# RX Family

## M3S-TFAT-Tiny Memory Driver Interface Module

## Firmware Integration Technology

### Introduction

This Application Note describes the M3S-TFAT-Tiny Memory Driver Interface module which uses Firmware Integration Technology (FIT). This module uses as memory driver interface to combine RX Family Open source FAT filesystem M3S-TFAT-Tiny FIT (TFAT FIT) with each memory drivers. In this document, this module is referred to as the TFAT driver FIT module.

Please refer to the following URL to know the details about FIT Modules.

https://www.renesas.com/en-us/solutions/rx-applications/fit.html

In this document, the terms are used as follows.

- TFAT FIT:

  RX Family Open Source FAT File System M3S-TFAT-Tiny Module FIT (R20AN0038)

- TFAT driver FIT:

  RX Family M3S-TFAT-Tiny Memory Driver Interface Module FIT (R20AN0335)

- TFAT:

  M3S-TFAT-Tiny or generic term for TFAT FIT and TFAT driver FIT

This Application Note provides the driver interface corresponding to SD memory card (SD mode), SD memory card (SPI mode), USB memory, USB Mini, eMMC, Serial Flash memory. Please use with following FIT Modules.

| Function | Product | Website |
|---|---|---|
| File system (*1) | TFAT FIT | http://www.renesas.com/mw/tfat-rx |
| SD memory card Drive (*2) | SD memory card Driver (SD mode) | https://www.renesas.com/driver/rtm0rx0000dsdd |
| | SPI mode SD memory card Driver | |
| USB Driver (*2) | USB Basic Host and Peripheral Driver | http://www.renesas.com/driver/usb |
| | USB Host Mass Storage Class Driver (HMSC) | |
| USB Mini Driver (*2) | USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) | http://www.renesas.com/driver/usb |
| | USB Host Mass Storage Class Driver for USB Mini Firmware | |
| eMMC Driver (*2) | MMC Mode MMCIF Driver | https://www.renesas.com/products/software-tools/software-os-middleware-driver/mmc/multimediacard-emmc-driver-for-rx-family.html |
| Serial Flash memory Driver (*2) | Clock Synchronous Control Module for Serial Flash Memory Access | https://www.renesas.com/products/software-tools/software-os-middleware-driver/serial-memory/spi-qspi-serial-flash-driver.html |

*1 This is required.

*2 One module is required.

## Target Device

- RX Family

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

## Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Confirmed Operation Environment".

## Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family Open Source FAT File System M3S-TFAT-Tiny Module Firmware Integration Technology (R20AN0038)
- RX Family SD Mode SD Memory Card Driver Firmware Integration Technology (R01AN4233)
- RX Family SPI Mode SD Memory Card Driver Firmware Integration Technology (R01AN6908)
- RX Family USB Basic Host and Peripheral Driver using Firmware Integration Technology (R01AN2025)
- RX Family USB Host Mass Storage Class Driver (HMSC) Firmware Integration Technology (R01AN2029)

RX Family USB Basic Mini Host and Peripheral Driver (USB Mini Firmware) Using Firmware Integration Technology (R01AN2166)

- RX Family USB Host Mass Storage Class Driver for USB Mini Firmware Using Firmware Integration Technology (R01AN2169)
- RX Family MMC Mode MMCIF Driver Firmware Integration Technology (R01AN4234)
- RX Family Clock Synchronous Control Module for Serial Flash Memory Access Firmware Integration Technology (R01AN2662)
- RX Family System Timer Module Firmware Integration Technology(R20AN0431)

## Contents

# 1. Overview

## 1.1 This Application Note

This Application describes memory driver interface combines TFAT FIT and each memory drivers. This Module can change the target of memory driver using config file.

The APIs provided by this module are called by TFAT FIT. It is no need to call by user.

The drive number controlled in TFAT FIT and the drive number controlled in device drivers (SD memory card driver etc.) are not equal. Therefore, this module has the conversion table for drive. Initial value can be configured, please refer to the section drv_change_alloc() if you change this as dynamic.

## 1.2 Structure of Application

### 1.2.1 Structure of Application

This application note includes the files below.

**Table 1.1 Structure of application note**

| file/folder name | description |
|---|---|
| FITModules | |
| driver_rx_v2.60.xml | FIT plug-in XML |
| driver_rx_v2.60_extend.mdf | Smart Configurator setting File |
| driver_rx_v2.60.zip | FIT plug-in ZIP |
| configuration (r_config) | |
| driver_rx_config.h | configuration file(default) |
| FIT Module (driver_rx) | |
| document(doc) | |
| English(en) | |
| r20an0335ej0260-rx-tfat.pdf | Application note (English) |
| Japanese(ja) | |
| r20an0335jj0260-rx-tfat.pdf | Application note (Japanese) |
| source code(src) | |
| readme (readme.txt) | readme |
| r_tfat_driver_rx_if.h | Header file |

### 1.2.2 Structure of Software

This product works with the TFAT FIT, the system timer module FIT, and various device driver FITs.

TFAT FIT is the main module of file system that contains open source the FatFs inside. The TFAT driver FIT has Wrapper functions inside and switches the I/O processing for the file system for each storage device. The user sets the storage device used by the TFAT driver FIT configuration settings and operates the file system via the TFAT FIT API.

