RX Family

R01AN2026EJ0127
Rev.1.27
Jul 31, 2019

USB Host Mass Storage Class Driver (HMSC) using Firmware Integration Technology

Introduction

This application note describes USB Host Mass Storage Class Driver (HMSC), which utilizes Firmware Integration Technology (FIT). This module operates in combination with the USB Basic Host and Peripheral Driver (USB-BASIC-FW FIT module). It is referred to below as the USB HMSC FIT module.

Target Device

RX65N/RX651 Group
RX64M Group
RX71M Group
RX66T Group
RX72T Group
RX72M Group

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

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
   http://www.usb.org/developers/docs/
4. RX64M Group User’s Manual: Hardware (Document number: R01UH0377)
5. RX71M Group User’s Manual: Hardware (Document number: R01UH0493)
6. RX65N/RX651 Group User’s Manual: Hardware (Document number: R01UH0590)
7. RX65N/RX651-2M Group User’s Manual: Hardware (Document number: R01UH0659)
8. RX66T User's Manual: Hardware (Document number. R01UH0749)
9. RX72T User's Manual: Hardware (Document number. R01UH0803)
10. RX72M User's Manual: Hardware (Document number. R01UH0804)
11. RX Family M3S-TFAT-Tiny: FAT file system software (Document number: R20AN0038)
12. RX Family M3S-TFAT-Tiny: Memory Driver Interface Module (Document number: R20AN0335)
13. USB Basic Host and Peripheral Driver using Firmware Integration Technology Application Note (Document number. R01AN2025）

Renesas Electronics Website

http://www.renesas.com/

USB Device Page

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

Contents

1. Overview

The USB HMSC FIT module, when used in combination with the USB-BASIC-FW FIT module, operates as a USB host mass storage class driver (HMSC).
The HMSC comprises a USB mass storage class bulk-only transport (BOT) protocol. When combined with a file system and storage device driver, it enables communication with a BOT-compatible USB storage device.
Note that please use the M3S-TFAT-Tiny (Document number: R20AN0038) and Memory driver interface module (Document numver: R20AN0335) in combination when using this driver.

This module supports the following functions.

1. Checking of connected USB storage devices (to determine whether or not operation is supported).
2. Storage command communication using the BOT protocol.
3. Support for SFF-8070i (ATAPI) USB mass storage subclass.
4. Sharing of a single pipe for IN/OUT directions or multiple devices.
5. Maximum 4 USB storage devices can be connected.

1.1 Please be sure to read

Please refer to the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver 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 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. This driver is confirmed for operation in combination with the following FAT.

   RX Family Open Source FAT File System [M3S-TFAT-Tiny] Module
   Firmware Integration Technology Rev.3.03

1.3 Limitation

1. Some MSC devices may be unable to be connected (because they are not recognized as storage devices).
2. MSC devices that return values of 1 or higher in response to the GetMaxLun command (mass storage class command) are not supported.
3. Maximum 4 USB storage devices can be connected.
4. USB storage devices with a sector size of 512 bytes can be connected.
5. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.

1.4 Terms and Abbreviations

APL	:	Application program
BOT	:	Mass Storage Class Bulk Only Transport
FSL	:	FAT File System Library
HCD	:	Host Control Driver for USB-BASIC-FW
HDCD	:	Host Device Class Driver (Device driver and USB class driver)
MGR	:	Peripheral Device State Manager for HCD
MSC	:	Mass Storage Class
Non-OS	:	USB Driver for OS-less
RSK	:	Renesas Starter Kits
RTOS	:	USB Driver for the real-time OS
TFAT	:	Tiny FAT file system software for microcontrollers (M3S-TFAT-Tiny-RX)
USB-BASIC-FW	:	USB Basic Host and Peripheral Driver

1.5 USB HMSC FIT Module

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

2. Software Configuration

HDCD (Host Device Class Driver) is the all-inclusive term for HMSDD (Host Mass Storage Device Driver) and HMSCD (USB Host Mass Storage Class Driver).

Figure 2-1 shows the HMSC software block diagram, with HDCD as the centerpiece. Table 2-1 describes each module.

