RX ファミリ

オープンソース FAT ファイルシステム M3S-TFAT-Tiny モジュール

Firmware Integration Technology

要旨

本アプリケーションノートでは、Firmware Integration Technology (FIT)を使用した RX ファミリ オープンソース FAT ファイルシステム M3S-TFAT-Tiny (注) モジュールについて説明します。

注：マイクロソフト社の Transaction-Safe FAT File System (TFAT)とは無関係です。

本書では以下のとおり用語を使い分けます。

・TFAT FIT：
RX ファミリ用 オープンソース FAT ファイルシステム M3S-TFAT-Tiny モジュール(R20AN0038)

・TFAT driver FIT：
M3S-TFAT-Tiny メモリドライバインタフェースモジュール Firmware Integration Technology（R20AN0335）

・TFAT：
M3S-TFAT-Tiny、または、TFAT FIT と TFAT driver FIT の総称

TFAT FIT は FAT ファイルシステムソフトウェアであり、オープンソースである FatFs が含まれています。

FAT ファイルシステムの LFN (long file name)機能は Microsoft Corporation の特許ですが、FatFs のアプリケーションノートによると、関連する特許はすべて期限切れになっており、LFN 機能の使用はどのプロジェクトでも無料になりました。

対象デバイス

・RX ファミリ
本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様に合わせて変更し、十分評価してください。

対象コンパイラ

・Renesas Electronics C/C++ Compiler Package for RX Family

・GCC for Renesas RX

・IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認内容については 7.1 動作確認環境を参照してください。

目次

1. 概要

TFAT は、省メモリタイプの FAT ファイルシステムソフトウェアです。

TFAT は、FatFs をベースに作成されています。

1.1 FatFs とは？

FatFs は小規模な組み込みシステム向けの汎用 FAT ファイルシステム・モジュールです。FatFs は ChaN 氏によって開発されたソフトウェアです。ChaN 氏は FatFs を含め、インターネット上にて無償で組み込み向けソフトウェアを提供されています。ChaN 氏および、FatFs の公式 Web ページは以下になります。

http://elm-chan.org/fsw/ff/00index_e.html (注)

また、FatFs 仕様に関するドキュメント（FatFs アプリケーションノート、構成オプション設定、API 関数詳説など）は、以下 URL からダウンロードできます。表 1.1 TFAT FIT の仕様の「オープンソースのベース」項目に記載されている FatFs バージョンに合致したドキュメントをダウンロードしてください。

http://elm-chan.org/fsw/ff/archives.html
または、TFAT FIT の製品ページにてダウンロードできる本製品に同梱されています。

https://www.renesas.com/jp/ja/products/software-tools/software-os-middleware-driver/file-system/m3s-tfat-tiny-for-rx.html

注：旧バージョンの FatFs に基づいて日本語に翻訳された Web ページも存在します。最新の FatFs との差異に十分注意してご参照ください。
http://irtos.sourceforge.net/FAT32_ChaN/doc/00index_j.html

1.2 TFAT 仕様

1.2.1 TFAT の仕様

TFAT の主な仕様を以下に示します。

表 1.1 TFAT の仕様

大項目	項目	仕様
環境	デバイス	RX ファミリ
	コンパイラ	CC-RX
		GCC
		IAR
	記録メディア	SD カード（SD モード）
		USB メモリ
		eMMC
		Serial Flash memory
	FIT 依存関係	TFAT FIT Rev.4.02 - TFAT driver FIT Rev.2.20
	RTOS	RTOS なし
		FreeRTOS（注 1）
		RI600V4（RX ファミリ用 µITRON）（注 1）
ファイルシステム	ベースプログラム	Fatfs R0.13c（オープンソース）
	ドライブ数	最大 10
	FAT サブタイプ	FAT16
		FAT32
	セクタサイズ	512 bytes
		4096 bytes
	ファイル名形式	8.3 形式（ファイル名：8 文字、拡張子：3 文字）
		LFN 形式（long file name、ファイル名：最大 255 文字）
	ファイルパス形式	絶対パス
	フォーマット機能	あり
	マルチパーティション機能	なし

注 1：SD カード（SD モード）、USB メモリのみ

1.2.2 モジュール構成

モジュール構成を以下に示します。

TFAT FIT は、TFAT driver FIT、システムタイマモジュール FIT、および、各種デバイス・ドライバ FIT を併用して動作します。

TFAT FIT はオープンソース FatFs を内部に含む、ファイルシステムの主モジュールです。TFAT driver FIT は内部に Wrapper 関数をもち、記憶デバイスごとにファイルシステム用の I/O 処理を切り替えます。ユーザは TFAT driver FIT のコンフィギュレーション設定で使用する記憶デバイスを設定し、TFAT FIT の API を介してファイルシステムを操作します。

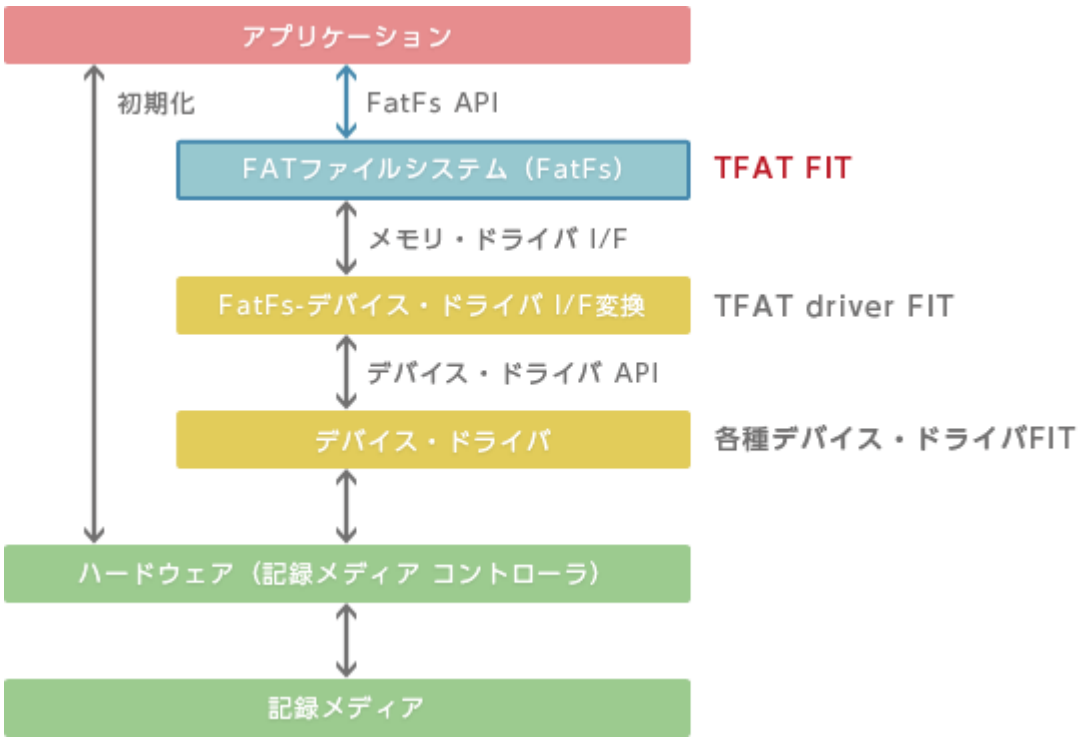

図 1.1 TFAT FIT の構成

1.2.3 FatFs 設定オプション

FatFs には複数の設定オプション（ファイル名など）があり、”ffconf.h”にて定義されています。これらの定義の値を変更することで使用可能な API や FatFs の機能をカスタマイズできます。スマート・コンフィグレータを使用した設定変更はできません。

弊社にて動作確認済みの FatFs 機能を表 1.2 ドライブ数に関連する FatFs 設定オプション ～ 表 1.7 フォーマット機能に関連する FatFs 設定オプション に示します。

また、設定オプションにて使用可能な API の一覧は 1.3 API の概要 をご覧ください。

(1) ドライブ数

ドライブ数は 1 以上 10 以下（FatFs 仕様）、かつ、TFAT driver FIT で定義する各記憶メディアの合計ドライブ数(注 1)以下（TFAT 仕様）である必要があります。

表 1.2 ドライブ数に関連する FatFs 設定オプション

定義名	定義値の許容値	定義値の意味	デフォルト値
FF_VOLUMES	1 以上 10 以下	ドライブ数(注 1)	1

注 1： TFAT_USB_DRIVE_NUM + TFAT_SDMEM_DRIVE_NUM +
TFAT_USB_MINI_DRIVE_NUM + TFAT_MMC_DRIVE_NUM +
TFAT_SERIAL_FLASH_DRIVE_NUM

(2) ファイル名形式

8.3 形式と 3 種類の LFN（long file name）形式、計 4 種類を選択できます。LFN を有効にすると、選択されたコード・ページに応じてモジュール・サイズが増大します。詳細は 1.1 FatFs とは？ にて記載した FatFs Web ページをご覧ください。

なお、FatFs のアプリケーションノートによると、LFN 形式はかつて Microsoft が特許を保持していましたが、現在は特許が切れており無料で利用できます。

表 1.3 ファイル名形式に関連する FatFs 設定オプション

定義名	定義値の許容値	定義値の意味	デフォルト値
FF_USE_LFN	0	8.3 形式(注 2)	0
	1	LFN 形式（BSS 上の静的ワーキングバッファを使用）(注 3)	
	2	LFN 形式（スタック上の動的ワーキングバッファを使用）	
	3	LFN 形式（ヒープ上の動的ワーキングバッファを使用）	

注 2：“FF_MAX_LFN”も無効扱いになります。
注 3：常にスレッドセーフではないため、RTOS 使用時は指定してはいけません。

(3) RTOS

RTOS 使用時、”FF_FS_REENTRANT”、”FF_FS_TIMEOUT”、”FF_SYNC_t”がソフトウェア上で自動的に定義されます。ユーザは”FF_FS_TIMEOUT”の値を任意に変更可能です。

表 1.4 RTOS 対応に関連する FatFs 設定オプション

定義名	定義値の許容値	定義値の意味	デフォルト値
FF_FS_REENTRANT	0	リエントラント性無効(注 4)	RTOS 未使用: 0 RTOS 使用: 1
	1	リエントラント性有効	
FF_FS_TIMEOUT	-1 以上(注 5)	API 関数が”FR_TIMEOUT”の値を返す（タイムアウト）までのミリ秒	1000(注 5)
FF_SYNC_t	型名	RTOS の同期オブジェクト（ミューテックス）の型名	FreeRTOS 使用: SemaphoreHandle_t RI600V4 使用: ID

注 4：”FF_FS_TIMEOUT”および”FF_SYNC_t”も無効扱いになります。
注 5：ユーザは任意の値に変更可能です。”0”を指定する場合、ポーリング待ちです。
さらに、RI600V4 使用時に”-1”を指定する場合、永久待ちとなります
（FreeRTOS は”-1”を使用できません）。

(4) セクタサイズ

FatFs 仕様では”FF_MIN_SS”および”FF_MAX_SS”には”512”, “1024”, “2048”, “4096”のいずれかを指定できますが、TFAT では”512”または”4096”のみサポートします。

TFAT V.4.02 では、ユーザが使用する記憶メディアの組み合わせによって”FF_MIN_SS”と”FF_MAX_SS”に指定する値は決まります（表 1.6 記憶メディアの組み合わせとセクタサイズ定義を参照）。

表 1.5 セクタサイズに関連する FatFs 設定オプション

定義名	定義値の許容値	定義値の意味	デフォルト値
FF_MIN_SS	512	最小セクタサイズ 512 バイト(注 6)	512
	4096	最小セクタサイズ 4096 バイト(注 6)	
FF_MAX_SS	512	最大セクタサイズ 512 バイト(注 6)	512
	4096	最大セクタサイズ 4096 バイト(注 6)	

注 6：”FF_MIN_SS”および”FF_MAX_SS”に指定する値は
SD カード、USB メモリ、eMMC を使用する場合は”512”を、
Serial Flash memory を使用する場合は”4096”を指定してください。

表 1.6 記憶メディアの組み合わせとセクタサイズ定義

記憶メディアの組み合わせ	セクタサイズ定義
Serial Flash memory が含まれない組み合わせ （SD カード、USB メモリ、eMMC のみ）	FF_MIN_SS = 512 FF_MAX_SS = 512
Serial Flash memory のみ	FF_MIN_SS = 4096 FF_MAX_SS = 4096
SD カード、USB メモリ、eMMC のいずれか + Serial Flash memory	FF_MIN_SS = 512 FF_MAX_SS = 4096