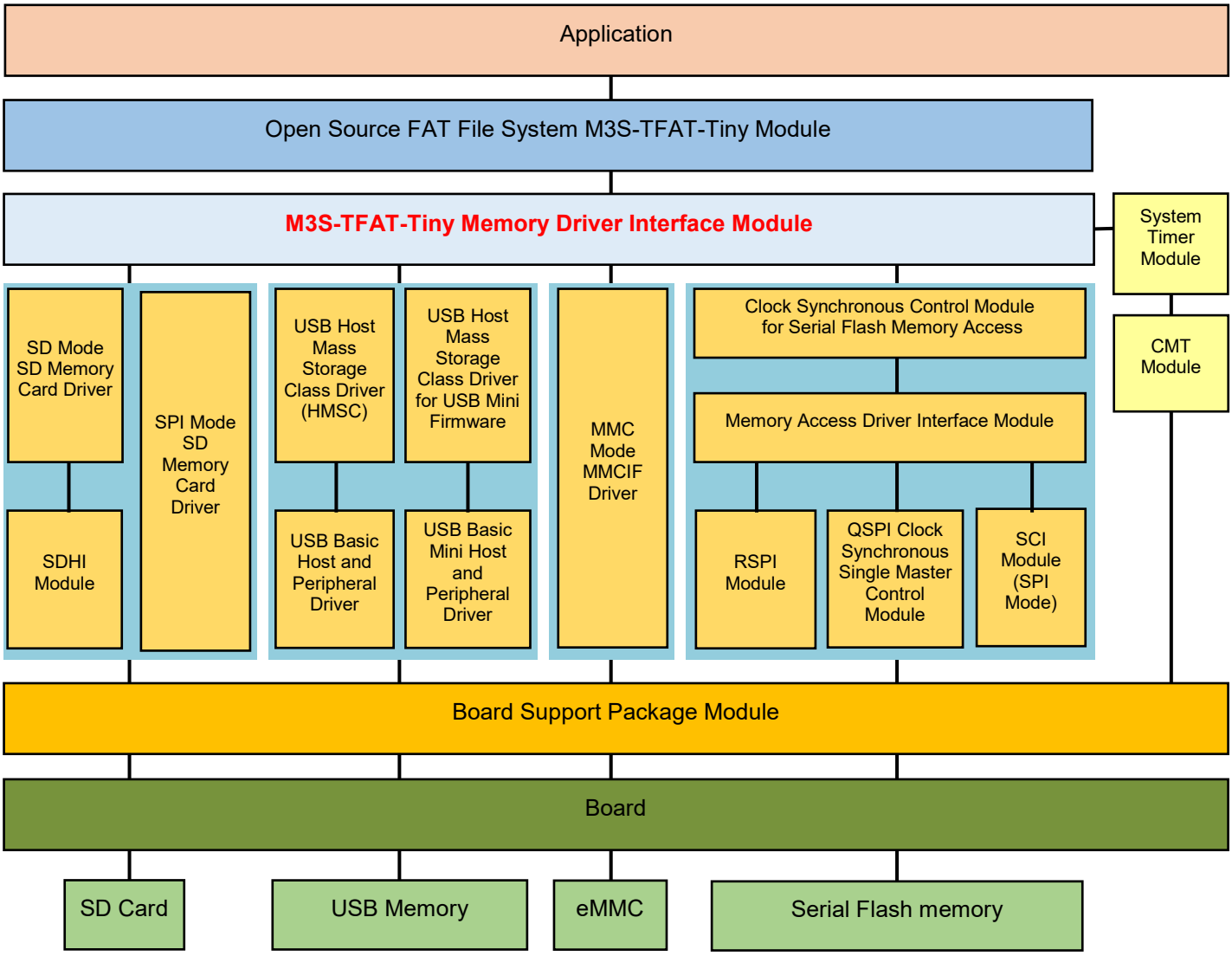


**Fig.1-1 Structure of software**

**Table 1.2 Using FIT Modules version**

| Storage Device | Product | version |
|---|---|---|
| Common | Board Support Package (BSP) | 7.51 |
| | TFAT | 4.13 |
| | System Timer module | 1.01 |
| | CMT Module | 5.70 |
| SD memory card (SD mode) | SD Mode SD Memory Card Driver | 3.00 |
| | SDHI Module | 2.11 |
| SD memory card (SPI mode) | SPI Mode SD Memory Card Driver | 1.10 |
| | RSPI Module | 3.50 |
| | SCI Module (SPI Mode) | 5.30 |
| USB memory | USB Basic Host and Peripheral Driver | 1.42 |
| | USB Host Mass Storage Class Driver (HMSC) | 1.42 |
| | USB Basic Mini Host and Peripheral Driver | 1.20 |
| | USB Host Mass Storage Class Driver for USB Mini Firmware | 1.20 |
| eMMC | MMC Mode MMCIF Driver | 1.10 |
| Serial Flash memory | Clock Synchronous Control Module for Serial Flash Memory Access | 3.30 |
| | Memory Access Driver Interface Module | 1.20 |
| | RSPI Module | 3.50 |
| | QSPI Clock Synchronous Single Master Control Module | 1.21 |
| | SCI Module (SPI Mode) | 5.30 |

## 1.3 API Overview

Table 1.3 shows the API Functions for this driver.

**Table 1.3 API Functions**

| Function | Functional Overview |
|---|---|
| disk_initialize() | Initialize disk drive. |
| disk_status() | Get the information about disk drive status. |
| disk_read() | Read the data from disk. |
| disk_write() | Write the data to disk. |
| disk_ioctl() | Control the drive. |
| get_fattime() | Get the time information. |
| drv_change_alloc() | Change the allocation of drive number between TFAT module and device driver. |

## 1.4 Limitations

(1) The target devices of TFAT are the devices supported by all the FITs that users use as lower layer than the TFAT. For the target devices of each FIT, refer to the respective application note.

(2) Use Rev.1.20 or later for the USB Basic Mini Host and Peripheral Driver FIT and the USB Host Mass Storage Class Driver for USB Mini Firmware FIT.
Because these revisions support RTOS.

# 2. API Information

## 2.1 Hardware Requirements

The microcontroller used must support the following functionality.

- USB
- SDHI
- CMT

## 2.2 Software Requirements

This FIT Module is dependent on the following packages:

- r_bsp(Rev. 5.52 or later)
- r_tfat_rx (Rev.4.12 or later)
- r_sys_time_rx (Rev.1.01 or later)
- r_cmt_rx (Rev.4.40 or later)

The kind of memory driver to use can be set in r_tfat_driver_rx_config.h

## 2.3 Supported Toolchain

The supported toolchains of this module are dependent on the toolchains of each memory driver.

## 2.4 Interrupt Vector

The TFAT driver FIT uses no interrupt vector.

## 2.5 Header Files

All API calls and their supporting interface definitions are located in "r_tfat_driver_rx_if.h".
Build-time configuration options are selected or defined in the file "r_tfat_driver_rx_config.h".

## 2.6 Integer Types

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

## 2.7 Configuration Overview

The configuration options in this module are specified in "r_tfat_driver_rx_config.h".
The option names and setting values are listed in the table below.

| Configuration options in r_tfat_driver_rx_config.h | |
|---|---|
| #define<br>TFAT_USB_DRIVE_NUM<br>- Default value = (0) | The number of drives for USB.<br>Please set (0) if user does not use USB. |
| #define<br>TFAT_SDMEM_DRIVE_NUM<br>- Default value = (0) | The number of drives for SD memory card.<br>Please set (0) if user does not use SD memory card. |
| #define<br>TFAT_SPI_SDMEM_DRIVE_NUM<br>- Default value = (0) | The number of drives for SPI mode SD memory card.<br>Please set (0) if user does not use SPI mode SD memory card. |
| #define<br>TFAT_USB_MINI_DRIVE_NUM<br>- Default value = (0) | The number of drives for USB Mini.<br>Please set (0) if user does not use USB Mini. |
| #define<br>TFAT_MMC_DRIVE_NUM<br>- Default value = (0) | The number of drives for eMMC.<br>Please set (0) if user does not use eMMC. |
| #define<br>TFAT_SERIAL_FLASH_DRIVE_NUM<br>- Default value = (0) | The number of drives for Serial Flash memory.<br>Please set (0) if user does not use Serial Flash memory. |
| #define<br>TFAT_FLASH_SECTOR_SIZE<br>- Default value = (4096) | The sector size configuration when using TFAT and Serial Flash memory.<br><br>512 bytes = (512)<br>1024 bytes = (1024)<br>2048 bytes = (2048)<br>4096 bytes = (4096) |
| #define<br>TFAT_DRIVE_ALLOC_NUM_i<br>i = 0-9<br>- Default value = (TFAT_CTRL_NONE) | This config allocates the device for each drive number.<br><br>The drive for USB = (TFAT_CTRL_USB)<br>The driver for SD memory card = (TFAT_CTRL_SDMEM)<br>The driver for USB Mini = (TFAT_CTRL_USB_MINI)<br>The driver for eMMC = (TFAT_CTRL_MMC)<br>The driver for Serial Flash memory<br>= (TFAT_CTRL_SERIAL_FLASH)<br>The driver for “not using” = (TFAT_CTRL_NONE)<br><br>This module uses these parameters for relating the drive number for TFAT FIT with the drive number of memory driver. The drive number is allocated ascending order. Please refer to the section 3.7 drv_change_alloc if user change this in dynamic. |

