Renesas USB MCU

Host Mass Storage Class Driver (HMSC) using USB Basic Mini Firmware

R01AN0624EJ0214
Rev. 2.14
Mar 28, 2016

Introduction

This document is an application note describing use of the Host Mass Storage Class Driver (HMSC) built using the USB Basic Mini Firmware.

Target Device

R8C/3MK, R8C/34K

This program can be used with other microcontrollers that have the same USB module as the above target devices. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

Contents

1. Overview

This document describes the use of the USB Host Mass Storage Class driver (HMSC). It also includes a simple example application that writes a file, then checks it repeatedly.

The software employs M3S-TFAT-Tiny (TFAT) and should be used in combination with the RX Family M3S-TFAT-Tiny FAT File System software (document No. R20AN0038EJ).

1.1 Functions and Features

The USB host mass storage class driver 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.

1.2 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. User's Manual: Hardware
5. USB Basic Mini Firmware Application Note (Document No. R01AN0326EJ)
6. RX Family M3S-TFAT-Tiny: FAT file system software (Document No.R20AN0038EJ)
7. Available from the Renesas Electronics Website
   - Renesas Electronics Website
     http://www.renesas.com/
   - USB Devices Page
     http://www.renesas.com/prod/usb/

1.3 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) FW
FSL	:	FAT File System Library
HCD	:	Host Control Driver of USB-BASIC-FW
HDCD	:	Host Device Class Driver (device driver and USB class driver)
hstd	:	Prefix of function and file for Host Basic (USB low level) F/W
HMSCD	:	Host Mass Storage Class Driver
HMSDD	:	Host Mass Storage Device Driver
non-OS	:	USB Basic Mini Firmware for non-OS based system
MGR	:	Peripheral device state manager of HCD
MSDCD	:	Mass Storage Device Class Driver (HMSDD + HMSCD)
PP	:	Pre-processed definition
Scheduler	:	Used to schedule functions, like a simplified OS.
Scheduler Macro	:	Used to call a scheduler function (non-OS)
SW1/SW2/SW3	:	User switches on the evaluation board.
Task	:	Processing unit
TFAT	:	Tiny FAT file system software for microcontrollers (M3S-TFAT-Tiny-RX)
USB	:	Universal Serial Bus
USB-BASIC-FW	:	USB Basic Mini Firmware for Renesas USB MCU

2. How to Register Class Driver

For the class driver layer to be used , it must be registered with the USB stack (HCD).

Please consult function usb_hapl_registration() in r_usb_hmsc_apl.c for registering a class driver with HCD.

For details, please refer to "How to register Host Class Driver" chapter of a Renesas USB device USB Basic Mini Firmware application note.

3. Software Configuration

3.1 Module 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 3-1 shows the HMSC software block diagram, with HDCD as the centerpiece. Table 3-1 describes each module.