(5) フォーマット機能

f_mkfs() API の有効/無効を定義しフォーマット機能の有無を選択します。

表 1.7 フォーマット機能に関連する FatFs 設定オプション

定義名	定義値の許容値	定義値の意味	デフォルト値
FF_USE_MKFS	0	f_mkfs() API 無効	0
	1	f_mkfs() API 有効	

1.2.4 RTOS 対応

本製品は、FreeRTOS および RI600V4 に対応しています。

(1) RTOS 使用時の動作概要

RTOS のマルチタスク動作を考慮するため、FatFs の各 API は、一部を除き、ボリュームに対してリエントラント性（排他制御）を確保する仕組みを備えています。具体的には、各 API は RTOS のタイムアウト付きミューテックスを利用しています。ミューテックスは f_mount 関数にて削除／生成され、各 API の実行直後に取得／実行完了直前に解放されます。

いずれかの API によってミューテックスが取得されている間、他のタスクなどから同じボリュームに対して API を実行した場合、後から実行した API はミューテックスの解放待ち状態に移行し、FatFs 設定オプションの FF_FS_TIMEOUT の定義値で指定した時間だけ待ちます。後から実行した API の戻り値は、指定時間（FF_FS_TIMEOUT）以内にミューテックスが解放された場合に FR_OK を、解放されなかった場合に FR_TIMEOUT を返します。

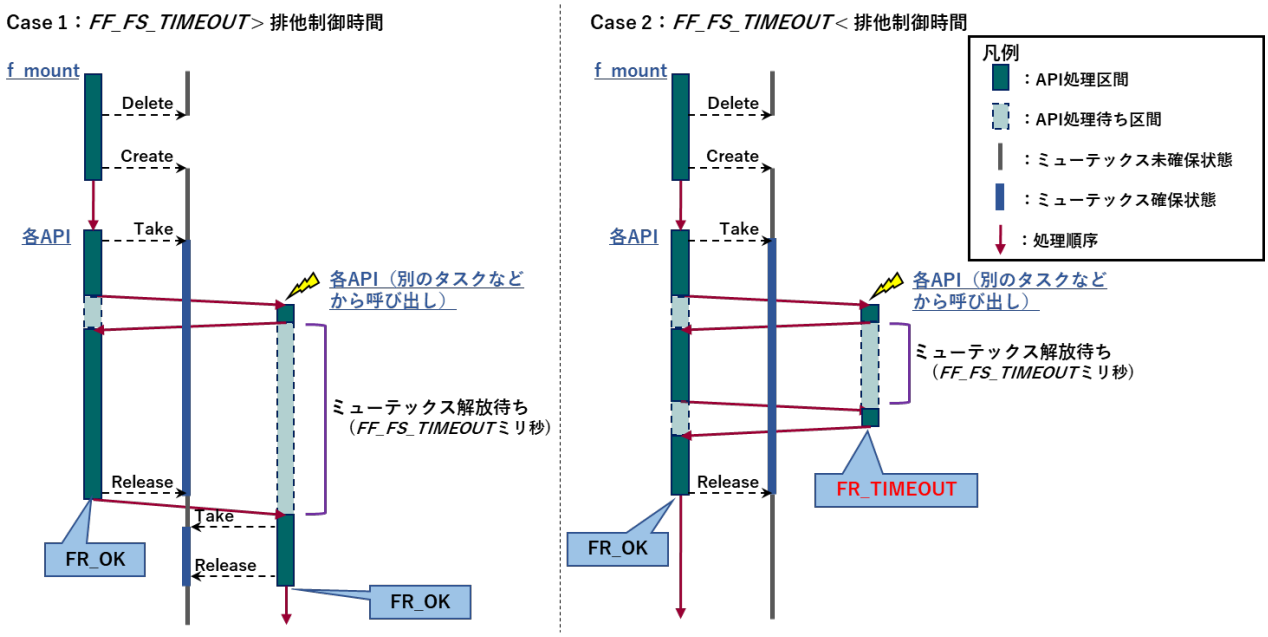

図 1.2 ボリュームに対する排他制御

(2) 排他制御の条件

FatFs の各 API が排他制御を適用する条件は FatFs 設定オプションやアクセス対象のボリュームによって異なります。

表 1.8 排他制御の条件（✓：排他制御可能、-：排他制御不可）

設定オプション	同じボリューム	異なるボリューム
FF_FS_REENTRANT = 0 && FF_USE_LFN != 1	-	✓
FF_FS_REENTRANT = 0 && FF_USE_LFN = 1	-	-
FF_FS_REENTRANT = 1 && FF_USE_LFN != 1	✓(注)	✓
FF_FS_REENTRANT = 1 && FF_USE_LFN = 1	-	-

注：f_mount、f_disk および f_mkfs 関数を除きます。これらの関数は同じボリュームに対して常にリエントラントでないため、他タスクなどによる同じボリュームへのアクセスをアプリケーション側処理で必ず防いでください。

なお、FatFs では、メモリ・ドライバ・インタフェース関数、および、その下位層のリエントラント性について関知していません。そのため、これらの低レベル I/O 関数では別途リエントラント性を確保するように実装する必要があります。

(3) RTOS 利用時の FatFs 設定オプション

RTOS 使用時にリエントラント性を確保するために、TFAT FIT は以下の FatFs 設定オプションを自動的に設定します（ffconf.h）。詳細は 1.2.3 (3) RTOS をご覧ください。

1.2.5 本製品の使用条件

本製品はオープンソース FatFs をベースとしています。したがって、FatFs のライセンス条項が定める使用条件を遵守する必要があります。

http://elm-chan.org/fsw/ff/doc/appnote.html#license

その他の使用条件は FIT のソフトウェア利用許諾契約書に準拠します。

https://www.renesas.com/jp/ja/common/disclaimers/disclaimer002.html

FatFs の内容に関わる質問については、FatFs のユーザ・フォーラムの利用をご検討ください。

http://elm-chan.org/fsw/ff/bd/

1.2.6 TFAT FIT のバージョン互換性

TFAT FIT Rev.4.00 以降はそれ以前の TFAT FIT と互換性がありません。FatFs の API 関数の仕様が異なるためです。

1.3 API の概要

TFAT FIT では、以下の API 関数を使用可能です。デフォルトの構成オプション設定で使用可能、および、弊社動作確認済みの API 関数を「✓」で示します。

表 1.9 API 関数

API 関数名	説明	デフォルト	動作確認済み
ファイル アクセス			
f_open()	ファイルのオープン・作成	✓	✓
f_close()	ファイルのクローズ	✓	✓
f_read()	ファイルの読み出し	✓	✓
f_write()	ファイルの書き込み	✓	✓
f_lseek()	リード/ライト・ポインタの移動、ファイルの拡張	✓	✓
f_truncate()	ファイル・サイズの切り詰め	✓	✓
f_sync()	キャッシュされたデータのフラッシュ	✓	✓
f_forward()	ファイル・データをストリームデバイスに転送		
f_expand()	ファイルに連続データ領域を準備または割り当て		
f_gets()	文字列の読み出し		
f_putc()	文字の書き込み		
f_puts()	文字列の書き込み		
f_printf()	書式化文字列の書き込み		
f_tell()	現在のリード/ライト・ポインタの取得	✓	✓
f_eof()	ファイル終端の有無の取得	✓	✓
f_size()	ファイル・サイズの取得	✓	✓
f_error()	ファイルのエラーの有無の取得	✓	✓
ディレクトリ アクセス			
f_opendir()	ディレクトリのオープン	✓	✓
f_closedir()	ディレクトリのクローズ	✓	✓
f_readdir()	ディレクトリの読み出し	✓	✓
f_findfirst()	ディレクトリでファイルを検索		
f_findnext()	次に一致するファイルを検索		
ファイル/ディレクトリ管理			
f_stat()	ファイル・ステータスの取得	✓	✓
f_unlink()	ファイル/ディレクトリの削除	✓	✓
f_rename()	ファイル/ディレクトリの名前変更・移動	✓	✓
f_chmod()	ファイル/ディレクトリの属性の変更		
f_utime()	ファイル/ディレクトリのタイムスタンプの変更		
f_mkdir()	ディレクトリの作成	✓	✓
f_chdir()	カレント・ディレクトリの変更		
f_chdrive()	カレント・ドライブの変更		
f_getcwd()	カレント・ディレクトリの取得		

ボリューム管理/システム構成			
f_mount()	ワークエリアの登録・削除	✓	✓
f_mkfs()	論理ドライブのフォーマット		✓
f_fdisk()	物理ドライブの分割		
f_getfree()	ボリューム空き領域の取得	✓	✓
f_getlabel()	ボリュームラベルを取得		
f_setlabel()	ボリュームラベルを作成		
f_setcp()	アクティブなコード・ページを設定		

1.4 メモリ・ドライバ・インタフェース関数の概要

TFAT FIT の API 関数は、以下のローレベル関数を使用します。しかし、これらの関数をアプリケーション・プログラム側から呼び出さないでください。

表 1.10 メモリ・ドライバ・インタフェース関数

ローレベル関数名	説明
disk_initialize()	ドライブの初期化
disk_status()	ドライブの状態取得
disk_read()	データの読み出し
disk_write()	データの書き込み
disk_ioctl()	その他のドライブ制御
get_fattime()	日付・時刻の取得

1.5 制限事項

(1) TFAT の対象デバイスは、TFAT より下位層でユーザが使用する全ての FIT でサポートされているデバイスです。各 FIT の対象デバイスはそれぞれのアプリケーションノートをご覧ください。

(2) TFAT FIT を RTOS と組み合わせて使用する場合、同一スレッド内で API を使用してください。

(3) f_mount、f_disk および f_mkfs 関数は同じボリュームに対してリエントラントではありません。使用する場合、アプリケーション側で排他制御をしなければなりません。

(4) TFAT 内部で以下の標準関数を使用しています。

・memset, memcmp, memcpy,

・malloc（注）, free（注）

注： FF_USE_LFN == 3 の場合

2. API 情報

2.1 ハードウェアの要求

なし

2.2 ソフトウェアの要求

このモジュールは以下の FIT モジュールに依存しています。

- ルネサスボードサポートパッケージ (r_bsp) Rev.5.52 以上
- M3S-TFAT-Tiny メモリ・ドライバ・インタフェースモジュール (r_tfat_driver) Rev.2.20 以上
- CMT モジュール Firmware Integration Technology (r_cmt) Rev.4.40 以上
- システムタイマモジュール Firmware Integration Technology (r_sys_time) Rev.1.01 以上

2.3 サポートされているツールチェーン

本 FIT モジュールは「7.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4 使用する割り込みベクタ

TFAT FIT では割り込みベクタを使用しません。

2.5 ヘッダファイル

API 呼び出しとメモリ・ドライバ・インタフェース定義は ff.h, diskio.h に記載しています。

2.6 コンパイル時の設定

TFAT FIT コンフィギュレーション・オプションの設定は、r_tfat_rx_config.h で行います（注）。

FatFs の構成オプション設定は、ffconf.h で行います。

注：TFAT FIT Rev.4.02 では設定可能な項目がありません。

2.7 コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。RX100 シリーズ、RX200 シリーズ、RX600 シリーズから代表して 1 デバイスずつ掲載しています。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.6 コンパイル時の設定」のコンフィギュレーション・オプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_tfat_rx Rev.4.02

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00
(統合開発環境のデフォルト設定に"-lang = c99"オプションを追加)
GCC for Renesas RX 8.3.0.202002
(統合開発環境のデフォルト設定に" -std=gnu99"オプションを追加)
IAR C/C++ Compiler for Renesas RX version 4.14.1
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ		
		Renesas Compiler	GCC	IAR Compiler
RX113	ROM(注)	6,487 バイト	13,168 バイト	8,443 バイト
	RAM(注)	26 バイト	28 バイト	58 バイト
	スタック(注)	368 バイト	-	380 バイト
RX231	ROM(注)	6,487 バイト	13,272 バイト	8,449 バイト
	RAM(注)	26 バイト	28 バイト	58 バイト
	スタック(注)	368 バイト	-	380 バイト
RX65N	ROM(注)	6,535 バイト	13,272 バイト	8,449 バイト
	RAM(注)	26 バイト	28 バイト	58 バイト
	スタック(注)	380 バイト	-	384 バイト

