Renesas USB MCU

R01AN2026EJ0110
Rev.1.10
Dec 26, 2014

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 firmware (USB-BASIC-F/W FIT module). It is referred to below as the USB HMSC FIT module.

Target Device

RX64M Group
RX71M 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 No.R01UH0377EJ)
5. RX71M Group User's Manual: Hardware (Document number No.R01UH0493EJ)
6. RX Family M3S-TFAT-Tiny: FAT file system software (Document No.R20AN0038EJ)
7. USB Basic Host and Peripheral firmware using Integration Technology Application Note
   (Document No. R01AN2025EJ)

Renesas Electronics Website
http://www.renesas.com/
USB Device Page
http://www.renesas.com/prod/usb/

Contents

1. Overview

The USB HCDC FIT module, when used in combination with the USB-BASIC-F/W 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 this software uses the M3S-TFAT-Tiny (TFAT). Therefore, it should be used in combination with RX Family M3S-TFAT-Tiny: FAT File System Software (document No. R20AN0038JJ).

This module supports the following functions.

- Checking of connected USB storage devices (to determine whether or not operation is supported).
- Storage command communication using the BOT protocol.
- Support for SFF-8070i (ATAPI) USB mass storage subclass.
- Maximum 4 USB storage devices can be connected.

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. Additional limitations apply to HMSDD. See section 10 for details.

Terms and Abbreviations

APL	:	Application program
BOT	:	Mass storage class Bulk Only Transport
cstd	:	Prefix of function and file for Peripheral & Host Common Basic (USB low level) F/W
FSL	:	FAT File System Library
HCD	:	Host Control Driver of USB-BASIC-F/W
HDCD	:	Host Device Class Driver (device driver and USB class driver)
HSTD	:	Function & File for Host Basic (USB low level) F/W
HUBCD	:	Hub Class Smple Driver
MGR	:	Peripheral device state manager of HCD
non-OS	:	USB basic firmware for non-OS based system
PP	:	Pre-processed definition
RSK	:	Renesas Starter Kits
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function (non-OS)
Task	:	Processing unit
TFAT	:	Tiny FAT file system software for microcontrollers (M3S-TFAT-Tiny-RX)
USB-BASIC-F/W	:	USB basic firmware for Renesas USB MCU (non-OS)
USB	:	Universal Serial Bus

1.1 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.
Please refer to the chapter “How to add the module” about how to add the module.

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.

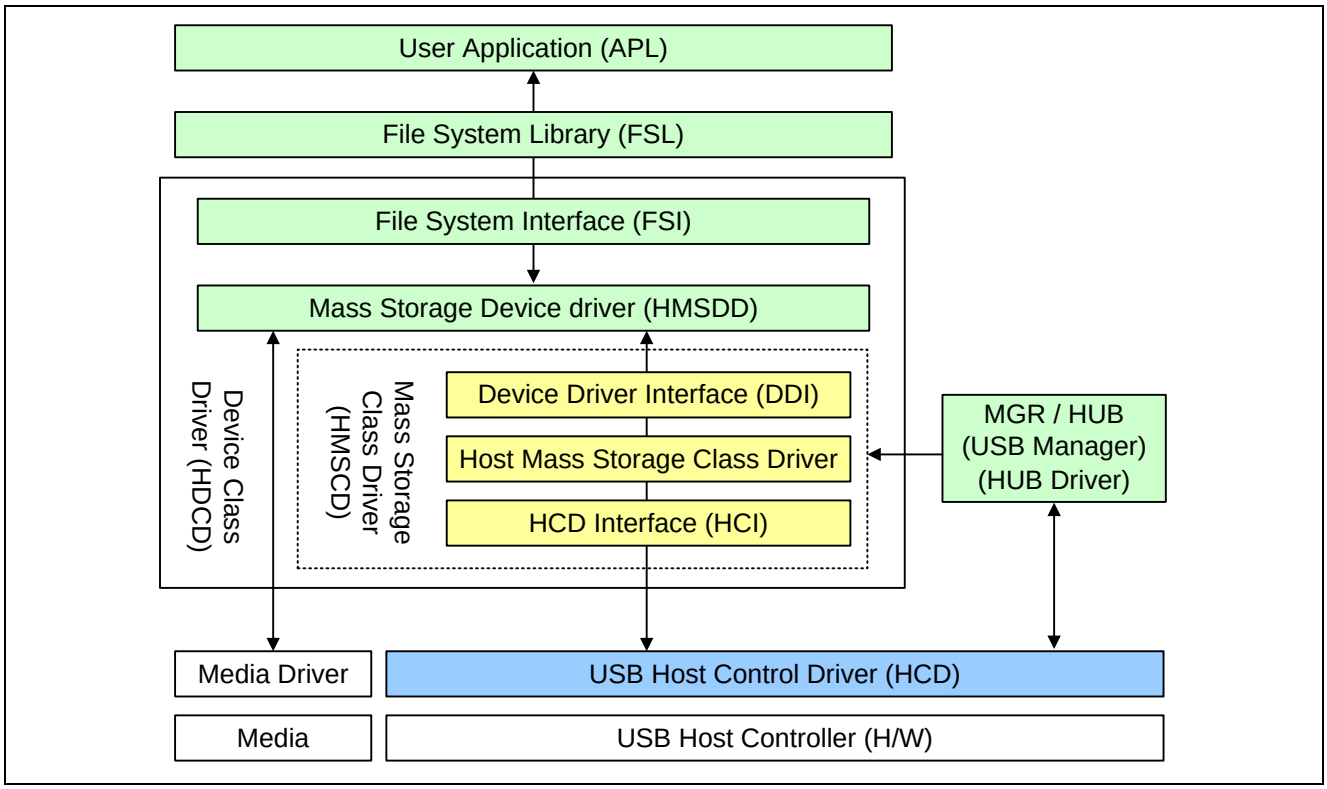

Figure 2-1 Software Block Diagram

Table 2-1 Module

Module	Description
APL	Calls FSL functions to implement storage functionality. Created by the customer to match the system specifications.
FSL	FAT file system with specifications defined by the user.
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.
Media Driver	Driver for non-USB storage devices.

