RX Family

USB Peripheral Mass Storage Class Driver (PMSC)

R01AN0514EJ0233
Rev.2.33
Jul 31, 2019

Introduction

This application note describes the USB peripheral mass storage class driver (PMSC). This driver operates in combination with the USB Basic Peripheral Driver (USB-BASIC-F/W). It is referred to below as the PMSC.

Targe Device

RX62N/RX621 Group
RX63N/RX631 Group
RX630 Group
RX63T Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Related Documents

1. USB Revision 2.0 Specification
2. USB Mass Storage Class Specification Overview Revision 1.1
3. USB Mass Storage Class Bulk-Only Transport Revision 1.0, “BOT” protocol
   http://www.usb.org/developers/docs/
4. RX62N/RX621 Group User’s Manual: Hardware (Document number .R01UH0033)
5. RX63N/RX631 Group User’s Manual: Hardware (Document number .R01UH0041)
6. RX630 Group User’s Manual: Hardware (Document number .R01UH0040)
7. RX63T Group User’s Manual: Hardware (Document number .R01UH0238)
8. USB Basic Host and Peripheral Driver Application Note. (Document number. R01AN0512)

Renesas Electronics Website
http://www.renesas.com/

USB Devices Page
http://www.renesas.com/prod/usb/

Content

1. Overview

PMSC, when used in combination with the USB-BASIC-F/W, operates as a USB peripheral mass storage class driver (PMSC). The USB peripheral mass storage class driver (PMSC) comprises a USB mass storage class bulk-only transport (BOT) protocol. When combined with a USB peripheral control driver and media driver, it enables communication with a USB host as a BOT-compatible storage device.

This module supports the following functions.

- Storage command control using the BOT protocol
- Response to mass storage device class requests from a USB host

1.1 Please be sure to read

Please refer to the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note* when creating an application program using this driver.
This document is located in the "**reference_documents**" folder within this package.

1.2 Note

This driver is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.

1.3 Limitations

PMSC is subject to the following limitations.

1. This driver return the value (0:zero) to the mass storage command (*GetMaxLun*) which is sent from USB Host.
2. The sector size which this driver supports is only 512 bytes.
3. This driver does not support DMA/DTC transfer.

1.4 Note

1. This driver is not guaranteed to provide USB communication operation. The customer should verify operation when utilizing it in a system and confirm the ability to connect to a variety of different types of devices.
2. Please be sure to use the documentations *(r01an0514jj0232_usb.pdf, r01an0512jj0232_usb.pdf)* under the "reference_documents" folder when using RX62N/RX621/ RX63T/RX630.
3. You need to create the media driver function to control the media which is used as the storage area.
4. This driver uses the following area as the storage media area.

MCU	Storage Media Area	File System
RX62N	SDRAM on RSK	FAT16
RX63N	SDRAM on RSK	FAT16
RX63T	Internal RAM	FAT12
RX630	Internal RAM	FAT12

Note:

When the following all conditions are satisfied, be sure to do the eject processing by "Safely Remove Hardware and Eject Media" display on Windows® task bar before plugging out MSC device.

(1). The storage media is formatted as FAT12.

(2). MSC Host is Windows® 8.1 or Windows® 10.

1.5 Terms and Abbreviations

APL	:	Application program
BOT	:	Mass storage class Bulk Only Transport
DDI	:	Device driver interface, or PMSDD API.
PCD	:	Peripheral control driver of
PCI	:	PCD interface
PDCD	:	Peripheral Device Class Driver
PMSCD	:	Peripheral mass storage USB class driver (PMSCF + PCI + DDI)
PMSCF	:	Peripheral mass storage class function
PMSDD	:	Peripheral mass storage device driver (sample ATAPI driver)
RSK	:	Renesas Starter Kits
USB	:	Universal Serial Bus
	:	USB Basic Host and Peripheral Driver for Renesas USB device

2. Software Configuration

PMSC comprises two layers: PMSCD and PMSDD.

PMSCD comprises three layers: PCD API (PCI), PMSDD API (DDI), and BOT protocol control and data sends and receives (PMSCF).

PMSCD uses the BOT protocol to communicate with the host via PCD.

PMSDD analyzes and executes storage commands received from PMSCD. PMSDD accesses media data via the media driver.

Figure 2-1 shows the configuration of the modules.

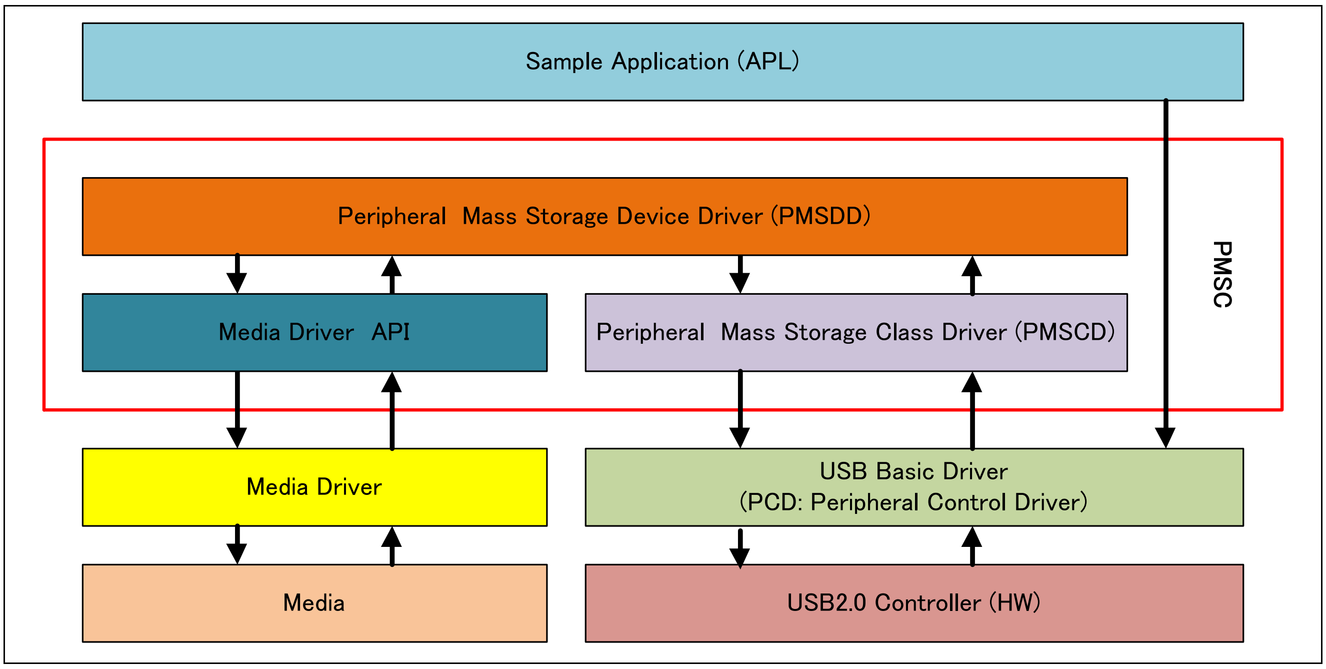

Sample Application (APL)
Peripheral Mass Storage Device Driver (PMSDD)
Media Driver API
Peripheral Mass Storage Class Driver (PMSCD)
PMSC
Media Driver
USB Basic Driver
(PCD: Peripheral Control Driver)
Media
USB2.0 Controller (HW)

Figure 2-1 Software Configuration Diagram

Table 2-1 Module Function Overview

Module	Description
PMSDD	Mass Storage Device Driver ・ Processes storage commands from the PMSCD ・ Accesses media via the media driver
DDI	PMSDD-PMSCD interface function
PMSCF	Mass Storage Class Driver ・ Controls BOT protocol data and responds to class requests. ・ Analyzes CBWs and transmits/receives data. ・ Generates CSWs together with the PMSDD/PCD.
PCI	PMSCD – PCD interface function
PCD	USB Peripheral H/W Control driver

3. API Information

This Driver API follows the Renesas API naming standards.

3.1 Hardware Requirements

This driver requires your MCU support the following features:

- USB

3.2 Operating Confirmation Environment

The following shows the operating confirmation environment of this driver.