注：TFAT driver FIT モジュールの ROM/RAM/スタックサイズが含まれています。

2.8 TFAT FIT の型定義

TFAT FIT で使用する型定義を以下に示します(C99 の場合。FIT は C99 を使用します)。

表 2.1 TFAT の型

Datatype	Typedef
unsigned char	BYTE
unsigned char	DSTATUS
uint16_t	WORD
uint16_t	WCHAR
unsigned int	UINT
uint32_t	DWORD
uint64_t	QWORD

また、TCHAR は設定により変化し、以下のように型定義されます。

・FF_USE_LFN && FF_LFN_UNICODE = 1 の場合は WCHAR

・FF_USE_LFN && FF_LFN_UNICODE = 2 の場合は char

・FF_USE_LFN && FF_LFN_UNICODE = 3 の場合は DWORD

・上記以外の場合は char

2.9 TFAT FIT の構造体

このセクションでは、TFAT FIT で使用する構造体について詳細に説明します。

2.9.1 FATFS – ファイルシステム・オブジェクト構造体

FatFs 構造体は、論理ドライブに必要なワークエリアを保持します。これは、アプリケーション・プログラムによって割り当てられ、f_mount 関数を使用して登録または登録解除されます。

この構造体のメンバはアプリケーション・プログラムから変更することはできません。

以下に FatFs 構造体のメンバの詳細を示します。

```
typedef struct {
      BYTE  fs_type;          /* Filesystem type (0:not mounted) */
      BYTE  pdrv;             /* Associated physical drive */
      BYTE  n_fats;                 /* Number of FATs (1 or 2) */
      BYTE  wflag;                  /* win[] flag (b0:dirty) */
      BYTE  fsi_flag;         /* FSINFO flags (b7:disabled, b0:dirty) */
      WORD  id;                     /* Volume mount ID */
      WORD  n_rootdir;        /* Number of root directory entries (FAT12/16) */
      WORD  csize;            /* Cluster size [sectors] */
#if FF_MAX_SS != FF_MIN_SS
      WORD  ssize;            /* Sector size (512, 1024, 2048 or 4096) */
#endif
#if FF_USE_LFN
      WCHAR*      lfnbuf;     /* LFN working buffer */
#endif
#if FF_FS_EXFAT
      BYTE* dirbuf;           /* Directory entry block scratchpad buffer for
                                 exFAT */
#endif
#if FF_FS_REENTRANT
      FF_SYNC_t   sobj;       /* Identifier of sync object */
#endif
#if !FF_FS_READONLY
      DWORD last_clst;        /* Last allocated cluster */
      DWORD free_clst;        /* Number of free clusters */
#endif
#if FF_FS_RPATH
      DWORD cdir;             /* Current directory start cluster (0:root) */
#if FF_FS_EXFAT
      DWORD cdc_scl;          /* Containing directory start cluster (invalid
                                 when cdir is 0) */
      DWORD cdc_size;         /* b31-b8:Size of containing directory, b7-b0:
                                 Chain status */
      DWORD cdc_ofs;          /* Offset in the containing directory (invalid
                                 when cdir is 0) */
#endif
#endif
      DWORD n_fatent;         /* Number of FAT entries (number of clusters+2)*/
      DWORD fsize;            /* Size of an FAT [sectors] */
      DWORD volbase;          /* Volume base sector */
      DWORD fatbase;          /* FAT base sector */
      DWORD dirbase;          /* Root directory base sector/cluster */
      DWORD database;         /* Data base sector */
#if FF_FS_EXFAT
      DWORD bitbase;          /* Allocation bitmap base sector */
```

```
#endif
      DWORD winsect;          /* Current sector appearing in the win[] */
      BYTE  win[FF_MAX_SS];   /* Disk access window for Directory, FAT (and file
                              data at tiny cfg) */
} FATFS;
```

2.9.2 DIR - ディレクトリ・オブジェクト構造体

DIR 構造体（ディレクトリ・オブジェクト）は、ディレクトリに関連するデータを保持します。

ディレクトリに関連するデータは、f_opendir および f_readdir 関数によって DIR 構造体に格納されます。

以下に DIR のメンバの詳細を示します。この構造体のメンバはアプリケーション・プログラムから変更することはできません。

```
typedef struct {
      FFOBJID     obj;        /* Object identifier */
      DWORD       dptr;       /* Current read/write offset */
      DWORD       clust;      /* Current cluster */
      DWORD       sect;       /* Current sector (0:Read operation has
                              terminated) */
      BYTE*       dir;        /* Pointer to the directory item in the win[] */
      BYTE        fn[12];     /* SFN (in/out) {body[8],ext[3],status[1]} */
#if FF_USE_LFN
      DWORD       blk_ofs;    /* Offset of current entry block being processed
                               (0xFFFFFFFF:Invalid) */
#endif
#if FF_USE_FIND
      const TCHAR* pat;       /* Pointer to the name matching pattern */
#endif
} DIR;
```

2.9.3 FIL - ファイル・オブジェクト構造体

FIL 構造体（ファイル・オブジェクト）はファイルの状態を保持します。f_open 関数によって作成され、f_close 関数によって破棄されます。この構造体の”cltbl”を除いたメンバはアプリケーション・プログラムから変更することはできません。

```
typedef struct {
      FFOBJID     obj;        /* Object identifier (must be the 1st member
                              to detect invalid object pointer) */
      BYTE        flag;       /* File status flags */
      BYTE        err;        /* Abort flag (error code) */
      FSIZE_t     fptr;       /* File read/write pointer (Zeroed on file open)*/
      DWORD       clust;      /* Current cluster of fpter (invalid when fptr
                               is 0) */
      DWORD       sect;       /* Sector number appearing in buf[] (0:invalid) */
#if !FF_FS_READONLY
      DWORD       dir_sect;   /* Sector number containing the directory entry
                               (not used at exFAT) */
      BYTE*       dir_ptr;    /* Pointer to the directory entry in the win[]
                               (not used at exFAT) */
#endif
#if FF_USE_FASTSEEK
      DWORD*      cltbl;      /* Pointer to the cluster link map table
                              (nulled on open, set by application) */
#endif
#if !FF_FS_TINY
      BYTE        buf[FF_MAX_SS];   /* File private data read/write window */
#endif
} FIL;
```

2.9.4 FILINFO - ファイル・ステータス構造体

FILINFO 構造体は、f_stat()および f_readdir()関数によって返されるファイル情報を保持します。

```
typedef struct {
      FSIZE_t     fsize;                      /* File size */
      WORD        fdate;                      /* Modified date */
      WORD        ftime;                      /* Modified time */
      BYTE        fattrib;                    /* File attribute */
#if FF_USE_LFN
      TCHAR       altname[FF_SFN_BUF + 1];    /* Altenative file name */
      TCHAR       fname[FF_LFN_BUF + 1];      /* Primary file name */
#else
      TCHAR       fname[12 + 1]               /* File name */
#endif
} FILINFO;
```

2.9.5 FFOBJID - オブジェクト ID と割り当て情報の構造体

FFOBJID 構造体は、オブジェクト ID と割り当て情報を保持します。

```
typedef struct {
      FATFS*      fs;         /* Pointer to the hosting volume of this object */
      WORD        id;               /* Hosting volume mount ID */
      BYTE        attr;             /* Object attribute */
      BYTE        stat;             /* Object chain status (b1-0: =0:not
                                     contiguous,=2:contiguous, =3:fragmented
                                     in this session,
                                    b2:sub-directory stretched) */
      DWORD       sclust;           /* Object data start cluster (0:no cluster
                                    or root directory) */
      FSIZE_t     objsize;          /* Object size (valid when sclust != 0) */
#if FF_FS_EXFAT
      DWORD       n_cont;           /* Size of first fragment - 1 (valid when
                                    stat == 3) */
      DWORD       n_frag;           /* Size of last fragment needs to be written
                                    to FAT (valid when not zero) */
      DWORD       c_scl;            /* Containing directory start cluster (valid
                                    when sclust != 0) */
      DWORD       c_size;           /* b31-b8:Size of containing directory,
                                    b7-b0: Chain status
                                    (valid when c_scl != 0) */
      DWORD       c_ofs;            /* Offset in the containing directory
                                     (valid when file object and sclust != 0) */
#endif
#if FF_FS_LOCK
      UINT        lockid;           /* File lock ID origin from 1
                                    (index of file semaphore table Files[]) */
#endif
} FFOBJID;
```

2.10 TFAT FIT 定数

このセクションでは、使用する定数について詳細に説明します。以下の定数は ff.h に定義されています。

2.10.1 FRESULT - API 関数戻り値

API 関数の戻り値は、enum 型で定義されます。

```
typedef enum
{
      FR_OK = 0,                    /* (0) Succeeded */
      FR_DISK_ERR,                  /* (1) A hard error occurred in the low
                                       level disk I/O layer */
      FR_INT_ERR,                   /* (2) Assertion failed */
      FR_NOT_READY,                 /* (3) The physical drive cannot work */
      FR_NO_FILE,                   /* (4) Could not find the file */
      FR_NO_PATH,                   /* (5) Could not find the path */
      FR_INVALID_NAME,              /* (6) The path name format is invalid */
      FR_DENIED,                    /* (7) Access denied due to prohibited
                                       access or directory full */
      FR_EXIST,                     /* (8) Access denied due to prohibited
                                       access */
      FR_INVALID_OBJECT,            /* (9) The file/directory object is
                                       invalid */
      FR_WRITE_PROTECTED,           /* (10) The physical drive is write
protected */
      FR_INVALID_DRIVE,             /* (11) The logical drive number is
                                       invalid */
      FR_NOT_ENABLED,               /* (12) The volume has no work area */
      FR_NO_FILESYSTEM,             /* (13) There is no valid FAT volume */
      FR_MKFS_ABORTED,              /* (14) The f_mkfs() aborted due to any
                                       problem */
      FR_TIMEOUT,                   /* (15) Could not get a grant to access
                                       the volume within defined period */
      FR_LOCKED,                    /* (16) The operation is rejected according
                                       to the file sharing policy */
      FR_NOT_ENOUGH_CORE,           /* (17) LFN working buffer could not be
                                       allocated */
      FR_TOO_MANY_OPEN_FILES,       /* (18) Number of open files > FF_FS_LOCK */
      FR_INVALID_PARAMETER          /* (19) Given parameter is invalid */
}FRESULT;
```

2.10.2 ファイル属性情報

以下のマクロは FILINFO 構造体の fattrib メンバに設定される各ビットの内容を示します。

表 2.2 ファイル属性情報のマクロ

名称	値	意味
AM_RDO	0x01	このフラグがセットされている場合、該当するファイルまたはディレクトリは読み出し専用であることを示します。
AM_HID	0x02	このフラグがセットされている場合、該当するファイル(またはディレクトリ)は隠しファイル(またはディレクトリ)であることを示します。
AM_SYS	0x04	このフラグがセットされている場合、該当するファイル(またはディレクトリ)はシステムファイル(またはディレクトリ)であることを示します。
AM_DIR	0x10	このフラグがセットされている場合、ディレクトリであることを示します。
AM_ARC	0x20	このフラグがセットされている場合、該当するファイル(またはディレクトリ)はアーカイブであることを示します。

2.10.3 ディスク・ステータスのためのマクロ

これらのマクロは DSTATUS 型に設定されるディスクのステータスを示します。メモリ・ドライバ・インタフェース関数は適切なマクロを設定し、結果を TFAT FIT に渡します。

表 2.3 ディスク・ステータスのためのマクロ

名称	値	意味
STA_NOINIT	0x01	このフラグは、ディスクドライブが初期化されていないことを示します。このフラグは、システムリセット時、ディスク取り出し時、disk_initialize 関数の失敗時にセットされ、disk_initialize 関数の成功時にクリアされます。
STA_NODISK	0x02	このフラグがセットされている場合、ドライブに記録メディアが挿入されていないことを示します。ドライブに記録メディアが挿入されている場合、このフラグはクリアされます。
STA_PROTECT	0x04	このフラグは、記録メディアが書き込み保護されていることを示します。これは、書き込み保護ノッチをサポートしていないドライブでは常にクリアされます。このフラグは、STA_NODISK がセットされていない場合は有効でありません。