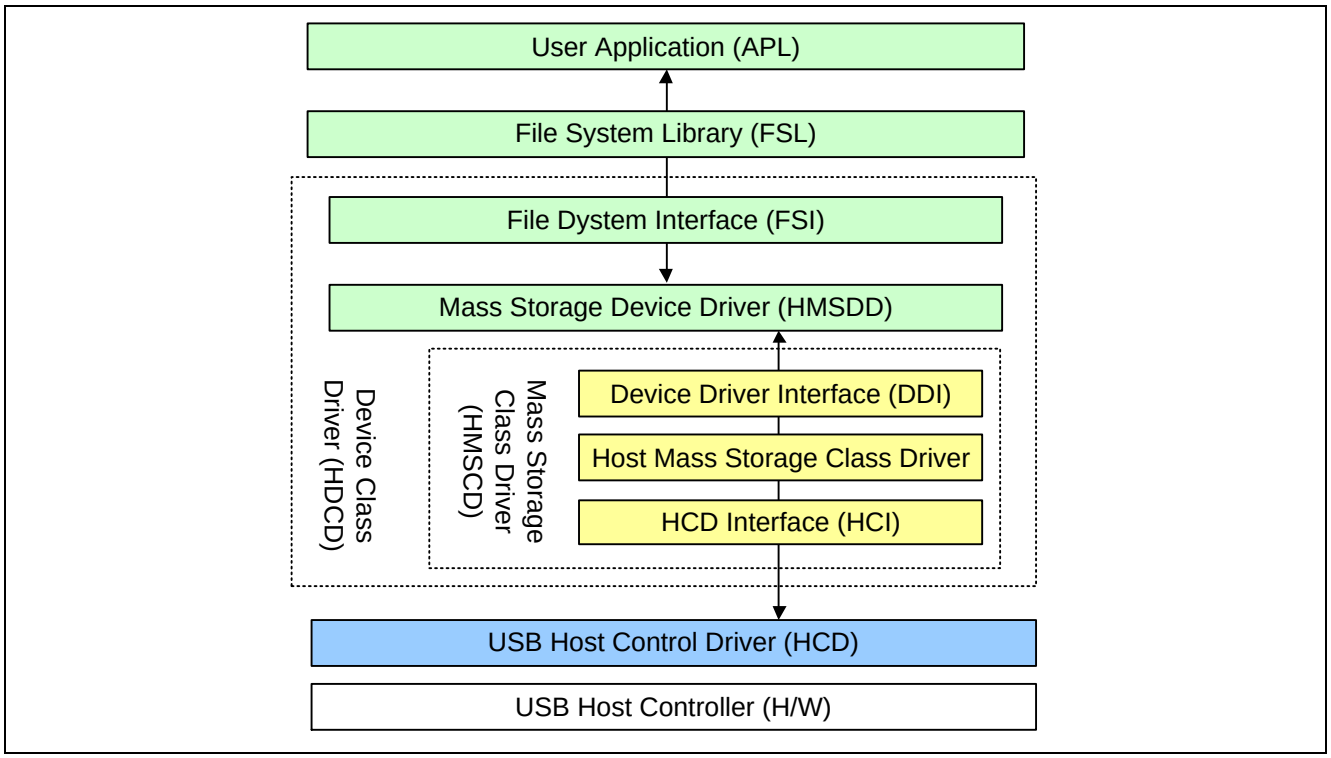

Figure 3-1 Software Block Diagram

Table 3-1 SW Modules..

Module	Description
APL	**Calls FSL functions to implement storage functionality. Created by the customer to match the system specifications.**
FSL	FAT file system library, with specifications defined by the user.
FSI	FSL-HMSDD interface functions. Translation of sector number and count, to logical block address and transfer data size. To be modified to match the file system type. FAT file system type is used in example code.
HMSDD	To be created (modified) by the customer to match the storage media. The FAT file system is used together with the ATAPI command set in the example code.
DDI	HMSDD-HMSCD interface functions. To be modified to match the storage media interface of HMSDD.The ATAPI command set is used in the sample code.
HMSCD	The USB host mass storage class driver. The BOT protocol is used to encapulate storage commands (ATAPI) and send 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 used in the example code.
HCI	HMSCD-HCD interface functions.
HCD	USB host hardware control driver.

3.2 File Structure

The folder structure in which the files are provided in HDCD is shown below.

```
[R8C]
  +(Project file etc)
    + src
    (HMSC  Code)
        +―――HMSC [Mass Storage Class driver and FAT]
        |       +―――src                  Host MSC driver
        |       +―――inc                  Host MSC header file
        |       +―――TFAT                 FAT File System library
     (Sample Code) [Class driver and sample application]
        +―――SmplMain
        |       +――― APL                 Sample application
    (Hardware Setting) [Hardware access layer; to initialize the MCU]
        +――― HwResource                   Hardware resource
    (USB-BASIC-FW code) [Common USB code that is used by all USB firmware]
        +―――USBSTDFW
            +――― inc                     Common header file of USB driver
            +――― src                     USB driver
```

3.2.1 File Structure

Table 3-2 File Structure

Folder Name	File Name	Description
APL	r_usb_hmsc_apl.c	Sample application
TFAT	r_tfat_drv_if.c	TFAT interface
HMSC	r_usb_hmsc_fsi.c	FSL interface for FAT
HMSC	r_usb_hstorage_driver.c	FAT based ATAPI command device driver (HMSDD)
HMSC	r_usb_hmsc_ddi.c	HMSDD interface for ATAPI (DDI)
HMSC	r_usb_hmsc_driver.c	USB class driver (host mass storage class)
HMSC	r_usb_hmsc_hci.c	HCD interface (HCI)

3.3 System Resources

3.3.1 Definitions

The system resources used by HMSC are listed below.

Table 3-3 Task Information

Function Name	Task ID	Priority	Description
usb_hstd_HcdTask	USB_HCD_TSK	USB_PRI_1	HCD task
usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	MGR task
usb_hhub_Task	USB_HUB_TSK	USB_PRI_3	HUBCD task
usb_hmsc_Task	USB_HMSC_TSK	USB_PRI_3	Mass storage task
usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	HMSDD task
usb_hmsc_SampleAplTask	USB_HMSCSMP_TSK	USB_PRI_4	APL task

Table 3-4 Mailbox Information

Mailbox Name	Using Task ID	Task Queue	Description
USB_HCD_MBX	USB_HCD_TSK	FIFO order	For HCD
USB_MGR_MBX	USB_MGR_TSK	FIFO order	For MGR
USB_HMSC_MBX	USB_HMSC_TSK	FIFO order	For HMSCD
USB_HSTRG_MBX	USB_HSTRG_TSK	FIFO order	For HMSDD
USB_HMSCSMP_MBX	USB_HMSCSMP_TSK	FIFO order	For Sample Application

Table 3-5 Memory Pool Mailbox Information

Memory Pool Name	Task Queue	Memory Block (note)	Description
USB_HCD_MPL	FIFO order	12bytes	For HCD
USB_MGR_MPL	FIFO order	12bytes	For MGR
USB_CLS_MPL	FIFO order	12bytes	For HDCD
USB_HMSCSMP_MPL	FIFO order	12bytes	For Sample application
USB_HMSC_MPL	FIFO order	12bytes	For HMSC

Note: The maximum number of memory blocks for the entire system is defined in USB_BLKMAX. The default value is 5.