Table 3-1 Operation Confirmation Environment

Item	Contents
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.07.00
	Compile Option : -lang = c99
Endian	Little Endian, Big Endian
Using Board	Renesas Starter Kits for RX63N
Host Environment	The operation of this USB Driver module connected to the following OSes has been confirmed. 1. Windows® 7 2. Windows® 8.1 3. Windows® 10

3.3 Usage of Interrupt Vector

The following shows the interrupt vector which this driver uses.

Table 3-2 List of Usage Interrupt Vectors

Device	Contents
RX63N/RX631	USBI0 Interrupt (Vector number: 35) / USBR0 Interrupt (Vector number: 90) USBI1 Interrupt (Vector number: 38) / USBR0 Interrupt (Vector number: 91)

3.4 Header Files

All API calls and their supporting interface definitions are located in r_usb_basic_if.h and r_usb_pmsc_if.h.

3.5 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in *stdint.h*.

3.6 Compile Setting

For compile settings, refer to chapter **7, Configuration (r_usb_pmsc_config.h)** in this document and chapter "Configuration" in the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note*.

3.7 ROM/RAM size

The follows show ROM/RAM size of this driver.

	Checks arguments	Does not check arguments
ROM size	20.1K bytes (Note 3)	19.6K bytes (Note 4)
RAM size	13.4K bytes	13.4K bytes

Note:

1. ROM/RAM size for USB Basic Driver is included in the above size.
2. The default option is specified in the compiler optimization option.
3. The ROM size of "Checks arguments" is the value when *USB_CFG_ENABLE* is specified to *USB_CFG_PARAM_CHECKING* definition in *r_usb_basic_config.h* file.
4. The ROM size of "Does not check arguments" is the value when *USB_CFG_DISABLE* is specified to *USB_CFG_PARAM_CHECKING* definition in *r_usb_basic_config.h* file.
5. The RAM size is the value when 8 (numeric value) is specified to *USB_CFG_PMSC_TRANS_COUNT* definition in *r_usb_pmsc_config.h* file.

3.8 Argument

For the structure used in the argument of API function, refer to chapter "**Structures**" in the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note*.

4. Class Driver Overview

4.1 Class Request

The following lists the class requests supported by this driver

Table 4-1 MSC Class Requests

Request	Code	Description
Bulk-Only Mass Storage Reset	0xFF	Resets the connection interface to the mass storage device.
Get Max Lun	0xFE	Reports the logical numbers supported by the device.

4.2 Storage Commands

The follwoing lists the storage commands supported by this driver. This driver send the STALL or FAIL error (CSW) to USB HOST when receiving other than the following command.

Table 4-2 Storage Commands

Command	Code	Description
TEST_UNIT_READY	0x00	Checks the state of the peripheral device.
REQUEST_SENSE	0x03	Gets the error information of the previous storage command execution result.
INQUIRY	0x12	Gets the parameter information of the logical unit.
READ_FORMAT_CAPACITY	0x23	Gets the formattable capacity.
READ_CAPACITY	0x25	Gets the capacity information of the logical unit.
READ10	0x28	Reads data.
WRITE10	0x2A	Writes data.
MODE_SENSE10	0x5A	Gets the parameters of the logical unit.

5. Peripheral Device Class Driver (PDCD)

5.1 Basic Functions

The functions of PDCD are to:

1. Supporting SFF-8070i (ATAPI)
2. Respond to mass storage class requests from USB host.
3. Respond to USB host storage commands which are encapsulated in the BOT protocol (Bulk Only Transport), see below)

5.2 BOT Protocol Overview

BOT (USB MSC Bulk-Only Transport) is a transfer protocol that, encapsulates command, data, and status (results of commands) using only two endpoints (one bulk in and one bulk out).
The ATAPI storage commands and the response status are embedded in a “Command Block Wrapper” (CBW) and a “Command Status Wrapper” (CSW).
The following shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

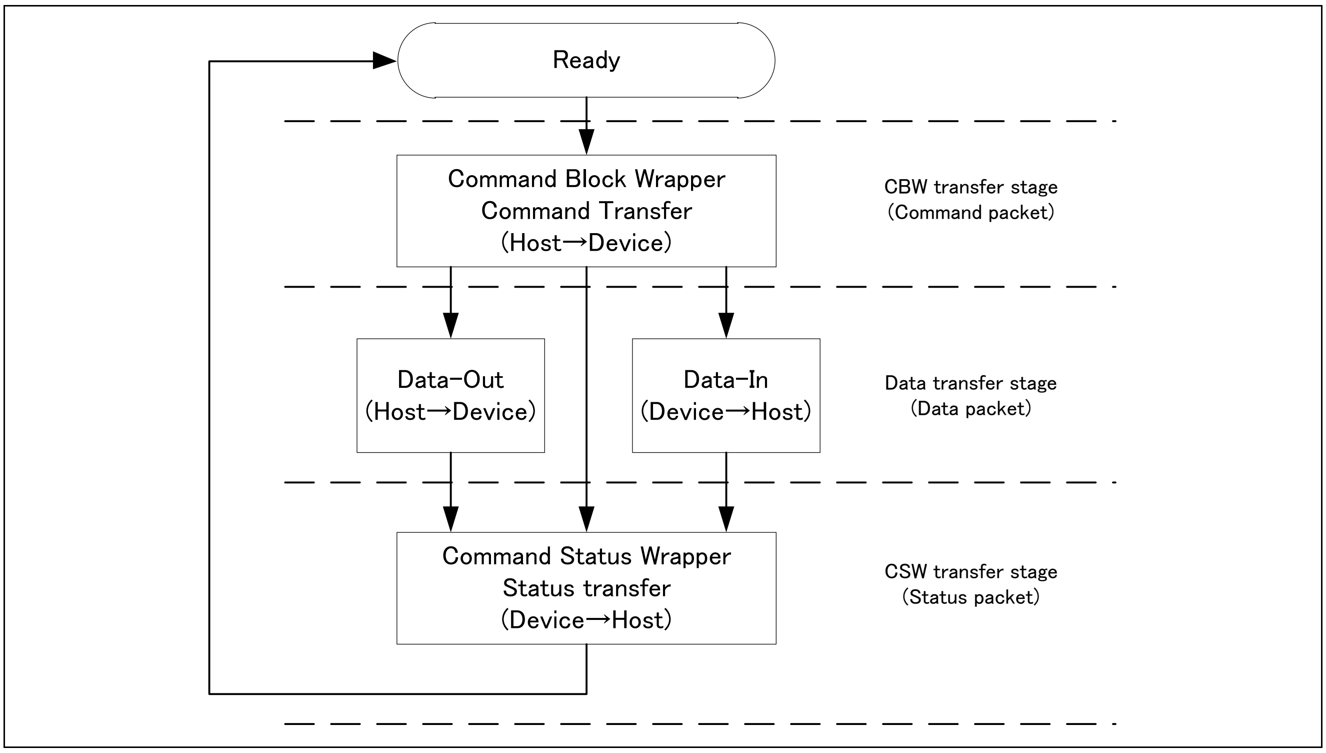


Figure 5-1 BOT protocol Overview
Command and status flow between USB host and peripheral.

6. API Functions

For API used in the application program, refer to chapter "**API Functions**" in the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note*.

7. Configuration (r_usb_pmsc_config.h)

Please set the following according to your system.

Note:

Be sure to set *r_usb_basic_config.h* file as well. For *r_usb_basic_config.h* file, refer to chapter "**Configuration**" in the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note*.

1. Setting pipe to be used

Set the pipe number (PIPE1 to PIPE5) to use for Bulk IN/OUT transfer. Do not set the same pipe number for the definitions of *USB_CFG_PMSC_BULK_IN* and *USB_CFG_PMSC_BULK_OUT*.

```
#define     USB_CFG_PMSC_BULK_IN          Pipe number (USB_PIPE1 to USB_PIPE5)
#define     USB_CFG_PMSC_BULK_OUT         Pipe number (USB_PIPE1 to USB_PIPE5)
```

2. Setting the response data for Inquiry command

This driver sends the data specified in the following definitions to the USB Host as the response data of Inquiry command.

