# RZ/A1H Group

## USB Peripheral Mass Storage Class Driver (PMSC)

R01AN3507EJ0120
Rev.1.20
Aug 31, 2017

## Introduction

This application note describes 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.

## Target Device

RZ/A1H Group

## Related Documents

1. Universal Serial Bus Revision 2.0 specification
2. RZ/A1H Group, RZ/A1M Group User's Manual: Hardware (Document No.R01UH0403EJ)
3. RZ/A1H Group USB Host and Peripheral Interface Driver (Document No.R01AN3291EJ)
4. RZ/A1H Group Downloading Program to NOR Flash Memory Using ARM® Development Studio 5(DS-5™) Semi hosting Function (for GENMAI) (Document No.R01AN1957EJ)
5. RZ/A1H Group I/O definition header file (Document No.R01AN1860EJ)
6. RZ/A1H Group Example of Initialization (for GENMAI) (Document No.R01AN1864EJ

— Renesas Electronics Website
https://www.renesas.com/
— USB Devices Page
http://japan.renesas.com/prod/usb/

## Contents

# 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.

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

## 1.1 Operating Confirmation Environment

The operation of the USB Driver module has been confirmed under the conditions listed in.

| Item | Description |
|---|---|
| MCU | RZ/A1H |
| Operating frequency | CPU clock (Iφ): 400 MHz<br>Image-processing clock (Gφ): 266.37 MHz<br>Internal bus clock (Bφ): 133.33 MHz<br>Peripheral clock 1 (P1φ): 66.67 MHz<br>Peripheral clock 0 (P0φ): 33.33 MHz |
| Operating voltage | Power supply voltage (I/O): 3.3 V<br>Power supply voltage (internal): 1.8 V |
| Integrated development environment | ARM Integrated Development Environment<br>ARM Development Studio (DS-5™) Version 5.26 |
| | IAR Integrated Development Environment<br>IAR Embedded Workbench for ARM Version 7.70 |
| Compiler | ARM C/C++ Compiler/Linker/Assembler Ver.5.03 [Build 102] |
| | KPIT GNUARM-RZ v14.01 |
| | IAR C/C++ Compiler for ARM 7.40 |
| Operating mode | Boot mode 0 (CS0-space 16-bit booting) |
| Board | GENMAI board<br>R7S72100 CPU board (RTK772100BC00000BR) |
| Device<br>(Functions used on the board) | USB1 connector, USB2 connector |

## 1.2 Limitations

This module is subject to the following restrictions.

1. Structures are composed of members of different types (Depending on the compiler, the address alignment of the structure members may be shifted).
2. This driver returns a value of 0 in response to the mass storage class command GetMaxLun sent from a USB host.
3. The only sector size supported by this driver is 512.

## 1.3 Note

1. This driver does not provide any guarantees with regard to USB communication activity. When employing this driver in a system, from the outset be sure to perform operation verification to confirm connection with a wide variety of devices.
2. You are responsible for implementing the media driver functions that are to control the media to be used as storage areas.

## 1.4 Terms and Abbreviations

| | | |
|---|---|---|
| APL | : | Application program |
| API | : | Application programing Interface |
| BOT | : | Mass storage class Bulk Only Transport |
| DDI | : | Device Driver Interface |
| PCD | : | Peripheral Control Driver |
| 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) |
| USB | : | Universal Serial Bus |
| USB-BASIC-FW | : | USB Basic Peripheral Driver |
| USB Host | : | PMC USB Host |
| data transfer | : | Control transfer, Bulk transfer, Interrupt transfer |

# 2. Operating environment

Figure 2.1 shows an example operating environment for PMSC. Refer to the associated instruction manuals for details on setting up the evaluation board and using the emulator, etc.