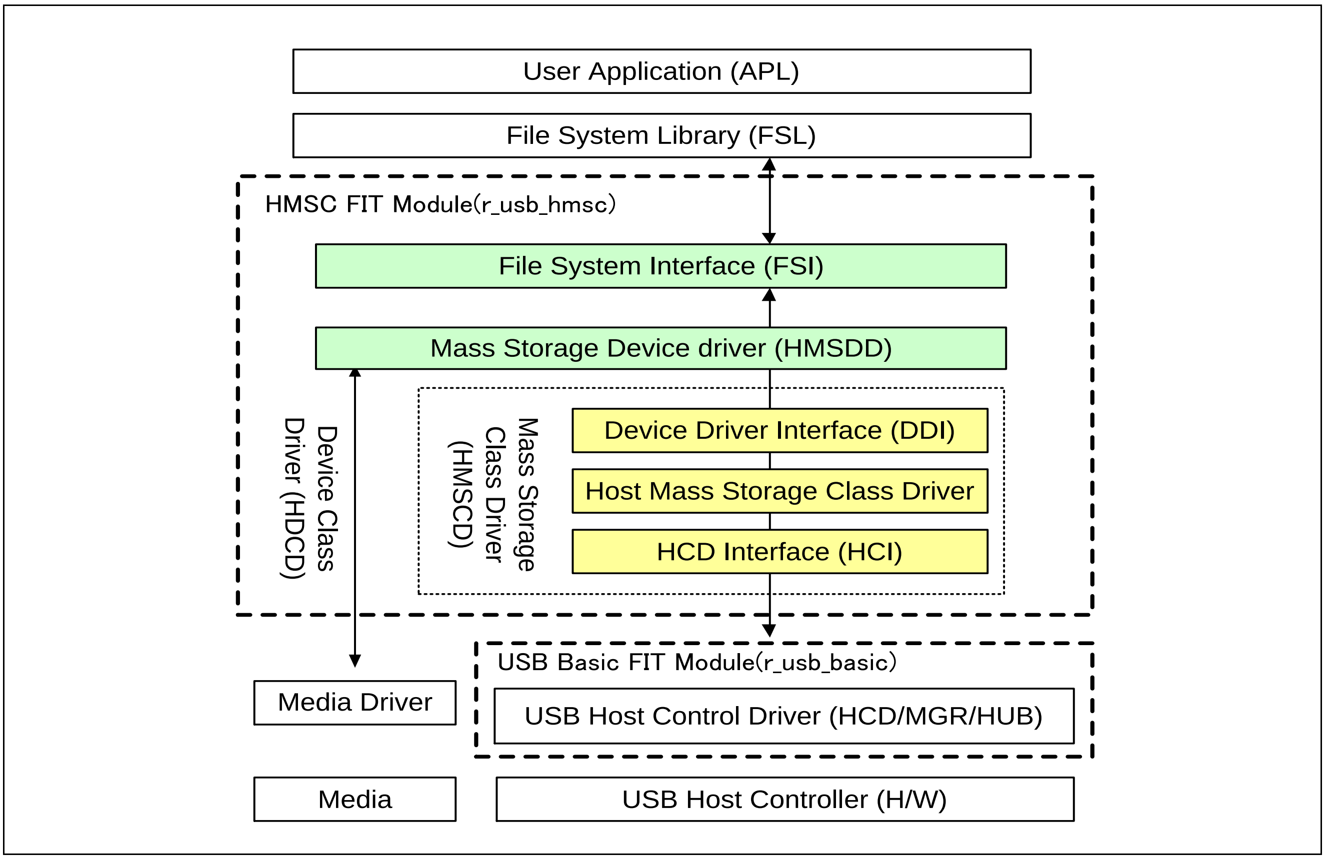

Figure 2-1 Software Module Structure

Table 2-1 Module

Module	Description
FSI	FSL-HMSDD interface functions. They should be modified to match FSL.
HMSDD	To be created (modified) by the customer to match the storage media.
DDI	HMSDD-HMSCD interface functions. They should be modified to match the storage media interface of HMSDD.
HMSCD	The USB host mass storage class driver. It appends BOT protocol information to storage commands and sends requests to HCD. It also manages the BOT sequence. The storage commands should be added (modified) by the customer to match the system specifications. SFF-8070i (ATAPI) is supported in the example code.
HCI	HMSCD-HCD interface functions.
MGR/HUB	Enumerates the connected devices and starts HMSCD. Also performs device state management.
HCD	USB host hardware 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

3.3 Operating Confirmation Environment

Table 3-1 shows the operating confirmation environment of this driver.