(1). Setting Vendor Information

Specify the vendor information which is response data of Inquiry command. <u>Be sure to enclose data of 8 bytes with double quotation marks.</u>

```
#define     USB_CFG_PMSC_VENDOR     Vendor Information
```

e.g)

```
#define     USB_CFG_PMSC_VENDOR     "Renesas "
```

(2). Setting Product Information

Specify the product information which is response data of Inquiry command. <u>Be sure to enclose data of 16 bytes with double quotation marks.</u>

```
#define     USB_CFG_PMSC_PRODUCT     Product Information
```

e.g)

```
#define     USB_CFG_PMSC_PRODUCT     "Mass Storage    "
```

(3). Setting Product Revision Level

Specify the product revision level which is response data of Inquiry command. <u>Be sure to enclose data of 4 bytes with double quotation marks.</u>

```
#define     USB_CFG_PMSC_REVISION     Product Information
```

e.g)

```
#define     USB_CFG_PMSC_REVISION     "1.00"
```

3. Setting the number of transfer sector

Specify the maximum sector size to request to PCD (Peripheral Control Driver) at one data transfer. This driver specifies the value of "1 sector (512) × USB_CFG_PMSC_TRANS_COUNT" bytes to PCD as the transfer size. By increasing this value, the number of data transfer requests to the PCD decreases, so the transfer speed performance may be improved. However, note that "1 sector (512) × USB_CFG_PMSC_TRANS_COUNT" bytes of RAM will be consumed.

```
#define   USB_CFG_PMSC_TRANS_COUNT      Number of transfer sectors (1 to 255)
```

e.g)

```
#define     USB_CFG_PMSC_TRANS_COUNT     4
```

8. Media Driver Interface

PMSC uses a common media driver API function to access to the media drivers with different specifications.

8.1 Overview of Media Driver API Functions

Media driver API functions are called by the PMSC and the API functions call the media driver function implemented by the user. This chapter explains the prototype of the media driver API function and the processing necessary for implementing each function.

Table 8-1 shows the list of the media driver API functions.

Table 8-1 Media Driver API

Media Driver API	Processing Description
R_USB_media_initialize	Initializes the media driver.
R_USB_media_open	Opens the media driver.
R_USB_media_close	Closes the media driver.
R_USB_media_read	Reads from the media.
R_USB_media_write	Writes to the media.
R_USB_media_ioctl	Processing the control instructions specific to the media device.

8.1.1 R_USB_media_initialize

Register the media driver function to the media driver

Format

bool R_USB_media_initialize(media_driver_t * p_media_driver);

Arguments

p_meida_driver	Point to the structure area for the media driver

Return Value

TRUE	Successfully completed
FALSE	Error generated

Description

This API registers the media driver function implemented by the user to the media driver.

Be sure to call this API at the initialization processing etc in the user application program.

Note

1. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.
2. For how to register of the media driver function implemented by the user, refer to the chapte**r 8.3, Registration of the storage media driver**.
3. This API does not do the media device initialization processing and does not do the starting operation processing of the media device. These processing is done by *R_USB_media_open* function.
4. PMSC does not support the function to register the multiple type media driver function.

Example

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
        /* Handle the error */
}
result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
        /* Process the error */
}
```

8.1.2 R_USB_media_open

Initialize the media driver and the media device

Format

usb_media_ret_t R_USB_media_open(void);

Arguments

--

Return Value

USB_MEDIA_RET_OK	Successfully completed
USB_MEDIA_RET_PARAERR	Parameter error
USB_MEDIA_RET_DEV_OPEN	The device was already opened
USB_MEDIA_RET_NOTRDY	The device is not responding or not present
USB_MEDIA_RET_OP_FAIL	Any other failure

Description

This API initializes the media device and the media driver and make the media device and the media driver the ready status.
Be sure to call this API at the initialization processing etc in the user application program.

Note

1. *R_USB_media_initialize* function has to be called before calling this API.
2. The number of calls this API is only once unless *R_USB_media_close* is called. After calling *R_USB_media_close* function, this API can be called again to return the device to the initial state.
3. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.

Example

```
if (!R_USB_media_initialize(&g_ram_mediadriver))
{
        /* Handle the error */
}

result = R_USB_media_open();
if (USB_MEDIA_RET_OK != result)
{
        /* Process the error */
}
```

8.1.3 R_USB_media_close

Release the resource for the media driver and return the media device to the non active state.

Format

usb_media_ret_t R_USB_media_close(void);

Arguments

--

Return Value

USB_MEDIA_RET_OK	Successfully completed
USB_MEDIA_RET_PARAERR	Parameter error
USB_MEDIA_RET_OP_FAIL	Any other failure

Description

This API releases the resource for the media driver and return the media device to the non active state.

Note

1. *R_USB_media_initialize* function has to be called before calling this API.
2. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.

Example

```
result = R_USB_media_close();
if (USB_MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

8.1.4 R_USB_media_read

Read the data blocks from the media device

Format

usb_media_ret_t R_USB_media_read(uint8_t *p_buf, uint32_t lba, uint8_t count);

Argument

p_buf	Pointer to the area to store the read data from the media device
lba	Read start logical block address
count	Number of read block (Number of sector)

Return Value

USB_MEDIA_RET_OK	Successfully completed
USB_MEDIA_RET_PARAERR	Parameter error
USB_MEDIA_RET_NOTRDY	The device is not ready state
USB_MEDIA_RET_OP_FAIL	Any other failure

Description

This API reads the data blocks from the media device. (Read the data blocks for the number of blocks specified by the third argument (*count*) from the LBA (Logical Block Address) specified by the second argument.)
The read data is stored in the specified area by the first argument (*p_buf*).

Note

1. *R_USB_media_initialize* function has to be called before calling this API.
2. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.

Example

```
result = R_USB_media_read(&buffer, lba, 1);
if (USB_MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

8.1.5 R_USB_media_write

Write the data block to the media device

Format

usb_media_ret_t R_USB_media_write(uint8_t *p_buf, uint32_t lba, uint8_t count);

Arguments

p_buf	Pointer to the area where data to be written to the media device is stored
lba	Write start logical block address
count	Number of write blocks (Number of sector)

Return Value

USB_MEDIA_RET_OK	Successfully completed
USB_MEDIA_RET_PARAERR	Parameter error
USB_MEDIA_RET_NOTRDY	The device is not ready state
USB_MEDIA_RET_OP_FAIL	Any other failure

Description

This API write the data blocks to the media device. (Write the data blocks for the number of blocks specified by the third argument (*count*) to the LBA (Logical Block Address) specified by the second argument.)
Store the write data in the area specified by the first argument (*p_buf*).

Note

1. *R_USB_media_initialize* function has to be called before calling this API.
2. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.

Example

```
result = R_USB_media_write(&buffer, lba, 1);
if (MEDIA_RET_OK != result)
{
    /* Process the error */
}
```

8.1.6 R_USB_media_ioctl

Get the information of the media driver etc

Format

```
usb_media_ret_t        R_USB_media_ioctl(ioctl_cmd_t command, void *p_data);
```

Arguments

command	Command code
p_data	Pointer to the area to store the media information

Return Value

USB_MEDIA_RET_OK	Successfully completed
USB_MEDIA_RET_PARAERR	Parameter error
USB_MEDIA_RET_NOTRDY	The device is not ready state
USB_MEDIA_RET_OP_FAIL	Any other failure

Description

This API gets the return information from the media driver by specifying the media driver specific command. PMSC uses the following commands as the command code to the media driver.

MEDIA_IOCTL_GET_NUM_BLOCKS	Number of block for the media area
MEDIA_IOCTL_GET_BLOCK_SIZE	1 block size

Note

1. *R_USB_media_initialize* function has to be called before calling this API.
2. The user can ndefine the command code specified in the argument(command) newly.
3. The user needs to implement the media driver function based on the contents described in the above "Arguments", "Return Value" and "Description" etc.

Example

```
uint32_t num_blocks;
uint32_t block_size;
uint64_t capacity;.

result = R_USB_media_ioctl(MEDIA_IOCTL_GET_NUM_BLOCKS, (void *)&num_blocks);
result = R_USB_media_ioctl(MEDIA_IOCTL_GET_BLOCK_SIZE, (void *)&block_size);

capacity = (uin64_t)block_size * (uint64_t)num_blocks;
```

8.2 Structure / Enum type definition

The following shows the structure and enum type used by the media driver API.
These are defined in *r_usb_media_driver_if.h* file.

8.2.1 usb_media_driver_t (Structure)

usb_media_driver_t is the structure to hold the pointer to the media driver function implemented by the user.
The following shows *usb_media_driver_t* structure.