3.4 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. shows the USB storage device access sequence. Note that APL is blocked until read or write is completed. No callbacks are implemented for read/write.

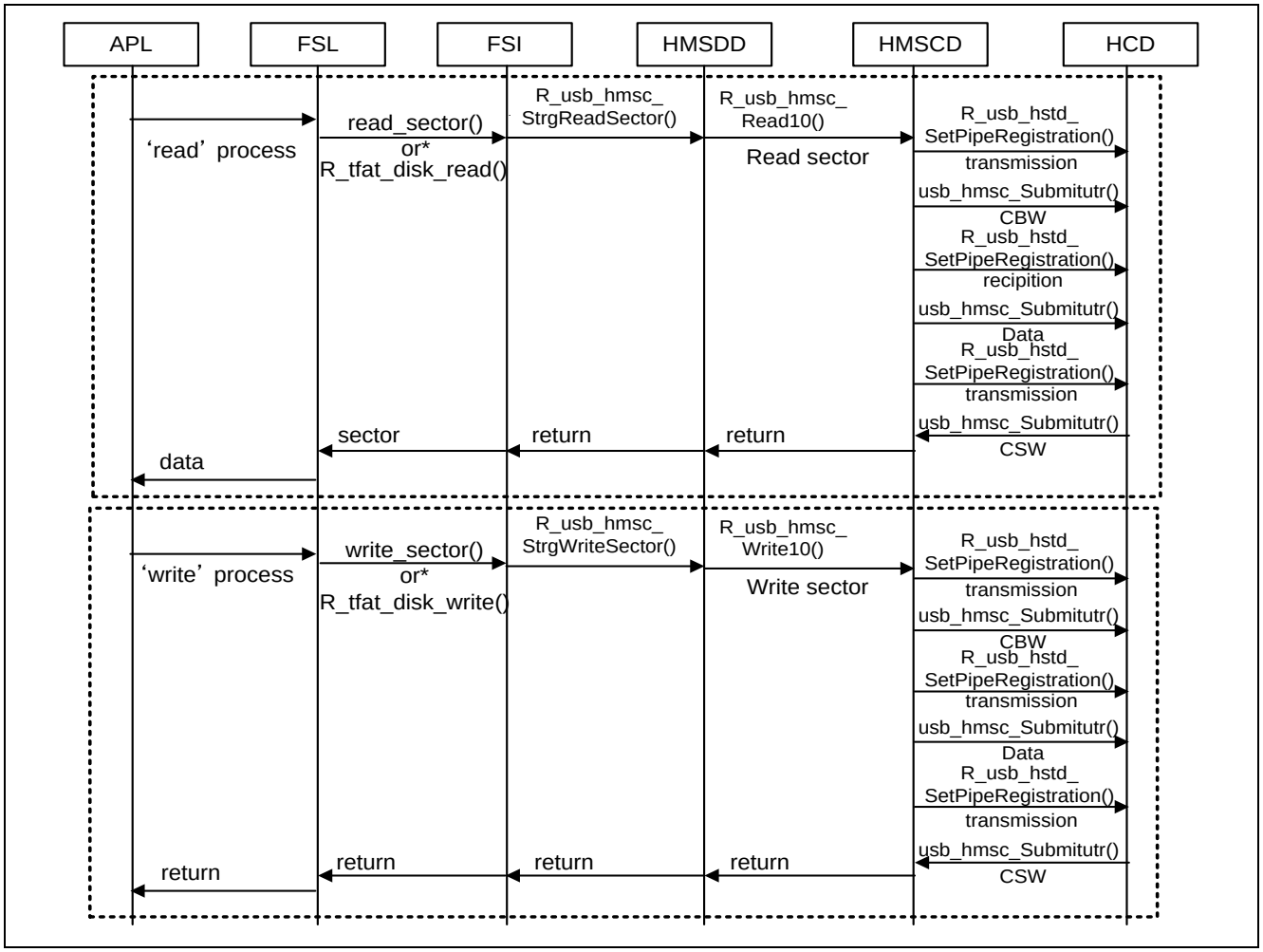

Figure 3-2 USB Storage Device Access Sequence

4. Sample Application Program (APL)

The following describes the HMSC sample application program (APL).

4.1 Application Program Function

The HMSC includes a sample application (APL) operated by switches on the board.

- Press SW2: After enumeration is complete, 512 bytes of character ‘a’ is written to a created file “hmscdemo.txt” on the USB mass storage function side. (Write one time).
- Press SW2 again to repeatedly read the data in”hmscdemo.txt”. (Read in loop). Each time the file is successfully read 100 times, LEDs 0~3 on the evaluation board are lit up in sequence.
- If an access error occurs on the connected device, the device will go to the SUSPEND state.

Operating mode and switch status are as follows.

Table 4-1 Operating Mode and Switch Status

Switch Label	Operation	Switch Number used on Evaluation Board
File operation switch	Press SW on the evaluation board, once, to write file to media. Press the switch again to continuously repeat the read process, until the switch is pressed again.	SW2

4.2 Operating Environment

Figure 4-1 shows operating environment example for the software.

