RZ/T1 Group

USB Host Mass Storage Class Driver (HMSC)

R01AN2634EJ0130
Rev.1.30
Dec 07, 2017

Introduction

This application note describes USB Host Mass Storage Class Driver (HMSC). This module performs hardware control of USB communication. It is referred to below as the USB-BASIC-F/W.
The sample program of this application note is created based on "RZ/T1 group Initial Settings Rev.1.30". Please refer to "RZ/T1 group Initial Settings application note (R01AN2554EJ0130)" about operating environment.

Target Device

RZ/T1 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 Class Definitions for Human Interface Devices Version 1.1
3. HID Usage Tables Version 1.1
   http://www.usb.org/developers/docs/
4. RZ/T1 Group User's Manual: Hardware (Document No.R01UH0483EJ0130)
5. RZ/T1 Group Initial Settings (Document No.R01AN2554EJ0130)
6. USB Host Basic Firmware (Document No.R01AN2633EJ0130)

Renesas Electronics Website
http://www.renesas.com/
USB Device Page
http://www.renesas.com/prod/usb/
FatFs Website
http://elm-chan.org/fsw/ff/00index_e.html

Contents

1. Overview

The HMSC, when used in combination with the USB-BASIC-F/W, operates as a USB host mass storage class driver. HMSC comprises a USB mass storage class bulk-only transport (BOT) protocol. When combined with a file system, it enables communication with a BOT-compatible USB storage device. HMSC has created on the assumption that the use of FatFs in the file system.

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) and SCSI USB mass storage subclass.
- Multiple 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. Only supported for Logical Unit Number 0 (LUN0).
3. USB storage devices with a sector size of 512 bytes can be connected.
4. A device that does not respond to the READ_CAPACITY command operates as a device with a sector size of 512 bytes.

Terms and Abbreviations

Terms and abbreviations used in this document are listed below.

APL	:	Application program
ATAPI	:	AT Attachment Packet Interface
BOT	:	Mass storage class Bulk Only Transport
CBW	:	Command Block Wrapper
CSW	:	Command Status Wrapper
FSI	:	File System Interface
HCD	:	Host Control Driver of USB-BASIC-F/W
HMSC	:	Host Mass Storage Class driver
HMSCD	:	Host Mass Storage Class Driver unit
HMSDD	:	Host Mass Storage Device Driver
HUBCD	:	Hub Class Smple Driver
LUN	:	Logical Unit Number
LBA	:	Logical Block Address
MGR	:	Peripheral device state manager of HCD
SCSI	:	Small Computer System Interface
Scheduler	:	Used to schedule functions, like a simplified OS.
Task	:	Processing unit
USB-BASIC-F/W	:	USB basic firmware for RZ/T1 Group (non-OS)
USB	:	Universal Serial Bus

2. Software Configuration

Figure 2-1 shows HMSC software block diagram. Table 2-1 describes each module.

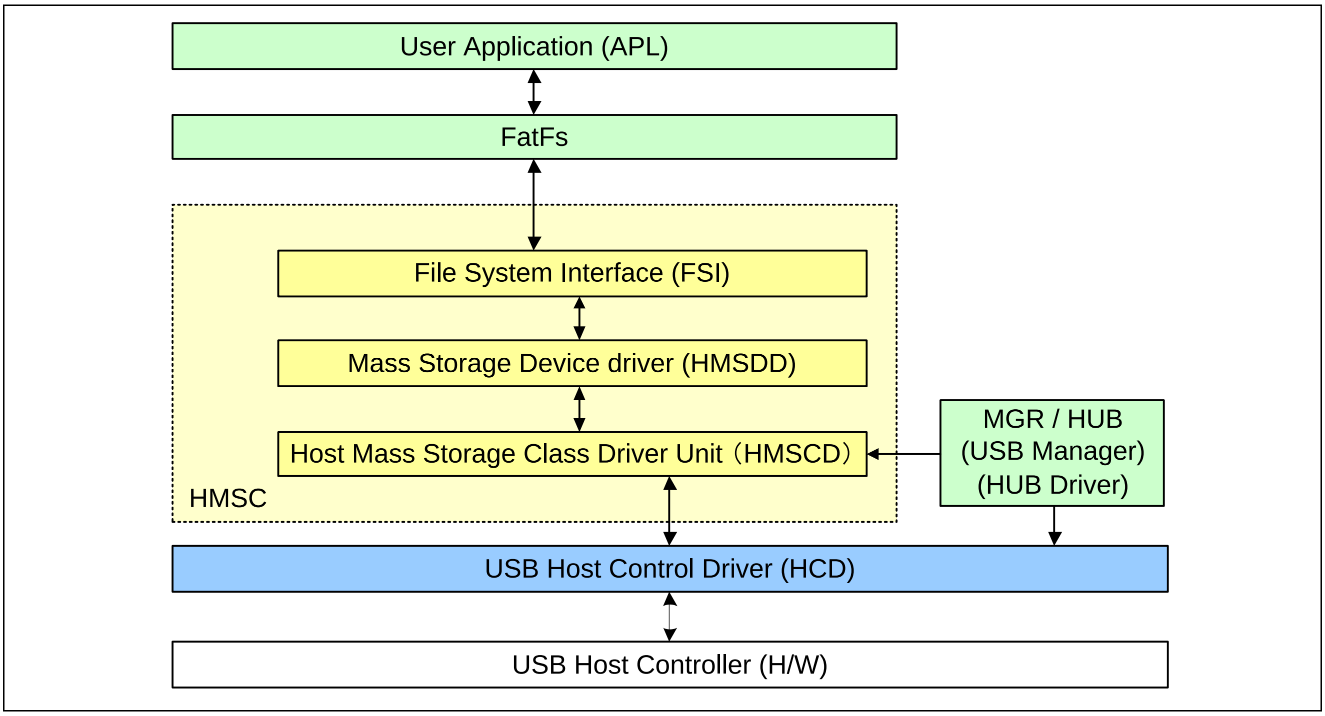


Figure 2-1 Software Block Diagram

Table 2-1 Module

Module	Description
APL	User application program. (Please prepare for your system)
FatFs	Generic FAT file system module. (Refer to 5. Importing procedure of FatFs)
HMSC	Host Mass Storage Class Driver Consists in the FSI and HMSDD and HMSCD.
FSI	File System Interface. (Provice a sample corresponding to the FatFs)
HMSDD	Host Mass Storage Device Driver - Searching USB storage device - Accessing USB storage device
HMSCD	Host Mass Storage Class Driver Unit - check MSC device - HCD communication required by BOT protocol - Management of BOT sequence
MGR/HUB	USB Manager / HUB class driver. (USB-BASIC-F/W) Enumerates the connected devices and starts HMSC. Performs device state management.
HCD	USB host Hardware Control Driver. (USB-BASIC-F/W)

3. Host Mass Storage Class Driver (HMSC)

HMSC has combined with the USB-BASIC-F/W and constitutes in FSI and HMSDD and HMSCD.
This chapter explains the functions of HMSC.

3.1 Class Request

Table 3-1 shows the class request HMSC supports.

Table 3-1 Class Request

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

3.2 Storage Command

Table 3-2 shows the storage command HMSC supports.

Table 3-2 Storage Command

Command	Code	Description	Supported
TEST_UNIT_READY	0x00	Verify state of peripheral device	YES
REQUEST_SENSE	0x03	Obtain peripheral device state	YES
FORMAT_UNIT	0x04	Format logic unit	NO
INQUIRY	0x12	Obtain logic unit parameter information	YES
MODE_SELECT6	0x15	Set parameters	YES
MODE_SENSE6	0x1A	Obtain logic unit parameters	NO
START_STOP_UNIT	0x1B	Enable/disable logic unit access	NO
PREVENT_ALLOW	0x1E	Enable/disable media removal	YES
READ_FORMAT_CAPACITY	0x23	Obtain format capacity	YES
READ_CAPACITY	0x25	Obtain logic unit capacity information	YES
READ10	0x28	Read data	YES
WRITE10	0x2A	Write data	YES
SEEK	0x2B	Move to logic block address	NO
WRITE_AND_VERIFY	0x2E	Write and verify data	NO
VERIFY10	0x2F	Verify data	NO
MODE_SELECT10	0x55	Set parameters	NO
MODE_SENSE10	0x5A	Obtain logic unit parameters	YES

3.3 Checking USB storage devices

USB-BASIC-F/W notify HMSC the information whch obtained from GET_DESCRIPTOR request at the time of enumeration by the callback function. HMSCD offers API function *R_usb_hmsc_ClassCheck()* for registration as this callback function. *R_usb_hmsc_ClassCheck()* analyzes the descriptor information, and notifies the collation result by the API function *R_usb_hstd_ReturnEnuMGR()* of USB-BASIC-F/W. If there is no problem in the result, USB-BASIC-F/W complete the enumeration.

3.4 Searching USB Storage Devices

After the enumeration is completed, HMSC is able to make the search process of USB storage devices by calling the API function *R_usb_hmsc_StrgDriveSearch()* of HMSDD. Completion of this process is notified by the callback function which registered at the time of *R_usb_hmsc_StrgDriveSearch()* call.
HMSC operate as a unit number 0 regardless of the response result of GetMaxLUN. Therefore, HMSC support LUN0 only.

Figure 3-1 shows the USB storage device search sequence.