```
typedef struct media_driver_t
{
        usb_media_open_t     pf_media_open;     /* Pointer to the open function */
        usb_media_close_t    pf_media_close;    /* Pointer to the close function   */
        usb_media_read_t     pf_media_read;     /* Pointer to the read function */
        usb_media_write_t    pf_media_write;    /* Pointer to the write function */
        usb_media_ioctl_t    pf_media_ctrl;     /* Pointer to the control function */
} usb_media_driver_t
```

8.2.2 usb_media_ret_t (Enum)

The return value is defined in *usb_media_ret_t* (Enum).

```
typedef enum
{
        USB_MEDIA_RET_OK = 0,           /* Successfully Completed */
        USB_MEDIA_RET_NOTRDY,           /* The device is not ready state */
        USB_MEDIA_RET_PARERR,           /* Parameter error */
        USB_MEDIA_RET_OP_FAIL,          /* Any other failure */
        USB_MEDIA_RET_DEV_OPEN,         /* The device was already opened */
} usb_media_ret_t
```

8.2.3 ioctrl_cmd_t (Enum)

The command code specified in the argument of the *R_USB_media_ioctl* function is defined in *ioctl_cmd_t* (Enum).

```
typedef enum
{
   USB_MEDIA_IOCTL_GET_NUM_BLOCKS,        /* Get the number of the logical block */
   USB_MEDIA_IOCTL_GET_BLOCK_SIZE,        /* Get the logical block size */
} ioctl_cmd_t
```

Note:

Please add the command code in the *ioctl_cmd_t* when adding the user own command code.

8.3 Registration of the storage media driver

To change the PMSC's storage media from RAM to something else, such as flash memory, the user has to implement media driver functions to handle reading from and writing to the new storage media and register them to the media driver API functions.

The example below shows the procedure for changing from RAM media to serial SPI flash.

1. **Creating Media Driver Functions**

   Assume that the following functions are implemented by the user as media driver functions for serial SPI flash.

```
1.    usb_media_ret_t      spi_flash_open (void)
2.    usb_media_ret_t      spi_flash_close (void)
3.    usb_media_ret_t      spi_flash_read(uint8_t *p_buf,uint32_t lba, uint8_t count)
4.    usb_media_ret_t      spi_flash_write(uint8_t *p_buf,uint32_t lba, uint8_t count)
```

5. usb_media_ret_t spi_flash_ioctl(ioctl_cmd_t ioctl_cmd,void * ioctl_data)

2. Registering the Media Driver Functions with the Media API

(1). Define the structure *usb_media_driver_t* for the serial SPI flash. As the members of this structure, specify pointers to the relevant media driver functions.

```
struct media_driver_t    g_spi_flash_mediadriver =
{
      &spi_flash_open,
      &spi_flash_close,
      &spi_flash_read,
      &spi_flash_write,
      &spi_flash_ioctl
};
```

(2). In the application program, specify the pointer to *usb_media_driver_t* structure to the argument in *R_USB_media_initialize* function (API), and perform initialization processing.

```
== Application Program ==
R_USB_media_initialize(& g_spi_flash_mediadriver );
```

The serial SPI flash function is registered as the media driver function called by the media drvier by doing the above order.

8.4 Implementation of the strorage media dirver

The user needs to implement the media driver function for controlling the storage media to be used.
The implemented media driver function is called from PMSC via the API described in chapter 8.1, **Overview of Media Driver API Functions** from PMSC.

Note:
For the necessary processing to implement the media driver function, refer to each API specification described in chapter **8.1, Overview of Media Driver API Functions.**

8.5 Prototype Declaration of Media Driver function

The following shows the prototype declaration of the media driver function.

