buffer\_ptr[buffer\_index]);

/\* Increment the buffer\_index, and wrap to zero if it exceeds the maximum number of buffers. \*/

buffer\_index = (buffer\_index + 1);

if(buffer\_index >= MAX\_NUM\_BUFFERS)

buffer\_index = 0;

}

}

}

# Chapter 3: USBX UVC API

### ux\_host\_class\_video\_control\_get

Get a specific control from the video control interface

**Prototype**

UINT **ux\_host\_class\_video\_control\_get**(UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video,

UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video\_control)

**Description**

This function reads static feature values (GET\_MIN/GET\_MAX/GET\_RES) in a specific control from the video control interface.

**Parameters**

**video** Pointer to the video class instance

**video\_control** Pointer to the video control structure

**Return Values**

**UX\_SUCCESS** (0x00) Successful get video control information

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x5B) The class instance is not attached to the class container.

**UX\_MEMORY\_INSUFFICIENT**

(0x12) Not enough memory for this controller.

**UX\_TRANSFER\_ERROR** (0x23) Transfer error while reading object

**Example**

UX\_HOST\_CLASS\_VIDEO\_CONTROL video\_control;

video\_control.ux\_host\_class\_video\_control = UX\_HOST\_CLASS\_VIDEO\_PU\_BRIGHTNESS\_CONTROL;

**ux\_host\_class\_video\_control\_get**(video\_ptr, &video\_control);

### ux\_host\_class\_video\_control\_value\_get

Get a specific control value from the video control interface

**Prototype**

UINT **ux\_host\_class\_video\_control\_value\_get**(UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video,

UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video\_control)

**Description**

This function obtains the current value (GET\_CUR) from a single video control.

**Parameters**

**video** Pointer to the video class instance

**video\_control** Pointer to the video control structure

**Return Values**

**UX\_SUCCESS** (0x00) Successful get video control value

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x5B) The class instance is not attached to the class container.

**UX\_MEMORY\_INSUFFICIENT**

(0x12) Not enough memory for this controller.

**UX\_TRANSFER\_ERROR** (0x23) Transfer error while reading object

**Example**

UX\_HOST\_CLASS\_VIDEO\_CONTROL video\_control;

video\_control.ux\_host\_class\_video\_control = UX\_HOST\_CLASS\_VIDEO\_PU\_BRIGHTNESS\_CONTROL;

ux\_host\_class\_video\_control\_get(g\_ux\_host\_class\_video, &video\_control);

**ux\_host\_class\_video\_control\_value\_get**(g\_ux\_host\_class\_video, &video\_control);ux\_host\_class\_video\_control\_value\_set

Set the value in the video control

**Prototype**

UINT **ux\_host\_class\_video\_control\_value\_set**(UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video,

UX\_HOST\_CLASS\_VIDEO\_CONTROL \*video\_control)

**Description**

This function sets the current value (SET\_CUR) of the video control.

**Parameters**

**video** Pointer to the video class instance

**video\_control** Pointer to the video control structure

**Return Values**

**UX\_SUCCESS** (0x00) Successful set video control value

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x5B) The class instance is not attached to the class container.

**UX\_MEMORY\_INSUFFICIENT**

(0x12) Not enough memory for this controller.

**UX\_TRANSFER\_ERROR** (0x23) Transfer error while reading object

**Example**

UX\_HOST\_CLASS\_VIDEO\_CONTROL video\_control;

video\_control.ux\_host\_class\_video\_control = UX\_HOST\_CLASS\_VIDEO\_PU\_BRIGHTNESS\_CONTROL;

video\_control.ux\_host\_class\_video\_control\_cur = 50;

**ux\_host\_class\_video\_control\_value\_set**(g\_ux\_host\_class\_video, &video\_control);

### ux\_host\_class\_video\_ioctl

IO Control function for the video class

**Prototype**

UINT \_ux\_host\_class\_video\_ioctl(UX\_HOST\_CLASS\_VIDEO \*video, ULONG ioctl\_function, VOID \*parameter)