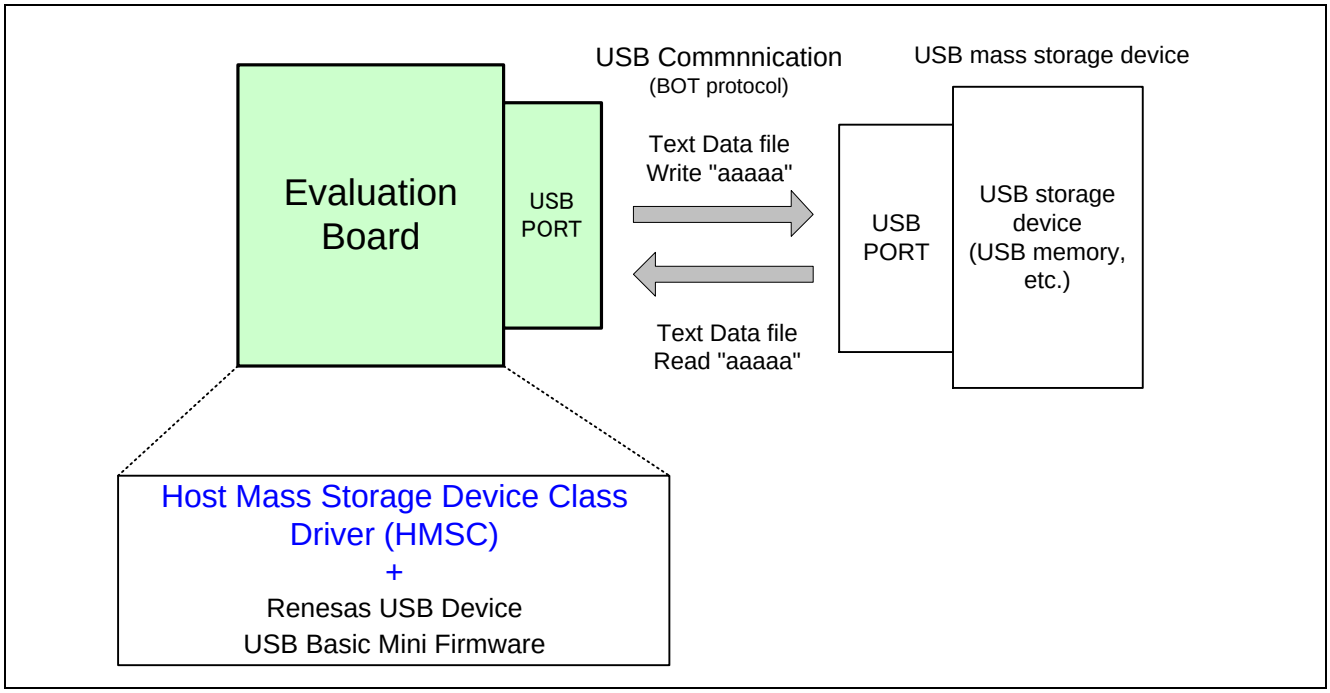

Figure 4-1 Operating Environment Example

4.3 APL Functions

Table 4-2 shows the list of APL functions

Table 4-2 APL functions

	Function Name	Description
1	usb_cstd_task_start()	Starts tasks
2	usb_apl_task_switch()	Switches tasks
3	usb_hapl_task_start()	Starts application task
4	usb_hapl_registration()	Registers HMSC driver with USB stack
5	usb_hmsc_PrAplTitle()	Display application title
6	usb_hmsc_AplClear()	Clears application (APL)
7	usb_shmsc_DemoStateChange()	Change demo state
8	usb_hmsc_SampleAplTask()	Application task
9	usb_hmsc_SmplMessageSend()	Message transmission to APL task
10	usb_hmsc_SampleAplSpecifiedPath()	APL selection