```
1. usb_media_ret_t  (*media_open_t) (uint8_t);                                  /* Open function type */
2. usb_media_ret_t  (*media_close_t)(uint8_t);                                  /* Close function type */
3. usb_media_ret_t (*media_read_t)(uint8_t, uint8_t*, uint32_t, uint8_t);     /* Read function type */
4. usb_media_ret_t (*media_write_t)(uint8_t, uint8_t*, uint32_t, uint8_t);    /* Write function type */
5. usb_media_ret_t  (*media_ioctl_t)(uint8_t, ioctl_cmd_t, void *);            /* Control function type */
```

9. Sample Application

9.1 Application Specification

The PMSC sample application (APL) runs on the RSK. When the RSK is connected to the host PC it is recognized as a removable disk, and data transfers, such as reading and writing files, can be performed.
Figure 9-1 shows an example PMSC operating environment, and Figure 9-2 shows a PMSC operation example.

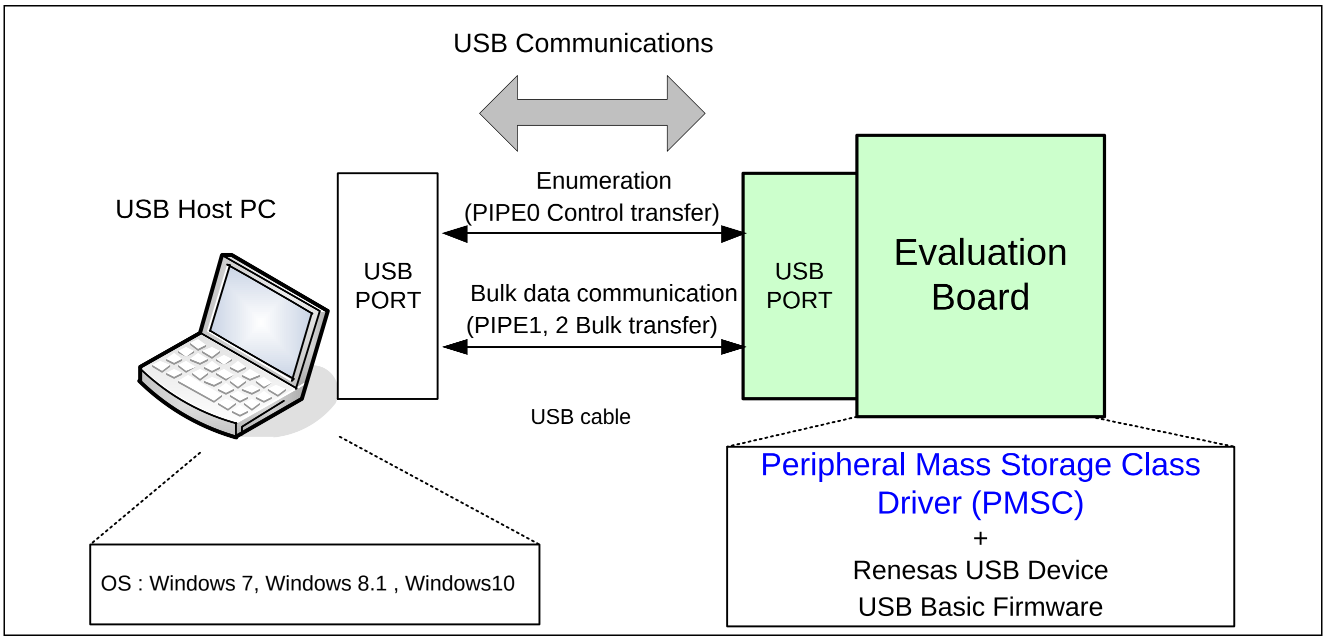
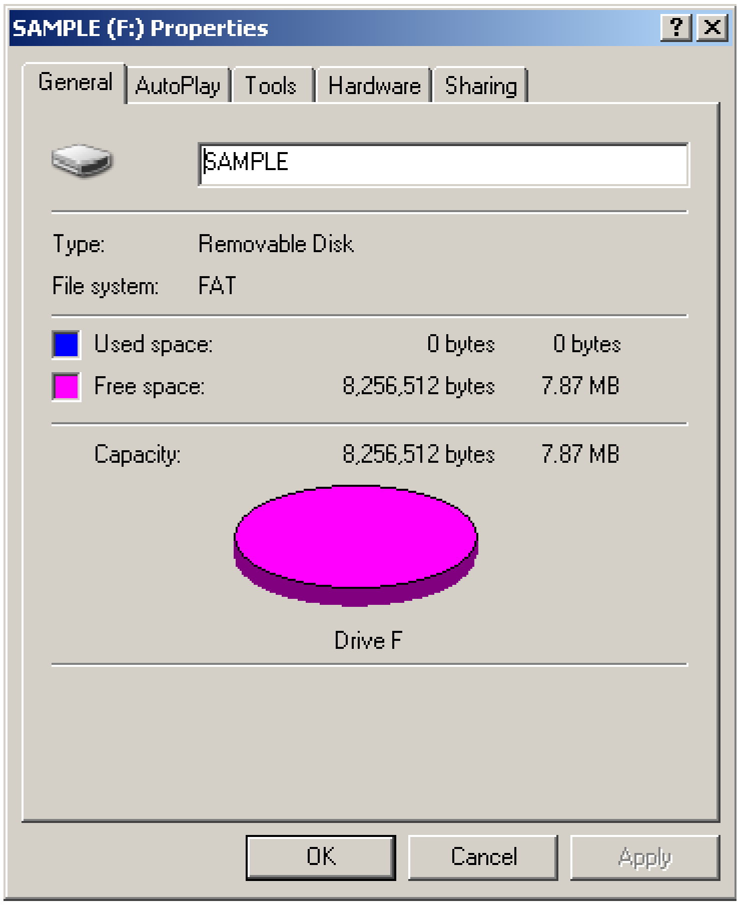


Figure 9-1 PMSC Operation Environment

9.1.1 Media Area on Removable Disk

The APL uses the SDRAM or Internal RAM area on the RSK as the media area of a removable disk.



SAMPLE

Type:	Removable Disk	
File system:	FAT	

🟦	Used space:	0 bytes	0 bytes
🟪	Free space:	8,256,512 bytes	7.87 MB
	Capacity:	8,256,512 bytes	7.87 MB

Drive F

Figure 9-2 PMSC Operation Example

9.2 Application Processing

The application comprises two parts: initial settings and main loop.

Initial setting	:	Makes MCU pin settings, initializes the USB controller, and initializes the USB driver.
Main loop	:	Calls R_USB_GetEvent function in the loop. If a Suspend request is received from the USB host or the USB host is detached while the loop is being processed, the APL transitions the MSC device (RSK) to low-power mode. For details of low-power mode, see **9.3, MCU Low power consumption processing**

PMSC controls processing by a mass storage class driver (MSCD) and mass storage device driver (MSDD) in response to requests from the USB host (PC). Therefore, the PMSC APL does not perform any processing on data transferred from the host. Aside from initialization processing, the only thing performed within the loop is calling the R_USB_GetEvene function. The APL does not write files to or read files from the PMSC storage area; this processing is all performed by the PMSC USB driver.