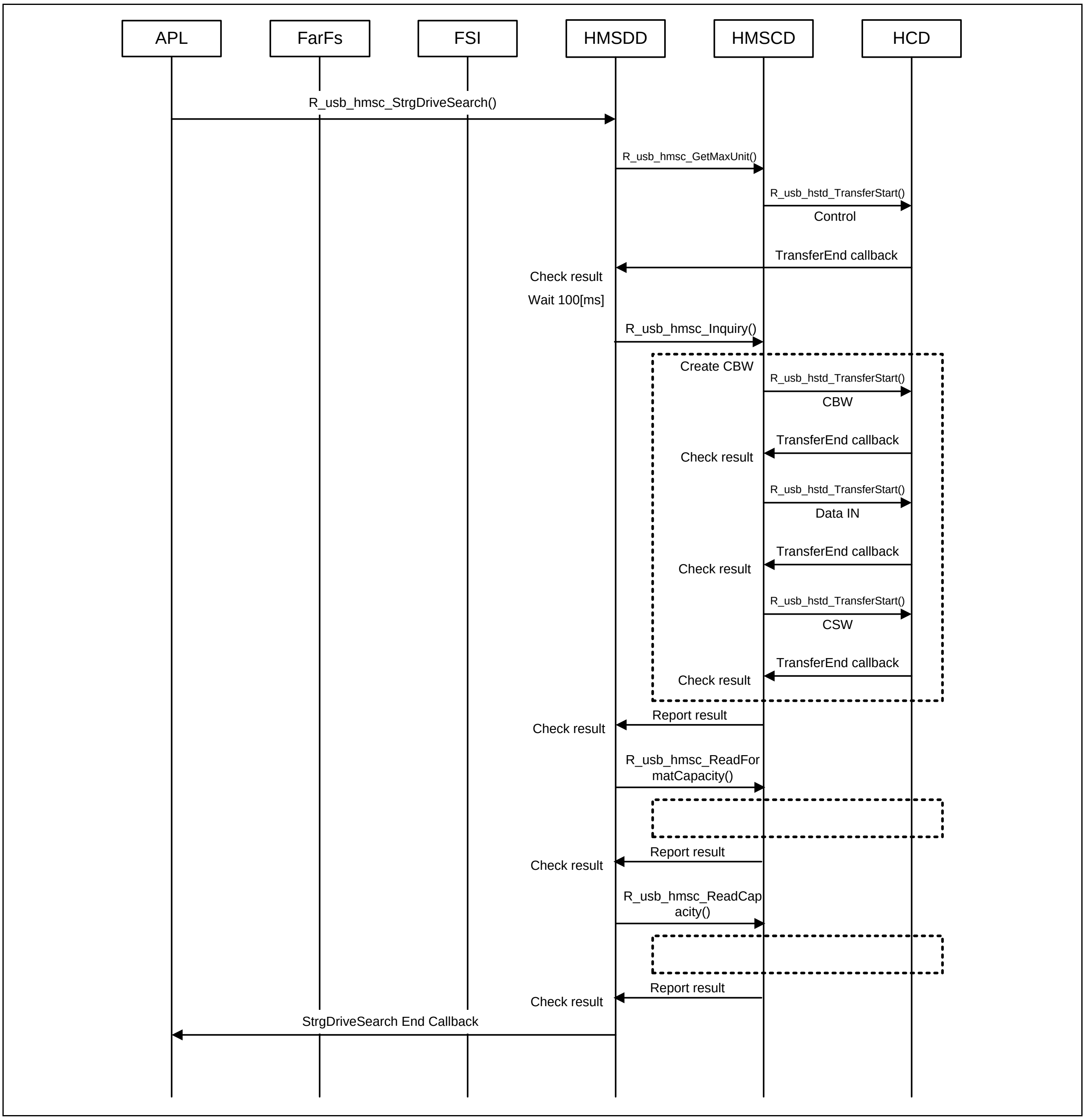


Figure 3-1 USB Storage Device Search Sequence

3.5 Accessing USB Storage device

After the information acquisition is complete, be able to access the USB storage devices in the API function of FatFs.
When call the API function of FatFs, FSI is called in the file system processing.
HMSDD calls the API function of HMSCD in accordance with the processing.
HMSDC issue a class request and create a USB packet in accordance with BOT protocol.
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.

Figure 3-2 shows the USB storage device access sequence.

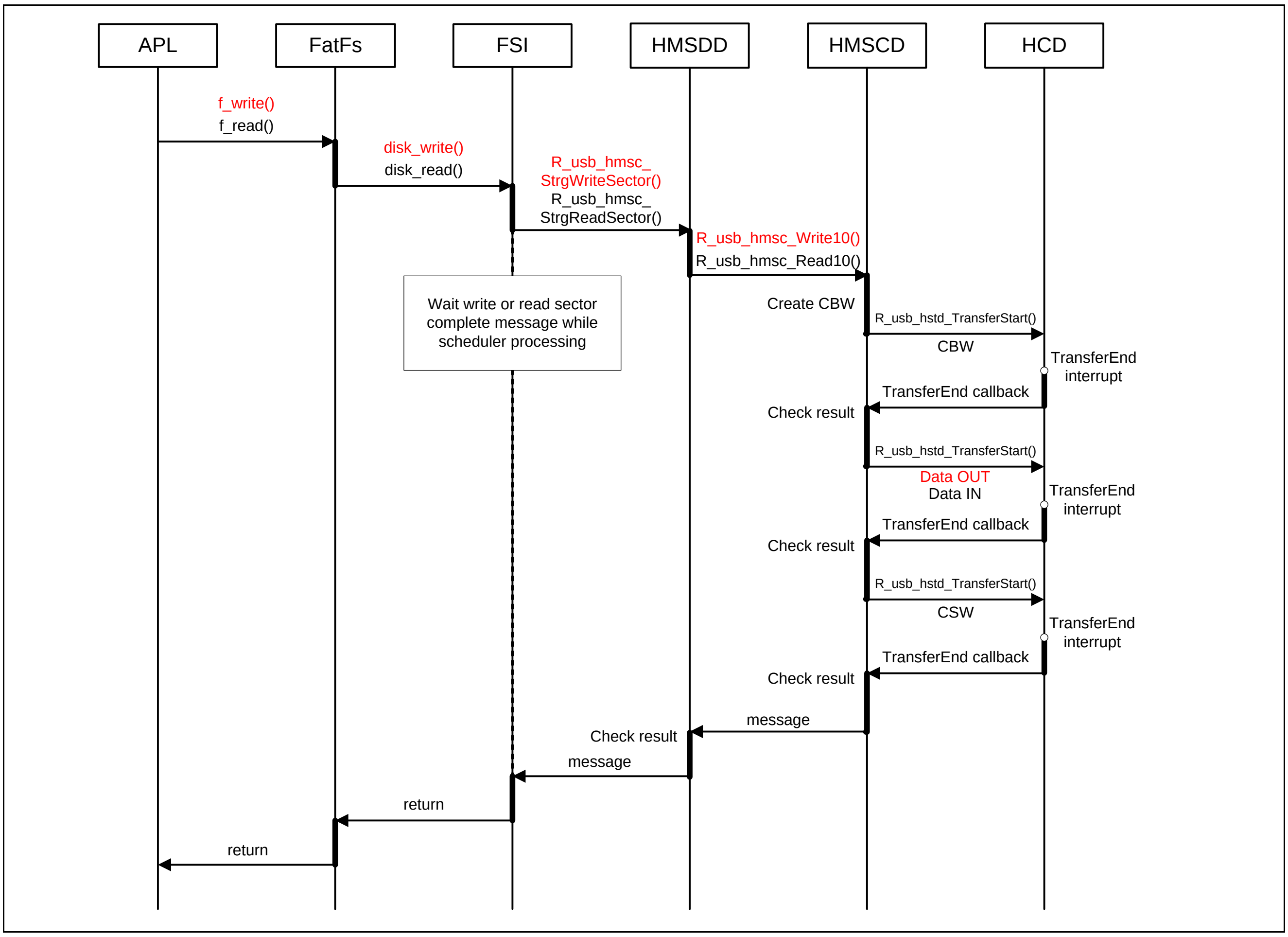


Figure 3-2 USB Storage Device Access Sequence

3.6 API functions of HMSCD

API functions of HMSCD are provided primarily for HMSDD.
Normally, the function to be used in the application are three below.

R_usb_hmsc_Task ()
R_usb_hmsc_driver_start ()
R_usb_hmsc_Class_Check ().

Table 3-3 lists the function of HMSCD.

Table 3-3 HMSCD Functions

Function Name	Description
R_usb_hmsc_Task()	Host Mass Storage Class Task
R_usb_hmsc_driver_start()	HMSC driver start processing.
R_usb_hmsc_ClassCheck()	Checks the descriptor table of the connected device to determine whether or not HMSCD can operate.
R_usb_hmsc_GetDevSts()	Returns HMSCD operation state.
R_usb_hmsc_Read10()	Issues the READ10 command.
R_usb_hmsc_Write10()	Issues the WRITE10 command.
R_usb_hmsc_GetMaxUnit()	GetMaxLUN request execution.
R_usb_hmsc_MassStorageReset()	MassStorageReset request execution.
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

3.6.1 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.