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
- r_tfat_rx

3.3 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v.2.01.00b

3.4 Header Files

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

3.5 Integer Types

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

3.6 Compile Setting

In order to use this module, it is necessary to set the USB-BASIC-F/W FIT module as a host. Refer to USB Basic Firmware application note (Document No.R01AN2025EJ) for information on USB-BASIC-F/W FIT module settings.

3.7 Argument

The structure for the arguments of the API functions is shown below. For details, please refer to USB Basic Firmware application note (Document No.R01AN2025EJ).

```
typedef struct USB_UTR
{
        USB_MH_t        msghead;                /* Message header (for RTOS) / No used */
        uint16_t        msginfo;                /* Message information */
        uint16_t        keyword;                /* Sub information(Port No/Pipe No etc）  */
        union {
                USB_REGADR_t        ipp;        /* USB IP address (for USBb module)*/
                USB_REGADR1_t       ipp1;       /* USB IP address (for USBA/USBAa module) */
        };
        uint16_t        ip;                     /* USB IP number*/
        uint16_t        result;                 /* USB transfer result */
        USB_CB_t        complete;               /* Pointer to the callback function */
        void            *tranadr;               /* Pointer to USB transfer buffer */
        uint32_t        tranlen;                /* USB data length*/
        uint16_t        *setup;                 /* Pointer to Setup packet data */
        uint16_t        status;                 /* USB status */
        uint16_t        pipectr;                /* PIPECTR register */
        uint8_t         errcnt;                 /* Error count for transferring */
        uint8_t         segment;                /* Segment information */
        void            *usr_data;              /* Other various information */
} USB_UTR_t;
```

3.8 How to add the module

This module must be added to an existing e^2 studio project. By using the e^2 studio plug-in, it is possible to update the include file path automatically. It is therefore recommended that this plug-in be used to add the project.

For instructions when using e^2 studio, refer to RX Family: Integration into e^2 studio, Firmware Integration Technology (document No. R01AN1723EU).

For instructions when using CS+, refer to RX Family: Adding Firmware Integration Technology Modules to CS+ Projects, Firmware Integration Technology (document No. R01AN1826EJ).

4. How to Register Class Driver

The class driver which the user created functions as a class driver by registering with a USB driver.
For details, please refer to the chapter "How to register Host Class Driver" of USB Basic Host and Peripheral firmware using Firmware Integration Technology application note (Document No.R01AN2025EJ).

5. System Resources

The resources used by HMSC are listed below.

Table 5-1 Task Information

Function Name	Task ID	Priority	Description
usb_hmsc_Task	USB_HMSC_TSK	USB_PRI_3	Mass storage task
usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	HMSDD task

Table 5-2 Mailbox Information

Mailbox Name	Using Task ID	Task Queue	Description
USB_HMSC_MBX	USB_HMSC_TSK	FIFO order	For HMSCD
USB_HSTRG_MBX	USB_HSTRG_TSK	FIFO order	For HMSDD

Table 5-3 Memory Pool Mailbox Information

Memory Pool Name	Task Queue	Memory Block (note)	Description
USB_HMSC_MPL	FIFO order	40byte	For HMSC
USB_HSTRG_MPL	FIFO order	40byte	For HDCD

Note: The maximum number of memory blocks for the entire system is defined in USB_BLKMAX. The default value is 20.

6. Target Peripheral List（TPL）

When using a USB host driver (USB-BASIC-F/W FIT module) and device class driver in combination, it is necessary to create a target peripheral list (TPL) for each device driver.
For details, see “Target Peripheral List,” in USB Basic Firmware application note (Document No.R01AN2025EJ).

7. Task ID etc and Priority Setting

Define the setting value for User task ID etc within the following range. Please use R_usb_cstd_SetTaskPri(API) about Task priority setting.

Task ID : 6 to (USB_IDMAX - 1)
Mailbox ID : Set the same value as Task ID
Memory pool ID : Set the same value as Task ID
Task Priority : 4 to (USB_PRIMAX - 1)

[Note]

1. USB_IDMAX and USB_PRIMAX is the defined by User in r_usb_config.h file.
2. Please define ID number for User task ID etc in the application file.

8. Accessing USB Storage Devices

FSI (File System Interface) converts sector numbers to logical block addresses and sector counts to transfer data sizes. From the drive number, HMSDD determines if a USB storage device is being accessed. HMSCD converts drive numbers to unit numbers and accesses USB storage devices via HCD according to the BOT protocol.

Figure8-1 shows the USB storage device access sequence.