**Description**

This function allows applications to obtain control information from the camera device. Valid control functions are:

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_INPUT\_TERMINAL

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_FORMAT\_NUMBER

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_FORMAT\_DATA

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_FRAME\_NUMBER

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_FRAME\_DATA

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_CHANNEL\_START

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_CHANNEL\_STOP

UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_FRAME\_INTERVAL

**Parameters**

**video** Pointer to the video class instance

**ioctl\_function** The ioctl function to execute

**parameter** The parameter passed to the ioctl function

**Return Values**

**UX\_SUCCESS** (0x00) Successful operation of the ioctl function

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x5B) The class instance is not attached to the class container.

**UX\_FUNCTION\_NOT\_SUPPORTED**

(0x54) Requested function not supported.

**Example**

UX\_HOST\_CLASS\_VIDEO\_PARAMETER\_INPUT\_TERMINAL input\_terminal;

**ux\_host\_class\_video\_ioctl**(g\_ux\_host\_class\_video, UX\_HOST\_CLASS\_VIDEO\_IOCTL\_GET\_INPUT\_TERMINAL, &input\_terminal);

### ux\_host\_class\_video\_read

Initiate the video read operation

**Prototype**

UINT **ux\_host\_class\_video\_read**(UX\_HOST\_CLASS\_VIDEO \*video,

UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST \*video\_transfer\_request)

**Description**

This function supplies a video frame buffer to the video device, and starts the video transfer.

**Parameters**

**video** Pointer to the video class instance

**video\_transfer\_request** Pointer to the video transfer request structure.

**Return Values**

**UX\_SUCCESS** (0x00) Successful operation of the ioctl function

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x5B) The class instance is not attached to the class container.

**UX\_FUNCTION\_NOT\_SUPPORTED**

(0x54) Requested function not supported.

**Example**

UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST video\_transfer\_request;

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request\_completion\_function = transfer\_request\_done\_callback;

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request\_class\_instance = g\_ux\_host\_class\_video;

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request\_next\_video\_transfer\_request = UX\_NULL;

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request\_requested\_length = 800;

/\* Uses the endpoint max packet size as packet size. \*/

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request.ux\_transfer\_request\_packet\_length = 0;

video\_transfer\_request.ux\_host\_class\_video\_transfer\_request\_data\_pointer = video\_buffer\_ptr;

**ux\_host\_class\_video\_read**(g\_ux\_host\_class\_video, &video\_transfer\_request);

### ux\_host\_class\_video\_start

Start the video streaming

**Prototype**

UINT **ux\_host\_class\_video\_start**(UX\_HOST\_CLASS\_VIDEO \*video)

**Description**

This function starts the video streaming. The video channel needs to be properly configured prior to calling this function.

**Parameters**

**video** Pointer to the video class instance

**Return Values**

**UX\_SUCCESS** (0x00) Successful starts video streaming.

**UX\_MEMORY\_INSUFFICIENT**

(0x12) Not enough memory for this controller.

**UX\_NO\_BANDWIDTH\_AVAILABLE**

(0x41) The USB controller doesn’t have enough bandwidth to support the selected frame parameters.

**Example**

/\* Starts the video channel. \*/

ux\_host\_class\_video\_start(video\_ptr);

### ux\_host\_class\_video\_stop

Stop the current video channel

**Prototype**

UINT **ux\_host\_class\_video\_stop**(UX\_HOST\_CLASS\_VIDEO \*video)

**Description**

This service stops the current video channel.