Note:

1. For a list of the storage commands supported by the PMSC, see **4.2, Storage Commands**.
2. Make sure to call the R_USB_GetEvent function from within the application program loop processing.

An overview of the processing performed by the APL is shown below:

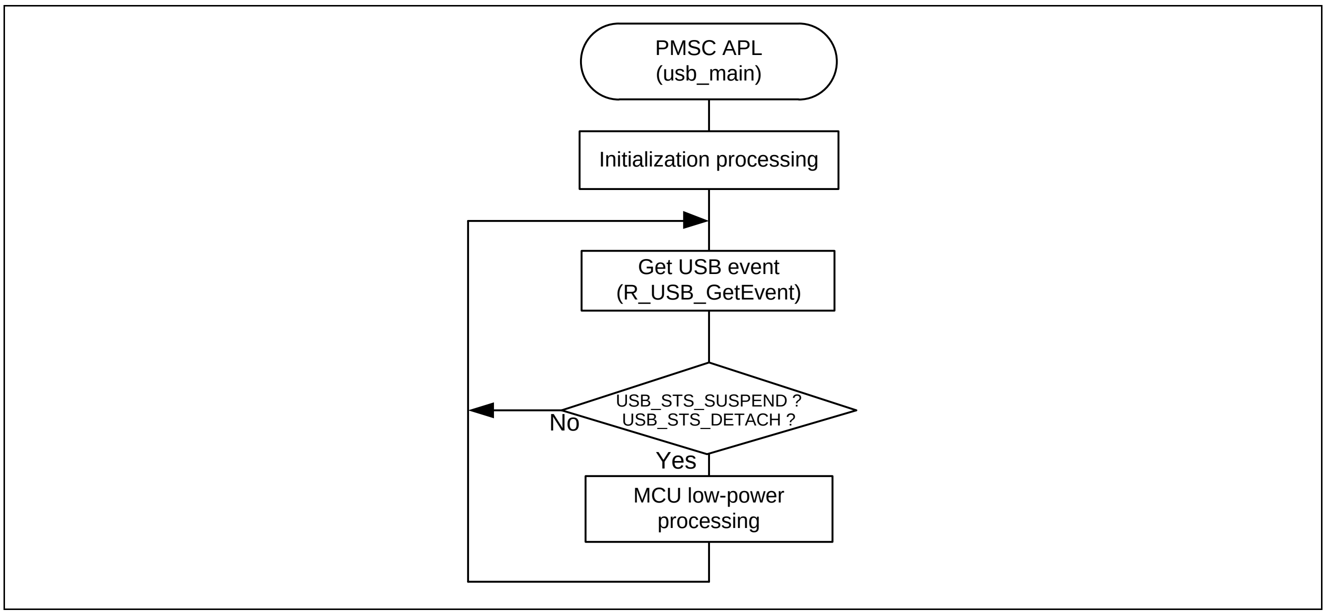


Figure 9-3 APL Processing Overview

9.3 MCU Low power consumption processing

MCU low-power processing occurs when the conditions in **Table 9-1** are met, causing a transition to low-power mode. Note that this processing is enabled by setting the USE_LPW definition in the r_usb_pmsc_apl_config.h file to USE_LPW.

Table 9-1 Conditions for Transition to Low-Power Mode

Transition Condition		**Transition Status**
VBUS	USB State	
OFF	—	Software standby mode
ON	Suspend Configured	Sleep mode
ON	Other Suspend Configured	Normal mode (program running)

(1). When the MSC device (RSK) detaches from the USB host (VBUS OFF), the APL performs processing to transition the MCU to software standby mode. Recovery from software standby mode occurs when the MSC device (RSK) attaches to the USB host.

(2). When a suspend signal sent by the USB host is received while the MSC device (RSK) is connected to the USB host, the APL performs processing to transition the MCU to sleep mode. Note that recovery from sleep mode occurs when a resume signal is received from the USB host.

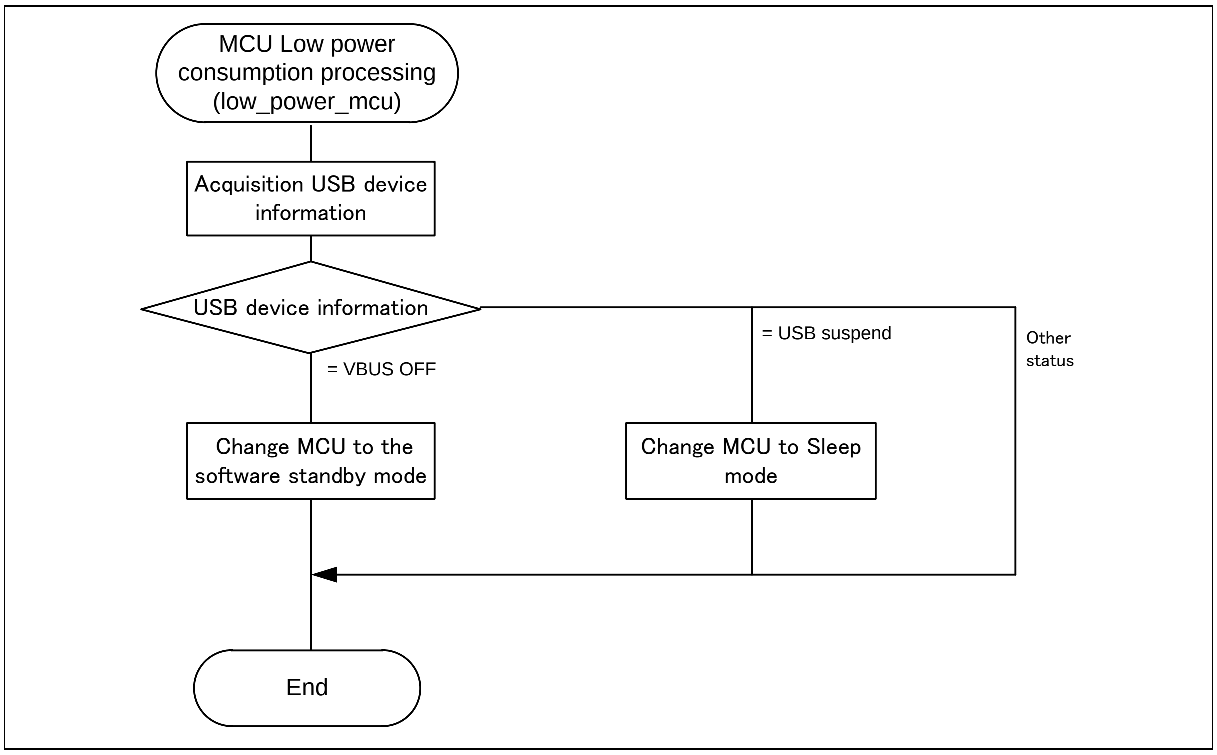

MCU Low power consumption processing (low_power_mcu)
Acquisition USB device information
USB device information
= VBUS OFF
= USB suspend
Other status
Change MCU to the software standby mode
Change MCU to Sleep mode
End

Figure 9-4 Flowchart of MCU Low Power Consumption Processing

9.4 Configuration File for the application program (r_usb_pmsc_apl_config.h)

Make settings for the definitions listed below.

1. USE_USBIP Definition

Specify the module number of the USB module you are using. Specify one of the following settings for the USE_USBIP definition.

