

RX Family

R01AN2172EJ0120
Rev.1.20
Jun 1, 2020

USB Peripheral Mass Storage Class Driver for USB Mini Firmware Using Firmware Integration Technology

Introduction

This application note describes USB peripheral mass storage class driver (PMSC), which utilizes Firmware Integration Technology (FIT). This module operates in combination with the USB Basic Mini Host and Peripheral Driver. It is referred to below as the USB PMSC FIT module.

Target Device

RX111 Group
RX113 Group
RX231 Group
RX23W 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. Universal Serial Bus 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. RX111 Group User’s Manual: Hardware (Document number .R01UH0365)
5. RX113 Group User’s Manual: Hardware (Document number.R01UH0448)
6. RX231 Group User’s Manual: Hardware (Document number .R01UH0496)
7. RX23W Group User’s Manual: Hardware (Document number .R01UH0823)
8. USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note (Document number.R01AN2166)

Renesas Electronics Website

<http://www.renesas.com>

USB Devices Page

<http://www.renesas.com/prod/usb/>

Contents

1. Overview	3
2. Software Configuration	4
3. API Information	5
4. Class Driver Overview.....	8
5. Peripheral Device Class Driver (PDCD)	9
6. API Functions	10
7. Configuration (r_usb_pmsc_mini_config.h).....	11
8. Configuration File (When using RI600V4)	12
9. Media Driver Interface	13
10. Creating an Application	23

1. Overview

The USB PMSC FIT module, when used in combination with the USB-BASIC-F/W FIT module, 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: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note* when creating an application program using this driver.

This document is located in the "**reference_documents**" folder within this package.

1.2 Limitation

1. This driver returns the value 0 (zero) to the mass storage command (*GetMaxLun*) sent from USB Host.
2. The sector size which this driver supports is 512 only.

1.3 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. The user needs to implement the media driver function which controls the media area used as the storage area.

1.4 Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

APL	:	Application program
BOT	:	USB mass storage class bulk only transport. See "Universal Serial Bus Mass Storage Class Bulk-Only Transport" at USB Implementers Forum..
DDI	:	Device driver interface, or PMSDD API.
IDE	:	Integrated Development Environment
PCD	:	Peripheral control driver of
PCI	:	PCD interface
PMSCD	:	Peripheral mass storage USB class driver (PMSCF + PCI + DDI)
PMSCF	:	Peripheral mass storage class function
PMSDD	:	Peripheral mass storage device driver (ATAPI driver)
RSK	:	Renesas Starter Kits
RTOS	:	USB Driver for the real-time OS
USB-BASIC-FW	:	USB Basic Mini Host and Peripheral Driver

1.5 USB PMSC FIT Module

User needs to integrate this module to the project using `r_usb_basic_mini`. User can control USB H/W by using this module API after integrating to the project.

2. Software Configuration

PMSC FIT module comprises two layers: PMSCD and PMSDD.

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.