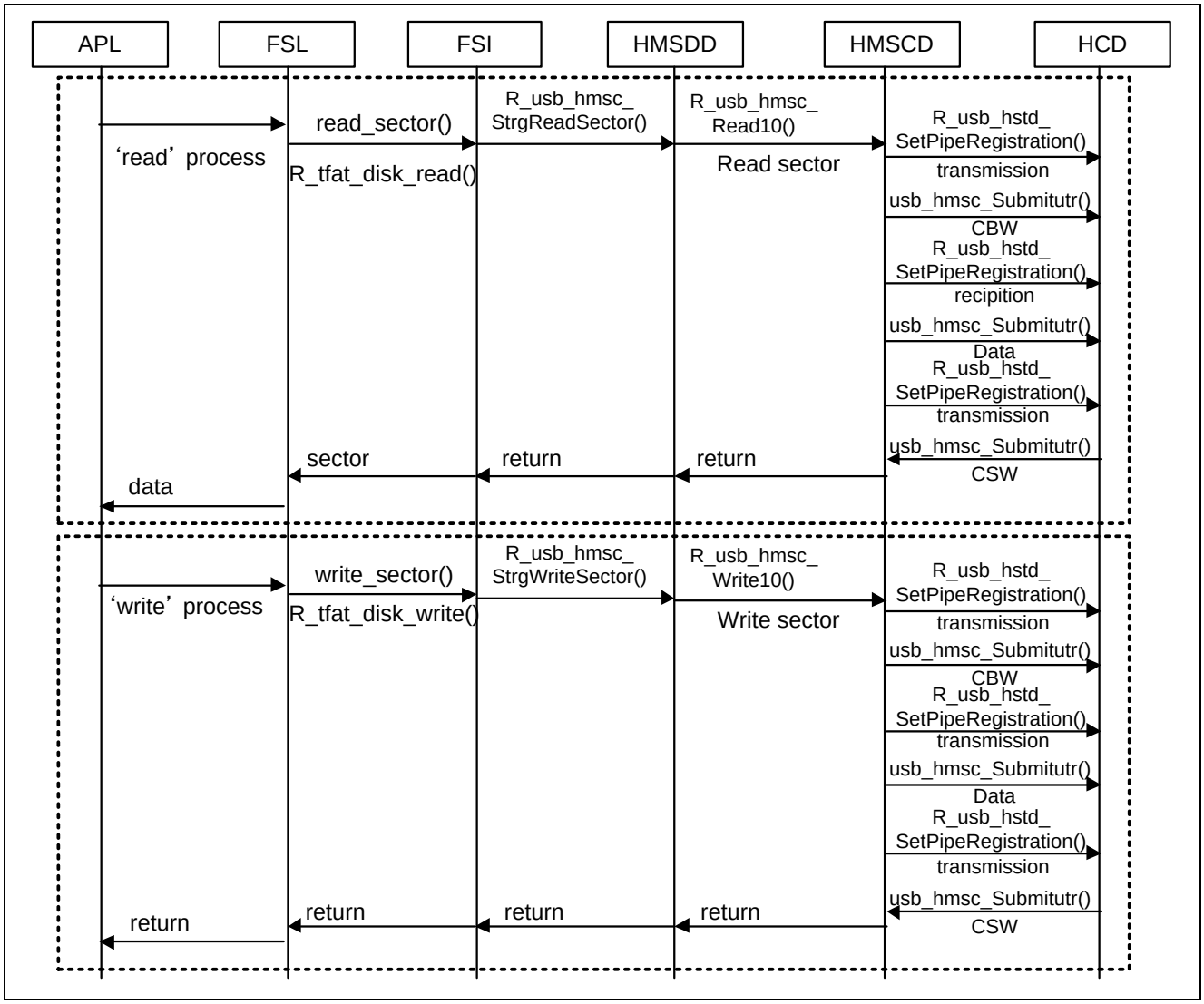

Figure8-1 USB Storage Device Access Sequence

9. File System Interface (FSI)

9.1 Functions and Features

Create the FSL to match the specifications of the interface.

9.2 FSI API

Table 9-1 lists the functions of the sample FSI.

R_tfat_disk_read/R_tfat_disk_write/R_usb_hmsc_WaitLoop API are supported by the application note "M3S-TFAT-Tiny Memory Driver Interface Module Firmware Integration Technology" (Document No.R20AN0335EJ).

Table 9-1 FSI Functions

Function Name	Description
R_usb_hmsc_DriveClose	Close Drive
R_tfat_disk_read / read_sector	Read data
R_tfat_disk_write / write_sector	Write data
R_usb_hmsc_WaitLoop	Wait for completion of DATA READ / DATA WRITE (For non-OS)
dev_open	Mount a drive (TFAT is not in use)
dev_close	Unmount a drive (TFAT is not in use)

9.2.1 R_usb_hmsc_DriveClose

Close Drive

Format

void	R_usb_hmsc_DriveClose(USB_UTR_t *ptr, uint16_t addr, uint16_t data2)

Argument

*ptr	Pointer to USB_UTR_t structure
addr	Device address
data2	Not used

Return Value

—	—

Description

This function initializes the pipe table information and global variables.

Notes

The function is registered in usb_hapl_registration() as the callback function in this F/W.

Example

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
    :
  /* Device detach */
  driver.devdetach = &R_usb_hmsc_DriveClose;
    :
}
```

10. Host Mass Storage Device Driver (HMSDD)

HMSDD, the “device driver”, is called by FAT when using HMSCD. HMSDD selects a storage device depending on a given drive number.

10.1 Limitation

- Maximum 4 USB storage devices can be connected.
- USB storage devices with a sector size of 512 bytes can be connected.
- A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.
- Some devices may be unable to be connected (because they are not recognized as storage devices).

10.2 Logical Unit Number (LUN)

If a device does not respond to a GetMaxUnit request and the unit count is undetermined, the device operates with a unit count setting of 0. HMSCD creates USB packets according to the BOT specification. The CBWCB field (storage command) of the data packet is created according to the SCSI specification. The LUN field settings in the bCBWLUN field of the CBW packet and in the storage command are listed in.Table 10-1.

Table 10-1 LUN Field Settings

	bCBWLUN Field	LUN Field in Command
usb_ghmsc_MaxLUN = 0	0	0
usb_ghmsc_MaxLUN = other than 0	Unit number	0

10.3 Changing Logical Block Addresses

Under the BOT specification, information is read and written according to logical block addresses. The data size is specified as the number of bytes of information that are read or written.
The logical block address is calculated from the sector number and offset sector number. The transfer size is calculated from the sector count and sector size.

Logical block address = logical sector number + offset sector number

Transfer size = sector count * sector size

10.4 HMSDD API Function

Table 10-2 lists the function of HMSDD.

Table 10-2 HMSDD 関数

Function Name	Description
R_usb_hmsc_StrgDriveTask()	Gets the storage device information
R_usb_hmsc_StrgDriveSearch()	Searchs the accessible drive
R_usb_hmsc_StrgReadSector()	Reads data.
R_usb_hmsc_StrgWriteSector()	Writes data.
R_usb_hmsc_StrgUserCommand()	Issues storage command.
R_usb_hmsc_alloc_drvno	Allocates the drive number.
R_usb_hmsc_free_drvno	Frees the drive number
R_usb_hmsc_ref_drvno	Refers the drive number

10.4.1 R_usb_hmsc_StrgDriveTask

Storage drive task

Format

void	R_usb_hmsc_StrgDriveTask(void)

Argument

—	—

Return Value

—	—

Description

This API does the processing to get the storage device information by sending SFF-8070i (ATAPI) command to the storage device..

Notes

1. Please call this function from the application program

Example