| Configuration options in r_tfat_driver_rx_config.h | |
|---|---|
| #define<br>RI600V4_MUTEX_ID_FOR_TFAT_DRIVE_ALLOC_NUM_i<br>i = 0~9<br>- Default value = (0) | When using RI600V4, input the mutex ID created by RI600V4 configuration.<br>This mutex is used by TFAT APIs to obtain the reentrancy (file/directory exclusive access) on a drive (logical volume).<br>Please set (0) if not use RI600V4 or the memory drive. Duplication with ID “0” is allowed.<br>Please set (1 to 255) for mutex ID if use RI600V4 and the memory drive. Duplication with ID for other using drives is not allowed. |

## 2.8 Code Size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the RX200 Series, and RX600 Series, respectively.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7 Configuration Overview

The values in the table below are confirmed under the following conditions.

Module Revision: r_tfat_driver_rx rev.2.60

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00

(The option of “lang = c99” is added to the default settings of the integrated development environment.)

GCC for Renesas RX 8.3.0.202405

(The option of “-std=gnu99” is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 5.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

**ROM, RAM and Stack Code Sizes**

| Device | Category | Memory Used | | |
|---|---|---|---|---|
| | | Renesas Compiler | GCC | IAR Compiler |
| RX113 | ROM(Note) | 6,516 bytes | 7,688 bytes | 8,657 bytes |
| | RAM(Note) | 26 bytes | 8 bytes | 56 bytes |
| | STACK (Note) | 192 bytes | - | 348 bytes |
| RX231 | ROM(Note) | 6,517 bytes | 7,688 bytes | 8,659 bytes |
| | RAM(Note) | 26 bytes | 8 bytes | 56 bytes |
| | STACK (Note) | 192 bytes | - | 348 bytes |
| RX65N | ROM(Note) | 6,517 bytes | 7,688 bytes | 8,652 bytes |
| | RAM(Note) | 26 bytes | 8 bytes | 58 bytes |
| | STACK (Note) | 192 bytes | - | 352 bytes |

Note. The sizes of ROM, RAM, and stack of TFAT FIT are included.

## 2.9 Arguments

Please use definition of drive number when calling TFAT FIT.

```
typedef enum
{
   TFAT_DRIVE_NUM_0 = 0x00,
   TFAT_DRIVE_NUM_1,
   TFAT_DRIVE_NUM_2,
   TFAT_DRIVE_NUM_3,
   TFAT_DRIVE_NUM_4,
   TFAT_DRIVE_NUM_5,
   TFAT_DRIVE_NUM_6,
   TFAT_DRIVE_NUM_7,
   TFAT_DRIVE_NUM_8,
   TFAT_DRIVE_NUM_9,
}TFAT_DRV_NUM;
```

## 2.10 Return Values

Return values are defined in "diskio.h" in TFAT FIT module.

```
/* Disk Status Bits (DSTATUS) */
typedef uint8_t DSTATUS;
```

- #define STA_NOINIT 0x01 /* Drive not initialized */
- #define STA_NODISK 0x02 /* No medium in the drive */
- #define STA_PROTECT 0x04 /* Write protected */

```
/* Results of Disk Functions */
typedef enum
{
        RES_OK = 0,    /* 0: Successful */
        RES_ERROR,  /* 1: R/W Error */
        RES_WRPRT,  /* 2: Write Protected */
        RES_NOTRDY,/* 3: Not Ready */
        RES_PARERR /* 4: Invalid Parameter */
} DRESULT;
```

## 2.11 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (2) or (4) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (3) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e$^2$ studio
By using the Smart Configurator in e$^2$ studio, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: e$^2$ studio (R20AN0451)” for details.

(2) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: CS+ (R20AN0470)” for details.

(3) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

(4) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to “RX Smart Configurator User’s Guide: IAREW (R20AN0535)” for details.

## 2.12 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows an example of description.

```
while statement example :
/* WAIT LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT LOOP */
for (i = 0; i < BSP REG PROTECT TOTAL ITEMS; i++)
{
    g protect counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy read(ether channel, PHY REG CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

## 3. API Functions

The following functions are called by TFAT FIT module. These functions (excluded some ones) call lower layer functions (4 Local API), which are prepared for each storage devices, according to the configuration. Note that application programs should not call the following APIs.

**Table 3.1 Functions List**

| Function Name | Function Overview |
|---|---|
| disk_initialize() | Initialize disk drive |
| disk_status() | Get disk status |
| disk_read() | Read sectors |
| disk_write() | Write sectors |
| disk_ioctl() | Control device dependent features |
| get_fattime() | Get current time |
| drv_change_alloc() | Change the allocation of drive number between TFAT module and device driver. |

## disk_initialize()

The disk_initialize() function is called to initialize the storage device.

### format

```
#include "ff.h"
DSTATUS disk_initialize (
  BYTE pdrv           /* [IN] Physical drive number */
);
```

### Parameters

pdrv

Physical drive number to identify the target device. Always zero at single drive system.

### Return Values

This function returns the current drive status flags as the result. For details of the drive status, refer to the disk_status() function.

### Properties

Prototyped in file “diskio.h”.

### Description

This function initializes the storage device and puts it ready to generic read/write. When the function succeeds, STA_NOINIT flag in the return value is cleared.

Remarks: This function needs to be under the control of FatFs module. Application program MUST NOT call this function, or FAT structure on the volume can be broken. To re-initialize the filesystem, use f_mount function instead.

### Example

None.

### Special Notes:

None.

## disk_status()

This function is called to inquire the current drive status.

### format