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_hstd_MgrTask();
    R_usb_hhub_Task();
    R_usb_hmsc_Task();
  }
}
```

3.6.2 R_usb_hmsc_driver_start

HMSC driver start

Format

void R_usb_hmsc_driver_start(void)

Argument

— —

Return Value

— —

Description

This function sets the priority of HMSC driver task.

Note

Call this function from the user application program during initialization.

Example

```
void usb_hstd_task_start( void )
{
  R_usb_hmsc_driver_start();     /* Host Class Driver Task Start Setting */
}
```

3.6.3 R_usb_hmsc_ClassCheck

Check descriptor

Format

void R_usb_hmsc_ClassCheck(uint16_t **table)

Argument

**table	Device information table [0] : Device Descriptor [1] : Configuration Descriptor [2] : Interface Descriptor [3] : Descriptor Check Result [4] : HUB Classification [5] : Port Number [6] : Transmission Speed [7] : Device Address

Return Value

— —

Description

This is a class driver registration function. It is registered to the driver registration structure member *classcheck*, as a callback function during HMSC registration at startup and called when a configuration descriptor is received during enumeration.
This function references the endpoint descriptor in the peripheral device configuration descriptor, then edits the pipe information table and checks the pipe information of the pipes to be used.

Note

—

Example

```
void  usb_hapl_registration(USB_UTR_t *ptr)
{
  USB_HCDREG_t driver;

  /* Driver check */
  driver.classcheck   = &R_usb_hmsc_ClassCheck;
}
```

3.6.4 R_usb_hmsc_GetDevSts

Returns HMSCD operation state

Format

uint16_t R_usb_hmsc_GetDevSts(uint16_t side)

Argument

side Drive number

Return Value

usb_ghmsc_AttSts USB_HMSC_DEV_ATT（Attach）
USB_HMSC_DEV_DET（Detach）

Description

Returns HMSCD operation state.

Note

The argument side, specify the drive number assigned by *R_usb_hmsc_alloc_drvno()*.

Example

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

  }
}
```

3.6.5 R_usb_hmsc_Read10

Issue 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

—

Example

```
void  usb_smp_task(void)
{
  uint16_t  result;

  /* Issuing READ10 */
  result = R_usb_hmsc_Read10(side, buff, secno, seccnt, trans_byte);
  if(result != USB_HMSC_OK)
  {
   /* Error Processing */
  }
}
```

3.6.6 R_usb_hmsc_Write10

Issue WRITE10 command

Format

uint16_t	R_usb_hmsc_Wead10(uint16_t side, const 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

—

Example

```
void  usb_smp_task(void)
{
  uint16_t   result;

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

3.6.7 R_usb_hmsc_GetMaxUnit

Issue GetMaxLUN request.

Format

USB_ER_t R_usb_hmsc_GetMaxUnit(uint16_t addr, USB_UTR_CB_t complete)

Argument

addr	Device address
complete	Callback function

Return Value

USB_OK	GET_MAX_LUN issued
USB_ERROR	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

—

Example

```
void  usb_smp_task(void)
{
  USB_ER_t   err;

  /* Getting Max unit number */
  err = R_usb_hmsc_GetMaxUnit(devadr, usb_hmsc_StrgCheckResult);
  if(err == USB_ERROR)
  {
    /* Error processing */
  }
}
```

3.6.8 R_usb_hmsc_MassStorageReset

Issue Mass Storage Reset request.

Format

USB_ER_t R_usb_hmsc_MassStorageReset(uint16_t addr, USB_UTR_CB_t complete)

Argument

addr	Drive address
complete	Callback function

Return Value

USB_OK	MASS_STORAGE_RESET issued
USB_ERROR	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

—

Example

```
void  usb_smp_task(void)
{
  USB_ER_t   err;

  /* Cansel the protocol error */
  err = R_usb_hmsc_MassStorageReset(devadr, usb_hmsc_CheckResult);
  if(err == USB_ERROR)
  {
    /* Error processing */
  }
}
```

3.6.9 R_usb_hmsc_alloc_drvno

Allocates the drive number

Format

uint16_t	R_usb_hmsc_alloc_drvno(uint16_t *side, uint16_t devadr)

Argument

*side	Drive number pointer
devadr	Device address of MSC device

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

This function allocate the drive number to the connected MSC device, and store in the argument (*side).
Drive number is assigned from 0 in the order.

Note

—

Example

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

3.6.10 R_usb_hmsc_free_drvno

Frees the drive number

Format

uint16_t	R_usb_hmsc_free_drvno(uint16_t side)

Argument

side	Drive number

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

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

Note

—

Example

```
void usb_smp_task(void)
{

    /* Frees the drive number */
    R_usb_hmsc_free_drvno(drvno);
}
```

3.6.11 R_usb_hmsc_ref_drvno

Refers the drive number

Format

uint16_t R_usb_hmsc_ref_drvno(uint16_t *side, uint16_t devadr)

Argument

devadr	Device address

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

This function refers the drive numer based on the device address which are specified by the argument (devadr), and stores in the argument (*side).

Note

—

Example

```
void usb_smp_task(void)
{
    /* refers the drive number */
    R_usb_hmsc_ref_drvno(&drvno, devadr);
}
```

3.7 API fucntions of HMSDD

Table 3-4 lists the function of HMSDD.

Table 3-4 HMSDD functions

Function Name	Description
R_usb_hmsc_StrgDriveTask()	Storage drive task
R_usb_hmsc_StrgDriveSearch()	Search drive
R_usb_hmsc_StrgDriveOpen()	Open drive
R_usb_hmsc_StrgDriveClose()	Close drive
R_usb_hmsc_StrgReadSector()	Read data
R_usb_hmsc_StrgWriteSector()	Write data
R_usb_hmsc_StrgCheckEnd()	Check Read/Write end
R_usb_hmsc_StrgUserCommand()	Issue storage command

3.7.1 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 storage command to the storage device.

Notes

Please call this function from a loop that executes the scheduler processing.

Example

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

    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_MgrTask();
        R_usb_hhub_Task();
        R_usb_hmsc_Task();
        R_usb_hmsc_StrgDriveTask();
    }
  }
}
```

3.7.2 R_usb_hmsc_StrgDriveSearch

Search drive

Format

uint16_t R_usb_hmsc_StrgDriveSearch(uint16_t addr, USB_UTR_CB_t complete)

Argument

addr	USB address
complete	Callback function

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

This API checks the follows by sending command to USB device which is specified in the argument (addr).
The callback function which is specified in the argument (complete) is called whencompleting the drive searching.
Refer to section 3.4.

Notes

Continue to operate the "Scheduler" from call this API until the callback function is called and do not the other USB processing.

Example

```
/* Callback function */
void usb_hmsc_StrgCommandResult( USB_UTR_t *mess )
{
      :
}

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

3.7.3 R_usb_hmsc_StrgDriveOpen

Open drive

Format

uint16_t	R_usb_hmsc_StrgDriveOpen(uint16_t *side, uint16_t addr)

Argument

*side	Drive number pointer
addr	USB address

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

Open the address specified in the argument.
Call after the enumeration is complete.

Notes

1. Use the R_usb_hmsc_alloc_drvno () in this function inside to assign the drive number.
2. Use the R_usb_hstd_GetPipeID() in this function inside to set the pipe number.

Example

```
void msc_configured(uint16_t devadr)
{
    R_usb_hmsc_StrgDriveOpen(&drvno, devadr);
}
```

3.7.4 R_usb_hmsc_StrgDriveClose

Close drive

Format

uint16_t R_usb_hmsc_StrgDriveClose(uint16_t side)

Argument

side Drive Number

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

Open the address specified in the argument.
Call after the enumeration is complete.

Notes

1. Use the R_usb_hmsc_free_drvno() in this function inside to open the drive number.
2. Use the R_usb_hstd_ClearPipe() in this function inside to clear the pipe information.

Example

```
void msc_detach(uint16_t devadr)
{
    R_usb_hmsc_StrgDriveClose(drv_no);
}
```

3.7.5 R_usb_hmsc_StrgReadSector

Read Sector

Format

uint16_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_OK	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. Use the R_usb_hmsc_Read10 () in this function inside to read sector information.
3. Use the R_usb_hmsc_GetDevSts () in this function inside, it has confirmed the connection status.If in a disconnected state, and terminates with an error before writing.

Example

```
DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)
{
   uint32_t       err;
   uint32_t       tran_byte;

   /* set transfer length */
   tran_byte = count * _MIN_SS;

   /* read function */
   err = R_usb_hmsc_StrgReadSector(pdrv, buff, sector, (uint16_t)count,
tran_byte);
   if (err != USB_OK)
   {
      return RES_ERROR;
   }
}
```

3.7.6 R_usb_hmsc_StrgWriteSector

Write Sector Information

Format

uint16_t R_usb_hmsc_StrgWriteSector(uint16_t side, const 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

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. Use the R_usb_hmsc_Write10 () in this function inside to write sector information.
3. Use the R_usb_hmsc_GetDevSts () in this function inside, it has confirmed the connection status.If in a disconnected state, and terminates with an error before writing.

Example

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
   uint32_t       err;
   uint32_t       tran_byte;

   /* set transfer length */
   tran_byte = count * _MIN_SS;

   /* write function */
   err = R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count,
tran_byte);
   if (err != USB_OK)
   {
      return RES_ERROR;
   }
}
```

3.7.7 R_usb_hmsc_StrgCheckEnd

Check Read/Write end

Format

uint16_t R_usb_hmsc_StrgCheckEnd(void)

Argument

—

Return Value

USB_FALSE	
USB_TRUE	Normal end
USB_ERROR	Error end

Description

Notes

disconnected state, and terminates with an error before writing.