```
void usb_hapl_mainloop(void)
{
   while(1)
   {
      R_usb_cstd_Scheduler();

      if(USB_FLGSET == R_usb_cstd_CheckSchedule())
      {
            R_usb_hstd_HcdTask((USB_VP_INT)0);
            R_usb_hstd_MgrTask((USB_VP_INT)0);
            R_usb_hhub_Task((USB_VP_INT)0);
            R_usb_hmsc_Task((USB_VP_INT)0)
            R_usb_hmsc_StrgDriveTask();
      }
                  :
   }
}
```

10.4.2 R_usb_hmsc_StrgDriveSearch

Searchs the accessible drive

Format

USB_ER_t R_usb_hmsc_StrgDriveSearch(USB_UTR_t *ptr, uint16_t addr, USB_CB_t complete)

Argument

*ptr	Pointer to USB_UTR_t structure
addr	USB address
complete	Callback function

Return Value

USB_DONE	Normal end
USB_ERROR	Error end

Description

This API checks the follows by sending the class request (GetMaxLun) and SFF-8070i (ATAPI) command to USB device which is specified in the argument (addr).

1. The number of unit of the storage device
2. Accesible the drive

The callback function which is specified in the argument (complete) is called whencompleting the drive searching.

Notes

2. Please call this function from the class driver or the FAT library I/F function.
3. Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
/* Callback function */
void usb_hmsc_StrgCommandResult( USB_UTR_t *mess, uint16_t data1, uint16_t data2 )
{
        :
}

void usb_hmsc_SampleAplTask(void)
{
        :
    R_usb_hmsc_StrgDriveSearch(mess, addr,(USB_CB_t)&usb_hmsc_StrgCommandResult);
        :
}
```

10.4.3 R_usb_hmsc_StrgReadSector

Read Sector Information

Format

USB_ER_t R_usb_hmsc_StrgWriteSector(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte))

Argument

ptr	Pointer to USB_UTR_t structure
side	Drive number
buff	Pointer to read data storage area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USB_DONE	Normal end
USB_ERROR	Error end

Description

Reads the sector information of the drive specified by the argument.
An error response occurs in the following cases.

1. When the sector information could not be read successfully from the storage device.

Notes

1. Please call this function from FAT library I/F function.
2. Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Numbe

Example

```
int read_sector(USB_UTR_t *ptr, int side, unsigned char *buff, unsigned long secno,
              long seccnt)
{
              :
    error = R_usb_hmsc_StrgReadSector(ptr, (uint16_t)side, buff, secno,
              (uint16_t)seccnt, trans_byte);

    if( error == USB_ERROR )
    {
              :
       return (-1);
    }
    return (0);
}
```

10.4.4 R_usb_hmsc_StrgWriteSector

Write Sector Information

Format

USB_ER_t R_usb_hmsc_StrgWriteSector(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte))

Argument

ptr	Pointer to USB_UTR_t structure
side	Drive number
buff	Pointer to write data storage area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

USB_DONE	Normal end
USB_ERROR	Error end

Description

Writes the sector information of the drive specified by the argument.
An error response occurs in the following cases.

1. When the sector information could not be read successfully from the storage device.

Notes

1. Please call this function from FAT library I/F function.
2. Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Numbe

Example

```
int write_sector(USB_UTR_t *ptr, int side, unsigned char *buff, unsigned long secno,
              long seccnt)
{
            :
    error = R_usb_hmsc_StrgWriteSector(ptr, (uint16_t)side, buff, secno,
              (uint16_t)seccnt, trans_byte);

    if( error == USB_ERROR )
    {
            :
        return (-1);
    }
    return (0);
}
```

10.4.5 R_usb_hmsc_StrgUserCommand

Issue Storage Command

Format

uint16_t R_usb_hmsc_StrgUserCommand(USB_UTR_t *ptr, uint16_t side, uint16_t command , uint8_t *buff, USB_CB_t complete)

Argument

*ptr	Pointer to USB_UTR_t structure
side	Drive number
command	Command to be issued
*buff	Data pointer
complete	Callback function

Return Value

USBC_DONE	Normal end
USBC_ERROR	Error end

Description

This function issues the storage command specified by the given argument, to the HMSC driver. The callback function which is specified in the argument (complete) is called when completing the issued storage command. Here are the storage commands supported:

Storage commands	Description
USB_ATAPI_TEST_UNIT_READY	Check status of peripheral device
USB_ATAPI_REQUEST_SENCE	Get status of peripheral device
USB_ATAPI_INQUIRY	Get parameter information of logical unit
USB_ATAPI_MODE_SELECT6	Specify parameters
USB_ATAPI_PREVENT_ALLOW	Enable/disable media removal
USB_ATAPI_READ_FORMAT_CAPACITY	Get formattable capacity
USB_ATAPI_READ_CAPACITY	Get capacity information of logical unit
USB_ATAPI_MODE_SENSE10	Get parameters of logical unit

Notes

Please call this function from the application program and the class driver.

Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
/* Callback function */
void  strgcommand_complete(USB_UTR_t *mess, uint16_t data1, uint16_t data2)
{
            :
}
void  usb_smp_task(void)
{
    :
  /* Issuing TEST_UNIT_READY */
  R_usb_hmsc_StrgUserCommand(ptr, side, USB_ATAPI_TEST_UNIT_READY, buf,
(USB_CB_t)complete);
    :
}
```

10.4.6 R_usb_hmsc_alloc_drvno

Allocates the driver number

Format

uint16_t R_usb_hmsc_alloc_drvno(uin16_t ipno, uint16_t devadr)

Argument

ipno	USB module number
devadr	Device address of MSC device

Return Value

–	Drive number

Description

This function allocate the drive number to the connected MSC device. c

Notes

1. The user can specifies the drive number which is allocated by this API in the argument of FAT API.
2. Specifies USB module number which MSC device is connected to in the argument "*ipno*".

USB Module	USB Module Number
USBb	USB_IP0
USBA/USBAa	USB_IP1

Example

```
void usb_smp_task(void)
{
            :
    /* Allocates the drive number */
    drvno = R_usb_hmsc_alloc_drvno(USB_IP1, devadr);
            :
}
```

10.4.7 R_usb_hmsc_free_drvno

Frees the driver number

Format

void R_usb_hmsc_free_drvno(uint16_t drvno)

Argument

drvno Drive number

Return Value

— —

Description

This function frees the drive number which is specified by the argument.

Notes

Specifies USB module number which MSC device is connected to in the argument "*ipno*".

USB Module	USB Module Number
USBb	USB_IP0
USBA/USBAa	USB_IP1

Example

```
void usb_smp_task(void)
{
            :
    /* Frees the drive number */
    R_usb_hmsc_free_drvno(drvno);
            :
}
```

10.4.8 R_usb_hmsc_ref_drvno

Refers the driver number

Format

void R_usb_hmsc_ref_drvno(uint16_t ipno, uint16_t devadr)

Argument

ipno	USB module number
devadr	Device address

Return Value

—	Drive number

Description

This function refers the drive numer based on USB module number and the device address which are specified by the argument.

Notes

—

Example