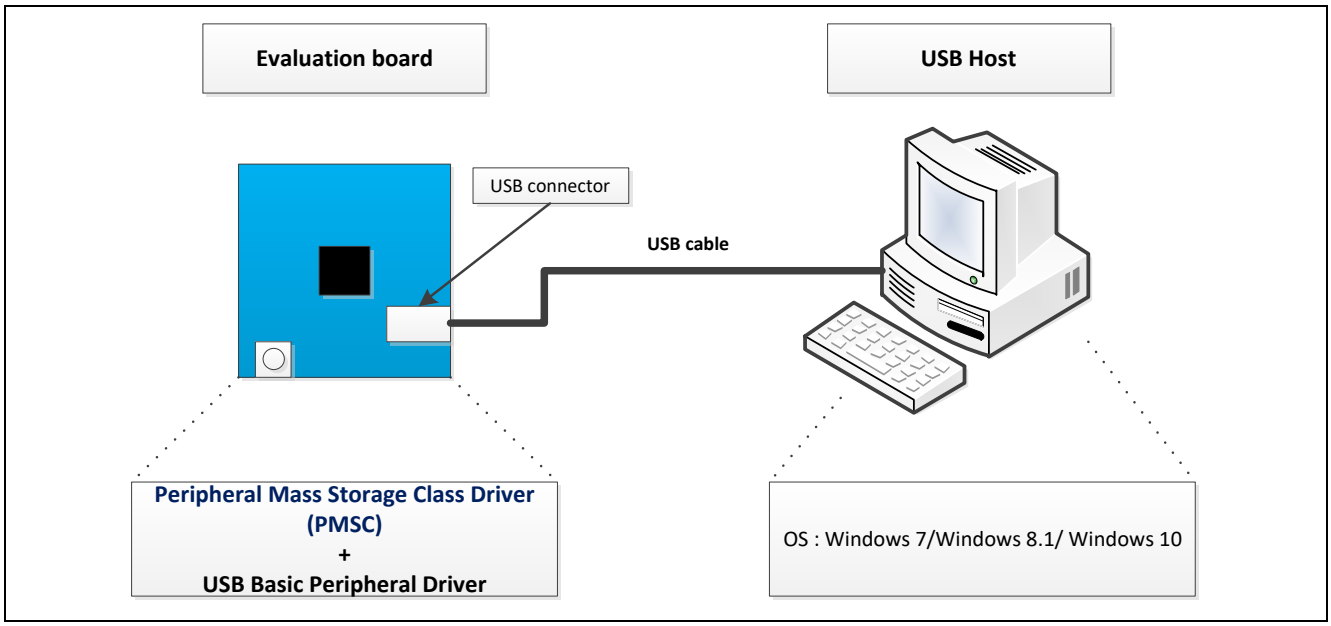


**Figure 2.1 Example Operating Environment**

## 3. File structure and directory

Figure 3.1 shows the software configuration of PMSC, and Table 3.1 shows the functional overview of each hierarchy.

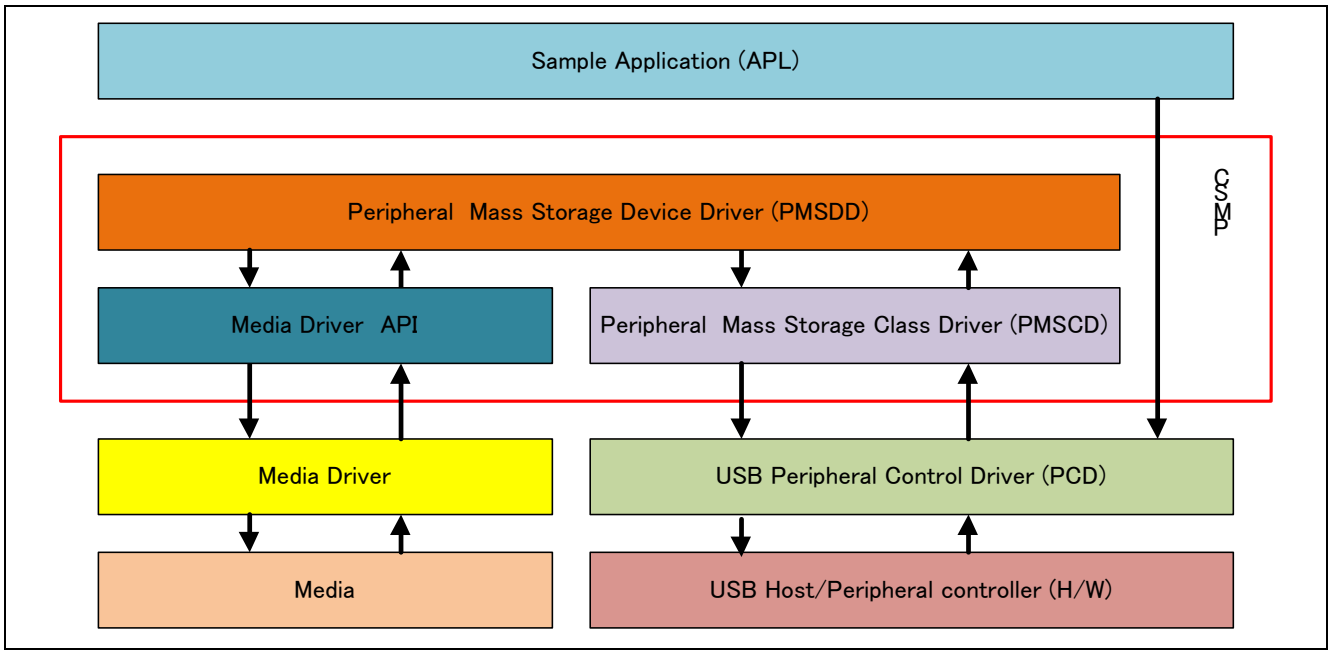


**Figure 3.1 Software Configuration**

**Table 3.1 Functional Overview of Each Hierarchy**

| Module | Description |
|---|---|
| APL | USB initial setting. |
| PMSDD | Processes storage commands from the PMSCD<br>Accesses media via the media driver. |
| PMSCD | Controls BOT protocol data and responds to class requests.<br>Analyzes CBWs and transmits/receives data.<br>Generates CSWs together with the PMSDD/PCD. |
| Media Driver API | Interface function between PMSDD and Media Driver. |
| Media Driver | Media control. |
| PCD | Interrupt processing.<br>Request analysis from USB Host.<br>Device state management.<br>Hardware control. |
| H/W | Hardware |

## 4. Compile Setting

When operating PMSC, it is necessary to set the USB-BASIC-F/W as a peripheral.

Refer to USB Basic Firmware application note (Document No. R01AN3291JEJ) for information on USB-BASIC-F/W settings.