2.10.4 メモリ・ドライバ・インタフェース関数の戻り値

ユーザがメモリ・ドライバ・インタフェース関数によって実行されたディスク動作の結果を示すためにこの enum を使用します。

表 2.4 DRESULT の値

名称	値	意味
RES_OK	0	関数の実行が成功しました。
RES_ERROR	1	関数実行時にエラーが発生しました。
RES_WRPRT	2	ディスクは書き込み保護されています。
RES_NOTRDY	3	ディスクドライブが初期化されていません。
RES_PARERR	4	無効な引数が関数に渡されました。

2.10.5 フォーマット・オプション

以下のマクロはフォーマット・オプションの内容を示します。f_mkfs の第 2 引数として用いられます。なお、実際にフォーマットされる FAT タイプは、クラスタ・サイズ(f_mkfs の第 3 引数)も影響します。

表 2.5 ファイル属性情報のマクロ

名称	値	意味
FM_FAT	0x01	クラスタサイズに応じて、 FAT12/FAT16/FAT32 の中から決定
FM_FAT32	0x02	FAT32
FM_EXFAT	0x04	EXFAT
FM_ANY	0x07	クラスタサイズに応じて、 FAT12/FAT16/FAT32/EXFAT の中から決定
FM_SFD	0x08	SFD 形式

2.11 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、スマート・コンフィグレータを使用した(1)、(3)、(5)の追加方法を推奨しています。ただし、スマート・コンフィグレータは、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e^2 studio 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
e^2 studio のスマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: e^2 studio 編 (R20AN0451)」を参照してください。

(2) e^2 studio 上で FIT コンフィグレータを使用して FIT モジュールを追加する場合
e^2 studio の FIT コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版スマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: CS+編 (R20AN0470)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

(5) IAREW 上でスマート・コンフィグレータを使用して FIT モジュールを追加する場合
スタンドアロン版スマート・コンフィグレータを使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0535)」を参照してください。

3. API 関数

この章では、弊社動作確認済みの API 関数について説明します。その他の API 関数については、1.1 FatFs とは？に記載されたドキュメントをご参照ください。

3.1 f_mount()

この関数はワークエリアの登録・削除を行います。

Format

```
FRESULT f_mount (
FATFS*      fs,    /* [IN] Filesystem object */
const TCHAR* path,  /* [IN] Logical drive number */
BYTE        opt    /* [IN] Initialization option */
);
```

Parameters

fs

登録および消去するファイルシステム・オブジェクトへのポインタ。NULL ポインタは登録されたファイルシステム・オブジェクトの登録を解除します。

path

論理ドライブを設定する NULL 終端文字列へのポインタ。ドライブ番号のない文字列はデフォルトのドライブを意味します。

opt

初期化オプション

0：今すぐマウントしない（ボリュームへの最初のアクセス時にマウントされます）

1：ボリュームを強制的にマウントして、作動可能かどうかを確認します。

Return Values

FRESULT：FR_OK, FR_INVALID_DRIVE, FR_DISK_ERR, FR_NOT_READY, FR_NOT_ENABLED, FR_NO_FILESYSTEM

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

FatFs は各論理ドライブ（FAT ボリューム）用のワークエリア（ファイルシステム・オブジェクト）を必要とします。ファイル/ディレクトリ操作を実行する前に、ファイルシステム・オブジェクトを f_mount 関数で論理ドライブに登録する必要があります。ファイル/ディレクトリ API 関数は、この手順の後に動作する準備が整います。論理ドライブ上にファイルまたはディレクトリのオープンオブジェクトがある場合、そのオブジェクトはこの機能によって無効にされます。

f_mount 関数は、次のとおりファイルシステム・オブジェクトを FatFs モジュールに登録/登録解除します。

1. パスで設定した論理ドライブを決定します。
2. ボリュームの登録済みワークエリアが存在する場合は、それをクリアおよび登録解除します。
3. fs が NULL でない場合は、新しいワークエリアをクリアしボリュームに登録します。
4. 強制マウントを設定した場合は、ボリュームへのボリュームマウント処理を行います。

強制マウントが設定されていない（opt = 0）場合、この関数は物理ドライブの状態に関係なく常に成功します。設定されたワークエリアをクリア（初期化前の状態に戻す）してそのアドレスを内部テーブルに登録するだけで、物理ドライブに対して影響を及ぼしません。以下の条件のいずれかが当てはまる場合、ボリュームマウントプロセス（対応する物理ドライブの初期化、物理ボリューム内の FAT ボリューム探索、ワークエリアの初期化）は後続の file / directroy 関数で試行されます（遅延マウント）。

·ファイルシステム・オブジェクトが初期化されていない状態。ファイルシステム・オブジェクトは f_mount 関数によって初期化前の状態に戻されます。

·物理ドライブが初期化されていない状態。物理ドライブはシステムリセットまたはメディアの取り外しによって初期化前の状態に戻されます。

強制マウント（opt = 1）の機能が FR_NOT_READY で失敗した場合は、ファイルシステム・オブジェクトは正常に登録されましたが、ボリュームは現在動作可能な状態ではありません。後続の file / directroy 関数でボリュームマウントプロセスが試行されます。

ディスク I / O 層の実装に非同期メディア変更の検出が欠けている場合、ファイルシステム・オブジェクトを強制的にクリアするために、アプリケーション・プログラムは各メディア変更後に f_mount 関数を実行する必要があります。ワークエリアの登録を解除するには、fs に NULL を設定し f_mount 関数を実行後、ワークエリアを破棄できます。

Example

```
FATFS *fs;    /* Ponter to the filesystem object */

fs = malloc(sizeof (FATFS));         /* Get work area for the volume */
f_mount(fs, “”, 0);                 /* Mount the default drive */
f_open(...                          /* Here any file API can be used */

...

f_mount(fs, “”, 0);                 /* Re-mount the default drive to
                                       reinitialize the filesystem */

...

f_mount(0, “”, 0);                  /* Unmount the default drive */
free(fs);                           /* Here the work area can be discarded */
...
```

Special Notes:

なし

3.2 f_open()

この関数は既存の、または、新しく作成したファイルを開きます。

Format

```
FRESULT f_open (
FIL* fp,          /* [OUT] Pointer to the file object structure */
const TCHAR* path, /* [IN] File name */
BYTE mode         /* [IN] Mode flags */
);
```

Parameters

fp

空のファイル・オブジェクト構造体へのポインタ。

path

開くファイルまたは作成するファイルの名前を設定する NULL 終端文字列へのポインタ。

mode

ファイルのアクセスの種類と開き方を設定するモードフラグ。以下のフラグの組み合わせで設定されます。

表 3.1 mode のパラメータ

値	説明
FA_READ	オブジェクトへの読み出しアクセスを設定します。ファイルからデータを読み取ることができます。
FA_WRITE	オブジェクトへの書き込みアクセスを設定します。データをファイルに書き込むことができます。リード/ライトアクセスの場合は FA_READ と組み合わせてください。
FA_OPEN_EXISTING	ファイルを開きます。ファイルが存在しない場合、関数は失敗します。（デフォルト）
FA_OPEN_ALWAYS	ファイルが存在する場合はそれを開きます。そうでない場合は、新しいファイルが作成されます。
FA_CREATE_NEW	新しいファイルを作成します。ファイルが存在する場合、関数は FR_EXIST で失敗します。
FA_CREATE_ALWAYS	新しいファイルを作成します。ファイルが存在する場合は、ファイル・サイズを切り詰めて上書きされます。
FA_OPEN_APPEND	リード/ライト・ポインタがファイルの末尾に設定されている点を除けば、FA_OPEN_ALWAYS と同じです。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_FILE, FR_NO_PATH, FR_INVALID_NAME, FR_DENIED, FR_EXIST, FR_INVALID_OBJECT, FR_WRITE_PROTECTED, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_LOCKED, FR_NOT_ENOUGH_CORE, FR_TOO_MANY_OPEN_FILES

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_open 関数はファイルを開き、ファイル・オブジェクトを作成します。ファイル・オブジェクトは、その後のファイルへのリード/ライト操作でファイルを識別するために使用されます。開いているファイルはファイルアクセス操作の後に f_close 関数で閉じるべきです。何らかの変更が行われたファイルがその後正しく閉じられなかった（電源切断、メディア取り外し、再マウント）場合、そのファイルが破損する場合があります。

すでに開かれているファイルを開く必要がある場合は、FatFs アプリケーションノートの多重アクセス制御を参照してください。ただし、書き込みモードフラグを設定してファイルを重複して開くことは常に禁止されています。

Example

```
FIL fil;       /* File object */
char line[100]; /* Line buffer */
FRESULT fr;    /* FatFs return code */


/* Register work area to the default drive */
f_mount(&FatFs, “”, 0);

/* Open a text file */
fr = f_open(&fil, “message.txt”, FA_READ);
if (fr) return (int)fr;

/* Read every line and display it */
while (f_gets(line, sizeof line, &fil)) {
   printf(line);
}

/* Close the file */
f_close(&fil);
```

Special Notes:

なし

3.3 f_close()

この関数はファイルを閉じます。

Format

```
FRESULT f_close (
FIL* fp    /* [IN] Pointer to the file object */
);
```

Parameters

fp

クローズ対象のファイル・オブジェクト構造体へのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_close 関数は、開いているファイル・オブジェクトを閉じます。ファイルが変更されている場合は、ファイルのキャッシュ情報がボリュームに書き戻されます。関数が成功すると、ファイル・オブジェクトは無効になり、破棄できます。ファイル・オブジェクトが読み出し専用モードで FF_FS_LOCK が有効になっていない場合は、この手順を実行せずにファイル・オブジェクトを破棄することもできます。しかしこれは将来の互換性のために推奨されません。

Example

3.2 f_open()に記載。

Special Notes:

なし

3.4 f_read()

この関数はファイルからデータを読み出します。

Format

```
FRESULT f_read (
 FIL* fp,     /* [IN] File object */
 void* buff,  /* [OUT] Buffer to store read data */
 UINT btr,    /* [IN] Number of bytes to read */
 UINT* br     /* [OUT] Number of bytes read */
);
```

Parameters

fp

開いているファイル・オブジェクトへのポインタ。

buff

読み出したデータを格納するためのバッファへのポインタ。

btr

UINT 型の範囲で読み込むバイト数。

br

読み出されたバイト数を受け取る UINT 型変数へのポインタ。この値は、関数の戻りコードに関係なく、関数呼び出しの後は常に有効です。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_DENIED, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

この関数は、ファイルのリード/ライト・ポインタが指す位置からデータの読み出しを開始します。リード/ライト・ポインタは、読み出されたバイト数だけ進みます。関数が成功した後、* br をチェックしてファイルの終わりを検出する必要があります。* br < btr の場合、読み出し操作中にリード/ライト・ポインタがファイルの終わりに達したことを意味します。

Example

```
  FATFS fs0, fs1;     /* Work area (filesystem object) for logical drives */
   FIL fsrc, fdst;     /* File objects */
   BYTE buffer[4096];   /* File copy buffer */
   FRESULT fr;         /* FatFs function common result code */
   UINT br, bw;        /* File read/write count */


   /* Register work area for each logical drive */
   f_mount(&fs0, “0:”, 0);
   f_mount(&fs1, “1:”, 0);

   /* Open source file on the drive 1 */
   fr = f_open(&fsrc, “1:file.bin”, FA_READ);
   if (fr) return (int)fr;

   /* Create destination file on the drive 0 */
   fr = f_open(&fdst, “0:file.bin”, FA_WRITE | FA_CREATE_ALWAYS);
   if (fr) return (int)fr;

   /* Copy source to destination */
   for (;;) {
      fr = f_read(&fsrc, buffer, sizeof buffer, &br);  /* Read a chunk of
                                                          source file */
      if (fr || br == 0) break; /* error or eof */
      fr = f_write(&fdst, buffer, br, &bw);          /* Write it to the
                                                          destination file */
      if (fr || bw < br) break; /* error or disk full */
   }

   /* Close open files */
   f_close(&fsrc);
   f_close(&fdst);

   /* Unregister work area prior to discard it */
   f_mount(0, “0:”, 0);
   f_mount(0, “1:”, 0);
```

Special Notes:

なし

3.5 f_write()

この関数はファイルにデータを書き込みます。

Format

```
FRESULT f_write (
 FIL* fp,         /* [IN] Pointer to the file object structure */
 const void* buff, /* [IN] Pointer to the data to be written */
 UINT btw,        /* [IN] Number of bytes to write */
 UINT* bw         /* [OUT] Pointer to the variable to return number of bytes written */
);
```