```
DSTATUS disk_status (
  BYTE pdrv     /* [IN] Physical drive number */
);
```

### Parameters

pdrv

Physical drive number to identify the target device. Always zero at single drive system.

### Return Values

The current drive status is returned in combination of status flags described below. FatFs refers only to STA_NOINIT and STA_PROTECT.

- STA_NOINIT: Indicates that the device has not been initialized and not ready to work. This flag is set on system reset, media removal or failure of disk_initialize() function. It is cleared on disk_initialize() function succeeded. Any media change that occurs asynchronously must be captured and reflected to the status flags, or auto-mount function will not work correctly. If the system does not support media change detection, the application program needs to explicitly re-mount the volume with f_mount() function after each media change.

- STA_NODISK: Indicates that there is no medium in the drive. This is always cleared at fixed disk drive. Note that FatFs does not refer to this flag.

- STA_PROTECT: Indicates that the medium is write-protected. This is always cleared at the drives without the write protect function. Not valid if STA_NODISK is set.

### Properties

Prototyped in file “diskio.h”.

### Description

None.

### Example

None.

### Special Notes:

None.

## disk_read()

This function is called to read data from the sector(s) of storage device.

### format

```
#include "ff.h"
DRESULT disk_read (
  BYTE pdrv,      /* [IN] Physical drive number */
  BYTE* buff,     /* [OUT] Pointer to the read data buffer */
  LBA_t sector,   /* [IN] Start sector number */
  UINT count      /* [IN] Number of sectors to read */
);
```

### Parameters

pdrv

Physical drive number to identify the target device.

buff

Pointer to the first item of the byte array to store read data. Size of read data will be the sector size * count bytes.

sector

Start sector number in 32-bit logical block address (LBA).

count

Number of sectors to read.

### Return Values

RES_OK

The function succeeded.

RES_ERROR

An unrecoverable hard error occurred during the read operation.

RES_PARERR

Invalid parameter.

RES_NOTRDY

The device has not been initialized.

### Properties

Prototyped in file “diskio.h”.

### Description

Read/write operation to the generic storage devices, such as memory card, hard disk and optical disk, is done in unit of block of data bytes called sector. FatFs supports the sector size in the range of 512 to 4096 bytes. When FatFs is configured for fixed sector size (FF_MIN_SS == FF_MAX_SS, this is the most case), the read/write function must work at that sector size. When FatFs is configured for variable sector size (FF_MIN_SS < FF_MAX_SS), the sector size of medium is inquired with disk_ioctl() function immediately following disk_initialize() function succeeded.

There are some considerations about the memory address passed via buff. It is not that always aligned to word boundary because the argument is defined as BYTE*. The unaligned transfer request can occur at direct transfer. If the bus architecture, especially DMA controller, does not allow unaligned memory access, it should be solved in this function. If it is the case, there are some workarounds described below to avoid this issue.

Convert word transfer to byte transfer with some method in this function. - Recommended.

On the f_read() calls, avoid long read request that includes a whole sector. - Any direct transfer never occurs.

On the f_read(fp, dat, btw, bw) calls, make sure that (((UINT)dat & 3) == (f_tell(fp) & 3)) is true. - Word alignment of buff is guaranteed.

Also, the memory area may be out of reach in DMA. This is the case if it is in tightly coupled memory which is usually used for stack. Use double buffered transfer or avoid to define any file I/O buffer includes FatFs and FIL structure as local variables where on the stack.

Generally, a multiple sector read request must not be split into single sector transactions to the storage device, or read throughput gets worse.

### Example

None.

### Special Notes:

None.

## disk_write()

This function is called to write data to the sector(s) of storage device.

### format

```
DRESULT disk_write (
  BYTE pdrv,        /* [IN] Physical drive number */
  const BYTE* buff, /* [IN] Pointer to the data to be written */
  LBA_t sector,     /* [IN] Sector number to write from */
  UINT count        /* [IN] Number of sectors to write */
);
```

### Parameters

pdrv

Physical drive number to identify the target device.

buff

Pointer to the first item of the byte array to be written. The size of data to be written is sector size * count bytes.

sector

Start sector number in 32-bit logical block address (LBA).

count

Number of sectors to write.

### Return Values

RES_OK

The function succeeded.

RES_ERROR

An unrecoverable hard error occurred during the read operation.

RES_WRPRT

The device is write-protected.

RES_PARERR

Invalid parameter.

RES_NOTRDY

The device has not been initialized.

### Properties
Prototyped in file “diskio.h”.

### Description
The specified memory address is not that always aligned to word boundary because the argument is defined as BYTE*. For more information, refer to the description of disk_read() function.

Generally, a multiple sector write request (count > 1) must not be split into single sector transactions to the storage device, or the file write throughput will be drastically decreased.

FatFs expects delayed write function of the disk control layer. The write operation to the media does not need to be completed at return from this function by what write operation is in progress or data is only stored into the write-back cache. But writing data on the buff is invalid after return from this function. The write completion request is done by CTRL_SYNC command of disk_ioctl() function. Therefore, if a delayed write function is implemented, the write throughput of the filesystem will be improved.

Remarks: Application program MUST NOT call this function, or FAT structure on the volume can be collapsed.

### Example
None.

### Special Notes:
This function is not needed when FF_FS_READONLY = 1.

## disk_ioctl()

This function is called to control device specific features and miscellaneous functions other than generic read/write.

### format

```
DRESULT disk_ioctl (
  BYTE pdrv,      /* [IN] Drive number */
  BYTE cmd,       /* [IN] Control command code */
  void* buff      /* [I/O] Parameter and data buffer */
);
```

### Parameters

pdrv
Physical drive number to identify the target device.

cmd
Command code.

buff
Pointer to the parameter depends on the command code. Do not care if the command has no parameter to be passed.

### Return Values

RES_OK

The function succeeded.

RES_ERROR

An error occurred.

RES_PARERR

The command code or parameter is invalid.

RES_NOTRDY

The device has not been initialized.

### Properties

Prototyped in file “diskio.h”.

### Description

The FatFs module requires only five device general-purpose commands described below.

**Table 3.2 general-purpose commands**

| Command | Describe |
|---|---|
| CTRL_SYNC | Make sure that the device has finished pending write process. If the disk I/O module or storage device has a write-back cache, the cached data marked dirty must be written back to the media immediately. Nothing to do for this command if each write operation to the media is completed within the disk_write function. |
| GET_SECTOR_SIZE | Returns number of available sectors on the drive into the DWORD variable pointed by buff. This command is used by f_mkfs and f_fdisk function to determine the volume/partition size to be created. Required at FF_USE_MKFS = 1. |
| GET_SECTOR_COUNT | Returns sector size of the device into the WORD variable pointed by buff. Valid return values for this command are 512, 1024, 2048 and 4096. This command is required only if FF_MAX_SS > FF_MIN_SS. When FF_MAX_SS = FF_MIN_SS, this command is never used and the device must work at that sector size. |
| GET_BLOCK_SIZE | Returns erase block size of the flash memory media in unit of sector into the DWORD variable pointed by buff. The allowable value is 1 to 32768 in power of 2. Return 1 if the erase block size is unknown or non flash memory media. This command is used by only f_mkfs function and it attempts to align data area on the erase block boundary. Required at FF_USE_MKFS = 1. |
| CTRL_TRIM | Informs the device the data on the block of sectors is no longer needed and it can be erased. The sector block is specified by a DWORD array {<start sector>, <end sector>} pointed by buff. This is an identical command to Trim of ATA device. Nothing to do for this command if this function is not supported or not a flash memory device. FatFs does not check the result code and the file function is not affected even if the sector block was not erased well. This command is called on remove a cluster chain and in the f_mkfs function. Required at FF_USE_TRIM = 1. |

### Example
None.

### Special Notes:
The disk_ioctl() function is not needed when FF_FS_READONLY = 1 and FF_MAX_SS = FF_MIN_SS.

## get_fattime()

This function is called to get the current time.

### format