Table 3-1 Operating Confirmation Environment

Item	Contents
C compiler	Renesas Electronics C/C++ compiler for RX Family V.3.01.00 (The option "-lang=C99" is added to the default setting of IDE)
	GCC for Renesas RX 4.08.04.201902 (The option "-std=gnu99" is added to the default setting of IDE)
	IAR C/C++ Compiler for Renesas RX version 4.10.01
Real-Time OS	FreeRTOS V.10.0.0
Endian	Little Endian, Big Endian
USB Driver Revision Number	Rev.1.27
Using Board	Renesas Starter Kits for RX64M Renesas Starter Kits for RX71M Renesas Starter Kits for RX65N, Renesas Starter Kits for RX65N-2MB Renesas Starter Kits for RX72T Renesas Starter Kits for RX72M

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
RX64M RX71M	USBI0 Interrupt (Vector number: 189, Interrupt source number : 62, Software Configurable Interrupt B) USB D0FIFO0 Interrupt (Vector number: 34) / USB D1FIFO0 Interrupt (Vector number: 35) USBR0 Interrupt (Vector number:90)
	USBAR Interrupt (Vector number: 94) USB D0FIFO2 Interrupt (Vector number: 32) / USB D1FIFO2 Interrupt (Vector number: 33)
RX65N RX651 RX72M	USBI0 Interrupt (Vector number: 185, Interrupt source number : 62, Software Configurable Interrupt B) USB D0FIFO0 Interrupt (Vector number: 34) / USB D1FIFO0 Interrupt (Vector number: 35) USBR0 Interrupt (Vector number:90)
RX66T RX72T	USBI0 Interrupt (Vector number: 174) / USBR0 Interrupt (Vector number: 90) USB D0FIFO0 Interrupt (Vector number: 34) / USB D1FIFO0 Interrupt (Vector number: 35)

3.5 Header Files

All API calls and their supporting interface definitions are located in r_usb_basic_if.h and r_usb_hmsc_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 "Configuration" in the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver 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	42.9K bytes (Note 4)	42.4K bytes (Note 5)
RAM size	23.6K bytes	23.6K bytes

(2). RTOS

	Checks arguments	Does not check arguments
ROM size	52.6K bytes (Note 4)	52.1K bytes (Note 5)
RAM size	48.7K bytes	48.7K bytes

2. GCC (Optimization Level: -O2)

	Checks arguments	Does not check arguments
ROM size	49.7K bytes (Note 4)	49.1K bytes (Note 5)
RAM size	23.5K bytes	23.5K bytes

3. IAR (Optimization Level: Medium)

	Checks arguments	Does not check arguments
ROM size	41.0K bytes (Note 4)	40.7K bytes (Note 5)
RAM size	22.1K bytes	22.1K bytes

[Note]

1. ROM/RAM size for BSP and USB Basic Driver is included in the above size.
2. ROM/RAM size for TFAT is not included in the above size.
3. The above is the size when specifying RX V2 core option.
4. The ROM size of “Checks arguments” is the value when *USB_CFG_ENABLE* is specified to *USB_CFG_PARAM_CHECKING* definition in *r_usb_basic_config.h* file.
5. The ROM size of “Does not check arguments” is the value when *USB_CFG_DISABLE* is specified to *USB_CFG_PARAM_CHECKING* definition in *r_usb_basic_config.h* file.
6. The result of RTOS includes the ROM/RAM size of the real-time OS.