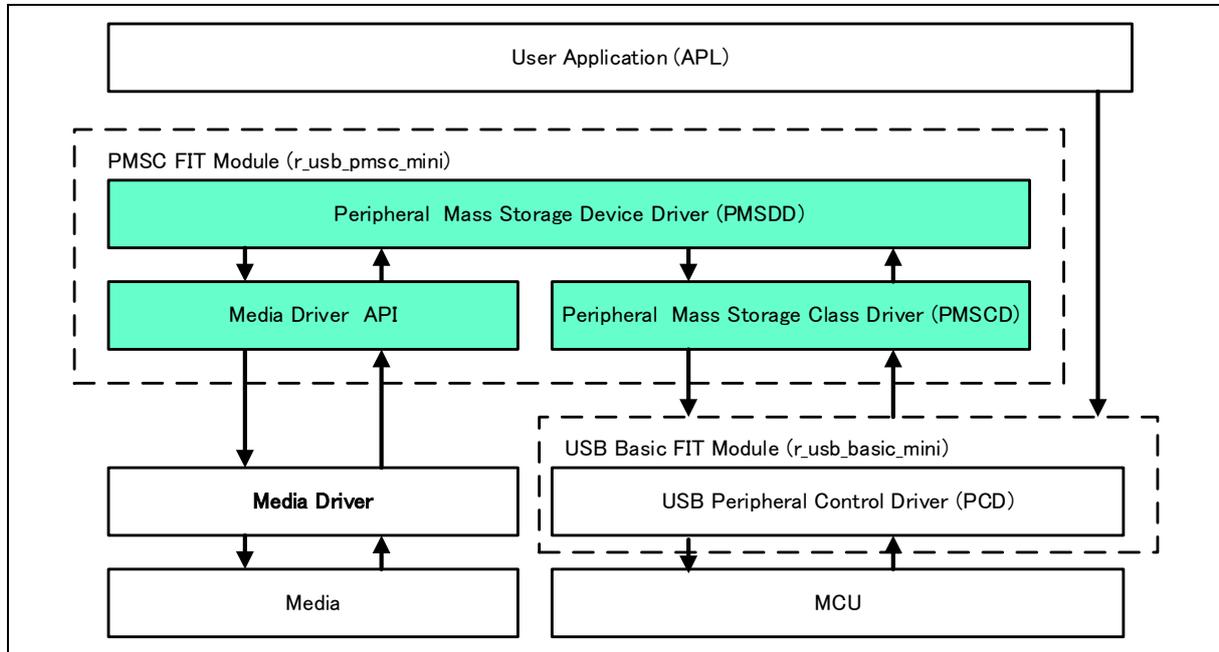


Figure 2-1 Software Module Structure

Table 2-1 Module Function Overview

Module	Description
PMSDD	Mass Storage Device Driver <ul style="list-style-type: none"> Processes storage commands from the PMSCD Accesses media via the media driver
PMSCD	Mass Storage Class Driver <ul style="list-style-type: none"> Controls BOT protocol data and responds to class requests. Analyzes CBWs and transmits/receives data. Generates CSWs together with the PMSDD/PCD.
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 Software Requirements

This driver is dependent upon the following packages:

- r_bsp
- r_usb_basic_mini

3.3 Operating Confirmation Environment

Table 3-1 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.3.02.00 (The option "-lang=C99" is added to the default setting of IDE)
	GCC for Renesas RX 8.3.0.201904 (The option "-std=gnu99" is added to the default setting of IDE)
	IAR C/C++ Compiler for Renesas RX version 4.14.1
Real-Time OS	FreeRTOS V.10.0.0 RI600V4 V.1.06
Endian	Little Endian, Big Endian
USB Driver Revision Number	Rev.1.20
Using Board	Renesas Starter Kit for RX111 Renesas Starter Kit for RX113 Renesas Starter Kit for RX231 Renesas Solution Starter Kit for RX23W
Host Environment	The operation of this USB Driver module connected to the following OSes has been confirmed. <ol style="list-style-type: none"> 1. Windows® 8.1 2. Windows® 10

3.4 Usage of Interrupt Vector

Table 3-2 shows the interrupt vector which this driver uses.

Table 3-2 List of Usage Interrupt Vectors

Device	Contents
RX111	USBIO Interrupt (Vector number: 36) / USBR0 Interrupt (Vector number: 90) USB D0FIFO0 Interrupt (Vector number: 36) / USB D1FIFO0 Interrupt (Vector number: 37)
RX113	
RX231	
RX23W	

3.5 Header Files

All API calls and their supporting interface definitions are located in *r_usb_basic_mini_if.h* and *r_usb_pmesc_mini_if.h*.

3.6 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*.

3.7 Compile Setting

For compile settings, refer to chapter 7, **Configuration (r_usb_pmesc_mini_config.h)** in this document and chapter "Configuration" in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note*.

3.8 ROM / RAM Size

The follows show ROM/RAM size of this driver.

1. CC-RX (Optimization Level: Default)

(1). Non-OS

	Checks arguments	Does not check arguments
ROM size	20.1K bytes (Note 3)	19.8K bytes (Note 4)
RAM size	8.0K bytes	8.0K bytes

(2). RI600V4

	Checks arguments	Does not check arguments
ROM size	36.7K bytes (Note 3)	36.4K bytes (Note 4)
RAM size	8.0K bytes	8.0K bytes

(3). FreeRTOS

	Checks arguments	Does not check arguments
ROM size	32.7K bytes (Note 3)	32.4K bytes (Note 4)
RAM size	19.1K bytes	19.1K bytes

2. GCC (Optimization Level: -O2)

	Checks arguments	Does not check arguments
ROM size	17.1K bytes (Note 3)	16.9K bytes (Note 4)
RAM size	6.9K bytes	6.9K bytes

3. IAR (Optimization Level: Medium)

	Checks arguments	Does not check arguments
ROM size	11.7K bytes (Note 3)	11.5K bytes (Note 4)
RAM size	2.8K bytes	2.8K bytes

[Note]

1. ROM/RAM size for BSP and USB Basic Driver is included in the above size.
2. The above is the size when specifying RX V2 core 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_mini_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_mini_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_mini_config.h` file.