4.4 APL Processing Overview

4.4.1 Application Level Processing Flow

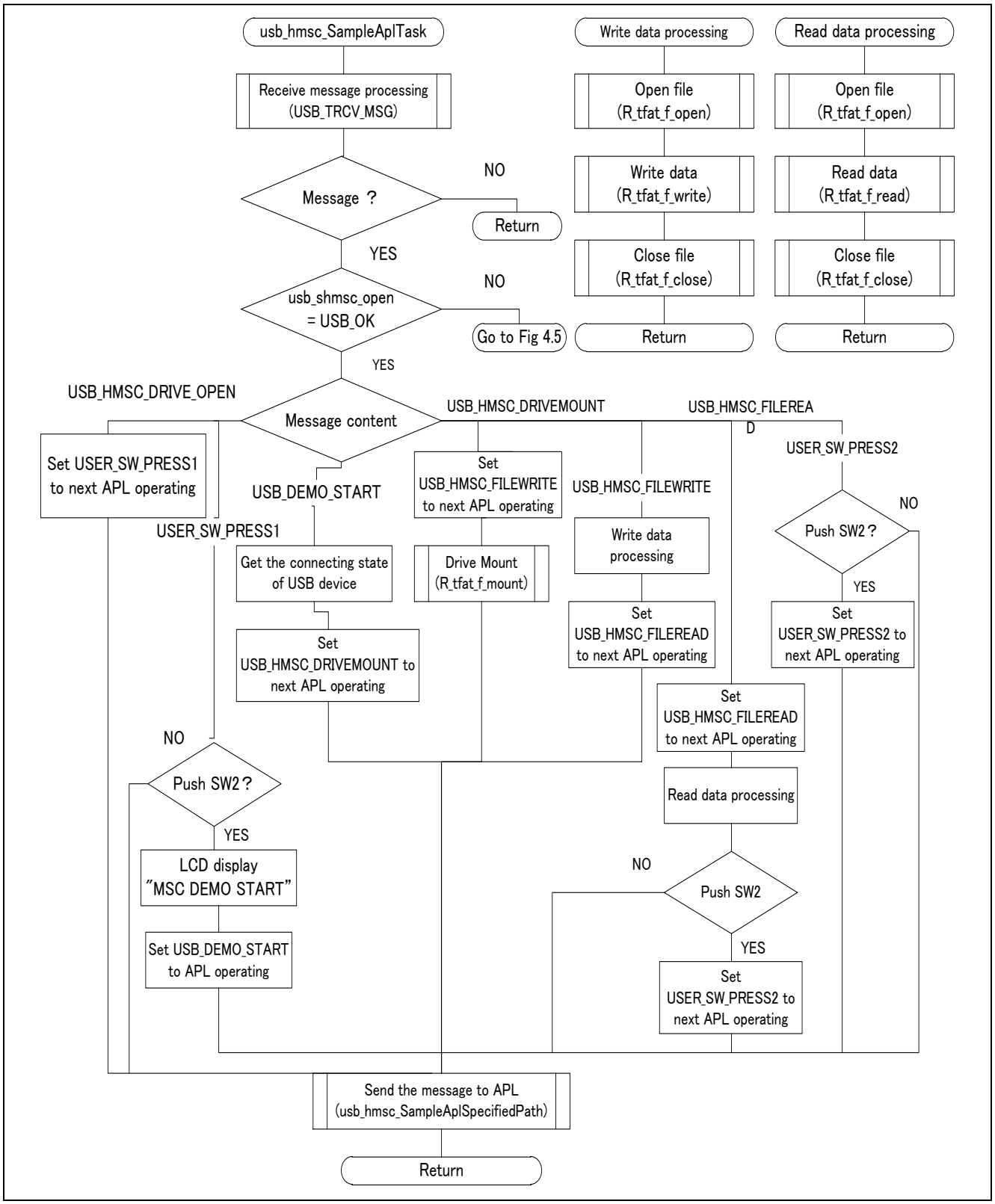

Figure 4-2 Application Processing Flow

5. File System Interface (FSI)

5.1 Functions and Features

FSI (File System Interface) converts sector number to logical block address and sector count to transfer data size.

Modify (modify) FSL to match the file system type. The FAT file system type is used in sample code..

5.2 FSI API

Table 5-1 lists the functions of the sample FSI.

Table 5-1 FSI Functions

	Function Name		Description
	Using TFAT	**Other File System**	
1	*R_tfat_disk_read*	*read_sector*	Read data
2	*R_tfat_disk_write*	*write_sector*	Write data

[Note]

R_usb_hmsc_WaitLoop function is called to wait for completion of DATA READ / DATA WRITE in R_tfat_disk_read/R_tfat_disk_write function.

6. Host Mass Storage Device Driver (HMSDD)

HMSDD, the “device driver”, is the application’s gateway to use HMSCD, the “class driver”.
The TFAT file system is used together with the ATAPI command set in the sample code.
A storage device is selected addording to given drive number.

6.1 Functions and Features

The drive start number of the USB storage device should be specified by the user with USB_DRIVE in the file r_usb_hmsc_define.h. Check the following items to enable proper storage commands.

a. The drive number is less than the total drive count.
b. The drive number is equal to or greater than USB_DRIVE.
c. The storage device is connected to a USB port.
d. The unit number can be searched by using the drive number.

6.1.1 Limitations

(a). Drive number

— Up to eight USB storage devices (up to one USB storage device in case of TFAT) can be connected (the number of times HMSCD can be registered).
— USB storage devices with up to four units (up to one unit in case of TFAT) can be connected.
— USB storage devices with up to 10 partitions (up to one partition in case of TFAT) can be connected.
(The max. unit count and max. partition count can be changed in the header file.)
— A device that does not respond to the GetMaxUnit request operates as a single-unit device.

(b). Drive features

— The only supported device type is direct access device (checked using the INQUIRY command).
— 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.
— The boot recorder **partition type** is determined based on the following value.
0x05/0x0F: Extended partition
0x01: Master boot recorder, FAT12
0x04/0x06/0x0E: Master boot recorder, FAT16
0x0B/0x0C: Master boot recorder, FAT32
Other than the above: Treated as partition boot recorder, JMP code, etc.
— Partition information (PBR/MBR) is obtained by searching method.
PBR: Total of start sector numbers in partition entries until PBR is found
MBR: Total of start sector numbers in partition entries until MBR is found + start sector number of MBR partition entry
— Some devices may be unable to be connected (because they are not recognized as storage devices).