Parameters

fp

開いているファイル・オブジェクト構造体へのポインタ。

buff

書き込まれるデータへのポインタ。

btw

UINT 型の範囲で書き込むバイト数。

bw

書き込まれたバイト数を受け取る UINT 型変数へのポインタ。この値は、関数の戻りコードに関係なく、関数呼び出しの後は常に有効です。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_DENIED, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

この関数は、ファイルのリード/ライト・ポインタが指す位置からデータの書き込みを開始します。リード/ライト・ポインタは、書き込まれたバイト数だけ進みます。関数が成功した後、* bw をチェックしてディスクがフルになったことを検出する必要があります。* bw < btw の場合は、書き込み操作中にボリュームがフルになったことを意味します。この機能は、ボリュームがフルになるか、フルに近づくと時間がかかることがあります。

Example

```
FATFS fs0, fs1;     /* Work area (filesystem object) for logical drives */
FIL fsrc, fdst;     /* File objects */
BYTE buffer[4096];   /* File copy buffer */
FRESULT fr;         /* FatFs function common result code */
UINT br, bw;        /* File read/write count */


/* Register work area for each logical drive */
f_mount(&fs0, “0:”, 0);
f_mount(&fs1, “1:”, 0);

/* Open source file on the drive 1 */
fr = f_open(&fsrc, “1:file.bin”, FA_READ);
if (fr) return (int)fr;

/* Create destination file on the drive 0 */
fr = f_open(&fdst, “0:file.bin”, FA_WRITE | FA_CREATE_ALWAYS);
if (fr) return (int)fr;

/* Copy source to destination */
for (;;) {
   fr = f_read(&fsrc, buffer, sizeof buffer, &br);  /* Read a chunk of
                                                       source file */
   if (fr || br == 0) break; /* error or eof */
   fr = f_write(&fdst, buffer, br, &bw);          /* Write it to the
                                                     destination file */
   if (fr || bw < br) break; /* error or disk full */
}

/* Close open files */
f_close(&fsrc);
f_close(&fdst);

/* Unregister work area prior to discard it */
f_mount(0, “0:”, 0);
f_mount(0, “1:”, 0);
```

Special Notes:

FF_FS_READONLY = 0 の場合に使用可能です。

3.6 f_lseek()

この関数は、開いているファイル・オブジェクトのファイルリード/ライト・ポインタを移動します。ファイル・サイズの拡大（クラスタの事前割り当て）にも使用できます。

Format

```
FRESULT f_lseek (
  FIL*    fp,  /* [IN] File object */
  FSIZE_t ofs  /* [IN] File read/write pointer */
);
```

Parameters

fp

開いているファイル・オブジェクトへのポインタ。

ofs

リード/ライト・ポインタを設定するためのファイルの先頭からのバイトオフセット。データ型 FSIZE_t は、構成オプション FF_FS_EXFAT に応じて、DWORD（32 ビット）または QWORD（64 ビット）のエイリアスです。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

開いたファイル・オブジェクトのリード/ライト・ポインタは次に読み出し・書き込みされるバイト数を指します。読み出し・書き込みされたバイト数だけリード/ライト・ポインタは進みます。f_lseek 関数はファイルへの読み出し・書き込み操作なしでリード/ライト・ポインタを移動します。

書き込みモードでファイル・サイズより大きな値を設定すると、指定されたオフセットまでファイル・サイズが拡張されます。この処理によってファイルにデータが書き込まれるのではないので、拡張された部分のデータは未定義となります。データを遅延無く高速に書き込みたいときに、予めこの関数で必要なサイズまでデータ領域を素早く事前に割り当てるのに適しています。f_lseek 関数が正常終了したあとは、リード/ライト・ポインタが正しく移動したかチェックするべきです。リード/ライト・ポインタが期待値ではないときは、次の原因が発生しています。

·ファイルが終端状態。ファイルが読み出し専用モードで開かれたため、設定した ofs がファイルの終端でクリップされた。

·ディスクがフル状態。ファイル拡張のためのボリューム上の空きスペースがない。

高速シーク機能は、オンメモリ CLMT（クラスター・リンク・マップ・テーブル）を使用することにより、FAT アクセスなしで高速な後方/ロング・シーク操作を可能にします。この機能は f_read および f_write 関数にも適用されますが、ファイルが高速シークモードの間は、f_write、f_lseek 関数ではファイル・サイズを拡張できません。

ファイル・オブジェクトの cltbl メンバが NULL でない場合、高速シークモードが有効になります。高速シーク動作を行う前に、DWORD 配列に CLMT を作成しておく必要があります。CLMT を作成するには、まずファイル・オブジェクトの cltbl メンバに DWORD 配列のアドレスをセットします。そして、配列の先頭要素にその配列のサイズ(要素数)を入れ、ofs == CREATE_LINKMAP を設定して f_lseek 関数を呼び出します。関数が成功すると CLMT が作成され、以降の f_read/f_write/f_lseek 関数では FAT へのアクセスは発生しません。先頭要素には使用された、または、必要とされた要素数が返されます。使用される要素数は、(ファイルの分割数 + 1) * 2 です。たとえば、ファイルが 5 つのフラグメントに分断されているときに必要な要素数は、12 となります。FR_NOT_ENOUGH_CORE で関数が失敗したときは、指定された配列サイズがファイルに対して不足しています。

Example

```
   /* Open file */
   fp = malloc(sizeof (FIL));
   res = f_open(fp, “file.dat”, FA_READ|FA_WRITE);
   if (res) ...

   /* Move to offset of 5000 from top of the file */
   res = f_lseek(fp, 5000);

   /* Move to end of the file to append data */
   res = f_lseek(fp, f_size(fp));

   /* Forward 3000 bytes */
   res = f_lseek(fp, f_tell(fp) + 3000);

   /* Rewind 2000 bytes (take care on wraparound) */
   res = f_lseek(fp, f_tell(fp) - 2000);;
```

```
   /* Using fast seek function */

   DWORD clmt[SZ_TBL];                 /* Cluster link map table buffer */

   res = f_open(fp, fname, FA_READ | FA_WRITE);   /* Open a file */

   res = f_lseek(fp, ofs1);             /* This is normal seek (cltbl is nulled
on file open) */

   fp->cltbl = clmt;                   /* Enable fast seek function (cltbl !=
NULL) */
   clmt[0] = SZ_TBL;                   /* Set table size */
   res = f_lseek(fp, CREATE_LINKMAP);    /* Create CLMT */
   ...

   res = f_lseek(fp, ofs2);             /* This is fast seek */
```

Special Notes:

FF_FS_MINIMIZE <= 2 の場合に使用可能です。高速シーク機能を使用するには、この機能を有効にするために FF_USE_FASTSEEK を 1 に設定する必要があります。

3.7 f_truncate()

この関数はファイル・サイズを切り詰めます。

Format

```
FRESULT f_truncate (
  FIL* fp    /* [IN] File object */
);
```

Parameters

fp

切り詰める対象の開いているファイル・オブジェクトへのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_DENIED, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_truncate 関数は、ファイル・サイズを現在のリード/ライト・ポインタに切り詰めます。ファイルのリード/ライト・ポインタがすでにファイルの末尾を指している場合、この関数は効果がありません。

Example

```
   /* Cluster pre-allocation (to prevent buffer overrun on streaming write) */

   res = f_open(fp, recfile, FA_CREATE_NEW | FA_WRITE);   /* Create a file */

   res = f_lseek(fp, PRE_SIZE);           /* Expand file size (cluster pre-
allocation) */
   if (res || f_tell(fp) != PRE_SIZE) ...   /* Check if the file has been
expanded successfly */

   res = f_lseek(fp, DATA_START);         /* Record data stream WITHOUT cluster
allocation delay */
   ...                                 /* Write operation should be aligned to
sector boundary to optimize the write throughput */

   res = f_truncate(fp);                  /* Truncate unused area */
   res = f_lseek(fp, 0);                  /* Set file header */
   ...

   res = f_close(fp);*/
```

Special Notes:

FF_FS_READONLY = 0 かつ FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.8 f_sync()

この関数は書き込み中のファイルのキャッシュ情報をフラッシュします。

Format

```
FRESULT f_sync (
  FIL* fp    /* [IN] File object */
);
```

Parameters

fp

フラッシュ対象の開いているファイル・オブジェクトへのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

この関数は f_close()と同じ処理を実行しますが、ファイルは引き続き開かれたままになり、リード/ライトを続行できます。ロギングなど、書き込みモードで長時間ファイルが開かれているアプリケーションにおいて、定期的または区切りの良いところでこの関数を使用することにより、不意の電源断やメディアの取り外しにより失われるデータを最小にすることができます。f_close()内ではこの関数を呼び出した後ファイル・オブジェクトを無効化しているだけなので、f_close()直前の f_sync()は意味がありません。

Example

なし

Special Notes:

FF_FS_READONLY = 0 の場合に使用可能です。

3.9 f_opendir()

この関数はディレクトリを開きます。

Format

```
FRESULT f_opendir (
  DIR* dp,          /* [OUT] Pointer to the directory object structure */
  const TCHAR* path  /* [IN] Directory name */
);
```

Parameters

dp

新しいディレクトリ・オブジェクトを作成するための空のディレクトリ・オブジェクトへのポインタ。

path

開くディレクトリ名を設定する NULL 終端文字列へのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_PATH, FR_INVALID_NAME, FR_INVALID_OBJECT, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_NOT_ENOUGH_CORE, FR_TOO_MANY_OPEN_FILES

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_opendir 関数は、既存のディレクトリを開き、後続の f_readdir 関数で使用するためにディレクトリ・オブジェクトを作成します。

Example

3.11 f_readdir()に記載。

Special Notes:

FF_FS_MINIMIZE <= 1 の場合に使用可能です。

3.10 f_closedir()

この関数は開いているディレクトリを閉じます。

Format

```
FRESULT f_closedir (
  DIR* dp    /* [IN] Pointer to the directory object */
);
```

Parameters

dp

閉じる対象の開いているディレクトリ・オブジェクト構造体へのポインタ。

Return Values

FRESULT：FR_OK, FR_INT_ERR, FR_INVALID_OBJECT, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_closedir 関数は、開いているディレクトリ・オブジェクトを閉じます。関数が成功すると、ディレクトリ・オブジェクトは無効になり、破棄できます。構成オプション FF_FS_LOCK が有効になっていない場合は、この手順を行わずにディレクトリ・オブジェクトを破棄することもできます。しかしこれは将来の互換性のために推奨されません。

Example

3.11 f_readdir()に記載。

Special Notes:

FF_FS_MINIMIZE <= 1 の場合に使用可能です。

3.11 f_readdir()

この関数はディレクトリの情報を読み出します。

Format

```
FRESULT f_readdir (
  DIR* dp,     /* [IN] Directory object */
  FILINFO* fno  /* [OUT] File information structure */
);
```

Parameters

dp

開いているディレクトリ・オブジェクトへのポインタ。

fno

読み出したファイル情報を保管するファイル・ステータス構造体へのポインタ。NULL ポインタはディレクトリの読み出しインデックスを最初に戻します。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_INVALID_OBJECT, FR_TIMEOUT, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_readdir 関数はオブジェクトに関する情報をディレクトリのアイテムから読み出します。ディレクトリ内のすべてのアイテムは、f_readdir 関数を連続して呼び出すことで、順番に読み取ることができます。サブディレクトリ内のドットエントリ（ “.”と “..”）は除外され、それらはアイテムとして読み取られません。すべてのディレクトリのアイテムが読み取られ、次に読み取るアイテムがない場合、エラーなしで NULL 文字列が fno-> fname []に保管されます。fno に NULL ポインタが与えられた場合、ディレクトリ・オブジェクトの読み出しインデックスは最初に戻ります。

長いファイル名（LFN）の構成オプションが有効になっている場合、オブジェクトの短いファイル名を格納するために、メンバ altname []がファイル・ステータス構造体に定義されています。以下の何らかの理由で長いファイル名にアクセスできない場合、短いファイル名は fname[]に格納され、altname[]は NULL 文字列を持ちます。