In addition, the configuration option setting of PMSC is done in r_usb_pmsc_config.h. Descriptions of option names and setting values are shown in the table below.

| Configuration options in r_usb_pmsc_config.h | | |
|---|---|---|
| Setting pipe to be used | | |
| | USB_CFG_PMSC_BULK_IN | Specifies the pipe number which is used at the data transfer (Bulk In). *1<br>Specifies any one from USB_PIPE1 to USB_PIPE5 |
| | USB_CFG_PMSC_BULK_OUT | Specifies the pipe number which is used at the data transfer (Bulk Out). *1<br>Specifies any one from USB_PIPE1 to USB_PIPE5 |
| Response data setting for Inquiry command | | |
| | USB_CFG_PMSC_VENDOR | Specify vendor information as response data for the Inquiry command. Be sure to specify this information as 8-byte data enclosed in double quotations.<br>(Ex)"Renesas " |
| | USB_CFG_PMSC_PRODUCT | Specify product information as response data for the Inquiry command. Be sure to specify this information as 16-byte data enclosed in double quotations.<br>(Ex)"Mass Storage " |
| | USB_CFG_PMSC_REVISION | Specify a product revision level as response data for the Inquiry command. Be sure to specify this information as 4-byte data enclosed in double quotations.<br>(Ex)"1.00" |
| Setting the number of data transfer sectors | | |
| | USB_CFG_PMSC_TRANS_COUNT | Specify the maximum sector size for a single data transfer to be requested of the PCD. This driver specifies to the PCD a value of "1 sector (512) × USB_CFG_PMSC_TRANS_COUNT" bytes as the transfer size. (1 to 127)<br>Increasing this value may reduce the number of data transfer requests to the PCD, which may improve the transfer speed performance, but note that "1 sector (512) × USB_CFG_PMSC_TRANS_COUNT" bytes of RAM is consumed. |

*1. Do not set the same pipe number for the definitions of USB_CFG_PMSC_BULK_IN and USB_CFG_PMSC_BULK_OUT.

# 5. Class Driver Overview

## 5.1 Class Request

Table 4 1 lists the class requests supported by this driver.

**Table 5.1 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. |

## 5.2 Storage Commands

The storage commands supported by this driver are shown in Table 5.2. In response to any commands other than those shown below, a STALL response or a FAIL wrapped in a CSW will be returned.

**Table 5.2 Storage Commands**

| Command | Code | Description |
|---|---|---|
| TEST_UNIT_READY | 0x00 | Checks the state of the peripheral device. |
| REQUEST_SENSE | 0x03 | Gets the state of the peripheral device. |
| 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. |

# 6. Device Class Driver (PMSCD)

## 6.1 Basic Functions

The functions of PMSCD are shown below.

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)

## 6.2 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.

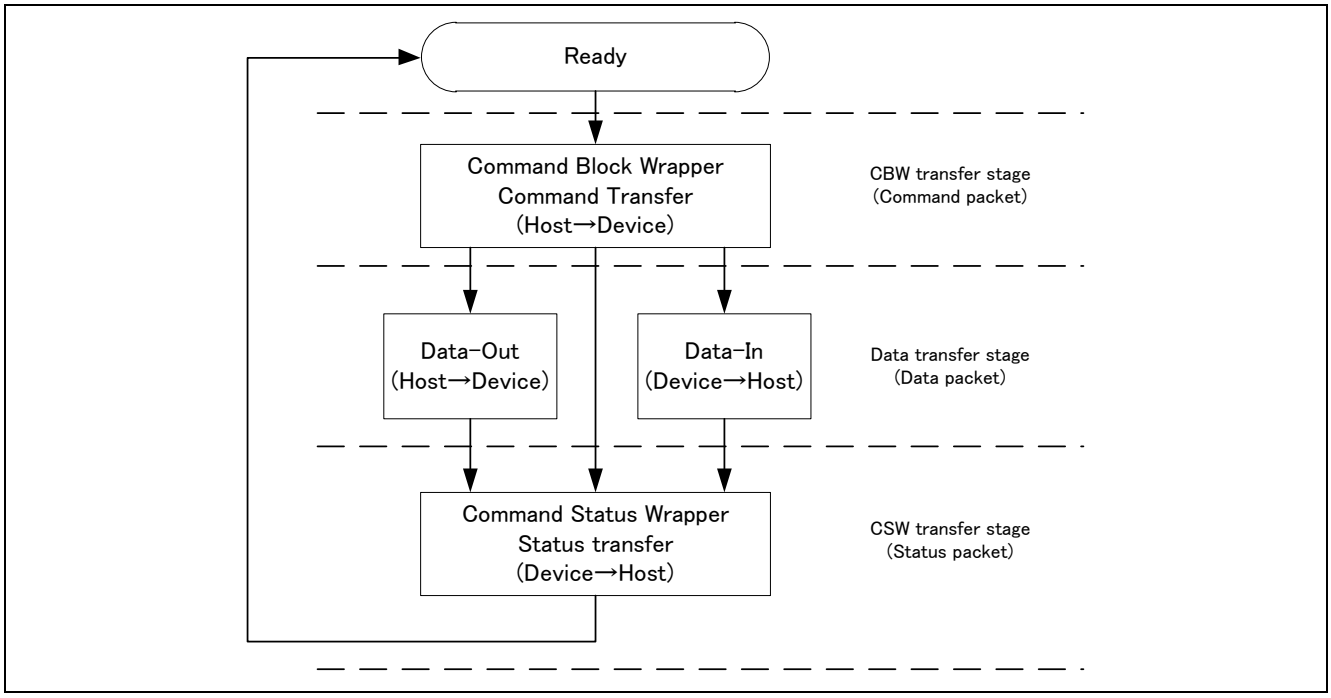


**Figure 6.1 BOT protocol Overview**

# 7. Media Driver Interface

With the PMSC, a common set of media driver API functions are used to make accesses to media drivers with different specifications simple.

## 7.1 Media Driver API Functions

The media driver API functions are called from the PMSC. The media driver API functions call media driver functions that have been implemented by the user. This section describes the prototypes of the media driver API functions, as well as the processing that is necessary to implement the respective functions.

Table 7.1 lists the media driver API functions.