```
#define     USE_USBIP          USE_USBIP0    // Specify USB_IP0.
#define     USE_USBIP          USE_USBIP1    // Specify USB_IP1.
```

[Note]
Specify *USE_USBIP0* when using RX63T or RX630.

2. USB_SUPPORT_SPEED Definition

Specify the USB operation speed (Full-speed) which MSC device (RSK) supports.

```
#define     USB_SUPPORT_SPEED  FULL_SPEED    // Full-Speed setting
```

3. Low-Power Function Definition

Specify whether or not the low-power function will be used. If the low-power function will be used, specify *USB_APL_ENABLE* to *USB_SUPPORT_LPW* definition.

```
#define     USE_SUPPORT_LPW    USB_APL_DISABLE   // No use the low-power function
#define     USB_SUPPORT_LPW    USB_APL_ENABLE    // Use the low-power function
```

4. Note

The above configuration settings apply to the application program. USB driver configuration settings are required in addition to the above settings. For information on USB driver configuration settings, refer to *USB Basic Host and Peripheral Driver Application Note* (Document number. R01AN0512EJ).

9.5 Descriptor

The PMSC's descriptor information is contained in r_usb_pmsc_descriptor.c. Also, please be sure to use your vendor ID.

10. Setup

10.1 Hardware

10.1.1 Example Operating Environment

Figure 10-1 shows an example operating environment for the PMSC. Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc

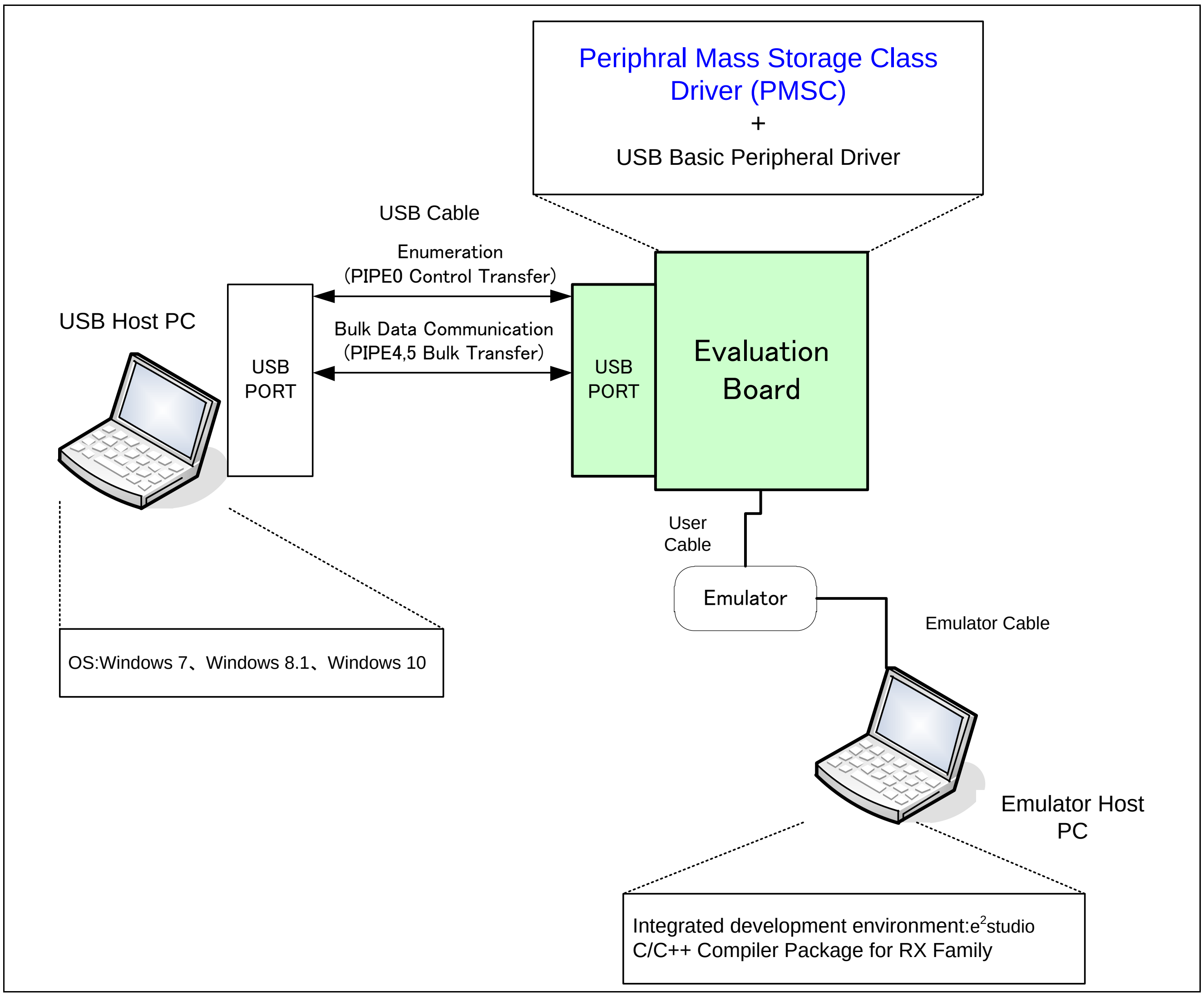

Periphral Mass Storage Class Driver (PMSC)
+
USB Basic Peripheral Driver
USB Cable
Enumeration (PIPE0 Control Transfer)
Bulk Data Communication (PIPE4,5 Bulk Transfer)
USB Host PC
USB PORT
USB PORT
Evaluation Board
User Cable
Emulator
Emulator Cable
OS:Windows 7、Windows 8.1、Windows 10
Emulator Host PC
Integrated development environment:e²studio
C/C++ Compiler Package for RX Family

Figure 10-1 Example Operating Environment

Note:

1. PC environment setting
   No special settings are required on the USB host PC so long as it is running Microsoft Windows 7 or later.

10.1.2 RSK Setting

It is necessary to set RSK to operate in the host mode. Please refer to the following.

Table 10-1 **RSK Setting**

RSK	Jumper Setting
RSK+RX63N	J3: Shorted Pin1-2 J4: Shorted Pin1-2 J18:Shorted Pin2-3

Note:

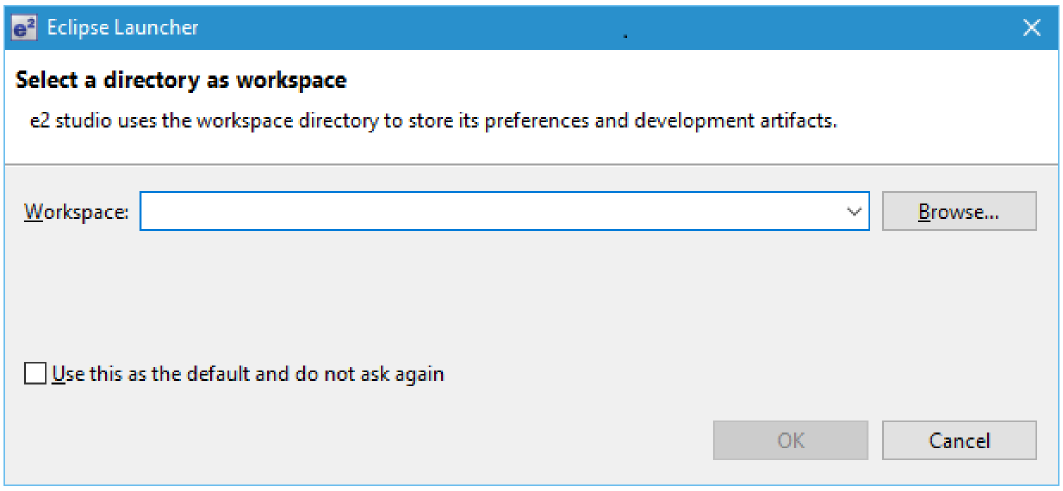
For the detail of RSK setting, refer to the user's manual of RSK.

10.2 Software

(1). Setup e^2 studio

a) Start e^2 studio

b) If you start up e^2 studio at first, the following dialog is displayed. Specify the folder to store the project in this dialog.


Eclipse Launcher

Select a directory as workspace

e2 studio uses the workspace directory to store its preferences and development artifacts.

Workspace: [ ] Browse...

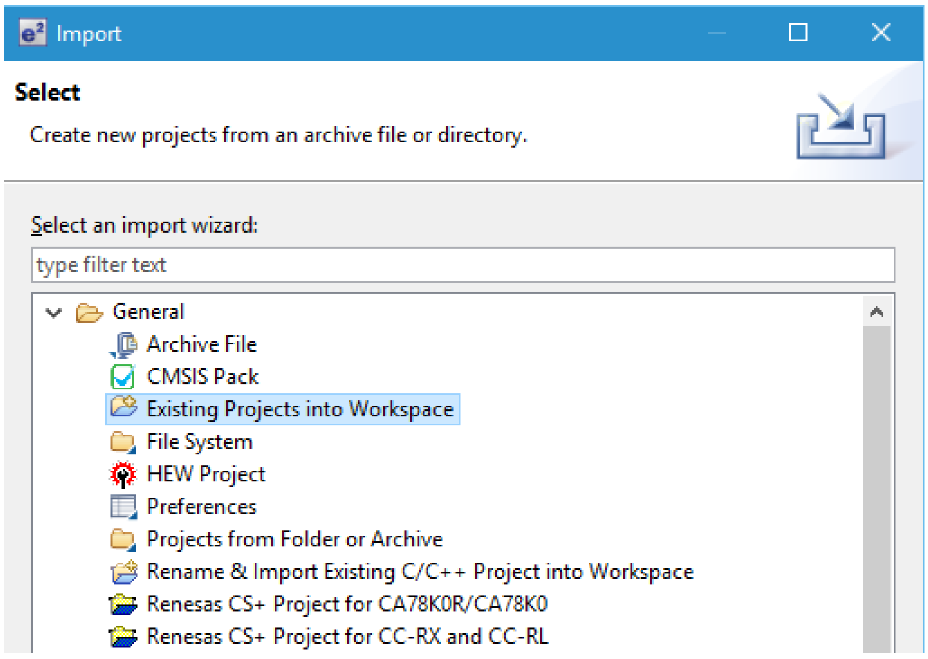
☐ Use this as the default and do not ask again

OK Cancel

(2). Import the project to the workspace

a) Select [File] > [Import]

b) Select [General] => [Existing Projects into Workspace]


Import

Select

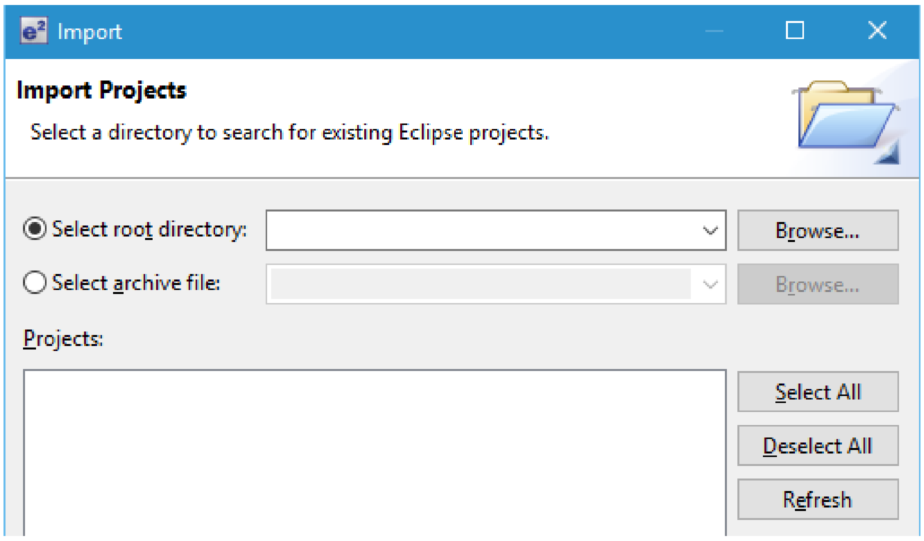
Create new projects from an archive file or directory.

Select an import wizard:

type filter text

- General
  - Archive File
  - CMSIS Pack
  - Existing Projects into Workspace
  - File System
  - HEW Project
  - Preferences
  - Projects from Folder or Archive
  - Rename & Import Existing C/C++ Project into Workspace
  - Renesas CS+ Project for CA78K0R/CA78K0
  - Renesas CS+ Project for CC-RX and CC-RL

c) Select the root directory of the project, that is, the folder containing the ".cproject" file.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

● Select root directory: [ ] Browse...

○ Select archive file: [ ] Browse...

Projects:

d) Click "Finish".

You have now imported the project into the workspace. Note that you can import other projects into the same workspace.

(3). Generate the binary target program by clicking the "Build" button.

(4). Connect the target board to the debug tool and download the executable. The target is run by clicking the "Run" button.

11. Creating an Application

Refer to the chapter “**Creating an Application Program**” in the document (Document number: R01AN0512) for *USB Basic Host and Peripheral Driver Application Note*.

Note:

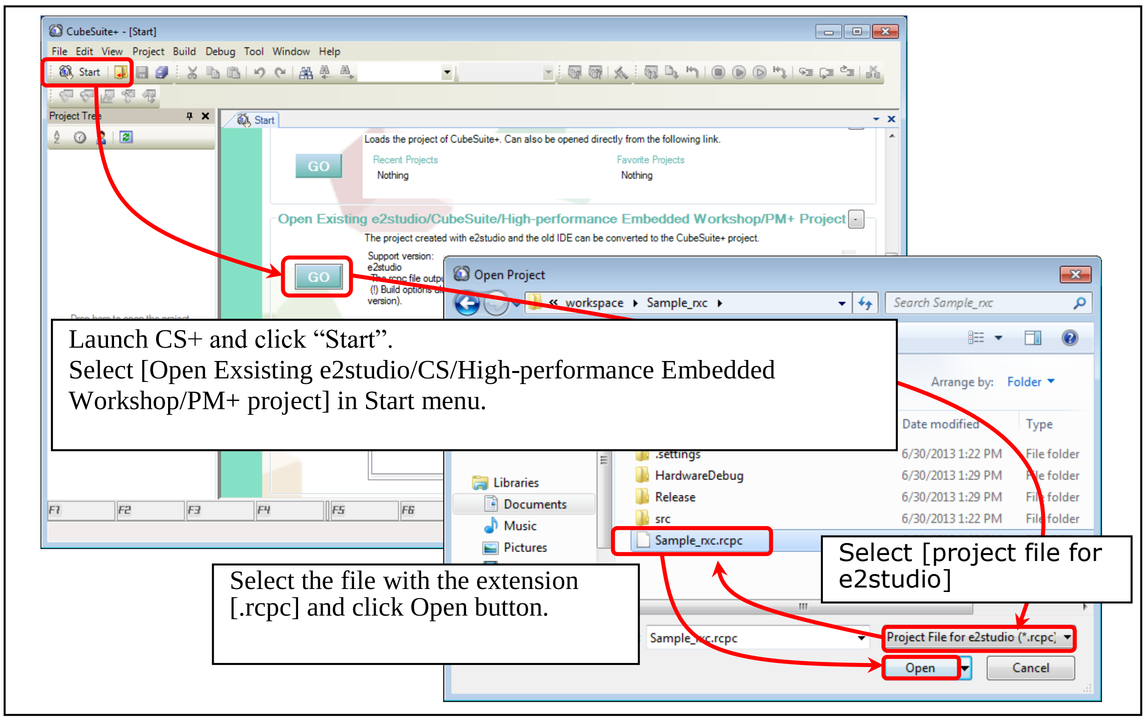
