User's Manual

# Supplement to USBX™ USB Host Video Class

User Guide

Renesas Synergy™ Platform
Synergy Software
Synergy Software (SSP) Component



Renesas Electronics
www.renesas.com

Nov 14, 2018

# Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

# Renesas Synergy Specific Information

If you are using Supplement to USBX™ USB Host Video Class for the Renesas Synergy platform, please use the following information.

## Customer Support

For Renesas Synergy platform support, please contact Renesas directly:

Support: https://synergygallery.renesas.com/support

America: https://www.renesas.com/en-us/support/contact.html

Europe: https://www.renesas.com/en-eu/support/contact.html

Japan: https://www.renesas.com/ja-jp/support/contact.html

## Isochronous Transfer Type Not Supported

**Page 7:** Isochronous transfer is only supported for USB Host. This transfer type is not supported for USB Device.


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

Part Number: 000-1010

Revision 5.8SP2

# *Chapter 1: Introduction to USBX UVC*

USBX UVC implements USB host Video Class. UVC allows an application to easily operate a USB camera device. In most cases, once the application obtains a video instance from USBX host stack, application only needs to specify the video format, resolution, and frame rate to get the video started. Other camera controls, such as exposure, brightness, color saturation can be achieved in the future.

Note that UBX UVC is designed to operate a USB video device, to obtain video streaming data from the video device. UVC does not encode or decode video data. Therefore, application is responsible for processing streaming data from the video device.

## 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  2

/* 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 two buffers. One buffer is used by video device while the
   application consumes data in the other buffer. */
UCHAR *buffer_ptr[2];

/* 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;

        /* Add buffer to the video device for video streaming data. */
        ux_host_class_video_transfer_buffer_add(video_ptr,
                                        buffer_ptr[buffer_index]);

    }

    /* Start video transfer.  */
    ux_host_class_video_start(video_ptr);

    buffer_index = 0;

    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_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. |

### 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_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);
```

Supplement to USBX™ USB Host Video Class

Publication Date: Rev.5.81 Nov 14, 2018

Published by: Renesas Electronics Corporation

**SALES OFFICES** Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.


**Renesas Electronics Corporation**
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

**Renesas Electronics America Inc.**
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

**Renesas Electronics Canada Limited**
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

**Renesas Electronics Europe Limited**
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

**Renesas Electronics Hong Kong Limited**
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

**Renesas Electronics Taiwan Co., Ltd.**
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

**Renesas Electronics Singapore Pte. Ltd.**
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

**Renesas Electronics Malaysia Sdn.Bhd.**
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

**Renesas Electronics India Pvt. Ltd.**
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

**Renesas Electronics Korea Co., Ltd.**
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338





Colophon 5.2

# Supplement to USBX™ USB Host Video Class

Renesas Electronics Corporation

R11UM0084EU0581