**Table 7.1 Media Driver API Function**

| 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 | Performs processing for command and control instructions specific to the media device. |

### 7.1.1 R_USB_media_initialize

**Registers media driver functions with the media driver.**

**Format**

bool R_USB_media_initialize(media_driver_t *p_media_driver)

**Arguments**

| | |
|---|---|
| usb_media_ret_t | Pointer to a media driver structure area |

**Return Value**

| | |
|---|---|
| true | Successful |
| false | Error |

**Description**

This function registers media driver functions implemented by the user with the media driver. Be sure to call this API when performing initialization processing and the like in a user application program.

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. For information regarding how to register media driver functions that have been implemented by the user, see 7.3 Storage Media Driver Registration.
3. This API does not initiate register initialization of the device or operation of the device. These types of processing are performed by the R_USB_media_open() function.
4. Only one media device can be registered using this API. Multiple media device registration is not supported.

**Examples**

```
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 */
}
```

### 7.1.2 R_USB_media_open

**Places a media driver and device in an operable state.**

**Format**

usb_media_ret_t R_USB_media_open(void)

**Arguments**

--

**Return Value**

| | |
|---|---|
| USB_MEDIA_RET_OK | Successful |
| USB_MEDIA_RET_PARAERR | Parameter error |
| USB_MEDIA_RET_DEV_OPEN | The device is already open |
| USB_MEDIA_RET_NOTRDY | The device does not respond or does not exist |
| USB_MEDIA_RET_OP_FAIL | Other error |

**Description**

This function initializes the hardware registers of the peripheral circuits to be used by the media driver, and places the device in an operable state. Be sure to call this API when performing initialization processing and the like in a user application program.

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. When calling the R_USB_media_open function, the registration processing described in note 1 above must be performed in order to enable media driver functions to be called. For information regarding how to register media driver functions, see 7.3 Storage Media Driver Registration.
3. The R_USB_media_initialize function must be called before calling this function.
4. Unless the R_USB_media_close function is called, R_USB_media_open function can only be called once. After the R_USB_media_close function has been called, the device settings revert to their initial state, so this function can be called again.

**Examples**

```
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 */
}
```

### 7.1.3 R_USB_media_close

**Frees resources used by the media driver, and returns the hardware to the inactive state.**

**Format**

usb_media_ret_t R_USB_media_close(void)

**Arguments**

--

**Return Value**

| | |
|---|---|
| USB_MEDIA_RET_OK | Successful |
| USB_MEDIA_RET_PARAERR | Parameter error |
| USB_MEDIA_RET_OP_FAIL | Other error |

**Description**

Releases the resources for the media driver and restores the hardware to inactive state.

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. When calling the R_USB_media_close function, the registration processing described in note 1 above must be performed in order to enable media driver functions to be called. For information regarding how to register media driver functions, see 7.3 Storage Media Driver Registration.

**Examples**

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

### 7.1.4 R_USB_media_read

**Reads 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)

**Arguments**

| | |
|---|---|
| p_buf | Pointer to the area where the data read from the media device is stored |
| lba | Read-out starting logical block address |
| count | Read-out block count (sector count) |

**Return Value**

| | |
|---|---|
| USB_MEDIA_RET_OK | Successful |
| USB_MEDIA_RET_PARAERR | Parameter error |
| USB_MEDIA_RET_NOTRDY | The device is not in the ready state |
| USB_MEDIA_RET_OP_FAIL | Other error |

**Description**

This function reads data blocks from the media device. (It reads the number of blocks specified in the third argument (count), starting with the LBA (Logical Block Address) specified in the second argument.)
The data that has been read is stored in the area specified by the first argument (p_buf).

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. When calling the R_USB_media_read function, the registration processing described in note 1 above must be performed in order to enable media driver functions to be called. For information regarding how to register media driver functions, see 7.3 Storage Media Driver Registration.

**Examples**

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

### 7.1.5 R_USB_media_write

**Writes 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 which stores the data to be written to the media driver |
| lba | Write-out starting logical block address |
| count | Write-out block count (sector count) |

**Return Value**

| | |
|---|---|
| USB_MEDIA_RET_OK | Successful |
| USB_MEDIA_RET_PARAERR | Parameter error |
| USB_MEDIA_RET_NOTRDY | The device is not in the ready state |
| USB_MEDIA_RET_OP_FAIL | Other error |

**Description**

This function writes data blocks to the media device. (It writes the number of blocks specified in the third argument (count), starting with the LBA (Logical Block Address) specified in the second argument.)
Place the write data in the area indicated by the first argument (p_buf).

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. When calling the R_USB_media_write function, the registration processing described in note 1 above must be performed in order to enable media driver functions to be called. For information regarding how to register media driver functions, see 7.3 Storage Media Driver Registration.

**Examples**

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

### 7.1.6 R_USB_media_ioctl

**Retrieves information about the media driver.**

**Format**

usb_media_ret_t R_USB_media_ioctl(ioctl_cmd_t command, void *p_data)

**Arguments**

| | |
|---|---|
| command | Command code for the media driver |
| p_data | Pointer to the area where media information is stored |

**Return Value**

| | |
|---|---|
| USB_MEDIA_RET_OK | Successful |
| USB_MEDIA_RET_PARAERR | Parameter error |
| USB_MEDIA_RET_NOTRDY | The device is not in the ready state |
| USB_MEDIA_RET_OP_FAIL | Other error |

**Description**

This function writes data blocks to the media device. (It writes the number of blocks specified in the third argument (count), starting with the LBA (Logical Block Address) specified in the second argument.)
Place the write data in the area indicated by the first argument (p_buf).