```
DWORD get_fattime (void);
```

### Parameters

None.

### Return Values

Current local time shall be returned as bit-fields packed into a DWORD value. The bit fields are as follows:

- bit31:25 Year origin from the 1980 (0..127, e.g. 37 for 2017)
- 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, e.g. 25 for 50)

### Properties

Prototyped in file “ff.h”.

### Description

The get_fattime() function shall return any valid time even if the system does not support a real time clock. If a zero is returned, the file will not have a valid timestamp.

### Example

None.

### Special Notes:

This function is not needed when FF_FS_READONLY = 1 or FF_FS_NORTC = 1.

## drv_change_alloc()

This function changes a drive's allocation. This function has nothing to do with FatFs and is a unique API of Renesas.

### format

```
DRESULT drv_change_alloc(TFAT_DRV_NUM tfat_drv, uint8_t dev_type,
                         uint8_t dev_drv_num);
```

### Parameters

tfat_drv

Th physical drive number for TFAT FIT.

dev_type

The device defines type (TFAT_USB_DRIVE_NUM, TFAT_SDMEM_DRIVE_NUM, or TFAT_USB_MINI_DRIVE_NUM).

dev_drv_num

The drive number/device channel for device driver.

### Return Values

RES_OK

The function succeeded.

RES_PARERR

The specified value of tfat_drv is invalid.

### Properties

Prototyped in file "r_tfat_driver_rx_if.h".

### Description

The drive used for TFAT FIT is specified by the TFAT_DRIVE_ALLOC_NUM_i definition in r_tfat_driver_rx_config.h, and the drive number for TFAT FIT is associated with the drive number of the memory driver.

Drive numbers for memory drivers are automatically assigned in ascending order.

Use this function if you want to change the association dynamically.

### Example

None.

### Special Notes:

None.

# 4. Local API

There are functions for SD memory card (SD mode), SD memory card (SPI mode), USB memory, USB Mini, eMMC, and Serial Flash memory. Each function calls memory driver functions.

## 4.1 For USB Memory

Table 4.1 Functions List are called when Section 2.7 Configuration Overview TFAT_USB_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i(i=0-9) have the settings “TFAT_CTRL_USB”.

**Table 4.1 Functions List**

| Function name | Function Overview |
|---|---|
| usb_disk_initialize() | Initialize disk drive |
| usb_disk_read() | Read sectors |
| usb_disk_write() | Write sectors |
| usb_disk_ioctl() | Control device dependent features |
| usb_disk_status() | Get disk status |

**Table 4.2 Other Functions List**

| Function name | Function Overview |
|---|---|
| R_usb_hmsc_WaitLoop() | Wait for read and write |

### 4.1.1 usb_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS usb_disk_initialize (uint8_t pdrv);
```

**Parameters**

| | | |
|---|---|---|
| pdrv | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This API does not call USB driver initialize function because of USB driver limitation (1 time call is only accepted). Please call USB driver initialize function in user program.

**Special Notes:**

None.

### 4.1.2 usb_disk_read()

This function reads the data from disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT usb_disk_read (   uint8_t   pdrv,
                          uint8_t   *buff,
                          uint32_t  sector,
                          uint8_t    count
);
```

**Parameters**

| | | |
|---|---|---|
| pdrv | input | Specifies the physical drive number. |
| buff | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector | input | Specifies the start sector number in logical block address (LBA). |
| count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

This function reads the data from disk drive. The position of read data is specified using this function argument.

**Special Notes:**

None.

### 4.1.3 usb_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT usb_disk_write (    uint8_t    pdrv,
                            uint8_t    *buff,
                            uint32_t   sector,
                            uint8_t     count
);
```

**Parameters**

| | | |
|---|---|---|
| pdrv | input | Specifies the physical drive number. |
| buff | input | Pointer to the data to be written. |
| sector | input | Specifies the start sector number in logical block address (LBA). |
| count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

This function writes the data to the disk drive. The position of write data is specified using this function argument.

**Special Notes:**

None.

### 4.1.4 usb_disk_ioctl()

This function controls the drive.

#### Format

```
#include "r_tfat_drv_if_dev.h"
DRESULT usb_disk_ioctl (  uint8_t   pdrv,
                          uint8_t   cmd,
                          void      *buff

);
```

#### Parameters

| | | |
|---|---|---|
| pdrv | input | Specifies the physical drive number. |
| cmd | input | Specifies the command code. The command code will always be 0. |
| buff | input | Pointer should always be a NULL pointer. |

#### Return Value

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

#### Description

The usb_disk_ioctl function is used only by the f_sync function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

For users who wish to use f_sync function in their applications, this particular driver interface function will have to be implemented. This driver function should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

#### Special Notes:

None.

### 4.1.5 usb_disk_status()

This function gets the information about disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS usb_disk_status (uint8_t  pdrv);
```

**Parameters**

| | | |
|---|---|---|
| pdrv | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

### 4.1.6 R_usb_hmsc_WaitLoop()

This function waits for the data read/write.

**Format**