3.9 Argument

For the structure used in the argument of API function, refer to chapter "**Structures**" in the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver 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 e^2 studio

By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio

By using the FIT Configurator in e^2 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 e^2 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. Target Peripheral List（TPL）

For the structure used in the argument of API function, refer to chapter " **How to Set the Target Peripheral List (TPL)**" in the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver using Firmware Integration Technology Application Note*.

5. Class Driver

5.1 Class Requet

This driver supports the following class request.

Table 5-1 Class Request

Request	Description
GetMaxLun	Gets the maximum number of units that are supported.
MassStrageReset	Cancels a protocol error.

5.2 Storage Command

This driver supports the following storage command.

1. TEST_UNIT_READY
2. REQUEST_SENSE
3. MODE_SELECT10
4. MODE_SENSE10
5. PREVENT_ALLOW
6. READ_FORMAT_CAPACITY
7. READ10
8. WRITE10

6. API Functions

The following are Host Mass Storage Class specific API functions

API	Desription
R_USB_HmscStrgCmd()	Issues a Mass Storage command.
R_USB_HmscGetDriveNo()	Obtains the drive number.
R_USB_HmscGetSem()	Gets a semaphore (Only RTOS)
R_USB_HmscRelSem()	Releases a semaphore (Only RTOS)

Note:

1. Uses the FAT (File Allocation Table) API to access storage media.
2. Refer to chapter "API" in the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver using Firmware Integration Technology Application Note*. when using other API.

6.1 R_USB_HmscStrgCmd

Issues a Mass Storage command

Format

usb_err_t R_USB_HmscStrgCmd(usb_ctrl_t *p_ctrl, uint8_t *p_buf, uint16_t command)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_buf	Pointer to data area
command	Mass storage command

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_NG	Other error

Description

1. Non-OS

The Mass Storage command assigned to the argument *(command)* is issued to the MSC device that is specifed by the members (*address* and *module*) in the argument (*p_ctrl*). An application program can check the completion of the Mass Storage command with the *USB_STS_MSC_CMD_COMPLETE* return value of the *R_USB_GetEvent* function.

If a Mass Storage command with response data is issued, after checking *USB_STS_MSC_CMD_COMPLETE* return value of the *R_USB_GetEvent* function, an application program can obtain the response data from the area indicated by the second argument (*p_buf*). Check the member (*size*) of the *usb_crtl_t* structure to get the size of the response data that was received.

2. RTOS

The Mass Storage command assigned to the argument *(command)* is issued to the MSC device that is specifed by the members (*address* and *module*) in the argument (*p_ctrl*). An application program can check whether the mass storage command complete by refering the argument (the member (*event*) of the *usb_ctrl_t* structure) in the callback function. This driver sets *USB_STS_MSC_CMD_COMPLETE* to the argument (the *event* member of the *usb_ctrl_t* structure) when completing a Mass Storage command.

If a Mass Storage command with response data is issued, after checking that *USB_STS_MSC_CMD_COMPLETE* is set to the argument (the member (*event*) of the *usb_ctrl_t* structure), an application program can obtain the response data from the area indicated by the second argument (*p_buf*). Check the member (*size*) of the *usb_crtl_t* structure to get the size of the response data that was received.

For both Non-OS and RTOS, assign the following to the argument (*command*).

Table 6-1 Mass Storage Command

MassStorage Command
USB_ATAPI_TEST_UNIT_READY
USB_ATAPI_REQUEST_SENSE
USB_ATAPI_INQUIRY
USB_ATAPI_MODE_SELECT10
USB_ATAPI_PREVENT_ALLOW
USB_ATAPI_READ_FORMAT_CAPACITY
USB_ATAPI_READ_CAPACITY
USB_ATAPI_MODE_SENSE10

Reentrant

1. Non-OS

This API is only reentrant for different USB module.

2. RTOS

This API is reentrant.