Example

```
DRESULT disk_write(BYTE pdrv, const BYTE* buff, DWORD sector, UINT count)
{
    uint32_t       err;

    /* write function */
    R_usb_hmsc_StrgWriteSector(pdrv, buff, sector, (uint16_t)count, tran_byte);

    /* Wait USB write sequence(WRITE10) */
    do
    {
        R_usb_cstd_Scheduler();
        if (R_usb_cstd_CheckSchedule() == USB_FLGSET)
        {
            R_usb_hstd_MgrTask();      /* MGR task */
            R_usb_hhub_Task();         /* HUB task */
            R_usb_hmsc_task();         /* HMSC Task */
            R_usb_hmsc_StrgDriveTask(); /* HSTRG Task */
        }
        err = R_usb_hmsc_StrgCheckEnd();
    }
    while (err == USB_FALSE);

    /* Set transfer result */
    if (err != USB_TRUE)
    {
        return RES_ERROR;
    }
    else
    {
        return RES_OK;
    }
}
```

3.7.8 R_usb_hmsc_StrgUserCommand

Issue Storage Command

Format

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

Argument

side	Drive number
command	Command to be issued
*buff	Data pointer
complete	Callback function

Return Value

USB_OK	Normal end
USB_ERROR	Error end

Description

This function issues the storage command specified by the given argument, to 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

1. Use the API function of HMSCD in this function inside to isuue the storage command.
2. Use the R_usb_hmsc_GetDevSts () in this function inside, it has confirmed the connection status.If in a disconnected state, and terminates with an error before writing.

Example

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

3.8 API functions of FSI

Since the FatFs module is a file system layer, it is completely separated from the storage devices.

FatFs requests the interface to the lower layer. Therefore, need to provide a control function corresponding the platforms and storage devices.

HMSC offers a sample of this control functions (FSI fucntions).

Check the specifications of FatFs, change to fit the system if necessary.

Table 3-5 lists the functions of the sample FSI.

Table 3-5 FSI Functions

Function Name	Description
disk_status	Get device status
disk_initialize	Initialize device
disk_read	Read sector(s)
disk_write	Write sector(s)
disk_ioctl	Control device dependent features
get_fattime	Get current time

3.8.1 disk_status

Get device status

Format

DSTATUS disk_status(BYTE pdrv)

Argument

pdrv [IN] Physical drive number

Return Value

The current drive status is returned in combination of status flags described below.

STA_NOINIT	Indicates that the device is not initialized
STA_NODISK	Indicates that no medium in the drive.
STA_PROTECT	Indicates that the medium is write protected. (not use in sample)

Description

This function return the value of the *R_usb_disk_status [pdrv]*.
If more than 10 is set to pdrv, return (*STA_NOINIT | STA_NODISK)*.

Notes

-

3.8.2 disk_initialize

Initialize device

Format

DSTATUS disk_initialize(BYTE pdrv)

Argument

pdrv [IN] Physical drive number

Return Value

The current drive status is returned in combination of status flags described below.

STA_NOINIT	Indicates that the device is not initialized
STA_NODISK	Indicates that no medium in the drive.
STA_PROTECT	Indicates that the medium is write protected. (not use in sample)

Description

This function return the value of the *R_usb_disk_status [pdrv]*.

Notes

This function is under control of FatFs module. Application program MUST NOT call this function, or FAT structure on the volume can be broken. To re-initialize the file system, use f_mount function instead.

3.8.3 disk_read

Read Sector(s)

Format

DRESULT disk_read(BYTE pdrv, BYTE* buff, DWORD sector, UINT count)

Argument

pdrv	[IN] Physical drive number
*buff	[OUT] Pointer to the read data buffer
sector	[IN] Start sector number
count	[IN] Number of sectros to read

Return Value

RES_OK	The function succeeded.
RES_ERROR	Any hard error occured during the read operation.
RES_PARERR	Invalid parameter. (not use in sample)
RES_NOTRDY	The device has not been initialized. (not use in sample)

Description

This function call the API function *R_usb_hmsc_StrgReadSector()* of HMSDD.
Arguments setting of *R_usb_hmsc_StrgReadSector()* is as follows.

Argument	value
uint16_t side	pdrv
uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

This function works the scheduler loops in this function until the USB read sequence is completed.
If it detects a USB disconnect in the middle, and then return the RES_ERROR.

Notes

-

3.8.4 disk_write

Write Sector(s)

Format

DRESULT disk_write(BYTE pdrv,const BYTE* buff, DWORD sector, UINT count)

Argument

pdrv	[IN] Physical drive number
*buff	[IN] Pointer to the data to be written
sector	[IN] Start sector number
count	[IN] Number of sectros to write

Return Value

RES_OK	The function succeeded.
RES_ERROR	Any hard error occured during the write operation.
RES_PARERR	Invalid parameter. (not use in sample)
RES_NOTRDY	The device has not been initialized. (not use in sample)

Description

This function call the API function *R_usb_hmsc_StrgWriteSector()* of HMSDD.
Arguments setting of *R_usb_hmsc_StrgWriteSector()* is as follows.

Argument	value
uint16_t side	pdrv
const uint8_t *buff	buff
uint32_t secno	sector
uint16_t seccnt	(uint16_t)count
uint32_t trans_byte	count * _MIN_SS

This function works the scheduler loops in this function until the USB write sequence is completed.
If it detects a USB disconnect in the middle, and then return the RES_ERROR.

Notes

-

3.8.5 disk_ioctl

Control device dependent features

Format

DRESULT	disk_ioctl(BYTE pdrv, BYTE cmd, void* buff)

Argument

pdrv	[IN] Physical drive number
cmd	[IN] Control command code
*buff	[I/O] Parameter and data buffer

Return Value

RES_OK	The function succeeded.
RES_ERROR	An error occured. (not use in sample)
RES_PARERR	The command code or parameter is invalid. (not use in sample)
RES_NOTRDY	The device has not been initialized. (not use in sample)

Description

This function return RES_OK without the processing for all of the command.

Notes

-

3.8.6 get_fattime

Get current time

Format

DWORD get_fattime(void)

Argument

-

Return Value

Currnet local time is returned with packed into a DWORD value. The bit field is as follows:

bit31:25	Year origin from the 1980 (0..127)
bit24:21	Month (1..12)
bit20:16	Day of the month(1..31)
bit15:11	Hour (0..23)
bit10:5	Minute (0..59)
bit4:0	Second / 2 (0..29)

Description

This function return 0x00000000 without setting the date and time information.

Notes

-

3.9 Scheduler settings

Table 3-6 lists the scheduler settings of HMSC.

Table 3-6 Scheduler settings

Function name	Task ID	Priority	Mailbox ID	Memory Pool ID	Desctiption
R_usb_hmsc_StrgDriveTask	USB_HSTRG_TSK	USB_PRI_3	USB_HSTRG_MBX	USB_HSTRG_MPL	HSTRG Task
R_usb_hmsc_task	USB_HMSC_TSK	USB_PRI_3	USB_HMSC_MBX	USB_HMSC_MPL	HMSC Task
R_usb_hub_task	USB_HUB_TSK	USB_PRI_3	USB_HUB_MBX	USB_HUB_MPL	HUB Task
R_usb_hstd_MgrTask	USB_MGR_TSK	USB_PRI_2	USB_MGR_MBX	USB_MGR_MPL	MGR Task
r_usb_hstd_HciTask	USB_HCI_TSK	USB_PRI_1	USB_HCI_MBX	USB_HCI_MPL	HCD Task

4. Sample Application

This section describes the initial settings necessary for using HMSC and USB-BASIC-F/W 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.

4.1 Example Operating Environment

Figure 4-1 shows an example operating environment for HMSC.