Be sure to call *R_USB_media_initialize* function (API) and *R_USB_media_open* function (API) at the initialize processing etc in the user application program.

12. Using the e^2 studio project with CS+

The PMSC contains a project only for e^2 studio. When you use the PMSC with CS+, import the project to CS+ by following procedures.

Note:

Uncheck the checkbox Backup the project composition files after conversion in Project Convert Settings window.

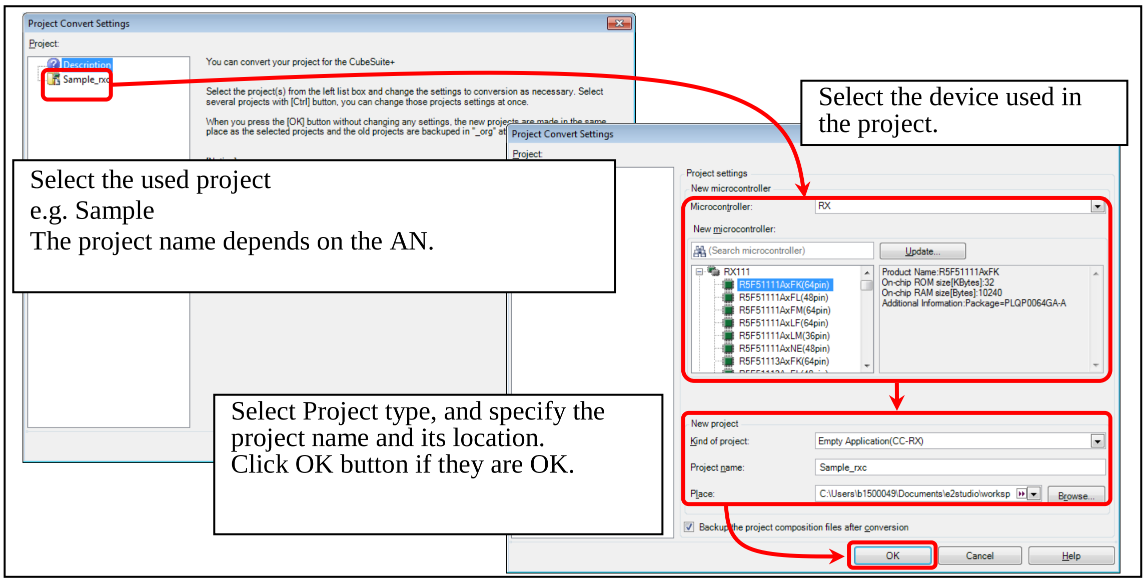


Launch CS+ and click "Start".
Select [Open Exsisting e2studio/CS/High-performance Embedded Workshop/PM+ project] in Start menu.

Select the file with the extension [.rcpc] and click Open button.

Select [project file for e2studio]

CubeSuite+ - [Start]
File Edit View Project Build Debug Tool Window Help
Start
Project Tree
Loads the project of CubeSuite+. Can also be opened directly from the following link.
GO
Recent Projects
Nothing
Favorite Projects
Nothing
Open Existing e2studio/CubeSuite/High-performance Embedded Workshop/PM+ Project
The project created with e2studio and the old IDE can be converted to the CubeSuite+ project.
Open Project
workspace ▸ Sample_rxc ▸
Search Sample_rxc
Arrange by: Folder
Date modified
Type
.settings 6/30/2013 1:22 PM File folder
HardwareDebug 6/30/2013 1:29 PM File folder
Release 6/30/2013 1:29 PM File folder
src 6/30/2013 1:22 PM File folder
Sample_rxc.rcpc
Libraries
Documents
Music
Pictures
Sample_rxc.rcpc
Project File for e2studio (*.rcpc)
Open
Cancel



Select the device used in the project.

Select the used project
e.g. Sample
The project name depends on the AN.

Select Project type, and specify the project name and its location.
Click OK button if they are OK.

Project Convert Settings
Project:
Description
Sample_rxc
You can convert your project for the CubeSuite+
Select the project(s) from the left list box and change the settings to conversion as necessary. Select several projects with [Ctrl] button, you can change those projects settings at once.
Project settings
New microcontroller
Microcontroller: RX
New microcontroller:
(Search microcontroller)
Update...
RX111
R5F51111AxFK(64pin)
R5F51111AxFL(48pin)
R5F51111AxFM(64pin)
R5F51111AxLF(64pin)
R5F51111AxLM(36pin)
R5F51111AxNE(48pin)
R5F51113AxFK(64pin)
Product Name:R5F51111AxFK
On-chip ROM size[KBytes]:32
On-chip RAM size[Bytes]:10240
Additional Information:Package=PLQP0064GA-A
New project
Kind of project: Empty Application(CC-RX)
Project name: Sample_rxc
Place: C:\Users\b1500049\Documents\e2studio\worksp
Browse...
Backup the project composition files after conversion
OK
Cancel
Help

Figure 12-1 Using the e^2 studio project with CS+

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar.09.11	—	First edition issued
1.10	Aug.10.11	—	Add Target Device RX630, R8A66597 Add the information on RX630 and R8A66597 (Hi-Speed USB)
		25	5.Peripheral HID Sample Application (APL) change to Application - Unused SW1 - SW description is changed to the function switch description
2.00	Sep.30.12	—	Revision of the document by firmware upgrade
2.10	Apr.1.13	—	First Release for V.2.10 Add Target Device RX63T and R8A66593. Add the information on RX63T and R8A66593.
2.20	Sep.30.15	—	Change the application program. Change the folder structure. RX63N, RX631, R8A66597 and R8A66593 are deleted from Target Device.
2.30	Sep 30, 2016	—	Supporting USB Host and Peripheral Interface Driver application note(Document No.R01AN3293EJ)
2.31	Sep 30, 2017	—	1. DMA/DTC transfer has been changed to unsupported. 2. The contents of USB Host and Peripheral Interface Driver application note (Document number: R01AN3293EJ) is moved to this document and USB Host and Peripheral Interface Driver application note is deleted.
2.32	Mar 31, 2018	—	The revision of USB Basic driver has been updated.
2.33	Jul 31, 2019	—	RX63N and RX631 are added in Target Device.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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

Renesas Electronics (China) Co., Ltd.
Room 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, 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, 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 No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

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

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