**Note**

1. The user is responsible for implementing media driver functions that conform to the information shown above in “Arguments”, “Return Value”, “Description”, etc.
2. When calling the R_USB_media_ioctl function, the registration processing described in note 1 above must be performed in order to enable media driver functions to be called. For information regarding how to register media driver functions, see 7.3 Storage Media Driver Registration.
3. The command codes indicated by the first argument (command) must be defined by the user.

**Examples**

```
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;
```

## 7.2 Structure and Enumeration Definition

This section describes the structures and enumerations that are used by the media driver API functions.

These are defined in the file r_usb_media_driver_if.h.

### 7.2.1 usb_media_driver_t (Structure)

usb_media_driver_t is a structure that holds the logical unit number of the media device and points to the functions that must be implemented by the media driver. The usb_media_driver_t structure is shown below.

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

### 7.2.2 usb_media_ret_t (Enumeration)

In usb_media_ret_t, the return values returned by the media driver API are defined.

The usb_media_ret_t enumeration is shown below.

```
typedef enum
{
    USB_MEDIA_RET_OK = 0,           /* Successful */
    USB_MEDIA_RET_NOTRDY,           /* The device is not in the ready state */
    USB_MEDIA_RET_PARERR,           /* Parameter error */
    USB_MEDIA_RET_OP_FAIL,          /* Other error */
    USB_MEDIA_RET_DEV_OPEN,         /* The device is already open */
    USB_MEDIA_RET_DEV_NO_INIT,      /* The device is not initialized */
} usb_media_ret_t
```

### 7.2.3 ioctl_cmd_t (Enumeration)

In ioctl_cmd_t, the command codes specified for the R_USB_media_ioctl function are defined.

The ioctl_cmd_t enumeration is shown below.

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

[Note]

1. To add user-specific command codes when implementing a media driver, add these to the ioctl_cmd_t enumeration described above.

## 7.3 Storage Media Driver Registration

To switch the storage media of the PMSC from EEPROM to another storage media such as flash memory, the user must implement media driver functions to read from and write to that storage media, and register these functions with the media driver API.

This registration procedure is shown below, with serial SPI flash memory media driver functions used as an example.

1. **Media driver functions to register**
   Assume that the user has implemented the following functions as media driver functions for serial SPI flash memory.

| | | |
|---|---|---|
| 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. **Registration to the media driver function media API**
   (1). Define the media_driver_s structure for serial SPI flash memory.
   In the respective 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 an application program, perform initialization by specifying the media_driver_s structure described above as an argument to the R_USB_media_initialize function (API).
   == Application program ==

```
R_USB_media_initialize(&g_spi_flash_mediadriver);
```

By performing the steps described above, the serial SPI flash memory functions that the media driver will call are registered.

## 7.4 Storage Media Driver Implementation

You are responsible for implementing the media driver functions to support the storage media to be used. The implemented media driver functions are called by the PMSC via the API described in 7.1 Media Driver API Functions.

[Note]

1. For information regarding the processing that is necessary to implement the media driver functions, see the respective API processing details in 7.1 Media Driver API Functions.

The five media driver functions to create are open, close, read, write, and control. The function prototypes for these are shown below.

```
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 */
```

# 8. Sample application (APL)

## 8.1 Application Specification

The PMSC sample application (APL) runs on the GENMAI board. When the GENMAI board 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.

For an operating environment example, see 2. Operating environment.

[Note]

1. USB communication can be performed with PCs that support Windows 7, Windows 8.1, and Windows 10.

### 8.1.1 Media area of removable disk

In APL, the SDRAM area on GENMAI is used as the media area of the removable disk. We format the media area with FAT32 and check the operation.

## 8.2 Configuration File for the application program (r_usb_pmsc_apl_config.h)

The operation setting of the application is set in r_usb_pmsc_apl_config.h. Below are the setting items to be done in r_usb_pmsc_apl_config.h.

1. USE_USBIP

   The USB module to use is selected. Specify either USB_IP0 or USB_IP1.
   Please set this setting to the setting of USB_CFG_USE_USBIP done in r_usb_basic_config.h.

```
#define    USE_USBIP    USE_IP0    // Specify USB_IP0
#define    USE_USBIP    USE_IP1    // Specify USB_IP1
```

2. USB_SPEED

   The operating speed of the PMSC is selected. Specify either USB_HS or USB_FS.

```
#define    USB_SPEED    USB_HS     // When operating with High speed
#define    USB_SPEED    USB_FS     // When operating with Full speed
```

3. Note

   R_usb_pmsc_apl_config.h is a configuration setting for application programs. In addition to the above settings, configuration settings of the USB driver are required. For the USB driver configuration settings, refer to "4. Compile Setting".

## 8.3 Application Processing

The APL consists of two parts: processing of initial settings and the main loop.

| | | |
|---|---|---|
| Initial settings | : | Initializes the USB controller, and initializes the USB driver. |
| Main loop | : | Call the R_USB_GetEvent function in the main loop to activate the USB driver |

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 “5.2 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:

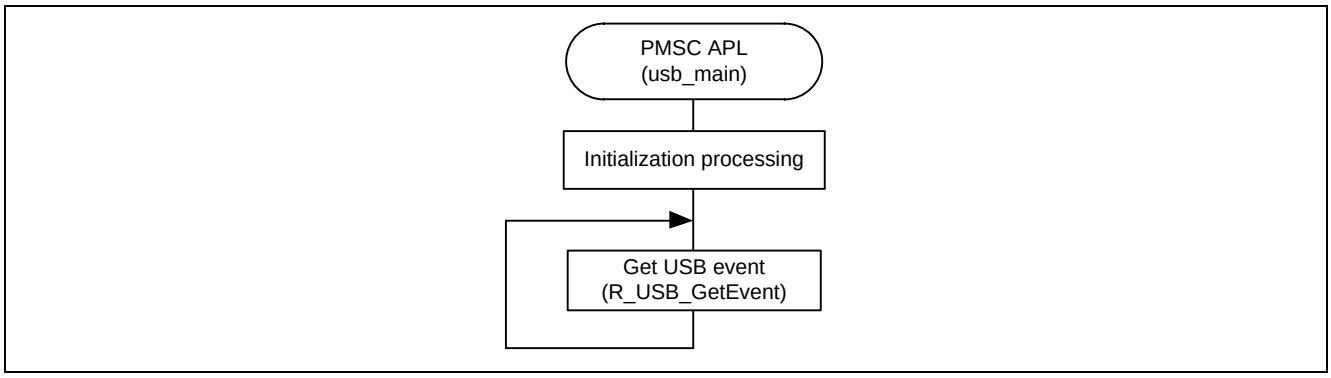


**Figure 8.1 APL Processing Overview**

## 8.4 Descriptor

The PMSC’s descriptor information is contained in r_usb_pmsc_descriptor.c.

[Note]

1. Please be sure to use your customer's Vender ID and Product ID.

The descriptor list of PMSC is shown in Table 8.1.

**Table 8.1 Descriptor list**

| Descriptor | variable |
|---|---|
| Device Descriptor | g_msc_device |
| Device Qualifier Descriptor | g_msc_qualifier_descriptor |
| Configuration Descriptor for FS | g_msc_fs_configuration *1 |
| Configuration Descriptor for FS | g_msc_fs_configuration *1 |
| Configuration Descriptor for FS | g_msc_fs_configuration *1 |
| Configuration Descriptor for HS | g_msc_hs_configuration *2 |
| Configuration Descriptor for HS | g_msc_hs_configuration *2 |
| Configuration Descriptor for HS | g_msc_hs_configuration *2 |
| String Descriptor | g_msc_string0 |
| | g_msc_string1 |
| | g_msc_string2 |
| | g_msc_string3 |
| | g_msc_string4 |
| | g_msc_string5 |
| | g_msc_string6 |

*1. g_msc_fs_configuration includes Configuration Descriptor for FS, Interface Descriptor for FS, Endpoint Descriptor for FS.

*2. G_msc_hs_configuration includes Configuration Descriptor for HS, Interface Descriptor for HS, Endpoint Descriptor for HS.

### 8.4.1 g_msc_device

Table 8.2 shows the Device Descriptor setting values.

**Table 8.2 Device Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_DD_BLENGTH | |
| 1 | bDescriptorType | USB_DT_DEVICE | |
| 2 | bcdUSB | USB_BCDNUM | Low : 1Byte |
| 3 | | USB_BCDNUM | High : 1Byte |
| 4 | bDeviceClass | 0x00 | |
| 5 | bDeviceSubClass | 0x00 | |
| 6 | bDeviceProtocol | 0x00 | |
| 7 | bMAXPacketSize0 | USB_DCPMAXP | |
| 8 | idVendor | USB_VENDORID | Low : 1Byte |
| 9 | | USB_VENDORID | High : 1Byte |
| 10 | idProduct | USB_PRODUCTID | Low : 1Byte |
| 11 | | USB_PRODUCTID | High : 1Byte |
| 12 | bcdDevice | USB_RELEASE | Low : 1Byte |
| 13 | | USB_RELEASE | High : 1Byte |
| 14 | iManufacturer | 0x01 | |
| 15 | iProduct | 0x02 | |
| 16 | iSerialNumber | 0x06 | |
| 17 | bNumConfigurations | USB_CONFIGNUM | |

### 8.4.2 g_msc_qualifier_descriptor

Table 8.3 shows the Device Qualifier Descriptor setting values.

**Table 8.3 Device Qualifier Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_QD_BLENGTH | |
| 1 | bDescriptorType | USB_DT_DEVICE_QUALIFIER | |
| 2 | bcdUSB | USB_BCDNUM | Low : 1Byte |
| 3 | | USB_BCDNUM | High : 1Byte |
| 4 | bDeviceClass | 0x00 | |
| 5 | bDeviceSubClass | 0x00 | |
| 6 | bDeviceProtocol | 0x00 | |
| 7 | bMAXPacketSize0 | USB_DCPMAXP | |
| 8 | bNumConfigurations | USB_CONFIGNUM | |
| 9 | bReserved | 0x00 | |

### 8.4.3 g_msc_fs_configuration

g_msc_fs_configuration includes Configuration Descriptor for FS, Interface Descriptor for FS, Endpoint Descriptor for FS. Table 8.4 shows the description of each descriptor.

**Table 8.4 g_msc_fs_configuration**

| offset | Descriptor |
|---|---|
| 0 | Configuration Descriptor |
| 9 | Interface Descriptor |
| 18 | Endpoint Descriptor |

**Table 8.5 Configuration Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_CD_BLENGTH | |
| 1 | bDescriptorType | USB_SOFT_CHANGE | |
| 2 | wTotalLength | USB_PMSC_CD_WTOTALLENGTH % 256 | Low : 1Byte |
| 3 | | USB_PMSC_CD_WTOTALLENGTH / 256 | High : 1Byte |
| 4 | bNumInterfaces | 0x01 | |
| 5 | bConfigurationValue | 0x01 | |
| 6 | iConfiguration | 0x04 | g_msc_string4[] |
| 7 | bmAttributes | USB_CF_RESERVED \| USB_CF_SELFP | Self-powered |
| 8 | bMaxPower | 10 / 2 | 10mA |