3.9 Argument

For the structure used in the argument of API function, refer to chapter "**Structures**" in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note*.

3.10 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

- (1) Adding the FIT module to your project using “Smart Configurator” on e² studio
By using the Smart Configurator in e² studio, the FIT module is automatically added to your project. Refer to “Renesas e² studio Smart Configurator User Guide (R20AN0451)” for details.
- (2) Adding the FIT module to your project using the FIT Configurator in e² studio
By using the FIT Configurator in e² studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.
- (3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e² studio Smart Configurator User Guide (R20AN0451)” for details.
- (4) Adding the FIT module to your project on CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

4. Class Driver Overview

4.1 Class Requests

Table 4-1 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

Table 4-2 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.

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).

Figure 5-1 shows an overview of how the BOT protocol progresses with command and status data flowing between USB host and peripheral.

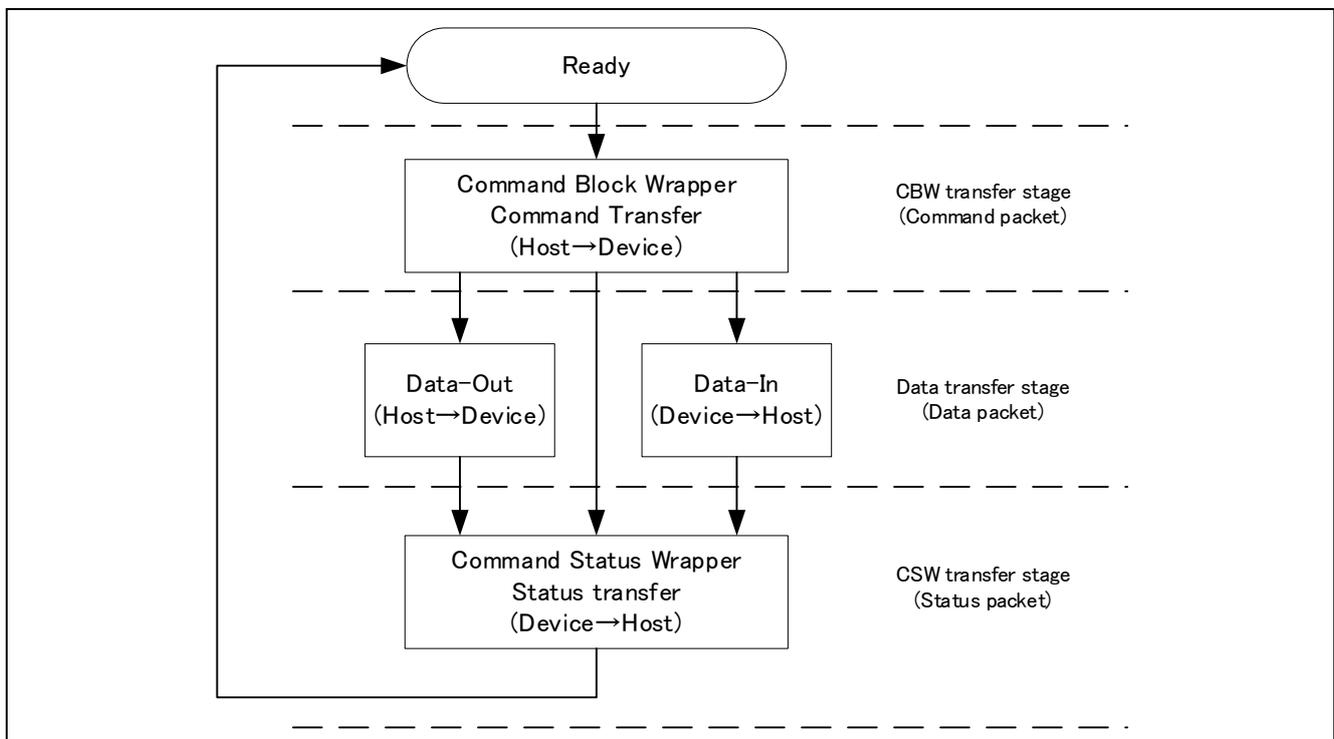


Figure 5-1 BOT protocol Overview.
Command and status flow between USB host and peripheral.