```
void  R_usb_hmsc_WaitLoop (void );
```

**Parameters**

None.

**Return Value**

None.

**Description**

Please refer to the USB driver document for details.

**Special Notes:**

None.

## 4.2 For SD Memory Card

Table 4.3 List of Functions are called when Section 2.7 Configuration Overview TFAT_SDMEM_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i (i=0-9) have the settings “TFAT_CTRL_SDMEM”.

**Table 4.3 List of Functions**

| Function Name | Outline |
|---|---|
| sdmem_disk_initialize() | Initialize disk drive |
| sdmem_disk_read() | Read sectors |
| sdmem_disk_write() | Write sectors |
| sdmem_disk_ioctl() | Control device dependent features |
| sdmem_disk_status() | Get disk status |

[Notice about SD memory card]

This module does not execute mount process and VDD power supply process. Please refer to the SD memory card module document and please implement. Otherwise, this module works abnormally.

### 4.2.1 sdmem_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS sdmem_disk_initialize (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function does not execute the SD memory card driver initialize. Please implement SD memory card initialize code in user code.

**Special Notes:**

None.

### 4.2.2 sdmem_disk_read()

This function reads the data from disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT sdmem_disk_read  (     uint8_t    drive,
                               uint8_t   *buffer,
                               uint32_t   sector_number,
                               uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Read data from SD memory by block.

**Special Notes:**

None.

### 4.2.3 sdmem_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT sdmem_disk_write (     uint8_t    drive,
                               uint8_t    *buffer,
                               uint32_t   sector_number,
                               uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | input | Pointer to the data to be written. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Write the data to the SD memory by block.

**Special Notes:**

None.

### 4.2.4 sdmem_disk_ioctl()

This function controls the drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT sdmem_disk_ioctl (     uint8_t  drive,
                               uint8_t  command,
                               void      *buffer
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| command | input | Specifies the command code. The command code will always be 0. |
| buffer | input | Pointer should always be a NULL pointer. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

The sdmem_disk_ioctl function is used only by the f_sync function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

For users who wish to use f_sync function in their applications, this particular driver interface function will have to be implemented. This driver function should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

**Special Notes:**

None.

### 4.2.5 sdmem_disk_status()

This function gets the disk drive status.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DSTATUS sdmem_disk_status (uint8_t  drive
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Other | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

## 4.3 For SPI mode SD Memory Card

Table 4.4 List of Functions are called when Section 2.7 Configuration Overview TFAT_SPI_SDMEM_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i (i=0-9) have the settings “TFAT_CTRL_SPI_SDMEM”.

**Table 4.4 List of Functions**

| Function Name | Outline |
|---|---|
| spi_sdmem_disk_initialize() | Initialize disk drive |
| spi_sdmem_disk_read() | Read sectors |
| spi_sdmem_disk_write() | Write sectors |
| spi_sdmem_disk_ioctl() | Control device dependent features |
| spi_sdmem_disk_status() | Get disk status |

[Notice about SPI mode SD memory card]

This module does not execute mount process and VDD power supply process. Please refer to the SPI mode SD memory card module document and please implement. Otherwise, this module works abnormally.

### 4.3.1 spi_sdmem_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS spi_sdmem_disk_initialize (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function does not execute the SPI mode SD memory card driver initialize. Please implement SPI mode SD memory card initialize code in user code.

**Special Notes:**

None.

### 4.3.2 spi_sdmem_disk_read()

This function reads the data from disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT spi_sdmem_disk_read  (     uint8_t    drive ,
                                   uint8_t   *buffer,
                                   uint32_t   sector_number,
                                   uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Read data from SPI mode SD memory by block.

**Special Notes:**

None.

### 4.3.3 spi_sdmem_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT spi_sdmem_disk_write (     uint8_t    drive,
                                   uint8_t    *buffer,
                                   uint32_t   sector_number,
                                   uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | input | Pointer to the data to be written. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Write the data to the SPI mode SD memory by block.

**Special Notes:**

None.

### 4.3.4 spi_sdmem_disk_ioctl()

This function controls the drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT spi_sdmem_disk_ioctl (     uint8_t  drive ,
                                   uint8_t  command ,
                                   void      *buffer
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| command | input | Specifies the command code. The command code will always be 0. |
| buffer | input | Pointer should always be a NULL pointer. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

The spi_sdmem_disk_ioctl function is used only by the f_sync function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

For users who wish to use f_sync function in their applications, this particular driver interface function will have to be implemented. This driver function should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

**Special Notes:**

None.

### 4.3.5 spi_sdmem_disk_status()

This function gets the disk drive status.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DSTATUS spi_sdmem_disk_status (uint8_t  drive
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

## 4.4 For USB Mini

Table 4.5 Functions List are called when Section 2.7 Configuration Overview TFAT_USB_MINI_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i(i=0-9) have the settings “TFAT_CTRL_USB_MINI”.

**Table 4.5 Functions List**

| Function name | Function Overview |
|---|---|
| usb_mini_disk_initialize() | Initialize disk drive |
| usb_mini_disk_read() | Read sectors |
| usb_mini_disk_write() | Write sectors |
| usb_mini_disk_ioctl() | Control device dependent features |
| usb_mini_disk_status() | Get disk status |

**Table 4.6 Other Function List**

| Function name | Function Overview |
|---|---|
| R_usb_mini_hmsc_WaitLoop() | Wait for read and write |

### 4.4.1 usb_mini_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS usb_mini_disk_initialize (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This API does not call USB driver initialize function because of USB driver limitation (1 time call is only accepted). Please call USB driver initialize function in user program.

**Special Notes:**

None.

### 4.4.2 usb_mini_disk_read()

This function reads the data from disk.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DRESULT usb_mini_disk_read (       uint8_t   drive ,
                                      uint8_t   *buffer ,
                                      uint32_t  sector_number ,
                                      uint8_t    sector_count
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

This function reads the data from disk drive. The position of read data is specified using this function argument.

**Special Notes:**

None.

### 4.4.3 usb_mini_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT usb_mini_disk_write (     uint8_t    drive ,
                                  uint8_t    *buffer ,
                                  uint32_t   sector_number ,
                                  uint8_t     sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | input | Pointer to the data to be written. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

This function writes the data to the disk drive. The position of write data is specified using this function argument.

**Special Notes:**

None.

### 4.4.4 usb_mini_disk_ioctl()

This function controls the drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT usb_mini_disk_ioctl (        uint8_t  drive,
                                     uint8_t  command,
                                     void      *buffer
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| command | input | Specifies the command code. The command code will always be 0. |
| buffer | input | Pointer should always be a NULL pointer. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

The usb_mini_disk_ioctl function is used only by the f_sync function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

For users who wish to use f_sync function in their applications, this particular driver interface function will have to be implemented. This driver function should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

**Special Notes:**

None.

### 4.4.5 usb_mini_disk_status()

This function gets the information about disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS usb_mini_disk_status (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

### 4.4.6 R_usb_mini_hmsc_WaitLoop()

This function waits for the data read/write.

**Format**