Note

1. Before calling this API, assign the module number to the member (*module)* and the device address to the member (*address)*. If something other than *USB_IP0* or *USB_IP1* is assigned to the member (*module*), then *USB_ERR_PARA* will be the return value.
2. If the MCU being used only supports one USB module, then do not assign *USB_IP1* to t the member (*module*). If *USB_IP1* is assigned, then *USB_ERR_PARA* will be the return value.
3. If *USB_NULL* is assigned to the argument (*p_ctrl*), then *USB_ERR_PARA* will be the return value.
4. Do not assign a pointer to the auto variable (stack) area to the arguments (*p_buf*).
5. Assign *USB_NULL* to the argument (*p_buf*) when issuing the mass storage command without the response data.
6. If a command other than the Mass Storage commands listed in Table 6-1 is assigned to the argument (*command*), then *USB_ERR_PARA* will be the return value.
7. When calling FAT API and this API after issuing the Mass storage command by this API, be sure to call these APIs after checking the return value (*USB_STS_CMD_COMPLETE*) of *R_USB_GetEvent* function.
8. Refer to chapter "**7. Return Value (USB_STS_MSC_CMD_COMPLETED) of a Mass Stoage Commnad**" about CSW.
9. The CSW information is set to the member (*status*) of the *usb_ctrl_t* structure. If the value of the member (*status*) is *USB_CSW_FAIL*, issue the "*Requeset Sense*" command to the MSC device using this API.
10. Set the page code (1 Byte) of the "*Mode Sense10*" command in the start address to the area indicated by the 2nd argument (*p_buf*).
11. Set the parameter data for the "*Mode Select10*" command to the area indicated by the 2nd argument (*p_buf*) based on the specification for USB Mass Storage Subclass (SFF-8070i etc).
12. This function can be called when the USB device is in the configured state. When the API is called in any other state, *USB_ERR_NG* is returned.

Example

1. **Non-OS**

```
void        usb_application( void )
{
        usb_ctrl_t ctrl;
        usb_err_t err;
                        :
        while (1)
        {
                switch (R_USB_GetEvent(&ctrl))
                {
                        :
                        case USB_STS_CONFIGURED:
                        :
                                g_buf[0] = 0x3F;      /* Page Code */
                                ctrl.module = USB_IP1;
                                ctrl.address = adr;
                                R_USB_HmscStrgCmd( &ctrl, &g_buf, USB_ATAPI_MODE_SENSE10 );
                        :
                        break;
                        case USB_STS_MSC_CMD_COMPLETE:
                                if( ctrl.status == USB_CSW_FAIL )
                                {
                                        R_USB_HmscStrgCmd(&ctrl, &g_buf, USB_ATAPI_REQUEST_SENSE);
                                }
                        :

                        break;
                        :
                }
        }
}
```

2. RTOS

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctr, usb_hdl_t hdl, uint8_t is_request)
{
    USB_APL_SND_MSG(USB_APL_MBX, (usb_msg_t *)p_ctrl);
}

void        usb_application_task( void )
{
    usb_ctrl_t      ctrl;
    usb_ctrl_t      *p_mess;
            :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                    :
            case USB_STS_CONFIGURED:
                    :
                g_buf[0] = 0x3F             /* Page Code */
                ctrl.module = USB_IP1;
                ctrl.address = adr;
                R_USB_HmscStrgCmd(&ctrl, &g_buf, USB_ATAPI_MODE_SENSE10);
                    :
            break;
            case USB_STS_MSC_CMD_COMPLETE:
                if (ctrl.status == USB_CSW_FAIL)
                {
                    R_USB_HmscStrgCmd(&ctrl, &g_buf, USB_ATAPI_REQUEST_SENSE);
                }
                    :
            break;
        }
    }
}
```

6.2 R_USB_HmscGetDriveNo

Obtains the drive number

Format

usb_err_t R_USB_HmscGetDriveNo(usb_ctrl_t *p_ctrl, uint8_t *p_drive)

Arguments

p_ctrl	Pointer to usb_ctrl_t structure area
p_drive	Pointer to the area to store the drive number

Return Value

USB_SUCCESS	Successfully completed
USB_ERR_PARA	Parameter error
USB_ERR_NG	Other error

Description

Based on the information assigned to the *usb_crtl_t* structure (the member *module* and *address*), obtains the related drive number. The drive number is stored in the area indicated by the argument (*p_drive*).

Reentrant

This API is reentrant.

Note

1. Before calling this API, assign the device address of the MSC device whose drive number is to be obtained, and the USB module number (*USB_IP0* or *USB_IP1*) connected to that MSC device, to the members (*address* and *module*) of the usb_crtl_t structure. If there is a problem with what is assigned to these members, then *USB_ERR_PARA* will be the return value.
2. If *USB_NULL* is assigned to the argument (*p_ctrl*), then USB_ERR_PARA will be the return value.
3. This function can be called when the USB device is in the configured state. When the API is called in any other state, *USB_ERR_NG* is returned.

Example