```
void usb_smp_task(void)
{
            :
    /* Frees the drive number */
    R_usb_hmsc_free_drvno(drvno);
            :
}
```

11. USB Mass Storage Class Driver (HMSCD)

11.1 Functions and Features

HMSCD executes storage command communication, if the USB storage devices are ready to operate. The BOT protocol is used, which encapsulates the storage commands as they pass through USB.
MSCD comprises three layers: HMSDD interface (DDI functions), HCD interface (HCI functions), and HMSCD itself.
HMSCD supports storage commands necessary for accessing USB storage devices and sample storage commands.
HMSCD has the following features.
Support for USB mass storage class BOT.
Support for SFF-8070i (ATAPI) and SCSI USB mass storage subclasses.
Sharing of a single pipe for IN/OUT directions or multiple devices.

11.2 Issuing Requests to HMSCD

The interface functions described below (Table 11-4and Table 11-5) are used when accessing USB storage devices.
HMSCD sends notification of results in response to requests from a higher layer in the return value of the registered callback function...

11.3 HMSCD Structures

Table 11-1 and Table 11-2 show the contents of the HMSCD structures.

Table 11-1 USB_MSC_CBW_t Structure

	Member Name	Description	Remarks
uint32_t	dCBWSignature	CBW Signature	0x55534243: USBC
uint32_t	dCBWTag	CBW Tag	Tag corresponding to CSW
uint8_t	dCBWDTL_Lo	CBW Data Transfer Length	Data length of transmit/receive data
uint8_t	dCBWDTL_ML		
uint8_t	dCBWDTL_MH		
uint8_t	dCBWDTL_Hi		
uint8_t	bmCBWFlags	CBW Direction	Data transmit/receive direction
uint8_t	bCBWLUN	Logical Unit Number	Unit number
uint8_t	bCBWCBLength	CBWCB Length	Command length
uint8_t	CBWCB[16]	CBWCB	Command block

Table 11-2 USB_MSC_CSW_t Structure

	Member Name	Description	Remarks
uint32_t	dCSWSignature	CSW Signature	0x55534253: USBS
uint32_t	dCSWTag	CSW Tag	Tag corresponding to CBW
uint8_t	dCSWDataResidue_Lo	CSW DataResidue	Data length used
uint8_t	dCSWDataResidue_ML		
uint8_t	dCSWDataResidue_MH		
uint8_t	dCSWDataResidue_Hi		
uint8_t	bCSWStatus	CSW Status	Command status
uint8_t	dummy	Dummy	Even number adjustment

11.4 USB Host Control Driver Interface (HCI) Functions

HCI is the interface function between HMSCD and HCD.
HCI uses the HMSCD area to send and receive messages. Note that two devices cannot be enumerated (or accessed) simultaneously.

11.5 Device Driver Interface (DDI) Functions

DDI is the interface function between HMSDD and HMSCD.
DDI functions comprise the HMSCD start function, end function, check connected device function, and sample storage command functions.

11.6 HMSC API

Application programming interface description for HMSCD

Table 11-3 HMSCD Functions

	Function Name	Description
1	R_usb_hmsc_SetDevSts()	Sets HMSCD operation state.
2	R_usb_hmsc_GetDevSts()	Returns HMSCD operation state.
3	R_usb_hmsc_Information()	Gets Pipe No that is Data transmission / reception uses.
4	R_usb_hmsc_Task()	Host Mass Storage Class Task
5	R_usb_hmsc_driver_start()	HMSC driver start processing.

Table 11-4 HCI Functions

	Function Name	Description
1	R_usb_hmsc_GetMaxUnit()	Get_MaxUnit request execution.
2	R_usb_hmsc_MassStorageReset()	MassStorageReset request execution.
3	R_usb_hmsc_ClearStall()	STALL release request execution to USB driver.

Table 11-5 DDI Functions

	Function Name	Description
1	R_usb_hmsc_Initialized()	Initializes Host Mass storage class
2	R_usb_hmsc_ClassCheck()	Checks the descriptor table of the connected device to determine whether or not HMSCD can operate.
3	R_usb_hmsc_Read10()	Issues the READ10 command.
4	R_usb_hmsc_Write10()	Issues the WRITE10 command.
5	R_usb_hmsc_DriveSpeed()	Returns the communication speed of the drive.
6	R_usb_hmsc_Release()	Releases Host Mass storage class driver.

11.6.1 R_usb_hmsc_SetDevSts

Sets HMSCD Operation State

Format

uint16_t R_usb_hmsc_SetDevSts(uint16_t drvno, uint16_t data)

Argument

drvno	Drive number
data	Operation state

Return Value

—	USB_DONE

Description

This function changes the HMSCD operation state according to the value specified in the argument.
The operation states are listed below.

Operation State	Description
USB_HMSC_DEV_DET	Detach state
USB_HMSC_DEV_ATT	Attach state（Complete Enumeration）
USB_HMSC_DEV_ENU	In Enumeration

Note

1. Please call this function from the user program or the class driver.
2. The return value is always USB_DONE.
3. Specifies the driver number which is allocated by *R_usb_hmsc_alloc_drvno* function in the argument "*drvno*".

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  /* Change the device state to Attach state */
  R_usb_hmsc_SetDevSts(drvno, (uint16_t)USB_HMSC_DEV_ATT);
    :
}
```

11.6.2 R_usb_hmsc_GetDevSts

Returns HMSCD operation state

Format

uint16_t R_usb_hmsc_GetDevSts(uint16_t drvno)

Argument

drvno Drive number

Return Value

usb_ghmsc_AttSts USB_HMSC_DEV_ATT（Attach）
USB_HMSC_DEV_DET（Detach）

Description

Returns the HMSCD operation state.

Note

1. Please call this function from the user application program or the class driver.
2. Specifies the driver number which is allocated by *R_usb_hmsc_alloc_drvno* function in the argument "*drvno*".