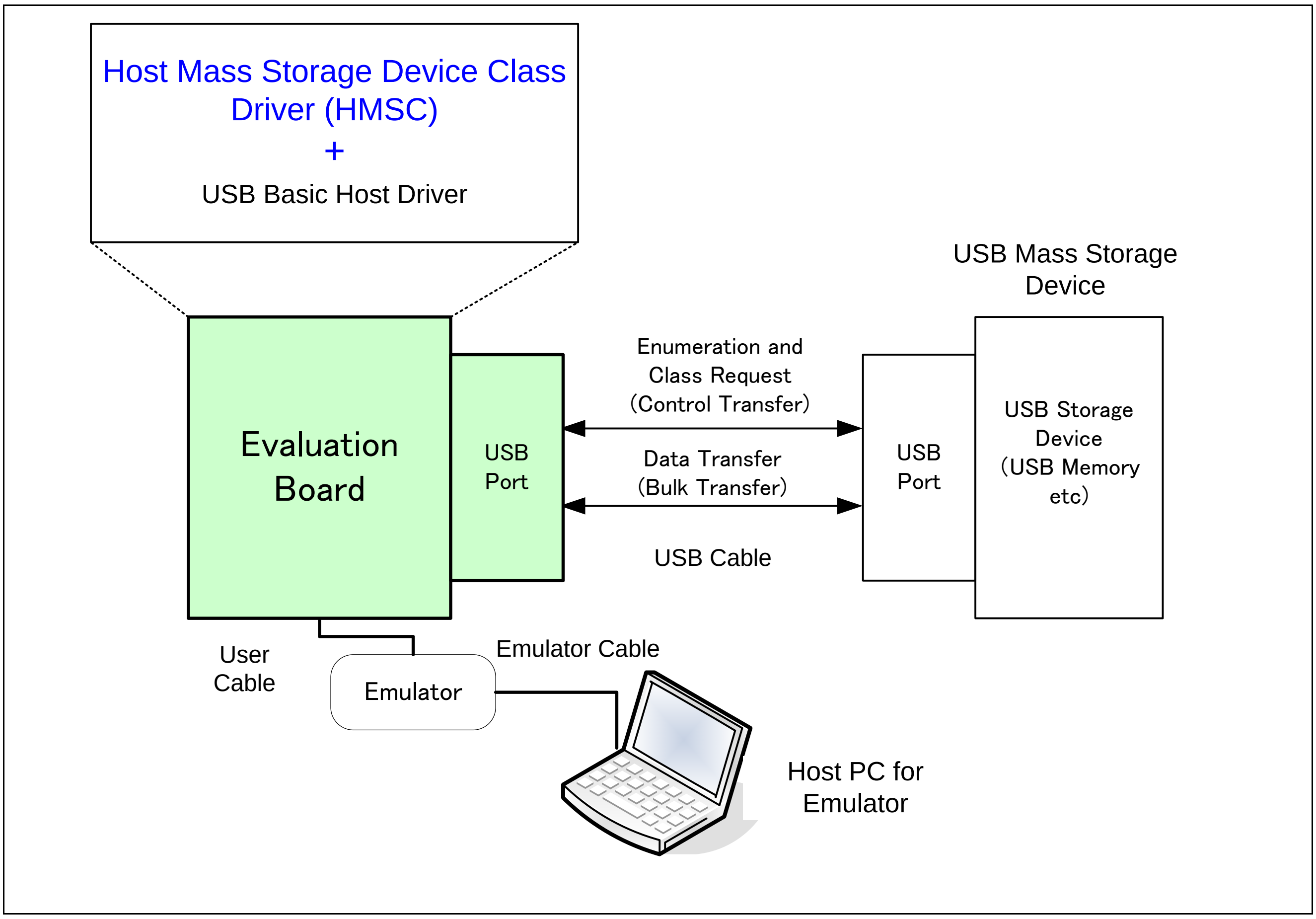


Figure 4-1 Example Operating Environment

4.2 Application Specifications

The main functions of HMSC sample application are as follows:

1. Processes the enumeration when the MSC device is connected.
2. Processes drive search when the enumeration of the MSC device is complete.
3. Writes a file with the 512 bytes of size to the MSC device.
4. Reads the file written in a MSC device.
5. Lights the LED that corresponds to the drive number if matched by comparing the file contents.

4.3 Initial Settings

Sample settings are shown below.

```
void usb_hmsc_apl(void)
{
   /* MCU Pin Setting (Refer to “4.3.1”) */
   usb_mcu_setting();

   /* USB Driver Setting (Refer to “4.3.2”) */
   R_usb_hstd_MgrOpen();
   R_usb_cstd_SetTaskPri(USB_HUB_TSK, USB_PRI_3);   // Note
   R_usb_hhub_Registration(USB_NULL);               // Note
   msc_registration();
   R_usb_hmsc_driver_start();

   /* Main routine (Refer to “4.4”) */
   usb_hapl_mainloop();
}
```

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

4.3.1 MCU Settings

Set the USB module according to the initial setting sequence of the hardware manual, the USB interrupt handler registration and USB interrupt enable setting.

4.3.2 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. The procedure is described below.

1. Call the USB-BASIC-F/W’s API function (R_usb_hstd_MgrOpen() to register the MGR task and the HCD task with the scheduler.
2. Call the class driver API function (R_usb_hhub_Registration()) to register the HUB task with the scheduler.
3. After specifying the necessary information in the members of the class driver registration structure (USB_HCDREG_t), call the USB-BASIC-F/W’s API function (R_usb_hstd_DriverRegistration() to register the class driver information.
4. Call the class driver HMSC’s API function (R_usb_hmsc_driver_start()) to register HMSC task and the HSTRG task with the scheduler.

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

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

   /* Class Code which is defined in the USB specification setting*/
   driver.ifclass     = (uint16_t)USB_IFCLS_MSCC;
   /* TPL setting */
   driver.tpl         = (uint16_t*)&usb_gapl_devicetpl; // Note 1
   /* Set the class check function which is called in the enumeration. */
   driver.classcheck  = &R_usb_hmsc_class_check;
   /* Set the function which is called when completing the enumeration */
   driver.devconfig   = &msc_configured;
   /* Set the function which is called when disconnecting USB device */
   driver.devdetach   = &msc_detach;
   /* Set the function which is called when changing the suspend state */
   driver.devsuspend  = &msc_suspend;
   /* Set the function which is called when resuming from the suspend state */
   driver.devresume   = &msc_resume;
```

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

[Note]

1. TPL(Target Peripheral List) need to be defined in the application program. Refer to USB Basic Firmware application note (Document No.R01AN2633EJ) about TPL.

4.4 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
  {
    // Confirming the event and getting (Note 1)
    R_usb_cstd_Scheduler();

    // Judgment whether the event is or not
    if(USB_FLGSET == R_usb_cstd_CheckSchedule())
    {
        R_usb_hstd_HcdTask((USB_VP_INT)0);    // HCD task                   (Note
        R_usb_hstd_MgrTask((USB_VP_INT)0);    // MGR task                   2)
        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
    }
    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.

4.4.1 APL

APL is managed by the state transition.
Table 4-1 shows list of states.

Table 4-1 List of States

State	Description
STATE_ATTACH	Wait attach
STATE_DRIVE	Search drive
STATE_READY	Mount drive
STATE_WRITE	FIle Write
STATE_READ	File Read
STATE_COMPLETE	Processing completion
STATE_ERROR	Error occurred

Figure 4-2 shows the process flowchart of APL.

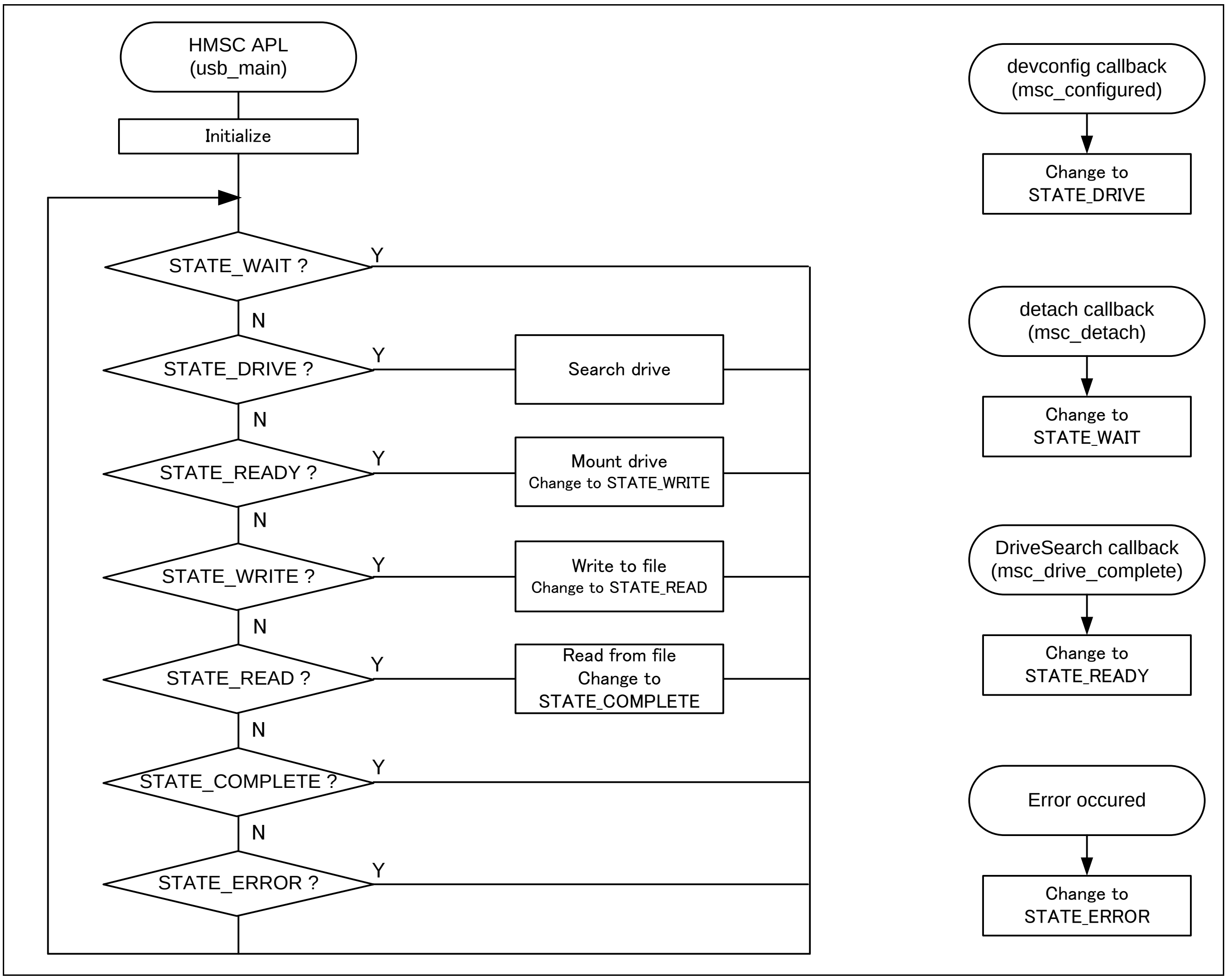


Figure 4-2 Main Loop flowchart

4.4.2 State Management

An overview of the processing associated with each state is provided below.

1) Wait Attach (STATE_WAIT)

== Outline ==

In this state, wait for the MSC device attach. When the enumeration is complete, then changes the state to STATE_DRIVE.

== Description ==

1. Initialization function sets the state to STATE_WAIT.
2. Continue to STATE_WAIT until the MSC device is connected.
3. When the MSC device is connected to enumeration is complete, The callback function *msc_configured()* is specified in the member *devconfig* of structure *USB_HCDREG_t* is called from the USB driver.
4. Changes the state to STATE_DRIVE.

2) Search drive (STATE_DRIVE)

== Outline ==

In this state, make the drive search process of the connected MSC device and change the state to STATE_READY.

== Description ==

1. Check the drive recognition flag variable *drive_search_lock*. Start the process if it's off.
2. Turn on the *drive_search_lock*.
3. Call *R_usb_hmsc_StrgDriveSearch()*, transmit class request GetMaxLUN and storage command to MSC device, and make the drive search process.
4. When completion of drive search process, *R_usb_hmsc_StrgDriveSearch()* callback function was registerd in *msc_drive_complete()* is called.
5. Change the state to STATE_READY.

3) Mount drive (STATE_READY)

== Outline ==

In this state, mount the recognized drive and change the state to STATE_WRITE.

== Description ==

1. Call *f_mount()*, mount in the recognized drive number.
2. Changes the state to STATE_WRITE.

4) File Write (STATE_WRITE)

== Outline ==

In this state, write the file to the connected MSC device and change the state to STATE_READ.

== Description ==

1. Call *f_open()*, Open the file in create and write mode.
2. Call *f_write()*, create the file of 512bytes of all 'a' (hmscdemX.txt). X in the file name corresponds to the drive number. For example, in the case of drive 1, file name is hmscdem1.txt.
3. Call *f_close()*, close the file.
4. Changes the state to STATE_READ.

5) File Read (STATE_READ)

== Outline ==

In this state, read the file from the connected MSC device and change the state to STATE_COMPLETE.

== Description ==

1. Call *f_open()*, open the file in read mode.
2. Call *f_read()*, read the file (hmscdemX.txt).
3. Check whether 512 bytes of data of all 'a'
4. Call *f_close()*, close the file.
5. Lights the LED that corresponds to the drive number.
6. Changes the state to STATE_COMPLETE.

6) Processing completion (STATE_COMPLETE)

== Outline ==

When the processing of the sample application is normally finished, will be in this state.

== Description ==

None processing.

7) Error occured (STATE_ERROR)

== Outline ==

When the processing of the sample application is abnormally terminated, will be in this state.

== Description ==

None processing.

8) Detach processing (STATE_DETACH)

When the connected MSC device is disconnected, the USB driver calls the callback function *msc_detach()*. This callback function perform to initialize variables and unmount drive and change the state to STATE_WAIT. The callback function *msc_detach()* is the function set in the member *devdetach* of the structure *USB_HCDREG_t*.

5. Importing procedure of FatFs

To build the sample program, the users need to import the FatFs.
The procedure to import FatFs is described below.

5.1 Obtains FatFs from the web site

FatFs is distributed at the following URL.
http://elm-chan.org/fsw/ff/00index_e.html

1. Scroll down the web page, then the download link will appear in "Resources" section.
2. To download "FatFs", click on the link "FatFs R0.13" and save it in arbitrary folder.

The sample program is created for the version R0.13. If FatFs has been updated, find this version from the link " Previous release".

5.2 Extracts FatFs at proper folder

Extracts the ZIP file (ff13.zip) of the downloaded FatFs, and move it into the workspace of the sample program as shown in the Figure 5-1.
Note that the sample program does not work without this folder structure.