**Parameters**

**video** Pointer to the video class instance

**Return Values**

**UX\_SUCCESS** (0x00) Successful stop the video channel.

**Example**

/\* Stop the device from streaming video data to the host. \*/

status = **ux\_host\_class\_video\_stop**(video\_ptr);

/\* If the return status is UX\_SUCCESS, the video streaming is stopped. \*/

### ux\_host\_class\_video\_frame\_parameters\_set

Configure the video channel parameters

**Prototype**

UINT \_**ux\_host\_class\_video\_frame\_parameters\_set**(UX\_HOST\_CLASS\_VIDEO \*video, ULONG frame\_format, ULONG width, ULONG height, ULONG frame\_interval)

**Description**

This function sets the video parameters for the video device.

**Parameters**

**video** Pointer to the video class instance

**frame\_format** Desired frame format. Valid values are:

*UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_UNCOMPRESSED*

*UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_MJPEG*

**width** Desired frame width, in pixels

**height** Desired frame height, in pixels

**frame\_interval** Desired frame intervals, in 100ns units

**Return Values**

**UX\_SUCCESS** (0x00) Successful configured the parameters for the video camera.

**UX\_HOST\_CLASS\_VIDEO\_PARAMETER\_ERROR**

(0x92) The desired video parameters are not supported by this camera.

**Example**

/\* The following example configures the video device to stream in MJPEG format, 640x480 frame resolution, at 30 frames-per-second. \*/

status = **ux\_host\_class\_video\_frame\_parameters\_set**(video\_ptr, UX\_HOST\_CLASS\_VIDEO\_VS\_FORMAT\_MJPEG, 640, 480, 333333);

/\* If return value is UX\_SUCCESS, the video device is configured properly. \*/

### ux\_host\_class\_video\_max\_payload\_get

Get the maximum transfer size in a single packet.

**Prototype**

UINT **ux\_host\_class\_video\_max\_payload\_get(UX\_HOST\_CLASS\_VIDEO \*video);**

**Description**

This function returns the maximum payload size for a given video parameter setting. After properly configures the video streaming parameters (such as video encoding, resolution, frame rate), application may use this function to obtain the maximum payload size. With the maximum payload size, application is able to allocate memory buffers for receiving incoming video frame data.

**Parameters**

**video** Pointer to the video class instance

**Return Values**

Maximum video data payload size, in number of bytes.

**Example**

/\* Find out the maximum payload size. \*/

ULONG payload\_size;

payload\_size = **ux\_host\_class\_video\_max\_payload\_get**(video\_ptr);

### ux\_host\_class\_video\_transfer\_buffer\_add

Add a data buffer for video transfer request.

**Prototype**

UINT **ux\_host\_class\_video\_tranfer\_buffer\_add(**UX\_HOST\_CLASS\_VIDEO \*video, UCHAR \*buffer)

**Description**

This function passes a buffer to the video device, which is used to store incoming video stream data. The size of the buffer must be at least the maximum of the video payload size, which can be obtained by calling ***ux\_host\_class\_video\_max\_payload\_get.***

**Parameters**

**video** Pointer to the video class instance

**buffer** Pointer to the buffer space to be used for receiving video data.

**Return Values**

**UX\_SUCCESS** (0x00) Successful setting video buffer.

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x59) The video instance is not valid.

**Example**

/\* Find the maximum payload size and allocate the buffer space for the video stream. \*/

#define MAX\_NUM\_BUFFERS 2

extern UCHAR \*data\_start;

ULONG max\_packet\_size;

UCHAR \*buffer\_ptr[MAX\_NUM\_BUFFERS];

UINT buffer\_index = 0;

max\_packet\_size = **ux\_host\_class\_video\_max\_payload\_get**(video\_ptr);

for(buffer\_index = 0; buffer\_index < MAX\_NUM\_BUFFERS; buffer\_index++)

{

buffer\_ptr[buffer\_index] = data\_start + max\_packet\_size \* buffer\_index;

}

buffer\_index = 0;

while(1)

{

**ux\_host\_class\_video\_transfer\_buffer\_add**(video\_ptr, buffer\_ptr[buffer\_index]);

/\* Wait for video data to be ready. \*/

/\* Consume video data \*/

buffer\_index++;

if(buffer\_index >= MAX\_NUM\_BUFFERS)

buffer\_index = 0;

}

### ux\_host\_class\_video\_transfer\_buffers\_add

Add data buffers for video transfer request.