**Table 8.6 Interface Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ID_BLENGTH | |
| 1 | bDescriptorType | USB_DT_INTERFACE | |
| 2 | bInterfaceNumber | 0x00 | |
| 3 | bAlternateSetting | 0x00 | |
| 4 | bNumEndpoints | 0x02 | |
| 5 | bInterfaceClass | USB_IFCLS_MAS | MSC |
| 6 | bInterfaceSubClass | USB_INTERFACE_SUBCLASS | |
| 7 | bInterfaceProtocol | USB_BOTP | |
| 8 | iInterface | 0x03 | g_msc_string3[] |

**Table 8.7 Endpoint Descriptor 0**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ED_BLENGTH | |
| 1 | bDescriptorType | USB_DT_ENDPOINT | |
| 2 | bEndpointAddress | USB_EP_IN \| USB_EP1 | b'7 : Direction<br>b'6 – b'4: Reserved<br>b'3 – b'0: Endpoint number |
| 3 | bmAttribute | USB_EP_BULK | b'5 – b'4: Usage type<br>b'3 – b'2: Synchronization Type<br>b'1 – b'0: Transfer type |
| 4 | wMaxPacketSize | 64 | |
| 5 | | 0 | |
| 6 | bInterval | 0x00 | |

**Table 8.8 Endpoint Descriptor 1**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ED_BLENGTH | |
| 1 | bDescriptorType | USB_DT_ENDPOINT | |
| 2 | bEndpointAddress | USB_EP_OUT \| USB_EP2 | b'7 : Direction<br>b'6 – b'4: Reserved<br>b'3 – b'0: Endpoint number |
| 3 | bmAttribute | USB_EP_BULK | b'5 – b'4: Usage type<br>b'3 – b'2: Synchronization Type<br>b'1 – b'0: Transfer type |
| 4 | wMaxPacketSize | 64 | |
| 5 | | 0 | |
| 6 | bInterval | 0x00 | |

### 8.4.4 g_msc_hs_configuration

g_msc_hs_configuration includes Configuration Descriptor for HS, Interface Descriptor for HS, Endpoint Descriptor for HS. Table 8.9 shows the description of each descriptor.

**Table 8.9 g_msc_hs_configuration**

| offset | Descriptor |
|---|---|
| 0 | Configuration Descriptor |
| 9 | Interface Descriptor |
| 18 | Endpoint Descriptor |

**Table 8.10 Configuration Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_CD_BLENGTH | |
| 1 | bDescriptorType | USB_SOFT_CHANGE | |
| 2 | wTotalLength | USB_PMSC_CD_WTOTALLENGTH % 256 | Low : 1Byte |
| 3 | | USB_PMSC_CD_WTOTALLENGTH / 256 | High : 1Byte |
| 4 | bNumInterfaces | 0x01 | |
| 5 | bConfigurationValue | 0x01 | |
| 6 | iConfiguration | 0x05 | g_msc_string5[] |
| 7 | bmAttributes | USB_CF_RESERVED \| USB_CF_SELFP | Self-powered |
| 8 | bMaxPower | 10 / 2 | 10mA |

**Table 8.11 Interface Descriptor**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ID_BLENGTH | |
| 1 | bDescriptorType | USB_DT_INTERFACE | |
| 2 | bInterfaceNumber | 0x00 | |
| 3 | bAlternateSetting | 0x00 | |
| 4 | bNumEndpoints | 0x02 | |
| 5 | bInterfaceClass | USB_IFCLS_MAS | MSC |
| 6 | bInterfaceSubClass | USB_INTERFACE_SUBCLASS | |
| 7 | bInterfaceProtocol | USB_BOTP | |
| 8 | iInterface | 0x03 | g_msc_string3[] |

**Table 8.12 Endpoint Descriptor 0**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ED_BLENGTH | |
| 1 | bDescriptorType | USB_DT_ENDPOINT | |
| 2 | bEndpointAddress | USB_EP_IN \| USB_EP1 | b'7 : Direction<br>b'6 – b'4: Reserved<br>b'3 – b'0: Endpoint number |
| 3 | bmAttribute | USB_EP_BULK | b'5 – b'4: Usage type<br>b'3 – b'2: Synchronization Type<br>b'1 – b'0: Transfer type |
| 4 | wMaxPacketSize | 0 | |
| 5 | | 2 | 512byte |
| 6 | bInterval | 0x00 | |

**Table 8.13 Endpoint Descriptor 1**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_ED_BLENGTH | |
| 1 | bDescriptorType | USB_DT_ENDPOINT | |
| 2 | bEndpointAddress | USB_EP_OUT \| USB_EP2 | b'7 : Direction<br>b'6 – b'4: Reserved<br>b'3 – b'0: Endpoint number |
| 3 | bmAttribute | USB_EP_BULK | b'5 – b'4: Usage type<br>b'3 – b'2: Synchronization Type<br>b'1 – b'0: Transfer type |
| 4 | wMaxPacketSize | 0 | |
| 5 | | 2 | 512byte |
| 6 | bInterval | 0x00 | |

### 8.4.5 g_msc_stringX (X = 0 - 6)

The setting values of String Descriptor are shown in Table 6.10 to Table 6.14.