```
RZ_T1_USBh_HMSC/
|-- inc/
`-- src/
    |-- common/
    |   |-- nor_boot/
    |   `-- serial_boot/
    |-- drv/
    |   `-- usbh/
    |       |-- basic/
    |       `-- hmsc/
    |           `-- ff13/   <--- "FatFs" here!
    |               |-- documents/
    |               |   |-- doc/
    |               |   `-- res/
    |               `-- source/
    |
    `-- sample/
```

Figure 5-1 The position to place "FatFs" folder.

5.3 Opens the workspace and builds the project

Open the workspace of the sample program, and build the sample project.
The sample program is set to link FatFs, but if the development environment is e2 studio or DS-5, please exclude the documents folder right under the ff13 folder and the diskio.c file under source from the build target.

5.4 Notice

When embedding FatFs into user's products, check the licence of FatFs and do it as user's responsibility.

Appendix A. Changes of initial setting

USB-BASIC-F/W has been changed to "RZ/T1 group initial setting Rev.1.30".
Sample program supports IAR embedded workbench for ARM (EWARM), DS-5 and e^2 studio.
This chapter describes the changes.

Folders and files

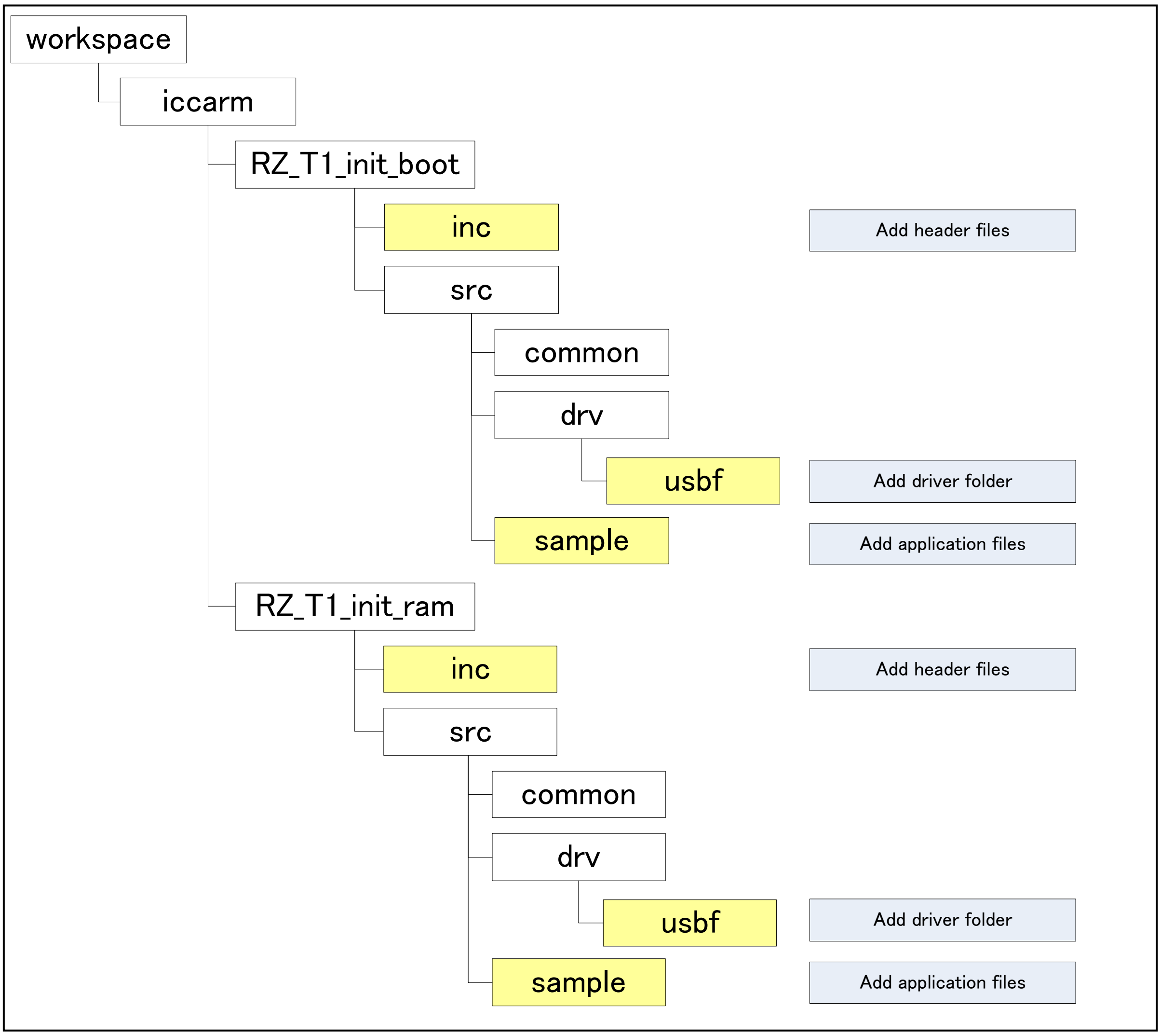
In the "RZ/T1 group initial setting Rev.1.30", different folder structure by the development environment and the boot method. Changes to each folder of all of the development environment and the boot method it is shown below.

- Add the following files in the “inc” folder.
  r_usb_basic_config.h
  r_usb_basic_if.h
  r_usb_hatapi_define.h
  r_usb_hmsc_config.h
  r_usb_hmsc_if.h

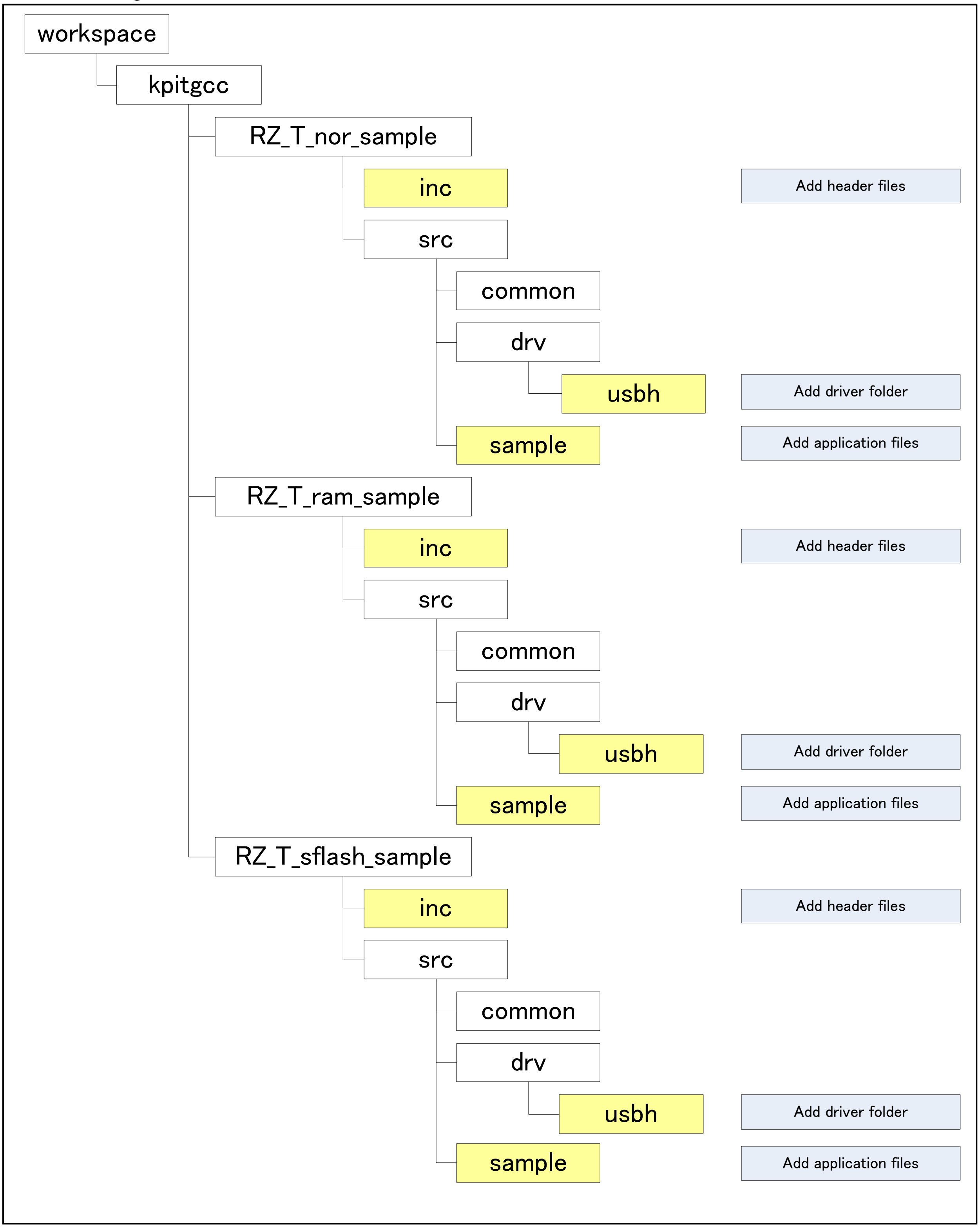
- Add the following files in the “sample” folder.
  r_usb_main.c
  r_usb_hmsc_apl.c