·アイテムに LFN がない。（exFAT ボリュームの場合は該当しません）

·FF_MAX_LFN の設定が、LFN を処理するのに不十分。（FF_MAX_LFN = 255 の場合は該当しません）

·FF_LFN_BUF の設定が LFN を格納するのに不十分。

·LFN に現在のコード・ページで定義されていない文字が含まれている。（FF_LFN_UNICODE> = 1 の場合は該当しません）

exFAT は短いファイル名をサポートしません。つまり、上記の条件が当てはまる時、ファイル名を返すことができない場合があります。その場合、オブジェクトにアクセスできないことを示すために、fname[]に “?” が返されます。この問題を回避するには、LFN 仕様の全機能をサポートするように FatFs FF_LFN_UNICODE> = 1 かつ FF_MAX_LFN = 255 を定義します。

Example

```
 FRESULT scan_files (
    char* path       /* Start node to be scanned (***also used as work area***)
*/
)
{
    FRESULT res;
    DIR dir;
    UINT i;
    static FILINFO fno;

    res = f_opendir(&dir, path);                    /* Open the directory */
    if (res == FR_OK) {
        for (;;) {
            res = f_readdir(&dir, &fno);                /* Read a directory item
*/
            if (res != FR_OK || fno.fname[0] == 0) break;  /* Break on error or
                                                            end of dir */
            if (fno.fattrib & AM_DIR) {                 /* It is a directory */
                i = strlen(path);
                sprintf(&path[i], “/%s”, fno.fname);
                res = scan_files(path);                 /* Enter the directory */
                if (res != FR_OK) break;
                path[i] = 0;
            } else {                                    /* It is a file. */
                printf(“%s/%s¥n”, path, fno.fname);
            }
        }
        f_closedir(&dir)
    }

    return res;
}

int main (void)
{
    FATFS fs;
    FRESULT res;
    char buff[256];


    res = f_mount(&fs, “”, 1);
    if (res == FR_OK) {
        strcpy(buff, “/”);
        res = scan_files(buff);
    }

    return res;
}
```

Special Notes:

FF_FS_MINIMIZE <= 1 の場合に使用可能です。

3.12 f_getfree()

この関数は、ボリューム上の空きクラスタの数を取得します。

Format

```
FRESULT f_getfree (
  const TCHAR* path,  /* [IN] Logical drive number */
  DWORD* nclst,       /* [OUT] Number of free clusters */
  FATFS** fatfs       /* [OUT] Corresponding filesystem object */
);
```

Parameters

path

論理ドライブを設定する NULL 終端文字列へのポインタ。NULL 文字列はデフォルトのドライブを意味します。

nclst

空きクラスタの数を格納するための DWORD 変数へのポインタ。

fatfs

対応するファイルシステム・オブジェクトへのポインタを格納するポインタへのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_getfree 関数は、ボリューム上の空きクラスタ数を取得します。ファイルシステム・オブジェクトのメンバ csize はクラスタあたりのセクタ数を示しているので、セクタ単位の空き容量はこの情報で計算できます。FAT32 ボリュームの FSINFO 構造体が同期していない場合、この関数は誤った空きクラスタ数を返す場合があります。この問題を回避するために、FF_FS_NOFSINFO 構成オプションで FatFs を強制的に完全 FAT スキャンすることができます。

Example

```
FATFS *fs;
DWORD fre_clust, fre_sect, tot_sect;

/* Get volume information and free clusters of drive 1 */
res = f_getfree(“1:”, &fre_clust, &fs);
if (res) die(res);

/* Get total sectors and free sectors */
tot_sect = (fs->n_fatent - 2) * fs->csize;
fre_sect = fre_clust * fs->csize;

/* Print the free space (assuming 512 bytes/sector) */
printf(“%10lu KiB total drive space.¥n%10lu KiB available.¥n”,
tot_sect / 2, fre_sect / 2);
```

Special Notes:

FF_FS_READONLY = 0 かつ FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.13 f_stat()

この関数はファイルまたはサブディレクトリの存在を確認します。

Format

```
FRESULT f_stat (
  const TCHAR* path,  /* [IN] Object name */
  FILINFO* fno       /* [OUT] FILINFO structure */
);
```

Parameters

path

情報取得対象のオブジェクトを設定する NULL 終端文字列へのポインタ。オブジェクトはルートディレクトリを設定できません。

fno

オブジェクトの情報を格納するための空の FILINFO 構造体へのポインタ。不要な場合は NULL ポインタを設定してください。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_FILE, FR_NO_PATH, FR_INVALID_NAME, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

f_stat 関数は、ファイルまたはサブディレクトリの存在を確認します。存在しない場合、関数は FR_NO_FILE を返します。存在する場合、関数は FR_OK で戻り、オブジェクトの情報、ファイル・サイズ、タイムスタンプ、および属性がファイル・ステータス構造体に格納されます。ファイル情報の詳細については、FILINFO 構造体と f_readdir 関数を参照してください。

Example

```
FRESULT fr;
FILINFO fno;


printf(“Test for 'file.txt'...¥n”);

fr = f_stat(“file.txt”, &fno);
switch (fr) {

case FR_OK:
   printf(“Size: %lu¥n”, fno.fsize);
   printf(“Timestamp: %u/%02u/%02u, %02u:%02u¥n”,
         (fno.fdate >> 9) + 1980, fno.fdate >> 5 & 15, fno.fdate & 31,
         fno.ftime >> 11, fno.ftime >> 5 & 63);
   printf(“Attributes: %c%c%c%c%c¥n”,
         (fno.fattrib & AM_DIR) ? 'D' : '-',
         (fno.fattrib & AM_RDO) ? 'R' : '-',
         (fno.fattrib & AM_HID) ? 'H' : '-',
         (fno.fattrib & AM_SYS) ? 'S' : '-',
         (fno.fattrib & AM_ARC) ? 'A' : '-');
   break;

case FR_NO_FILE:
   printf(“It is not exist.¥n”);
   break;

default:
   printf(“An error occured. (%d)¥n”, fr);
}
```

Special Notes:

FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.14 f_mkdir()

この関数はディレクトリを新規作成します。

Format

```
FRESULT f_mkdir (
  const TCHAR* path /* [IN] Directory name */
);
```

Parameters

path

作成するディレクトリ名を設定する NULL 終端文字列へのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_PATH, FR_INVALID_NAME, FR_DENIED, FR_EXIST, FR_WRITE_PROTECTED, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

この関数は新しいディレクトリを作成します。ディレクトリを削除するには、f_unlink 関数を使います。

Example

```
res = f_mkdir(“sub1”);
if (res) die(res);
res = f_mkdir(“sub1/sub2”);
if (res) die(res);
res = f_mkdir(“sub1/sub2/sub3”);
if (res) die(res);
```

Special Notes:

FF_FS_READONLY = 0 かつ FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.15 f_unlink()

この関数はボリュームからファイルまたはサブディレクトリを削除します。

Format

```
FRESULT f_unlink (
  const TCHAR* path  /* [IN] Object name */
);
```

Parameters

path

削除するファイルまたはサブディレクトリを設定する NULL 終端文字列へのポインタ。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_FILE, FR_NO_PATH, FR_INVALID_NAME, FR_DENIED, FR_WRITE_PROTECTED, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_LOCKED, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

削除対象の条件が以下の条件に該当する場合、削除処理は拒否されます。

·ファイル/サブディレクトリが読み出し専用属性（AM_RDO）の場合（FR_DENIED を返します。）

·ディレクトリが空でない場合、またカレント・ディレクトリである場合（FR_DENIED を返します。）

·ファイル/サブディレクトリを開いている場合。開いている場合、FAT 構造が破壊される可能性があります。また、ファイルロック機能が有効（FF_FS_LOCK = 1）の場合、安全に拒否されます。

Example

なし

Special Notes:

FF_FS_READONLY = 0 かつ FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.16 f_rename()

この関数はファイルまたはサブディレクトリの名前を変更、または移動します。

Format

```
FRESULT f_rename (
  const TCHAR* old_name, /* [IN] Old object name */
  const TCHAR* new_name  /* [IN] New object name */
);
```

Parameters

old_name

名前を変更する既存のファイルまたはサブディレクトリを設定する、NULL 終端文字列へのポインタ。

new_name

新しいオブジェクト名を設定する NULL 終端文字列へのポインタ。この文字列にドライブ番号を設定することはできますが、無視され、old_name の同じドライブと見なされます。old_name 以外のこのパス名を持つオブジェクトが存在してはいけません。すでに存在する場合、関数は FR_EXIST で失敗します。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_INT_ERR, FR_NOT_READY, FR_NO_FILE, FR_NO_PATH, FR_INVALID_NAME, FR_EXIST, FR_WRITE_PROTECTED, FR_INVALID_DRIVE, FR_NOT_ENABLED, FR_NO_FILESYSTEM, FR_TIMEOUT, FR_LOCKED, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

同じボリューム内のファイルやサブディレクトリの名前を変更し、または他のディレクトリに移動することもできます。名前を変更するオブジェクトは、開いているオブジェクトにしないでください。FAT 構造が破壊される可能性があります。ファイルロック機能が有効（FF_FS_LOCK = 1）の場合、誤った操作は安全に拒否されます。

Example

```
/* Rename an object in the default drive */
f_rename(“oldname.txt”, “newname.txt”);

/* Rename an object in the drive 2 */
f_rename(“2:oldname.txt”, “newname.txt”);

/* Rename an object and move it to another directory in the drive */
f_rename(“log.txt”, “old/log0001.txt”);
```

Special Notes:

FF_FS_READONLY = 0 かつ FF_FS_MINIMIZE = 0 の場合に使用可能です。

3.17 f_tell()

この関数はファイル上の現在のリード/ライト・ポインタを取得します。

Format

```
FSIZE_t f_tell (
  FIL* fp   /* [IN] File object */
);
```

Parameters

fp

開いているファイル・オブジェクト構造体へのポインタ。

Return Values

ファイルの現在のリード/ライト・ポインタ(ファイル先頭からのバイト単位のオフセット)が返ります。

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

現リビジョンでは、f_tell 関数はマクロとして実装されています。

```
#define f_tell(fp) ((fp)->fptr)
```

Example

なし

Special Notes:

なし

3.18 f_eof()

この関数はファイルのリード/ライト・ポインタがファイル終端に達しているかどうか調べます。

Format

```
int f_eof (
  FIL* fp   /* [IN] File object */
);
```

Parameters

fp

開いているファイル・オブジェクト構造体へのポインタ。

Return Values

リード/ライト・ポインタがファイルの終わりに達した場合、f_eof 関数は 0 以外の値を返します。達していない場合は 0 を返します。

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

現リビジョンでは f_eof 関数はマクロとして実装されています。

#define f_eof(fp) ((int)((fp)->fptr == (fp)->fsize))

Example

なし

Special Notes:

なし

3.19 f_size()

この関数はファイルのサイズを取得します。

Format

```
FSIZE_t f_size (
  FIL* fp   /* [IN] File object */
);
```

Parameters

fp

開いているファイル・オブジェクト構造体へのポインタ。

Return Values

バイト単位のファイル・サイズが返ります。

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

現リビジョンでは f_size 関数はマクロとして実装されています。

#define f_size(fp) ((fp)->obj.objsize)

Example

なし

Special Notes:

なし

3.20 f_error()

この関数はファイルに対するエラー発生の有無を調べます。

Format

```
int f_error(
  FIL* fp   /* [IN] File object */
);
```

Parameters

fp

開いているファイル・オブジェクト構造体へのポインタ。

Return Values

ハードエラーが発生した場合は 0 以外の値を返します。発生していない場合は 0 を返します。

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

現リビジョンでは f_error 関数はマクロとして実装されています。

#define f_error(fp) ((fp)->err)

Example

なし

Special Notes:

なし

3.21 f_mkfs()

この関数は論理ドライブ上に FAT/exFAT ボリュームを作成（フォーマット）します。

Format

```
FRESULT f_mkfs(
  const TCHAR*  path,  /* [IN] Logical drive number */
  const MKFS_PARM* opt,/* [IN] Format options */
  void*  work,        /* [-]  Working buffer */
  UINT  len          /* [IN] Size of working buffer */
);
```

Parameters

path

フォーマットする論理ドライブを指定する Null 終端文字列へのポインタ。ドライブ番号が含まれていない場合は、デフォルトのドライブをフォーマットします。論理ドライブは、フォーマットプロセス用にマウントされている場合とマウントされていない場合があります。

opt

フォーマット・オプションを指定する構造体（MKFS_PARM）へのポインタ。Null ポインタを指定する場合、関数にすべてのオプションがデフォルト値で渡されます。フォーマット・オプションの構造体には、次に説明する 5 つのメンバがあります。