```
void  R_usb_mini_hmsc_WaitLoop (void );
```

**Parameters**

None.

**Return Value**

None.

**Description**

Please refer to the USB driver document for details.

**Special Notes:**

None.

## 4.5 For eMMC

Table 4.7 List of Functions are called when Section 2.7 Configuration Overview TFAT_MMC_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i (i=0-9) have the settings “TFAT_CTRL_MMC”.

**Table 4.7 List of Functions**

| Function Name | Outline |
|---|---|
| mmcif_disk_initialize() | Initialize disk drive |
| mmcif_disk_read() | Read sectors |
| mmcif_disk_write() | Write sectors |
| mmcif_disk_ioctl() | Control device dependent features |
| mmcif_disk_status() | Get disk status |

### 4.5.1 mmcif_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS mmcif_disk_initialize (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function does not execute the eMMC driver initialize. Please implement eMMC initialize code in user code.

**Special Notes:**

None.

### 4.5.2 mmcif_disk_read()

This function reads the data from disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT mmcif_disk_read  (     uint8_t    drive ,
                               uint8_t   *buffer,
                               uint32_t   sector_number,
                               uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Read data from eMMC by block.

**Special Notes:**

None.

### 4.5.3 mmcif_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT mmcif_disk_write (     uint8_t    drive ,
                               uint8_t    *buffer,
                               uint32_t   sector_number,
                               uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | input | Pointer to the data to be written. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Write the data to the eMMC by block.

**Special Notes:**

None.

### 4.5.4 mmcif_disk_ioctl()

This function controls the drive.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DRESULT mmcif_disk_ioctl (     uint8_t  drive,
                                  uint8_t  command,
                                  void      *buffer
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| command | input | Specifies the command code. The command code will always be 0. |
| buffer | input | Pointer should always be a NULL pointer. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

The mmcif_disk_ioctl function is used only by the f_sync or f_mkfs function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

The command CTRL_SYNC should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

For other commands, refer to Table 3.2 general-purpose commands.

**Special Notes:**

None.

### 4.5.5 mmcif_disk_status()

This function gets the disk drive status.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DSTATUS mmcif_disk_status (uint8_t  drive
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

## 4.6 For Serial Flash Memory

Table 4.8 List of Functions are called when Section 2.7 Configuration Overview TFAT_SERIAL_FLASH_DRIVE_NUM and TFAT_DRIVE_ALLOC_NUM_i (i=0-9) have the settings “TFAT_CTRL_SERIAL_FLASH”.

**Table 4.8 List of Functions**

| Function Name | Outline |
|---|---|
| flash_spi_disk_initialize() | Initialize disk drive |
| flash_spi_disk_read() | Read sectors |
| flash_spi_disk_write() | Write sectors |
| flash_spi_disk_ioctl() | Control device dependent features |
| flash_spi_disk_status() | Get disk status |

**Table 4.9 Other Functions List**

| Function name | Function Overview |
|---|---|
| flash_spi_1ms_interval() | Update internal timer per 1 ms |

### 4.6.1 flash_spi_disk_initialize()

This function initializes the disk drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DSTATUS flash_spi_disk_initialize (uint8_t drive);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the initialize drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function does not execute the Serial Flash memory driver initialize. Please implement Serial Flash memory initialize code in user code.

**Special Notes:**

None.

### 4.6.2 flash_spi_disk_read()

This function reads the data from disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT flash_spi_disk_read  (     uint8_t    drive,
                                   uint8_t   *buffer,
                                   uint32_t   sector_number,
                                   uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | output | Pointer to the read buffer to store the read data. A buffer of the size equal to the number of bytes to be read is required. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to read. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Read data from Serial Flash memory by block.

**Special Notes:**

None.

### 4.6.3 flash_spi_disk_write()

This function writes the data to the disk.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT flash_spi_disk_write (     uint8_t    drive,
                                   uint8_t    *buffer,
                                   uint32_t   sector_number,
                                   uint8_t    sector_count
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| buffer | input | Pointer to the data to be written. |
| sector_number | input | Specifies the start sector number in logical block address (LBA). |
| sector_count | input | Specifies number of sectors to write. The value can be 1 to 255. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

Write the data to the Serial Flash memory by block.

**Special Notes:**

None.

### 4.6.4 flash_spi_disk_ioctl()

This function controls the drive.

**Format**

```
#include "r_tfat_drv_if_dev.h"
DRESULT flash_spi_disk_ioctl (     uint8_t  drive,
                                   uint8_t  command,
                                   void      *buffer
);
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |
| command | input | Specifies the command code. The command code will always be 0. |
| buffer | input | Pointer should always be a NULL pointer. |

**Return Value**

| | |
|---|---|
| DRESULT | Result of the function execution as explained in section 2.10 Return Values. |

**Description**

The flash_spi_disk_ioctl function is used only by the f_sync or f_mkfs function amongst all the TFAT FIT functions. Users who do not plan to use f_sync function in their applications can skip the implementation for this particular driver interface function.

For users who wish to use f_sync function in their applications, the command CTRL_SYNC has to be implemented.

The command CTRL_SYNC should consist of the code to finish off any pending write process. If the disk i/o module has a write back cache, the dirty sector must be flushed immediately. The f_sync function will perform a save operation to the unsaved data related to the file object passed as argument.

For other commands, refer to Table 3.2 general-purpose commands.

**Special Notes:**

None.

### 4.6.5 flash_spi_disk_status()

This function gets the disk drive status.

**Format**

```
   #include "r_tfat_drv_if_dev.h"
   DSTATUS flash_spi_disk_status (uint8_t  drive
 );
```

**Parameters**

| | | |
|---|---|---|
| drive | input | Specifies the physical drive number. |

**Return Value**

| | |
|---|---|
| TFAT_RES_OK | Normal termination. |
| Others | DSTATUS status of the disk after function execution as explained in section 2.10 Return Values. |

**Description**

This function should consist of the code that checks the disk and returns the current disk status. The disk status can have any of the three values as explained in section 2.10 Return Values. The disk status can be returned by updating the return value with the macros related to disk status.

**Special Notes:**

None.

### 4.6.6 flash_spi_1ms_interval()

This function updates the internal timer per 1 ms.

**Format**

```
#include "r_tfat_drv_if_dev.h"
void flash_spi_1ms_interval (void);
```

**Parameters**

None.

**Return Value**

None.

**Description**

This function updates the internal timer of the TFAT driver FIT per 1 ms. This timer is used to check the busy status of the serial flash memory.

**Special Notes:**

None.

## 5. Pin Settings

The TFAT driver FIT has no pin settings.

# 6. Appendices

## 6.1 Confirmed Operation Environment

This section describes the operation confirmation environment for TFAT driver FIT.

**Table 6.1 Confirmed Operation Environment (Rev.1.05 for SD Memory Card Driver and USB Mini Driver)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio V6.3.0 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00 |
| | Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.1.05 |
| Board used | Renesas Starter Kit+ for RX64M (product No.:R0K50564MSxxxxx)<br>Renesas Starter Kit for RX231 (product No.:R0K505231Sxxxxx) |
| RTOS | None |

**Table 6.2 Confirmed Operation Environment (Rev.1.05 for USB Driver)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio V7.3.0 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 |
| | Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.1.05 |
| Board used | Renesas Starter Kit+ for RX64M (product No.:R0K50564MSxxxxx) |
| RTOS | None |

**Table 6.3 Confirmed Operation Environment (Rev. 2.00)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 7.7.0<br>IAR Embedded Workbench for Renesas RX 4.13.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.201904<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.13.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.00 |
| Board used | Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx) |
| RTOS | FreeRTOS V10.0.00<br>RI600V4 V1.06.00 |

**Table 6.4 Confirmed Operation Environment (Rev. 2.10)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 7.8.0<br>IAR Embedded Workbench for Renesas RX 4.14.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.201904<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.14.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.10 |
| Board used | Renesas Starter Kit for RX231 (product No.: RTK55231xxxxxxxxxx)<br>Renesas Starter Kit+ for RX64M (product No.: RTK5564Mxxxxxxxxxx) |
| RTOS | FreeRTOS V10.0.00<br>RI600V4 V1.06.00 |