  r_usb_hmsc_apl.h

- Add the “usbh” folder and the following files “usbh” folder in the “drv” folder.

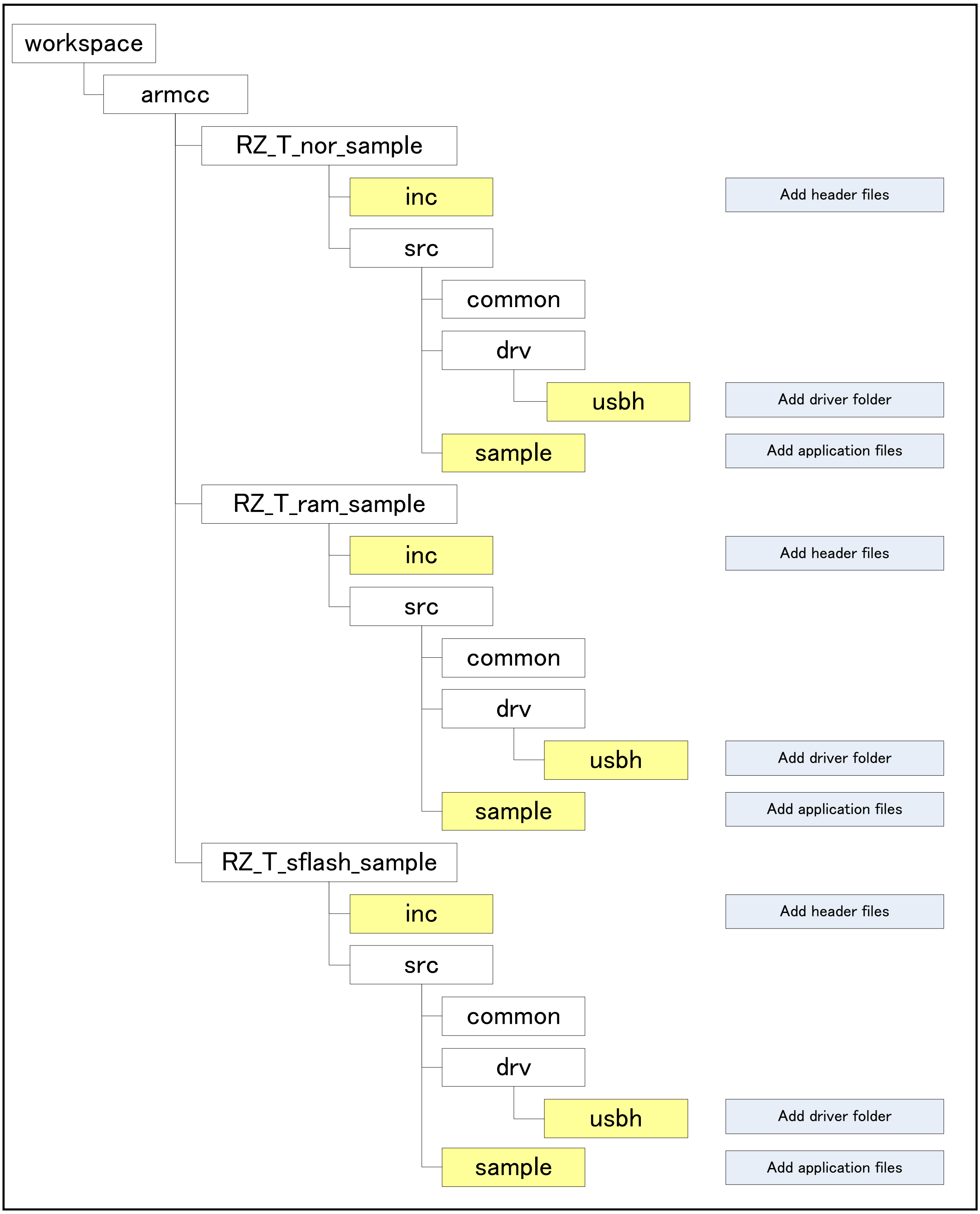
The following is the folder structure of EWARM.



The following is the folder structure of e[2] studio.



The following is the folder structure of DS-5.



Section

Modify the section size of the code area and a data area, and add the following section.

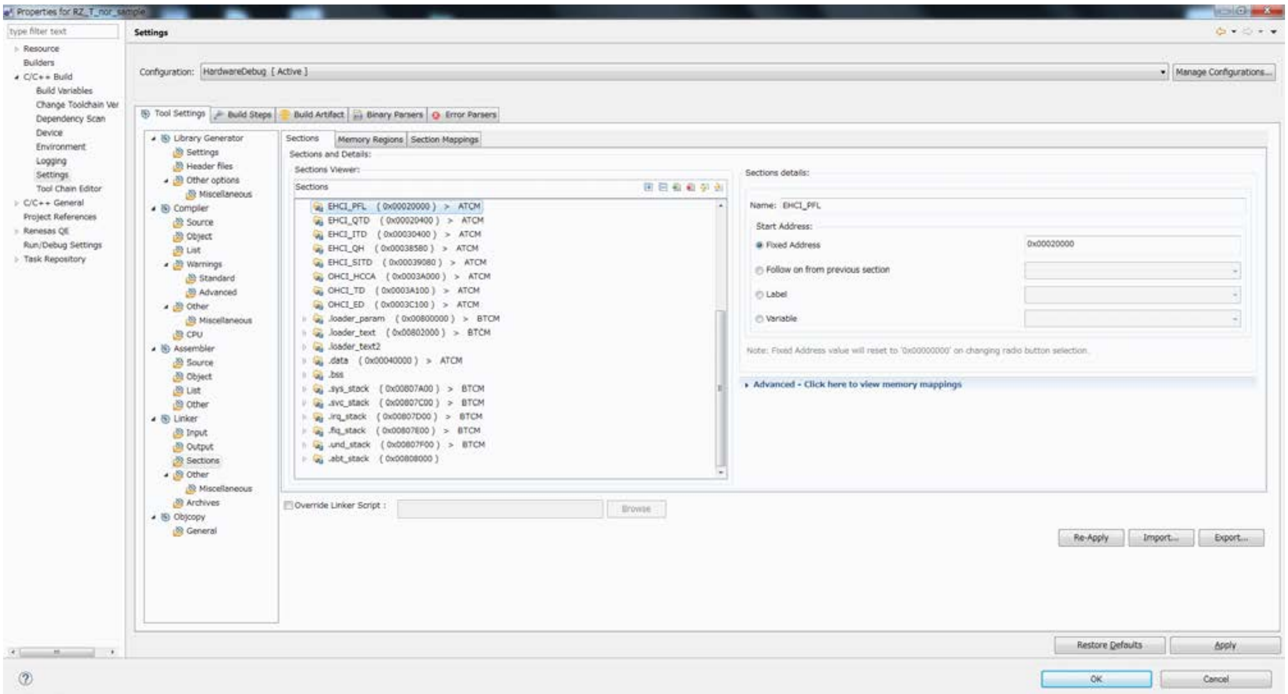
Section name	Address	variable	file
EHCI_PFL	0x00020000	ehci_PeriodicFrameList	r_usb_hEhciMemory.c
EHCI_QTD	0x00020400	ehci_Qtd	
EHCI_ITD	0x00030400	ehci_Itd	
EHCI_QH	0x00038580	ehci_Qh	
EHCI_SITD	0x00039080	ehci_Sitd	
OHCI_HCCA	0x0003A000	ohci_hcca	r_usb_hOhciMemory.c
OHCI_TD	0x0003A100	ohci_TdMemory	
OHCI_ED	0x0003c100	ohci_EdMemory	

e^2 studio

e^2 studio sets the section in the configuration screen.
Changes are as follows:

・Fixed address of ".data" section from 0x0007F000 to 0x00040000
・Add section setting of EHCI and OHCI.

Refer to [Project] → [Properties] → [C/C++ Build] → [Settings] → [Sections].



Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __GNUC__
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ]
                __attribute__ ((section ("EHCI_PFL")));
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ]
                __attribute__ ((section ("EHCI_QH")));
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ]
                __attribute__ ((section ("EHCI_QTD")));
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ]
                __attribute__ ((section ("EHCI_ITD")));
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ]
                __attribute__ ((section ("EHCI_SITD")));
#endif  /* __GNUC__ */
```