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  /* Checking the device state */
  if(R_usb_hmsc_GetDevSts(drvno) == USB_HMSC_DEV_DET)
  {
   /* Detach processing */

  }
    :
}
```

11.6.3 R_usb_hmsc_Information

Gets pipe number used for data transmit / receive

Format

uint16_t R_usb_hmsc_Information(uint16_t ipno, uint16_t pipe_offset)

Argument

ipno	USB module number
pipe_offset	Pipe Information Table

Return Value

—	Pipe no

Description

This function gets the pipe number used for data transmit/receive from the pipe information table.

Note

1. Please call this function from the user application program or the class driver.
2. Specifies USB module number which MSC device is connected to in the argument "*ipno*".

USB Module	USB Module Number
USBb	USB_IP0
USBA/USBAa	USB_IP1

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t  pipeno;
    :
  /* Getting Pipe No */
  pipeno = R_usb_hmsc_Information(usb_ghmsc_InPipe[msgnum][0]);
    :
}
```

11.6.4 R_usb_hmsc_Task

Host Mass Storage Class task

Format

void	R_usb_hmsc_Task(void)

Argument

—	—

Return Value

—	—

Description

This function is a task for HMSCD.
This function controls BOT.

Note

Please call this function from a loop that executes the scheduler processing for non-OS operations.
Please refer to the chapter "Operation Flow in Static State" in the Basic F/W application note for more information about this loop.

Example

```
void  usb_smp_mainloop(void)
{
  while(1)
  {
  /* Scheduler processing */
  R_usb_cstd_Scheduler();
  /* Checking flag */
  if(USB_FLGSET == R_usb_cstd_CheckSchedule())
  {
       :
    usb_hmsc_Task();
       :
  }
    :
}
```

11.6.5 R_usb_hmsc_driver_start

HMSC driver start

Format

void	R_usb_hmsc_driver_start(USB_UTR_t *ptr)

Argument

*ptr	Pointer to USB Transfer structure

Return Value

—	—

Description

This function sets the priority of HMSC driver task.
The sent and received of message are enable by the priority is set.

Note

1. Call this function from the user application program during initialization.
2. Please set the following member of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void usb_hstd_task_start( void )
{
  USB_UTR_t  *ptr;
    :
  ptr->ip = USB_HOST_USBIP_NUM;   /* USB IP No */
  ptr->ipp = R_usb_cstd_GetUsbIpAdr( ptr->ip );  /* USB IP base address */
    :
  R_usb_hmsc_driver_start( ptr );    /* Host Class Driver Task Start Setting */
    :
}
```

11.6.6 R_usb_hmsc_GetMaxUnit

Issue GetMaxLUN request.

Format

USB_ER_t R_usb_hmsc_GetMaxUnit(USB_UTR_t *ptr, uint16_t addr, USB_CB_t complete)

Argument

*ptr	Pointer to USB_UTR_t structure
addr	Device address
complete	Callback function

Return Value

USB_E_OK	GET_MAX_LUN issued
USB_E_ERROR	GET_MAX_LUN not issued
USB_E_QOVR	GET_MAX_LUN not issued

Description

This function issues the GET_MAX_LUN request and gets the maximum storage unit count. The callback function which is specified in the argument (complete) is called when completing this request.

Note

Please call this function from the user application program or the class driver.

1. Please set the following members of USB_UTR_t structure.
   USB_REGADR_t ipp : USB register base address
   uint16_t ip : USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  USB_ER_t  err;
    :
  /* Getting Max unit number */
  err = R_usb_hmsc_GetMaxUnit(mess, addr, (USB_CB_t)usb_hmsc_StrgCheckResult);
  if(err == USB_E_QOVR)
  {
    /* Error processing */
  }
    :
}
```

11.6.7 R_usb_hmsc_MassStorageReset

Issue Mass Storage Reset request.

Format

USB_ER_t R_usb_hmsc_MassStorageReset(USB_UTR_t *ptr, uint16_t drvnum, USB_CB_t complete)

Argument

*ptr	Pointer to USB_UTR_t structure
drvnum	Drive number
complete	Callback function

Return Value

USB_E_OK	MASS_STORAGE_RESET issued
USB_E_ERROR	MASS_STORAGE_RESET not issued
USB_E_QOVR	MASS_STORAGE_RESET not issued

Description

This function issues the MASS_STORAGE_RESET request and cancels the protocol error. The callback function which is specified in the argument (complete) is called when completing this request.

Note

Please call this function from the user application program or the class driver.
Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  USB_ER_t  err;
    :
  /* Cansel the protocol error */
  err = R_usb_hmsc_MassStorageReset(ptr, drvnum, (USB_CB_t)usb_hmsc_CheckResult);
  if(err == USB_E_QOVR)
  {
    /* Error processing */
  }
    :
}
```

11.6.8 R_usb_hmsc_ClearStall

Cancel pipe STALL state

Format

void	R_usb_hmsc_ClearStall(USB_UTR_t *ptr, uint16_t type, uint16_t msgnum , USB_CB_t complete)

Argument

*ptr	Pointer to USB_UTR_t structure
type	IN/OUT communication direction
msgnum	Driver number
complete	Callback function

Return Value

—	—

Description

This function releases the pipe from the STALL state.

Note

Please call this function from the user application program or the class driver.

Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
    :
  /* Cansel STALL state of the device */
  R_usb_hmsc_ClearStall(ptr, (uint16_t)USB_DATA_NONE, msgnum,
    (USB_CB_t)usb_hmsc_ClearStallCheck2);
    :
}
```

11.6.9 R_usb_hmsc_Initialized

Initialize Host Mass Storage class

Format

void	R_usb_hmsc_Initialized(USB_UTR_t *ptr, uint16_t data1, uint16_t data2)

Argument

*ptr	Pointer to USB_UTR_t structure
data1	Not used
data2	Not used

Return Value

—	—

Description

Initializes an internal variable of HMSCD.