6. API Functions

For API used in the application program, refer to chapter "**API Functions**" in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note*.

7. Configuration (r_usb_pmesc_mini_config.h)

Please set the following according to your system.

Note:

Be sure to set *r_usb_basic_mini_config.h* file as well. For *r_usb_basic_mini_config.h* file, refer to chapter "**Configuration**" in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology 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_PMESC_BULK_IN* and *USB_CFG_PMESC_BULK_OUT*.

```
#define    USB_CFG_PMESC_BULK_IN        Pipe number (USB_PIPE1 to USB_PIPE5)
#define    USB_CFG_PMESC_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. Be sure to enclose data of 8 bytes with double quotation marks.

```
#define    USB_CFG_PMESC_VENDOR        Vendor Information
e.g)
#define    USB_CFG_PMESC_VENDOR        "Renesas "
```

(2). Setting Product Information

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

```
#define    USB_CFG_PMESC_PRODUCT       Product Information
e.g)
#define    USB_CFG_PMESC_PRODUCT       "Mass Storage "
```

(3). Setting Product Revision Level

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

```
#define    USB_CFG_PMESC_REVISION      Product Revision Level
e.g)
#define    USB_CFG_PMESC_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_PMESC_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_PMESC_TRANS_COUNT*" bytes of RAM will be consumed.

```
#define    USB_CFG_PMESC_TRANS_COUNT   Number of transfer sectors (1 to 32)
e.g)
#define    USB_CFG_PMESC_TRANS_COUNT   4
```

8. Configuration File (When using RI600V4)

It is necessary to register the OS resource used by USB PMSC driver to RI600V4 when using RI600V4. Please add the following definition in the configuration file. For how to create the configuration file, refer to the chapter, "**RI600V4(Configuration File Creation)**" in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology Application Note*.

8.1 Task Definition

name	:	ID_USB_RTOS_PMSC_TSK
entry_address	:	usb_pstd_pmsc_task()
stack_size	:	512
initial_start	:	OFF
exinf	:	0

8.2 Mailbox Definition

name	:	ID_USB_RTOS_PMSC_MBX
wait_queue	:	TA_FIFO
message_queue	:	TA_MFIFO

9. Media Driver Interface

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

9.1 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 9-1 shows the list of the media driver API functions.

Table 9-1 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.

9.1.1 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 chapter **9.3, 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 */
}
```

9.1.2 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 */
}
```

9.1.3 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 */  
}
```

9.1.4 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 */
}
```

9.1.5 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 */
}
```

9.1.6 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;
```

9.2 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.

9.2.1 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
```

9.2.2 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
```

9.2.3 ioctl_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.

9.3 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 driver by doing the above order.

9.4 Implementation of the storage media driver

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 9,

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 9,

Overview of Media Driver API Functions.

9.5 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 */*

10. Creating an Application

Refer to the chapter “**Creating an Application Program**” in the document (Document number: R01AN2166) for *USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) using Firmware Integration Technology 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.

Website and Support

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/inquiry>

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Dec 1, 2014	—	First edition issued
1.01	Jun 1, 2015	—	RX231 is added in the target device.
1.02	Dec 28, 2015	—	Checked the operation with Windows® 10.
1.10	Nov 30, 2018	—	<ol style="list-style-type: none">1. Supporting Smart Configurator.2. The following chapters are added.<ol style="list-style-type: none">(1). 4. Class Driver Overview(2). 6. API Functions(3). 7. Configuration (r_usb_pmsc_mini_config.h)(4). 8. Media Driver Interface3. The following chapters are changed.<ol style="list-style-type: none">(1). 3. API Information(2). 9. Creating an Application4. The following chapters are deleted. "How to Register Class Driver", "System Resources", "Task ID and Priority Setting", " USB Peripheral Mass Storage Class Driver", " Peripheral Mass Storage Device Driver ".
1.11	May 31, 2019	—	Support GCC compiler and IAR compiler.
1.12	Jun 30, 2019	—	RX23W is added in the target device.
1.20	Jun 1, 2020	—	Support the real time OS.

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.

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)



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. Milpitas Campus

1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.

Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics America Inc. San Jose Campus

6024 Silver Creek Valley Road, San Jose, CA 95138, USA

Tel: +1-408-284-8200, Fax: +1-408-284-2775

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, #06-02 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 Pengerutarcua 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