r_usb_hOhciMemory.c

```
#ifdef __GNUC__
static USB_OHCI_HCCA_BLOCK              ohci_hcca
                __attribute__ ((section ("OHCI_HCCA")));
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR ohci_TdMemory[USB_OHCI_NUM_TD]
                __attribute__ ((section ("OHCI_TD")));
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR ohci_EdMemory[USB_OHCI_NUM_ED]
                __attribute__ ((section ("OHCI_ED")));
#endif  /* __GNUC__ */
```

EWARM

EWARM sets the section in the linker setting file (.icf file).

Changes are as follows:
・Start address of RAM region from 0x00070000 to 0x00040000.
・End address of USER_PRG region from 0x0006FFFF to 0x0001FFFF.

```
define symbol __ICFEDIT_region_RAM_start__    = 0x00040000;

define symbol __region_USER_PRG_end__         = 0x0001FFFF;
```

・To the EHCI and OHCI to fixed address, adds memory region definition.

```
define region EHCI_MEM1_region = mem:[from 0x00020000 to 0x000203FF];
define region EHCI_MEM2_region = mem:[from 0x00020400 to 0x00039FFF];

define region OHCI_MEM1_region = mem:[from 0x0003A000 to 0x0003A0FF];
define region OHCI_MEM2_region = mem:[from 0x0003A100 to 0x0003FFFF];

do not initialize  { section EHCI_PFL, section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section
EHCI_SITD, section OHCI_HCCA, section OHCI_TD, section OHCI_ED };

place in EHCI_MEM1_region { section EHCI_PFL };
place in EHCI_MEM2_region { section EHCI_QH, section EHCI_QTD, section EHCI_ITD, section EHCI_SITD };

place in OHCI_MEM1_region { section OHCI_HCCA };
place in OHCI_MEM2_region { section OHCI_TD, section OHCI_ED };
```

Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __ICCARM__
#pragma location="EHCI_PFL"
static uint32_t                ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma location="EHCI_QH"
static USB_EHCI_QH             ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma location="EHCI_QTD"
static USB_EHCI_QTD            ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma location="EHCI_ITD"
static USB_EHCI_ITD            ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma location="EHCI_SITD"
static USB_EHCI_SITD           ehci_Sitd[ USB_EHCI_NUM_SITD ];
#endif  /* __ICCARM__ */
```

r_usb_hOhciMemory.c

```
#ifdef __ICCARM__
#pragma location="OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                 ohci_hcca;
#pragma location="OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR    ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma location="OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR    ohci_EdMemory[USB_OHCI_NUM_ED];
#endif  /* __ICCARM__ */
```

DS-5

DS-5 sets the section in the linker setting file (.icf file).

Changes are as follows:

・Start address of RAM region from 0x00040000and BSS region(0clear init memory region) follow RAM region.

```
DATA            0x00040000    UNINIT
{
    * (+RW)
}
BSS             +0
{
    * (+ZI)
}
```

・To the EHCI and OHCI to fixed address, adds memory region definition.

```
EHCI_PERIODIC_FRAMELIST 0x00020000    0x400
{
        r_usb_hEhciMemory.o(EHCI_PFL)
}
EHCI_QTD         +0
{
        r_usb_hEhciMemory.o(ehci_Qtd)
}
EHCI_ITD         +0
{
        r_usb_hEhciMemory.o(ehci_Itd)
}
EHCI_QH          +0
{
        r_usb_hEhciMemory.o(ehci_Qh)
}
EHCI_SITd        +0
{
        r_usb_hEhciMemory.o(ehci_Sitd)
}
OHCI_HCCA        0x0003A000      0x100
{
        r_usb_hOhciMemory.o(OHCI_HCCA)
}
OHCI_TDMEMORY    +0
{
        r_usb_hOhciMemory.o(OHCI_TD)
}
OHCI_EDMEMORY    +0
{
        r_usb_hOhciMemory.o(OHCI_ED)
}
```

Variable definitions in the code are as follows.

r_usb_hEhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "EHCI_PFL"
static uint32_t         ehci_PeriodicFrameList[ USB_EHCI_PFL_SIZE ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QH"
static USB_EHCI_QH      ehci_Qh[ USB_EHCI_NUM_QH ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_QTD"
static USB_EHCI_QTD     ehci_Qtd[ USB_EHCI_NUM_QTD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_ITD"
static USB_EHCI_ITD     ehci_Itd[ USB_EHCI_NUM_ITD ];
#pragma arm section zidata
#pragma arm section zidata = "EHCI_SITD"
static USB_EHCI_SITD    ehci_Sitd[ USB_EHCI_NUM_SITD ];
#pragma arm section zidata
#endif
```

r_usb_hOhciMemory.c

```
#ifdef __CC_ARM
#pragma arm section zidata = "OHCI_HCCA"
static USB_OHCI_HCCA_BLOCK                  ohci_hcca;
#pragma arm section zidata
#pragma arm section zidata = "OHCI_TD"
static USB_OHCI_HCD_TRANSFER_DESCRIPTOR    ohci_TdMemory[USB_OHCI_NUM_TD];
#pragma arm section zidata
#pragma arm section zidata = "OHCI_ED"
static USB_OHCI_HCD_ENDPOINT_DESCRIPTOR    ohci_EdMemory[USB_OHCI_NUM_ED];
#pragma arm section zidata
#endif
```

Call the USB-BASIC-F/W function

Adds the usbh_main() of USB-BASIC-F/W in the main() of “¥src¥sample¥int_main.c”.

```
extern void usbh_main(void);

int main (void)
{
    /* Initialize the port function */
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();

    /* USBh main */
    usbh_main();

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
```

Website and Support

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

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Aug 21, 2015	—	First edition issued
1.10	Dec 25, 2015	43	Added Appendix A
1.20	Feb 29, 2016	45,49,50	Added DS-5 setting
1.30	Dec 07, 2017	—	Corresponds to RZ / T1 initial setting Ver 1.30
		42	Changed FatFs version and folder structure diagram

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

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

1. Handling of Unused Pins

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

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

2. Processing at Power-on

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

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

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

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

4. Clock Signals

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

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

5. Differences between Products

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

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

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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

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

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

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

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

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

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

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