6.2 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 6-1.

Table 6-1 LUN Field Settings

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

6.3 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

6.4 HMSDD API Function

Table 6-2 lists the function of HMSDD.

Table 6-2 HMSDD Functions

	Function Name	Description
1.	R_usb_hmsc_StrgDriveOpen()	Mounts drive.
2.	R_usb_hmsc_StrgDriveClose()	Unmounts drive.
3.	R_usb_hmsc_StrgReadSector()	Reads data.
4.	R_usb_hmsc_StrgWriteSector()	Writes data.
5.	R_usb_hmsc_StrgDriveSearch()	Reads partition information and searches for drive.

R_usb_hmsc_StrgDriveOpen

Mounts drive

Format

usb_er_t R_usb_hmsc_StrgDriveOpen(uint16_t side)

Argument

side Drive number

Return Value

USB_DONE Normal end

USB_ERROR Error end

Description

Mounts the drive specified by the argument. (The sector information is initialized.)

An error response occurs in the following cases.

1. When the partition number could not be found
2. When the drive information could not be read successfully from the storage device

Notes

1. Please call this function from the class driver or the FAT library I/F function..

Example

```
int dev_open(USB_UTR_t *ptr, int side)
{
            :
    /* Mounts Drive */
    error = R_usb_hmsc_StrgDriveOpen((uint16_t)side);

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

R_usb_hmsc_StrgDriveClose

Unmounts drive

Format

usb_er_t R_usb_hmsc_StrgDriveClose(uint16_t side)

Argument

side Drive number

Return Value

USB_DONE Normal end

USB_ERROR Error end

Description

Unmounts the drive specified by the argument. (The sector information is cleared.)

An error response occurs in the following cases.

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

Notes

1. Please call this function from the class driver or the FAT library I/F function..

Example

```
int dev_close(int side)
{
                :
    /* close function */
    error = R_usb_hmsc_StrgDriveClose((uint16_t)side);

   /* Condition compilation by the difference of the operating system */
    if( error == USB_ERROR )
    {
             :
        return (-1);
    }
    return (0);
}
```

R_usb_hmsc_StrgReadSector

Read Sector Information

Format

usb_er_t R_usb_hmsc_StrgReadSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

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 specified sector data of specified drive.

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.

Example

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

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

R_usb_hmsc_StrgWriteSector

Write Sector Information

Format

usb_er_t	R_usb_hmsc_StrgWriteSector(uint16_t side, uint8_t *buff, uint32_t secno, uint16_t seccnt, uint32_t trans_byte)

Argument

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

Write specified sector data of specified drive.

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.

Example

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

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

R_usb_hmsc_StrgDriveSeach

Reads partition information and searches for drive.

Format

uint16_t R_usb_hmsc_StrgDriveSearch(uint16_t addr)

Argument

addr	Device address

Return Value

USB_DONE	Drive detected
USB_ERROR	Drive not detected

Description

This function checks the storage device unit count and issues an INQUIRY command to each storage device unit to confirm that access to the device is possible.
The function reads the logical block address = 0x00 area of all accessible units and increments the drive count for each.

Notes

1. This function only determines the number of accessible drives. It does not mount a drive..

Example