**Table 6.5 Confirmed Operation Environment (Rev. 2.20)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 2020-07<br>IAR Embedded Workbench for Renesas RX 4.14.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202002<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.14.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.20 |
| Board used | Renesas Starter Kit for RX231 (product No.: RTK55231xxxxxxxxx)<br>Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxx)<br>Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxx) |
| RTOS | FreeRTOS V10.0.03<br>RI600V4 V1.06.00 |

**Table 6.6 Confirmed Operation Environment (Rev. 2.30)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 2023-07<br>IAR Embedded Workbench for Renesas RX 4.20.3 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202305<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.20.3<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.30 |
| Board used | Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxx) |
| RTOS | FreeRTOS V10.4.3<br>RI600V4 V1.06.01 |

**Table 6.7 Confirmed Operation Environment (Rev. 2.40)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 2023-10<br>IAR Embedded Workbench for Renesas RX 4.20.3 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202305<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 4.20.3<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.40 |
| Board used | Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx) |
| RTOS | None |

**Table 6.8 Confirmed Operation Environment (Rev. 2.50)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 2024-07<br>IAR Embedded Workbench for Renesas RX 5.10.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202405<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 5.10.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.50 |
| Board used | Renesas Target Board for RX140 (product No.: RTK5RX1400xxxxxxxx)<br>Renesas Starter Kit+ for RX140 (product No.: RTK551406Bxxxxxxxx)<br>Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx)<br>Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxxx)<br>Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx) |
| RTOS | FreeRTOS V10.4.3 |

**Table 6.9 Confirmed Operation Environment (Rev. 2.60)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 2024-07<br>IAR Embedded Workbench for Renesas RX 5.10.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 8.3.0.202405<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99 |
| | IAR C/C++ Compiler for Renesas RX version 5.10.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.60 |
| Board used | None |
| RTOS | None |

## 6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”

- Using e$^2$ studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_tfat_driver_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_tfat_driver_rx_config.h” may be wrong. Check the file “r_tfat_driver_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7 Configuration Overview for details.

(4) Q: The pin setting is supposed to be done, but this does not look like it.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 5 Pin Settings for details.

## 7. Reference Documents

User's Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

## Related Technical Updates

This module reflects no technical updates.

## Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Dec 01, 2014 | - | First edition issued |
| 1.01 | Jan 05, 2015 | - | Added support MCUs. |
| 1.02 | Jun 30, 2015 | - | Added support MCU RX231. |
| 1.03 | Oct 01, 2016 | - | Added support RX family. |
| 1.04 | Jun 29, 2018 | - | 1.2.2 Fig.1-1 Added System timer and CMT modules.<br>1.3 Added API Overview<br>2.6 Changed SD memory card define name.<br>2.7 Added Code Size<br>3.6 Modified get_fattime() description.<br>4.2.1-4.2.5 Changed API name.<br>5 Added Appendices<br>6 Added Reference Documents |
| 1.05 | Dec 14, 2018 | - | Revision up by USB driver supporting RTOS. |
| 2.00 | Feb. 25, 2020 | - | Supported the following compilers.<br>- GCC for Renesas RX<br>- IAR C/C++ Compiler for Renesas RX<br>Supported the following RTOS.<br>- FreeRTOS<br>- RI600V4<br>Removed "R_TFAT_" from the function names. |
| 2.10 | Jul. 27, 2020 | - | Supported the following storage devices.<br>- eMMC<br>- Serial Flash memory<br>Supported the format function for following storage devices.<br>- eMMC<br>- Serial Flash memory<br>Supported sector size 4096 bytes.<br>Supported the following RTOS with using USB mini FIT.<br>- FreeRTOS<br>- RI600V4 |
| 2.20 | Sep. 10, 2020 | -<br><br><br>Program | Supported the format function for following storage devices.<br>- SD memory card<br>- USB<br>Modified the TFAT driver FIT module due to the software issue<br><br>[Description]<br>The BLOCK SIZE value of eMMC obtained by the GET_BLOCK_SIZE command is incorrect.<br><br>[Conditions]<br>The following two conditions are met:<br>- Rev.2.10 version of the TFAT driver FIT module is used.<br>- Format eMMC.<br><br>[Workaround]<br>Please use Rev.2.20 or a later version of the TFAT driver FIT module. |

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 2.20 | Sep. 10, 2020 | Program | Modified the TFAT driver FIT module due to the software issue<br><br>[Description]<br>The BLOCK SIZE value of Serial Flash memory obtained by the GET_BLOCK_SIZE command is incorrect.<br>If the size of the Serial Flash memory is very large, the format processing time will be long and the available memory will be small.<br><br>[Conditions]<br>The following two conditions are met:<br>- Rev.2.10 version of the TFAT driver FIT module is used.<br>- Format Serial Flash memory.<br><br>[Workaround]<br>Please use Rev.2.20 or a later version of the TFAT driver FIT module. |
| 2.30 | Aug. 31, 2023 | 14<br><br>18, 20<br>Program | Deleted the description of FIT configurator from "2.11 Adding the FIT Module to Your Project"<br>Updated the open source base version from V0.13c to V0.15.<br>Updated the open source base version from V0.13c to V0.15. |
| 2.40 | Dec. 15, 2023 | 8<br>12<br><br>13<br>16<br>65<br><br>Program | Updated FIT module version for Table 1.2<br>Updated the sector size configuration option when using TFAT and Serial Flash memory.<br>Updated the section of 2.8 Code Size.<br>Added 2.12 “for”, “while” and “do while” statements.<br>6.1 Confirmed Operation Environment:<br>Added Table for Rev.2.40<br>Added support Serial Flash FAT sector size selectable. |
| 2.50 | Sep 16, 2024 | 2, 8, 29<br>3<br>7<br>9<br><br><br>12<br><br>14<br>42-47<br>72<br><br>Program | Updated to support SPI mode SD card.<br>Added related document for SPI mode SD card module.<br>Updated FIT module version<br>Updated to support SPI mode SD card module for Table 1.2<br>Updated FIT module version for Table 1.2<br>Added configuration option when using TFAT and SPI mode SD card memory.<br>Updated the section of 2.8 Code Size.<br>Added support SPI mode SD card<br>6.1 Confirmed Operation Environment:<br>Added Table for Rev.2.50<br>Added support SPI mode SD card. |

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 2.60 | Nov 01, 2024 | 7, 9<br>14<br>73<br><br>Program | Updated FIT module version<br>Updated the section of 2.8 Code Size.<br>6.1 Confirmed Operation Environment:<br>Added Table for Rev.2.60<br>Changed the comment of API functions to the Doxygen style. |

# 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. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

# Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

© 2024 Renesas Electronics Corporation. All rights reserved.