Note

—

Example

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
    :
  /* Driver init */
  R_usb_hmsc_Initialized( ptr, d1, d2 );
    :
}
```

11.6.10 R_usb_hmsc_ClassCheck

Compare connected device with interface descriptor

Format

void	R_usb_hmsc_ClassCheck(USB_UTR_t *ptr, uint16_t **table)

Argument

*ptr	Pointer to USB_UTR_t structure
**table	Not used

Return Value

—	—

Description

Checks the device count and drive count, and analyzes the interface descriptor table. Confirms that the items listed below match HMSCD, and reads the serial number if the operation is possible. Updates the pipe information table with information from the bulk endpoint descriptor table (endpoint address, max. packet size, etc.).

Interface descriptor information check

bInterfaceSubClass = USB_ATAPI / USB_SCSI
bInterfaceProtocol = USB_BOTP
bNumEndpoint > USB_TOTALEP

String descriptor information check
Serial number of 12 or more characters (warning indication in case of error)
Endpoint Descriptor information check

bmAtributes = 0x02 (bulk endpoint required)
bEndpointAdress (endpoints required for both IN and OUT)

One of the following check results is returned as Table [3].

USB_DONE: HMSCD operation possible
USB_ERROR: HMSCD operation not possible

Note

1. In application program, register this API as the callback function by setting this API in the member *classcheck*.
2. The result of USB receiving processing is set to the argument "*ptr" of this function.
3. The maximum connectable storage device count is defined by USB_MAXSTRAGE. (Refer to r_usb_hmsc_define.h)
4. The maximum operable drive count is defined by USB_MAXDRIVE.(Refer to r_usb_hmsc_define.h)

Example

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
    :
  /* Driver check */
  driver.classcheck   = &R_usb_hmsc_ClassCheck;
    :
}
```

11.6.11 R_usb_hmsc_Read10

Issue a READ10 command

Format

uint16_t R_usb_hmsc_Read10(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

*ptr	Pointer to USB_UTR_t structure
side	Drive number
*buff	Read data area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

–	Error code

Description

Creates and executes the READ10 command.
When a command error occurs, the REQUEST_SENSE command is executed to get error information.

Note

1. Please call this function from the user application program or the class.driver.
2. Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint32_t   result;
    :
  /* Issuing READ10 */
  result = R_usb_hmsc_read10(ptr, side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* Error Processing */
  }
    :
}
```

11.6.12 R_usb_hmsc_Write10

Issue a WRITE10 command

Format

uint16_t R_usb_hmsc_Wead10(USB_UTR_t *ptr, uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

*ptr	Pointer to USB_UTR_t structure
side	Drive number
*buff	Write data area
secno	Sector number
seccnt	Sector count
trans_byte	Transfer data length

Return Value

–	Error code

Description

Creates and executes the WRITE10 command.
When a command error occurs, the REQUEST_SENSE command is executed to get error information.

Note

Please call this function from the user program or the class driver.
Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint32_t  result;
    :
  /* Issuing WRITE10 */
  result = R_usb_hmsc_Write10(ptr, side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
    /* Error processing */
  }
    :
}
```

11.6.13 R_usb_hmsc_DriveSpeed

Check drive communication speed

Format

uint16_t R_usb_hmsc_DriveSpeed(USB_UTR_t *ptr, uint16_t side)

Argument

*ptr	Pointer to USB_UTR_t structure
side	Drive number

Return Value

—	Communication speed

Description

Returns the communication speed of the drive.

Note

Please call this function from the application program or the class driver.
Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t  result;
    :
  /* Getting the USB speed */
  result = R_usb_hmsc_DriveSpeed(ptr, side);
    :
}
```

11.6.14 R_usb_hmsc_Release

Release Host Mass Storage Class Driver

Format

void R_usb_hmsc_Release(USB_UTR_t *ptr)

Argument

*ptr Pointer to USB_UTR_t structure

Return Value

— —

Description

Cancel HMSCD registration from HCD.

Note

Please call this function from the application program or the class driver,
Please set the following members of USB_UTR_t structure.

USB_REGADR_t	ipp	: USB register base address
uint16_t	ip	: USB IP Number

Example

```
void  usb_smp_task(USB_UTR_t *ptr)
{
  uint16_t  result;
    :
  /* Cancel HMSCD registration */
  R_usb_hmsc_Release(ptr);
    :
}
```

12. Creating an Application

This section describes the initial settings necessary for using the USB HMSC FIT module and USB-BASIC-F/W FIT module in combination as a USB driver and presents an example of data transfer by means of processing by the main routine and the use of API functions.

12.1 Initial Settings of USB Driver

The following settings are required in order to use the USB driver.

- MCU pin settings
- USB controller startup and settings
- USB driver settings

Sample settings are shown below.

```
/* USB Transfer Structure for HMSC */
USB_UTR_t   usb_ghmsc_utr

void usb_hmsc_apl(void)
{
    /* USB transfer structure */
    USB_UTR_t       *ptr;
    USB_ER_t        err;
    USB_HCDREG_t reg;

    /* MCU Pin Setting (Refer to "12.1.1 MCU Pin Settings") */
    usb_mcu_setting();

    /* USB Driver Setting (Refer to "12.1.2 USB Driver Settings") */
    ptr         = &usb_ghmsc_utr;
    ptr->ip     = USB_HOST_USBIP_NUM;
    ptr->ipp    = R_usb_cstd_GetUsbIpAdr(ptr->ip);

    usb_cstd_ScheInit();
    R_usb_hstd_MgrOpen(ptr);
    R_usb_hstd_HcdOpen(ptr);

    reg.tpl = usb_ghhub_TPL;
    reg.pipetbl = usb_ghhub_DefEPTbl;
    R_usb_hhub_Registration(ptr, &reg); // Note

    usb_hapl_registration(ptr);
    R_usb_hmsc_driver_start(ptr);

    /* USB module starting and setting (Refer to "12.1.3 USB Module Startup and Settings")  */
    err = R_USB_Open( USB_IP0 );
    if(err != USB_E_OK)
    {
        /* Error processing */
    }
    R_usb_cstd_UsbIpInit(ptr, USB_HOST);