BYTE fmt

FAT タイプのフラグ（FM_FAT、FM_FAT32、FM_EXFAT、およびこれら 3 つのビットの論理和である FM_ANY）の組み合わせ。exFAT が無効の場合、FM_EXFAT フラグは無視されます。これらのフラグは、ボリュームに作成する FAT タイプを指定します。2 つ以上のタイプが指定されている場合、ボリュームサイズと au_size に応じて、そのうちの 1 つが選択されます。FM_SFD フラグは、SFD フォーマットでドライブにボリュームを作成することを指定します。デフォルト値は FM_ANY です。

DWORD au_size

バイト単位のアロケーションユニット（クラスタ）サイズ。有効な値は、FAT / FAT32 ボリュームの場合は 1 * セクタサイズから 128 * セクタサイズまでの 2 の累乗で、exFAT ボリュームの場合は最大 16 MB です。ゼロ（デフォルト値）または無効な値が指定された場合、デフォルトのアロケーションユニットサイズは使用されるボリュームサイズに依存します。

UINT n_align

セクタ単位のボリュームデータ領域（ファイル割り当てプール、通常はフラッシュメディアのイレーズブロック境界）の配置。このメンバの有効な値は、1 から 2768 までの 2 の累乗です。ゼロ（デフォルト値）または無効な値が指定された場合、関数は disk_ioctl 関数を使用して下位層からブロックサイズを取得します。

BYTE n_fat

FAT / FAT32 ボリューム上の FAT コピーの数。このメンバの有効な値は 1 または 2 です。デフォルト値（0）と無効な値は 1 になります。FAT タイプが exFAT の場合、このメンバは無効です。

UINT n_root
FAT ボリューム上のルートディレクトリエントリの数。このメンバの有効な値は最大 32768 で、セクタサイズ/ 32 に揃えられます。デフォルト値（0）および無効な値は 512 になります。FAT タイプが FAT32 または exFAT の場合、このメンバは無効です。

work
フォーマットプロセスに使用される作業バッファへのポインタ。 FF_USE_LFN == 3 で Null ポインタが指定されている場合、関数はこの関数で作業バッファ用のメモリブロックを取得します。

len
バイト単位の作業バッファのサイズ。 少なくとも FF_MAX_SS である必要があります。 十分な作業バッファにより、ドライブへの書き込みトランザクションの数が減り、フォーマットプロセスが迅速に完了します。

Return Values

FRESULT：FR_OK, FR_DISK_ERR, FR_NOT_READY, FR_WRITE_PROTECTED, FR_INVALID_DRIVE, FR_MKFS_ABORTED, FR_INVALID_PARAMETER, FR_NOT_ENOUGH_CORE

Properties

ファイル ff.h にプロトタイプ宣言されています。

Description

Microsoft 社によって発行された FAT 仕様によれば、exFAT を除く FAT ボリュームの FAT サブタイプ FAT12 / FAT16 /FAT32 は、ボリューム上のクラスタの数のみによって決定され、それ以外は何も決定されません。したがって、作成されるボリュームの FAT サブタイプは、ボリュームサイズとクラスタサイズに依存します。引数で指定された FAT タイプとクラスタサイズの組み合わせがボリューム上で有効でない場合、関数は FR_MKFS_ABORTED で失敗します。クラスタともアロケーションユニットは、ファイルのディスク領域割り当て単位です。アロケーションユニットのサイズが 32768 バイトの場合、サイズが 100 バイトのファイルは 32768 バイトのディスク領域を占有します。アロケーションユニットのサイズが大きくなると、ディスク使用のスペース効率は低下しますが、読み取り/書き込みパフォーマンスは向上します。したがって、アロケーションユニットのサイズは、スペース効率とパフォーマンスの間のトレードオフです。GB 単位の大容量ストレージの場合、ボリューム上に非常に多くの小さなファイルが作成されるケースを除き、32768 バイト以上（これはデフォルトで自動的に選択されます）のアロケーションユニットのサイズが推奨されます。

フォーマットする論理ドライブが物理ドライブに関連付けられている（FF_MULTI_PARTITION = 0）、かつ、FM_SFD フラグが指定されていない場合、パーティションがディスク領域全体を占有し、FAT ボリュームがパーティションに作成されます。FM_SFD フラグが指定されている場合、FAT ボリュームはディスクのパーティション化なしで作成されます。

フォーマットする論理ドライブが複数パーティション機能によって特定のパーティションに関連付けられている場合（FF_MULTI_PARTITION = 1）、FAT ボリュームはボリュームマッピングテーブルで指定されたパーティションに作成され、FM_SFD フラグは無視されます。この関数で FAT ボリュームを作成する前に、物理ドライブを f_fdisk 関数または任意のパーティションツールでパーティション分割する必要があります。

MBR、GPT、SFD の 3 つの標準ディスクパーティションフォーマット形式があります。MBR フォーマットは FDISK フォーマットとも呼ばれ、通常、ハードディスク、メモリカード、U ディスクに使用されます。パーティションテーブルを使用して、物理ドライブを 1 つ以上のパーティションに分割できます。GPT（GUID パーティションテーブル）は、大規模なストレージデバイス用に新しく定義されたパーティション分割形式です。 FatFs は、64 ビット LBA が有効な場合にのみ GPT をサポートします。SFD（スーパーフロッピーディスク）は、パーティション分割されていないディスクフォーマットです。 FAT ボリュームは、物理パーティション全体を占有し、ディスクのパーティション化は行いません。通常、フロッピーディスク、光ディスク、ほとんどのスーパーフロッピーメディアに使用されます。システムと記憶メディアの一部の組み合わせは、パーティション形式または非パーティション形式のどちらかのみをサポートし、その他はサポートされていません。一部のシステムは、非標準形式でオンボードストレージのパーティションを管理します。パーティションは、disk_～関数内の pdrv によって識別される物理ドライブとしてマップされます。このようなシステムの場合、パーティションに FAT ボリュームを作成するには、SFD フォーマットが適しています。

Example

```
/* Format default drive and create a file */
int main (void)
{
   FATFS fs;          /* Filesystem object */
   FIL fil;           /* File object */
   FRESULT res;       /* API result code */
   UINT bw;           /* Bytes written */
   BYTE work[FF_MAX_SS]; /* Work area (larger is better for processing time) */


   /* Format the default drive with default parameters */
   res = f_mkfs(“”, 0, work, sizeof work);
   if (res) ...

   /* Gives a work area to the default drive */
   f_mount(&fs, “”, 0);

   /* Create a file as new */
   res = f_open(&fil, “hello.txt”, FA_CREATE_NEW | FA_WRITE);
   if (res) ...

   /* Write a message */
   f_write(&fil, “Hello, World!¥r¥n”, 15, &bw);
   if (bw != 15) ...

   /* Close the file */
   f_close(&fil);

   /* Unregister work area */
   f_mount(0, “”, 0);

   ...
```

Special Notes:

FF_FS_READOLNY == 0、かつ、FF_USE_MKFS == 1 の時、この関数を使用できます。

4. メモリ・ドライバ・インタフェース関数

メモリ・ドライバ・インタフェース関数は、diskio.h または ff.h にてプロトタイプ宣言されています。関数の実体は、TFAT driver FIT の r_tfat_drv_if.c に存在します。

これら関数の詳細は、TFAT driver FIT のアプリケーション・ノート（r20an0335xxxxxx）をご覧ください。

5. 端子設定

TFAT FIT に端子設定はありません。

6. サンプルプログラム

6.1 概要

サンプルプログラムは、7.1 動作確認環境に記載された評価ボード (以降 CPU ボードと呼ぶ)で動作する e^2 studio のプロジェクトです。またサンプルプログラムは、メモリ・ドライバ・インタフェースとして、SD モード SD カードドライバと USB ドライバを実装したものを用意しています。さらに、それぞれ None RTOS、FreeRTOS、RI600V4 を使用したものを用意しています。

- ドキュメント No. : R01AN3852
- ドキュメントタイトル : RX ファミリ SDHI モジュール Firmware Integration Technology: アプリケーションノート

- ドキュメント No. : R01AN4233
- ドキュメントタイトル : RX ファミリ SD モード SD メモリカードドライバ Firmware Integration Technology: アプリケーションノート

- ドキュメント No. : R01AN2025
- ドキュメントタイトル : USB Basic Host and Peripheral Driver Firmware Integration Technology: アプリケーションノート

- ドキュメント No. : R01AN2026
- ドキュメントタイトル : USB Host Mass Storage Class Driver (HMSC) Firmware Integration Technology: アプリケーションノート

- ドキュメント No. : R01AN2166
- ドキュメントタイトル : USB Basic Mini Host and Peripheral Driver Firmware Integration Technology(USB Mini Firmware): アプリケーションノート

- ドキュメント No. : R01AN2169
- ドキュメントタイトル : USB Host Mass Storage Class Driver (HMSC) for USB Mini Firmware Integration Technology: アプリケーションノート

6.2 動作

6.2.1 SD カードドライバ使用したサンプルプログラム

プログラムが実行されると、FAT ファイルシステムワークエリアを登録します。ディレクトリおよびファイルは記録メディア上に作成され、2KB のテキストデータをファイルに書き込みます。その後、ファイルを閉じます。書き込んだデータを確認するために、ファイルを読み出しモードで再び開きます。ファイルの内容全体を読み出し、プログラムの書き込みバッファデータと比較します。データの内容が一致しているかどうかをデバッガ上のデバッグコンソール(Renesas Debug Virtual Console)上に表示します。

表 6.1 デバッグコンソール表示の意味

表示文字列	意味
Detected attached SD card.	SD カードの挿入を検知しました。
Detected detached SD card.	SD カードの抜去を検知しました。
!!! Attach SD card. !!!!	SD カードを挿入してください。
!!! Detach SD card. !!!!	SD カードを抜去してください。
Start TFAT sample	サンプルプログラムを実行します。
Finished TFAT sample	サンプルプログラムが正常終了しました。
!!!!! TFAT error !!!!!	サンプルプログラムでエラーが発生しました。

ファイルリード/ライトのためのデータは、r_data_file.c に定義しています。デフォルトでは”Renesas¥n”という文字列を繰り返し書き込みます。書き込むデータは合計 2KB（2048 バイト）です。必要に応じて、このデータおよび対応するマクロ FILESIZE を編集してください。

サンプルプログラムにて使用されている主な関数は以下のとおりです。RTOS の使用有無に関わらず処理内容は同じです。ただし、FreeRTOS および RI600V4 はタスクとして実行されます。

表 6.2 サンプルプログラムの主な関数

処理内容	None RTOS での 関数名	FreeRTOS での タスク関数名	RI600V4 での タスク関数名
初期化	main()	main_task()	main_task()
デバイス検知アイドル	idle_sdc_detection()	idle_detection_task()	idle_detection_task()
TFAT FIT API 実行	tfat_sample()	tfat_sample_task() (注)	tfat_sample_task() (注)