```
void usb_application( void )
{
    usb_ctrl_t ctrl;
    uint8_t    drive;
                    :
    while (1)
    {
        switch (R_USB_GetEvent(&ctrl))
        {
                    :
            case USB_STS_CONFIGURED:
                    :
                ctrl.module = USB_IP0;
                ctrl.address = adr;
                R_USB_HmscGetDriveNo( &ctrl, &drive );
                    :
            break;
                    :
        }
    }
}
```

6.3 R_USB_HmscGetSem

Gets a semaphore (Only RTOS)

Format

void R_USB_HmscGetSem(void)

Arguments

none

Return Value

none

Description

Gets a specific semaphore which is used in this driver.

Reentrant

This API is reentrant.

Note

1. Be sure to call this API before calling the FAT file open function (e.g *R_tfat_f_open*).
2. If this API is called when a semapohre counter value is zero, the user task which calls this API shift to a semaphore waiting status.
3. The creation processing of a semaphore which this API uses is performed in USB driver.

Example

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctr, usb_hdl_t hdl, uint8_t is_request)
{
     USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}
void          usb_application_task( void )
{
    usb_ctrl_t         ctrl;
    usb_ctrl_t         *p_mess;
              :
    while(1)
    {
         USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
         ctrl = *p_mess;
         switch (ctrl.event)
         {
                         :
               case USB_STS_CONFIGURED:
                         :
                    R_USB_HmscGetSem();
                    R_tfat_f_open(&file, (const char *) &g_msc_file[drvno][0],
                                   (TFAT_FA_CREATE_ALWAYS | TFAT_FA_WRITE));
                    R_tfat_f_write(&file, g_file_data, sizeof(g_file_data), &file_size);
                    R_tfat_f_close(&file);
                    R_USB_HmscRelSem();
                         :
               break;
                         :
          }
     }
}
```

6.4 R_USB_HmscRelSem

Releases a semaphore (Only RTOS)

Format

void R_USB_HmscRelSem(void)

Arguments

none

Return Value

none

Description

Releases a specific semaphore which is used in this driver.

Reentrant

This API is reentrant.

Note

1. Be sure to call this API after calling the FAT file close function (e.g *R_tfat_f_close*).
2. An application task during a semaphore waiting status by *R_USB_HmscGetSem* function is released the semaphore waiting status by this API.
3. The creation processing of a semaphore which this API uses is performed in USB driver.

Example

```
/* Callback function */
void usb_apl_callback (usb_ctrl_t *p_ctr, usb_hdl_t hdl, uint8_t is_request)
{
    USB_APL_SND_MSG(hdl, (usb_msg_t *)p_ctrl);
}
void        usb_application_task( void )
{
    usb_ctrl_t        ctrl;
    usb_ctrl_t        *p_mess;
              :
    while(1)
    {
        USB_APL_RCV_MSG(USB_APL_MBX, (usb_msg_t **)&p_mess);
        ctrl = *p_mess;
        switch (ctrl.event)
        {
                      :
            case USB_STS_CONFIGURED:
                      :
                R_USB_HmscGetSem();
                R_tfat_f_open(&file, (const char *) &g_msc_file[drvno][0],
                              (TFAT_FA_CREATE_ALWAYS | TFAT_FA_WRITE));
                R_tfat_f_write(&file, g_file_data, sizeof(g_file_data), &file_size);
                R_tfat_f_close(&file);
                R_USB_HmscRelSem();
                      :
            break;
                      :
        }
    }
}
```

7. Return Value (USB_STS_MSC_CMD_COMPLETED) of a Mass Stoage Commnad

(1). Non-OS

After the completion of a Mass Storage command is checked with the *R_USB_HmscStrgCmd* function, if the *R_USB_GetEvent* function is called, then *USB_STS_MSC_CMD_COMPLETE* will be the return value.