**Table 8.14 g_msc_string0**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | wLANGID | 0x09 | Low : 1Byte |
| 3 | | 0x04 | High : 1Byte |

**Table 8.15 g_msc_string1**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | 'R' | |
| 3 | | 0x00 | |
| 4 | | 'E' | |
| 5 | | 0x00 | |
| 6 | | 'N' | |
| 7 | | 0x00 | |
| 8 | | 'E' | |
| 9 | | 0x00 | |
| 10 | | 'S' | |
| 11 | | 0x00 | |
| 12 | | 'A' | |
| 13 | | 0x00 | |
| 14 | | 'S' | |
| 15 | | 0x00 | |

**Table 8.16 g_msc_string2**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | 'U' | |
| 3 | | 0x00 | |
| 4 | | 'S' | |
| 5 | | 0x00 | |
| 6 | | 'B' | |
| 7 | | 0x00 | |
| 8 | | ' ' | |
| 9 | | 0x00 | |
| 10 | | 'M' | |
| 11 | | 0x00 | |
| 12 | | 'a' | |
| 13 | | 0x00 | |
| 14 | | 's' | |
| 15 | | 0x00 | |
| 16 | | 's' | |
| 17 | | 0x00 | |
| 18 | | ' ' | |
| 19 | | 0x00 | |
| 20 | | 'S' | |
| 21 | | 0x00 | |
| 22 | | 't' | |
| 23 | | 0x00 | |
| 24 | | 'o' | |
| 25 | | 0x00 | |
| 26 | | 'r' | |
| 27 | | 0x00 | |
| 28 | | 'a' | |
| 29 | | 0x00 | |
| 30 | | 'g' | |
| 31 | | 0x00 | |
| 32 | | 'e' | |
| 33 | | 0x00 | |

**Table 8.17 g_msc_string3**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | 'R' | |
| 3 | | 0x00 | |
| 4 | | 'e' | |
| 5 | | 0x00 | |
| 6 | | 'm’ | |
| 7 | | 0x00 | |
| 8 | | 'o' | |
| 9 | | 0x00 | |
| 10 | | 'v' | |
| 11 | | 0x00 | |
| 12 | | 'a' | |
| 13 | | 0x00 | |
| 14 | | 'b' | |
| 15 | | 0x00 | |
| 16 | | 'l' | |
| 17 | | 0x00 | |
| 18 | | 'e' | |
| 19 | | 0x00 | |
| 20 | | ' ' | |
| 21 | | 0x00 | |
| 22 | | 'D' | |
| 23 | | 0x00 | |
| 24 | | 'r' | |
| 25 | | 0x00 | |
| 26 | | 'i' | |
| 27 | | 0x00 | |
| 28 | | 'v' | |
| 29 | | 0x00 | |
| 30 | | 'e' | |
| 31 | | 0x00 | |

**Table 8.18 g_msc_string4**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | 'F' | |
| 3 | | 0x00 | |
| 4 | | 'u' | |
| 5 | | 0x00 | |
| 6 | | 'l' | |
| 7 | | 0x00 | |
| 8 | | 'l' | |
| 9 | | 0x00 | |
| 10 | | '-' | |
| 11 | | 0x00 | |
| 12 | | 'S' | |
| 13 | | 0x00 | |
| 14 | | 'p' | |
| 15 | | 0x00 | |
| 16 | | 'e' | |
| 17 | | 0x00 | |
| 18 | | 'e' | |
| 19 | | 0x00 | |
| 20 | | 'd' | |
| 21 | | 0x00 | |

**Table 8.19 g_msc_string5**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | 'H' | |
| 3 | | 0x00 | |
| 4 | | 'I' | |
| 5 | | 0x00 | |
| 6 | | '-' | |
| 7 | | 0x00 | |
| 8 | | 'S' | |
| 9 | | 0x00 | |
| 10 | | 'p' | |
| 11 | | 0x00 | |
| 12 | | 'e' | |
| 13 | | 0x00 | |
| 14 | | 'e' | |
| 15 | | 0x00 | |
| 16 | | 'd' | |
| 17 | | 0x00 | |

**Table 8.20 g_msc_string6**

| Offset | Field | Value | Remarks |
|---|---|---|---|
| 0 | bLength | USB_PMSC_SD0_BLENGTH | |
| 1 | bDescriptorType | USB_DT_STRING | |
| 2 | bString | '0' | |
| 3 | | 0x00 | |
| 4 | | '0' | |
| 5 | | 0x00 | |
| 6 | | '0' | |
| 7 | | 0x00 | |
| 8 | | '0' | |
| 9 | | 0x00 | |
| 10 | | '0' | |
| 11 | | 0x00 | |
| 12 | | '0' | |
| 13 | | 0x00 | |
| 14 | | '0' | |
| 15 | | 0x00 | |
| 16 | | '0' | |
| 17 | | 0x00 | |
| 18 | | '0' | |
| 19 | | 0x00 | |
| 20 | | '0' | |
| 21 | | 0x00 | |
| 22 | | '0' | |
| 23 | | 0x00 | |
| 24 | | '1' | |
| 25 | | 0x00 | |

## Website and Support

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

Inquiries
http://www.renesas.com/contact/

## Revision History <revision history,rh>

| | | Description | |
|---|---|---|---|
| **Rev.** | **Date** | **Page** | **Summary** |
| 1.20 | Aug. 31. 8 | - | First edition issued |

## 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 disputes 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, drawing, chart, program, algorithm, 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 otherwise misappropriate 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, copy or otherwise misappropriation of Renesas Electronics products.
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; and 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.

   Renesas Electronics products are neither intended nor 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 damages (space and 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 third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the 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 radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond 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. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of 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 by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate 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. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use 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 not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
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 majority-owned subsidiaries.

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

(Rev.3.0-1 November 2016)

RENESAS

## SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

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


**Renesas Electronics America Inc.**
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

**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: +44-1628-585-100, Fax: +44-1628-585-900

**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, P. R. China 200333
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 II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

**Renesas Electronics Korea Co., Ltd.**
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141




Colophon 6.0