    /* Main routine */
    usb_hapl_mainloop();
}
```

[Note]
It is only necessary to call this function when the HUB will be used.

12.1.1 MCU Pin Settings

It is necessary to make USB I/O pin settings in order to use the USB controller. Sample USB I/O pin settings are shown below.

Table12-1 USB I/O Pin Settings for Host Operation

Pin Name	I/O	Description
USB_VBUSEN	Output	USB VBUS output enable pin
USB_OVRCURA	Input	USB overcurrent detection pin

[Note]

The USB HMSC FIT module and USB-BASIC-F/W FIT module do not make microcontroller pin settings. If pin settings are necessary, refer to the user's manual of the specific microcontroller and make pin settings to match the specifications of the evaluation board to be used.

12.1.2 USB Driver Settings

The USB driver settings consist of registering a task with the scheduler and registering class driver information for the USB-BASIC-F/W FIT module. The procedure for registering the class driver information and registering the task with the scheduler is described below.

1. Specify the IP number of the USB module as a member of the USB communication structure declared by the USB_UTR_t type.
2. Call R_usb_cstd_GetUsbIpAdr(), and set the base address of the USB register in the USB communication structure.
3. Call the USB-BASIC-F/W FIT module's API functions (R_usb_hstd_MgrOpen() and R_usb_hstd_HcdOpen()) to register the HCD task and MGR task with the scheduler.
4. Call the class driver function (R_usb_hhub_Registration()) to register the HUB task with the scheduler.(Note1)
5. After specifying the necessary information in the members of the class driver registration structure (USB_HCDREG_t), call R_usb_hstd_DriverRegistration() to register the class driver information for the USB-BASIC-F/W FIT module.
6. Call the class driver API function (R_usb_hmsc_driver_start()) to register the MSC task with the scheduler.

A sample of information specified in the structure declared by USB_HCDREG_t is shown below.

```
void usb_hapl_registration(USB_UTR_t *ptr)
{
    USB_HCDREG_t driver;   /* Structure for the class driver registration */

    /* Class Code which is defined in the USB specification setting*/
    driver.ifclass       = (uint16_t)USB_IFCLS_MAS;
    /* TPL setting */
    driver.tpl           = (uint16_t*)&usb_gapl_devicetpl; // Note 2
    /* Pipe information table setting */
    driver.pipetbl       = (uint16_t*)&usb_ghmsc_smp_eptbl [0]; // Note 3
    /* Set the function which is called when registering the class driver */
    driver.classinit     = (USB_CB_INFO_t)&usb_cstd_DummyFunction;
    /* Set the class check function which is called in the enumeration. */
    driver.classcheck = (USB_CB_CHECK_t)&R_usb_hmsc_ClassCheck;
    /* Set the function which is called when completing the enumeration */
    driver.devconfig   = (USB_CB_INFO_t)&usb_hmsc_smp_open;     // Note 4
    /* Set the function which is called when disconnecting USB device */
    driver.devdetach  = (USB_CB_INFO_t)&usb_hmsc_smp_close;     // Note 5
    /* Set the function which is called when changing the suspend state */
    driver.devsuspend = (USB_CB_INFO_t)&usb_cstd_DummyFunction;
    /* Set the function which is called when resuming from the suspend state */
    driver.devresume = (USB_CB_INFO_t)&usb_cstd_DummyFunction;

    /* Register the class driver information to HCD */
    R_usb_hstd_DriverRegistration(ptr, &driver);
}
```

[Note]

1. It is only necessary to call this function when the HUB will be used.

2. TPL(Target Peripheral List) need to be defined in the application program. Refer to USB Basic Firmware application note (Document No.R01AN2025EJ) about TPL.
3. The pipe information table need to be defined in the application program. Refer to USB Basic Firmware application note (Document No.R01AN2025EJ) about the pipe information table.

= Example of Pipe Information Table =

```
uint16_t usb_ghmsc_smp_eptbl[] =
{
        USB_NONE,
        /* TYPE/BFRE/DBLB/CNTMD/SHTNAK/DIR/EPNUM */
        USB_NONE | USB_BFREOFF | USB_DBLBON | USB_CNTMDON | USB_SHTNAKON
        | USB_NONE | USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_CUSE,

        USB_NONE,
        /* TYPE/BFRE/DBLB/CNTMD/SHTNAK/DIR/EPNUM */
        USB_NONE | USB_BFREOFF | USB_DBLBON | USB_CNTMDON | USB_SHTNAKON
        | USB_NONE | USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_NONE,
        USB_CUSE,

    /* Pipe end */
        USB_PDTBLEND,
};
```

4. Iinitialize global variables etc in the function which is set in this member.
5. Initialize the pipe information table etc in the function which is set in this member.

12.1.3 USB Module Startup and Settings

The USB module startup and setting procedure is described below.

1. Call R_USB_Open() to launch the USB module. This API function should only be called once, when making initial settings.
2. Call R_usb_cstd_UsbIpInit() to make initial settings for the USB module.

12.2 Processing by Main Routine

After the USB driver initial settings, call the scheduler (R_usb_cstd_Scheduler()) from the main routine of the application. Calling R_usb_cstd_Scheduler() from the main routine causes a check for events. If there is an event, a flag is set to inform the scheduler that an event has occurred. After calling R_usb_cstd_Scheduler(), call R_usb_cstd_CheckSchedule() to check for events. Also, it is necessary to run processing at regular intervals to get events and perform the appropriate processing.

```
void usb_hapl_mainloop(void)
{
   while(1)   // Main routine
   {
      R_usb_cstd_Scheduler();              // Confirming the event and getting (Note 1)

      if(USB_FLGSET == R_usb_cstd_CheckSchedule())   // Judgment whether the event is or not
      {
            R_usb_hstd_HcdTask((USB_VP_INT)0); // HCD task
            R_usb_hstd_MgrTask((USB_VP_INT)0); // MGR task
            R_usb_hhub_Task((USB_VP_INT)0);    // HUB task (Note 3)
            R_usb_hmsc_task((USB_VP_INT)0);    // MSC task
            R_usb_hmsc_StrgDriveTask();        // STRG driver task
      }                                                                    } (Note 2)
      hmsc_application();          // User application program
   }
}
```

[Note]

1. If, after getting an event with R_usb_cstd_Scheduler() and before running the corresponding processing, R_usb_cstd_Scheduler() is called again and gets another event, the first event is discarded. After getting an event, always call the corresponding task to perform the appropriate processing.
2. Be sure to describe these processes in the main loop for the application program.
3. It is only necessary to call this function when the HUB will be used.

12.3 Data Transfer

Use TFAT API when accessing to the media. Please refer to TFAT application note(Document No.R20AN0038EJ) about TFAT API.

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

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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 an MPU or MCU 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of 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. 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 should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended 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 systems; anti-disaster systems; anti-crime systems; and safety 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 (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable 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. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages 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 its 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 be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, 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 use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may 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 should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be 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, or if you have any other inquiries.

(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.

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, 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 Rd., Putuo District, Shanghai, China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, 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 906, 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 Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 3.0