```
void usb_hmsc_DriveOpen( uint16_t addr, uint16_t data2)
{
      uin16_t err;
                  :
      err = R_usb_hmsc_StrgDriveSearch( addr );
                  :
}
```

6.5 Drive Management

HMSDD sets USB_DRIVE as the initial drive number of a connected storage device. HMSDD also supports connection of storage devices with multiple units and multiple drives. It is only necessary to specify the drive number from the application to access the relevant partition block of any unit. A storage device may have up to four units, and each unit may have up to 10 partition blocks. The unit count and partition block count can be changed by specifying new values in the header file.

For example, the drives are managed as shown below when USB_DRIVE is set to 1, the USB storage device has three units, unit 0 has three partitions (total of three partitions including extensions), unit 1 has no media, and unit 2 has two partitions.

Table 6-3 Drive Numbers

Unit	Partition	Drive	Remarks
0	0	1 (B)	FAT32
0	1		Not classified (no drive)
0	2		Extended partition (continuous information read)
0	2 to 0	2 (C)	FAT32 (continuous information of partition 2)
0	2 to 0		Not classified (no drive)
0	3	3 (D)	FAT32
1			No media
2	0	4 (E)	FAT16
2	1	5 (F)	FAT16

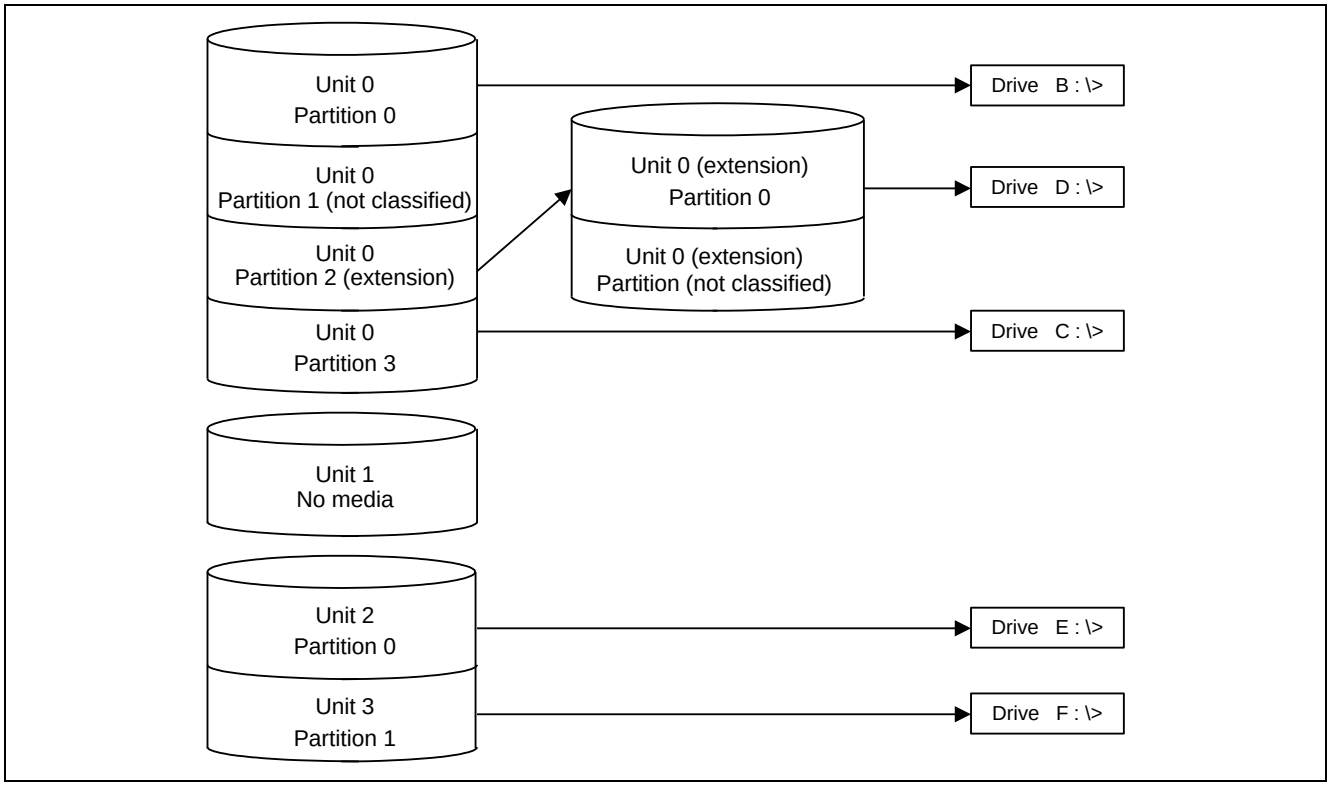

Figure 6-1 Drive Management

7. USB Mass Storage Class Driver (HMSCD)

7.1 Functions and Features

HMSCD executes storage command communication, if the USB storage device is ready to operate.

The BOT protocol is used, which encapsulates the storage commands (ATAPI, see below) as they pass through USB.

MSCD comprises three layers: HMSDD interface (DDI functions), HCD interface (HCI functions), and HMSCD itself.

HMSCD encapsulates the storage commands necessary for accessing USB storage devices. 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.

7.2 Issuing Requests to HMSCD

The interface functions described below (Table 7-4 and Table 7-5) 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...

7.3 HMSCD Structures

Table 7-1and Table 7-2 show the contents of the HMSCD structures. For more on the BOT protocol, see Related Documents 3.

Table 7-1 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[49]	CBWCB	Command block

Table 7-2 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[51]	Dummy	Even number adjustment

7.4 USB Host Control Driver Interface (HCI) Functions

HCI is the interface function between HMSCD and HCD.

When using an RTOS, until the drive is mounted, HCI uses the HCD class area to send and receive messages. When mounting of the drive is finished, it uses the HMSCD dedicated area. In non-OS operations, HCI uses the HMSCD area to send and receive messages. Note that two devices cannot be enumerated (or accessed) simultaneously.

7.5 Device Driver Interface (DDI) Functions

DDI is the interface function between HMSDD and HMSCD. The ATAPI command set is used in sample code, but this could be changed by the user.

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

7.6 HMSC API

Application programming interface description for HMSCD.

Table 7-3 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

Table 7-4 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 7-5 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.

R_usb_hmsc_SetDevSts

Sets HMSCD Operation State

Format

uint16_t R_usb_hmsc_SetDevSts(uint16_t data)

Argument

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

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

The return value is always USB_DONE.

Example

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

R_usb_hmsc_GetDevSts

Returns HMSCD operation state

Format

uint16_t R_usb_hmsc_GetDevSts(void)

Argument

— —

Return Value

USB_HMSC_DEV_ATT（Attach）

USB_HMSC_DEV_DET（Detach）

Description

Returns the HMSCD operation state.

Note

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

Example

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

R_usb_hmsc_Information

Gets pipe number used for data transmit / receive

Format

uint16_t R_usb_hmsc_Information(uint16_t pipe_offset)

Argument

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

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

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]);
    :
}
```

R_usb_hmsc_Task

The 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.