(2). RTOS

When a Mass Storage command completes, the callback function that has been registered using the *R_USB_Callback* function will be called by the USB driver. At this time, *USB_STS_MSC_CMD_COMPLETE* will be set to the member (*event*) in the argument (the pointer to the *usb_ctrl_t* structure) of this callback function.

The following shows the information which is set to the member in the *usb_ctrl_t* structure when completing Mass Storage command.

module	:	USB module number where Mass Storage command has been completed.
address	:	Device address of USB device where Mass Storage command has been completed.
size	:	Size of response data
status	:	CSW information

Note:

1. The member (*module*) of the *usb_ctrl_t* structure has the USB module number (USB_IP0 / USB_IP1) connected to that USB device. The member (*address*) has the device address of the USB device where the Mass Storage command has been completed.
2. The member (*size*) has the size of the response data sent from MSC device.
3. The member (*status*) has bCSWStatus of the CSW (Command Status Wrapper):

USB_CSW_SUCCESS	(Value: 00H)	:	Successful
USB_CSW_FAIL	(Value: 01H)	:	Failed
USB_CSW_PHASE	(Value: 02H)	:	Phase error

8. Creating an Application

Refer to the chapter “**Creating an Application Program**” in the document (Document number: R01AN2025) for *USB Basic Host and Peripheral Driver using Firmware Integration Technology Application Note*.

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summery
1.00	Aug 1, 2014	—	First eddition issued.
1.10	Dec 26, 2014	—	1. RX71M is supported newly. 2. The following APIs are added. R_usb_hmsc_alloc_drvno, R_usb_hmsc_free_drvno R_usb_hmsc_ref_drvno 3. The argument “drvno” is added to the following APIs. R_usb_hmsc_SetDevSts, R_usb_hmsc_GetDevSts 4. The argument “ipno” is added to the following APIs. R_usb_hmsc_Information 5. The multiple connecting of MSC device is supported.
1.11	Sep 30, 2015	—	RX63N and RX631 are added in Target Device
1.20	Sep 30, 2015	—	1. RX65N and RX651 are added in Target Device. 2. Supporting DMA transfer. 3. Supporting USB Host and Peripheral Interface Driver application note(Document No.R01AN3293EJ)
1.21	Mar 31, 2017	—	1. Supported Technical Update (Document number. TN-RX*-A172A/E) 2. The API other than the chapter ***API Functions*** is moved to the document (Document number: R01AN2025) of *USB Basic Host and Peripheral Driver Firmware Integration Technology*.
1.22	Sep 30, 2017	—	Supporting RX65N/RX651-2M
1.23	Mar 31, 2018	—	Supporting the Smart Configurator.
1.24	Dec 28, 2018	—	1. Supporting RTOS. 2. Supporting R_USB_HmscGetSem/R_USB_HmscRelSem function.
1.25	Apr 16, 2019	—	Added RX66T/RX72T in Target Device.
1.26	May 31, 2019	—	1. Support GCC compiler and IAR compiler. 2. Remove RX63N from Target Device.
1.27	Jul 31, 2019	—	RX72M is added in Target Device.

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

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

1. Handling of Unused Pins

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

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

2. Processing at Power-on

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

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

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

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

Renesas Electronics Corporation

SALES OFFICES

http://www.renesas.com

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

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

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

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

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

Renesas Electronics (China) Co., Ltd.
Room 101-T01, Floor 1, Building 7, Yard No. 7, 8th Street, Shangdi, Haidian District, Beijing 100085, China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit No 3A-1 Level 3A Tower 8 UOA Business Park, No 1 Jalan Pengaturcara U1/51A, Seksyen U1, 40150 Shah Alam, Selangor, Malaysia
Tel: +60-3-5022-1288, Fax: +60-3-5022-1290

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

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

Colophon 8.0