**Prototype**

UINT **ux\_host\_class\_video\_tranfer\_buffers\_add(**UX\_HOST\_CLASS\_VIDEO \*video, UCHAR \*\*buffers, ULONG num\_buffers)

**Description**

This function passes a buffer list to the video device, which is used to store incoming video stream data.

The size of each buffer in the list must be at least the maximum of the video payload size, which can be obtained by calling ***ux\_host\_class\_video\_max\_payload\_get.***

It’s usually used on start of a high-bandwidth video stream. The transfer doesn’t start until the whole list is ready, so when adding more buffer by ***ux\_host\_class\_video\_transfer\_buffer\_add***, the previous added buffers are still in progress. This helps to improve the performance.

The maximum number of buffers can be buffered is ***UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST\_COUNT*** – 1.

**Parameters**

**video** Pointer to the video class instance

**buffers** Pointer to the buffer list to be used for receiving video data.

**num\_buffers** Number of buffers in the list.

**Return Values**

**UX\_SUCCESS** (0x00) Successful setting video buffer.

**UX\_HOST\_CLASS\_INSTANCE\_UNKNOWN**

(0x59) The video instance is not valid.

**Example**

/\* Find the maximum payload size and allocate the buffer space for the video stream. \*/

#define MAX\_NUM\_BUFFERS (UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST\_COUNT-1)

extern UCHAR \*data\_start;

ULONG max\_packet\_size;

UCHAR \*buffer\_ptr[MAX\_NUM\_BUFFERS];

UINT buffer\_index = 0;

max\_packet\_size = **ux\_host\_class\_video\_max\_payload\_get**(video\_ptr);

for(buffer\_index = 0; buffer\_index < MAX\_NUM\_BUFFERS; buffer\_index++)

{

buffer\_ptr[buffer\_index] = data\_start + max\_packet\_size \* buffer\_index;

}

buffer\_index = 0;

/\* Prepare buffer list for the stream. \*/

**ux\_host\_class\_video\_transfer\_buffers\_add**(video\_ptr, buffer\_ptr, MAX\_NUM\_BUFFERS);

while(1)

{

/\* Wait for buffer transfer done and video data to be ready (read). \*/

/\* Consume video data \*/

/\* Reuse this free buffer for transfer. \*/

**ux\_host\_class\_video\_transfer\_buffer\_add**(video\_ptr, buffer\_ptr[buffer\_index]);

buffer\_index++;

if(buffer\_index >= MAX\_NUM\_BUFFERS)

buffer\_index = 0;

}

### ux\_host\_class\_video\_transfer\_callback\_set

Sets video transfer done callback function

**Prototype**

UINT **ux\_host\_class\_video\_transfer\_callback\_get**(UX\_HOST\_CLASS\_AUDIO \*video,

VOID(\*callback\_function)(UX\_TRANSFER\*))

**Description**

This function sets the video transfer callback function. This callback function is invoked once a transfer request has been fulfilled, and the application is ready to consume the video data.

**Parameters**

**video** Pointer to the video class instance

**callback\_function** User-supplied transfer done callback function

**Return Values**

**None**

**Example**

VOID video\_transfer\_done(UX\_HOST\_CLASS\_VIDEO\_TRANSFER\_REQUEST \*transfer\_request)

{

/\* Transfer request is complete. Data is stored in transfer\_request -> ux\_host\_class\_video\_transfer\_request\_data\_pointer. \*/

/\* Note that the callback function executes in USB thread. Post a semaphore and let application thread process the video data. \*/

tx\_semaphore\_put(&frame\_ready\_semaphore);

}

**ux\_hsot\_class\_video\_transfer\_callback\_set**(video\_ptr, video\_transfer\_done);