Please refer to "7.2.2 Operation of Host Sample Program" in the Basic Mini 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())
  {
       :
    R_usb_hmsc_Task();
       :
  }
    :
}
```

R_usb_hmsc_GetMaxUnit

Issue GetMaxLUN request.

Format

usb_er_t R_usb_hmsc_GetMaxUnit(uint16_t addr, usb_cb_t complete)

Argument

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.

Note

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

Example

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

R_usb_hmsc_MassStorageReset

Issue Mass Storage Reset request.

Format

usb_er_t R_usb_hmsc_MassStorageReset(uint16_t drvnum, usb_cb_t complete)

Argument

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.

Note

1. Please call this function from the user application program or the class driver.
2. Please refer to USB Basic Mini FW application note about usb_cb_t structure.

Example

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

R_usb_hmsc_ClearStall

Cancel pipe STALL state

Format

void R_usb_hmsc_ClearStall(uint16_t type, uint16_t msgnum , usb_cb_t complete)

Argument

type	IN/OUT communication direction
msgnum	Driver number
complete	Callback function

Return Value

—

Description

This function releases the pipe from the STALL state.

Note

1. Please call this function from the user application program or the class driver.
2. Please refer to USB Basic Mini FW application note about usb_cb_t structure.

Example

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

R_usb_hmsc_Initialized

Initialize Host Mass Storage class

Format

void R_usb_hmsc_Initialized(void)

Argument

— —

Return Value

— —

Description

Initializes an internal variable of HMSCD.

Note

—

Example

```
USB_STATIC void usb_hmsc_device_state(uint16_t data, uint16_t state)
{
    switch( state )
    {
              :
        case USB_STS_DEFAULT:
            R_usb_hmsc_Initialized();
        break;
              :
     }
}    /* eof usb_hsmpl_device_state() */
```

R_usb_hmsc_ClassCheck

Compare connected device with interface descriptor

Format

void	R_usb_hmsc_ClassCheck(uint16_t **table)

Argument

**table	Pointer to the area which the device information is stored in.

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

This function is registered using usb_hapl_registration() as a callback function.

The maximum connectable storage device count is defined by USB_MAXSTRAG
(Refer to r_usb_hmsc_define.h)

The maximum operable drive count is defined by USB_MAXDRIV(Refer to r_usb_hmsc_define.h)

Example

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

R_usb_hmsc_Read10

Issue a READ10 command

Format

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

Argument

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.

Example

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

R_usb_hmsc_Write10

Issue a WRITE10 command

Format

uint16_t R_usb_hmsc_Write10(uint16_t side, uint8_t *buff, uint32_t secno,
uint16_t seccnt, uint32_t trans_byte)

Argument

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.

Example

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

R_usb_hmsc_DriveSpeed

Check drive communication speed

Format

uint16_t R_usb_hmsc_DriveSpeed(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.

Example

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

R_usb_hmsc_Release

Release Host Mass Storage Class Driver

Format

void	R_usb_hmsc_Release(void)

Argument

—	—

Return Value

—	—

Description

Cancel HMSCD registration from HCD.

Note

Please call this function from the application program or the class driver,

Example

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

8. Target Peripheral List (TPL)

When acting as Host, a class driver is not required to support operation with all types of USB peripherals of the class. It is up to the manufacturer of the host to determine what peripherals to support and provide a list of those peripherals. This is the called the device's "Targeted Peripheral List (TPL)".

TPL is composed of VID and PID. So as not to check it by VID (/PID), it is specified USB_NOVENDOR (/USB_NOPRODUCT). Please describe TPL which is supported in usb_gapl_devicetpl[] is described in usb_hmsc_apl.c file.

9. Limitations

The following limitations apply to HMSCD.

1. Structures are composed of members of different types (Depending on the compiler, the address alignment of the structure members may be shifted).
2. When connecting multiple storage devices, HMSCD must be registered the same number of times as the number of connectable devices.
3. Additional limitations apply to HMSDD. See section 6 for details.
4. There is no guarantee that a storage device meeting the functional requirements listed in section 6 can be successfully connected.
5. TFAT supports only one device; see TFAT instruction manual for details.

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
2.10	Aug 1 2013	—	First edition USB Mini FW..
2.11	—	—	Missing number.
2.12	Mar 31 2014	—	R8C is supported. Error is fixed.
2.13	Mar 16, 2015	—	RX111 is deleted from Target Device.
2.14	Mar 28, 2016		Upgrading of this USB driver by upgrading of "USB Basic Mini Firmware (R01AN0326)".

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.
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, HALII 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 5.0