注：関数を実行時に排他制御を行うべきですが、サンプルプログラムの簡略化を優先し省略しています。

6.2.2 処理フロー（SD カードドライバ）

SD カードドライバを使用したサンプルプログラムの処理フローは下図のとおりです。

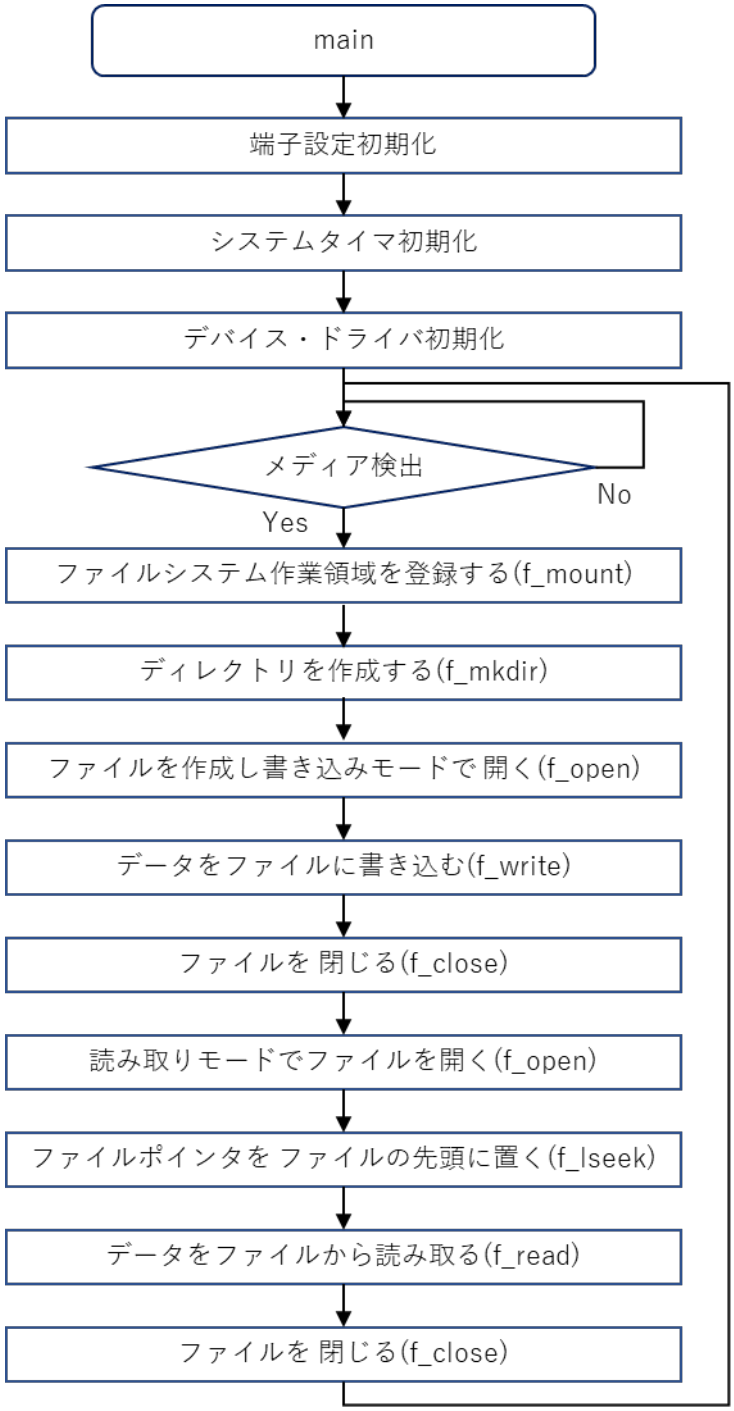

図 6.1 サンプルプログラム（SD カードドライバ）の処理フロー

6.2.3 USB ドライバを使用したサンプルプログラム

プログラムが実行されると、FAT ファイルシステムワークエリアを登録します。ディレクトリおよびファイルは記録メディア上に作成され、2KB のテキストデータをファイルに書き込みます。その後、ファイルを閉じます。書き込んだデータを確認するために、ファイルを読み出しモードで再び開きます。ファイルの内容全体を読み出し、プログラムの書き込みバッファデータと比較します。データの内容が一致しているかどうかをデバッガ上のデバッグコンソール(Renesas Debug Virtual Console)上に表示します。

表 6.3 デバッグコンソール表示の意味

表示文字列	意味
Detected attached USB memory.	USB メモリの挿入を検知しました。
Detected detached USB memory.	USB メモリの抜去を検知しました。
!!! Attach USB memory. !!!!	USB メモリを挿入してください。
!!! Detach USB memory. !!!!	USB メモリを抜去してください。
Start TFAT sample	サンプルプログラムを実行します。
Finished TFAT sample	サンプルプログラムが正常終了しました。
!!!!! TFAT error !!!!!	サンプルプログラムでエラーが発生しました。

ファイルのリード/ライトのためのデータは、r_data_file.c に定義しています。デフォルトでは”Renesas¥n”という文字列を繰り返し書き込みます。書き込むデータは合計 2KB（2048 バイト）です。必要に応じて、このデータおよび対応するマクロ FILESIZE を編集してください。

サンプルプログラムにて使用されている主な関数は以下のとおりです。RTOS の使用有無に関わらず処理内容は同じです。ただし、FreeRTOS および RI600V4 はタスクとして実行されます。

表 6.4 サンプルプログラムの主な関数

処理内容	None RTOS での 関数名	FreeRTOS での タスク関数名	RI600V4 での タスク関数名
初期化	main()	main_task()	main_task()
デバイス検知アイドル	idle_usb_detection()	idle_detection_task()	idle_detection_task()
TFAT FIT API 実行	tfat_sample()	tfat_sample_task() (注)	tfat_sample_task() (注)

注：関数を実行時に排他制御を行うべきですが、サンプルプログラムの簡略化を優先し省略しています。

6.2.4 処理フロー（USB ドライバ）

USB ドライバを使用したサンプルプログラムの処理フローは下図のとおりです。

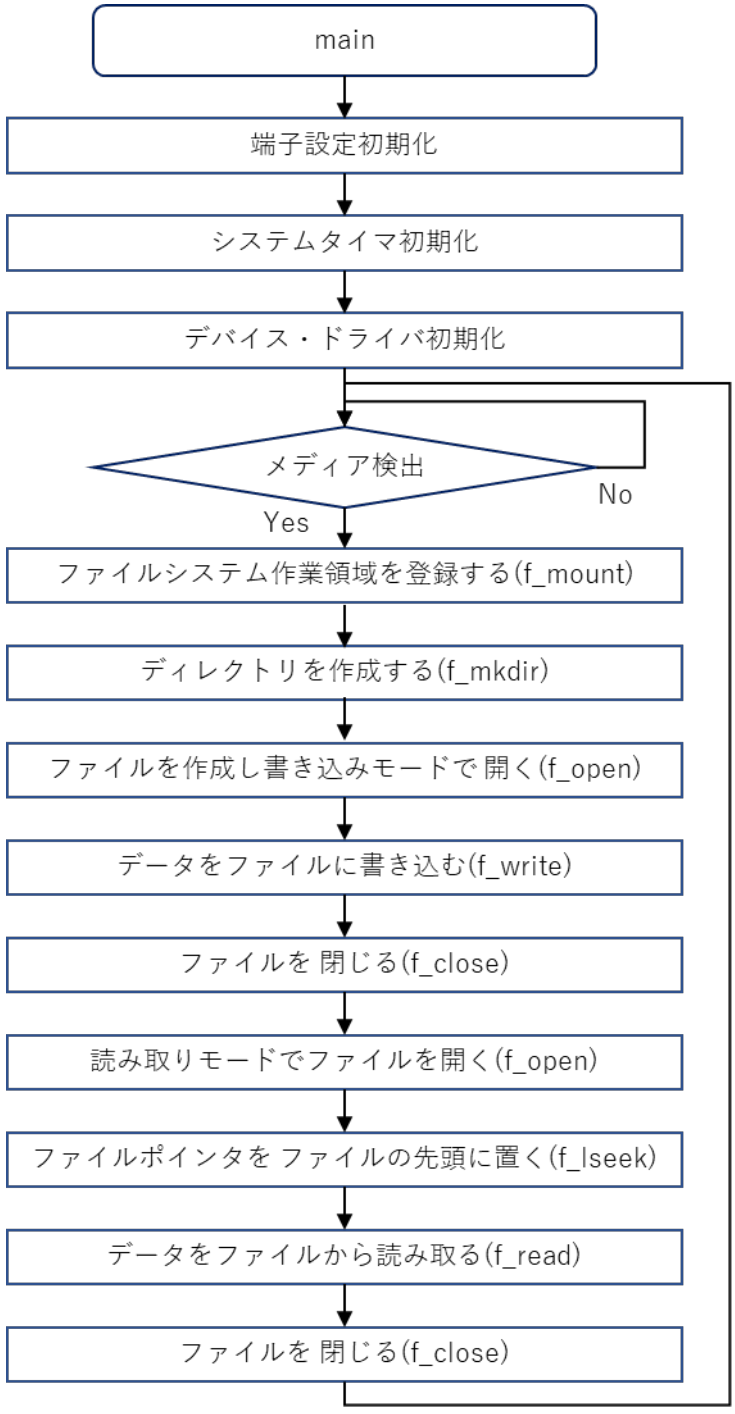

図 6.2 サンプルプログラム（USB ドライバ）の処理フロー

7. 付録

7.1 動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 7.1 動作確認環境 (Rev.3.04)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e^2 studio V7.1.0
Cコンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.3.00.00
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	リトルエンディアン
モジュールのリビジョン	Rev.3.04
使用ボード	Renesas Starter Kit for RX231 (型名：R0K505231Sxxxxxxx) Renesas Starter Kit+ for RX65N-1MB (型名：R0K50565NSxxxxxxx)
RTOS	なし

表 7.2 動作確認環境 (Rev.4.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V7.7.0 IAR Embedded Workbench for Renesas RX 4.13.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 コンパイルオプション：統合開発環のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.201904 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.13.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.00
使用ボード	Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx)
RTOS	FreeRTOS V10.0.00 RI600V4 V1.06.00

表 7.3　動作確認環境 (Rev.4.01)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V7.8.0 IAR Embedded Workbench for Renesas RX 4.14.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 コンパイルオプション：統合開発環のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.201904 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.01
使用ボード	Renesas Starter Kit for RX231 (型名：RTK55231xxxxxxxxxx) Renesas Starter Kit+ for RX64M (型名：RTK5564Mxxxxxxxxxx)
RTOS	FreeRTOS V10.0.00 RI600V4 V1.06.00

表 7.4　動作確認環境 (Rev.4.02)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio 2020-07 IAR Embedded Workbench for Renesas RX 4.14.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family V3.02.00 コンパイルオプション：統合開発環のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202002 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのリビジョン	Rev.4.02
使用ボード	Renesas Starter Kit for RX231 (型名：RTK55231xxxxxxxxxx) Renesas Starter Kit+ for RX72N (型名：RTK5572Nxxxxxxxxxx) Renesas Starter Kit+ for RX72M (型名：RTK5572Mxxxxxxxxxx)
RTOS	FreeRTOS V10.0.03 RI600V4 V1.06.00

7.2 トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file “platform.h”」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e^2 studio を使用している場合
  アプリケーションノート RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r_sdc_sd_rx module.」エラーが発生します。

A：追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

(3) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行するとコンフィグ設定が間違っている旨のエラーが発生します。

A：“r_tfat_rx_config.h”ファイルの設定値が間違っている可能性があります。“r_tfat_rx_config.h”ファイルを確認して正しい値を設定してください。詳細は「2.6 コンパイル時の設定」を参照してください。

(4) Q：端子設定していない場合発生する現象が発生します。

A：正しく端子設定が行われていない可能性があります。本 FIT モジュールを使用する場合は端子設定が必要です。詳細は「5 端子設定」を参照してください。

8. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）
テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）
ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデートの対応について

- テクニカルアップデートはありません。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.10.08	—	初版発行
1.01	2012.09.01	—	RX210 対応
1.02	2013.11.08	—	ドキュメントのタイトルを変更 章構成を変更 ライブラリソースに Fatfs の著作権表示を追加
1.03	2013.11.30	—	オープンソースのベースバージョンを V0.06→V0.09b に変更
3.00	2014.04.01	—	FIT モジュール対応
3.01	2014.12.28	—	RX71M,RX113 に対応 FIT 用 xml ファイルを更新しました
3.02	2015.03.01	—	RX231 に対応 FIT 用 xml ファイルを更新しました
3.03	2016.10.01	−	RX ファミリに対応 FIT 用 xml ファイルを更新しました
3.04	2018.11.30	−	2.4 制限事項 Real Time OS 使用時の注意事項を追加しました。 5.デモプロジェクト、6.付録 を追加しました。 FIT 用 xml ファイルを更新しました
4.00	2020.2.25	-	提供方法をライブラリ形式からソースコード形式に変更 オープンソースのベースバージョンを V0.09b→V0.13c に変更 以下のコンパイラに対応 ・GCC for Renesas RX ・IAR C/C++ Compiler for Renesas RX 以下の RTOS に対応（USB mini FIT を使用するデバイスを除く） ・FreeRTOS ・RI600V4 関数名から「R_TFAT_」を削除 ソースに Fatfs の著作権表示を追加 サンプルプログラムを RTOS 用に更新

Rev.	発行日	改訂内容	
		ページ	ポイント
4.01	2020.7.27	-	・以下の記憶メディアに対応 ・eMMC ・Serial Flash memory ・以下の記憶メディアに対しフォーマット機能（f_mkfs）を追加 ・eMMC ・Serial Flash memory ・セクタサイズ 4096 バイトに対応 ・USB mini FIT を使用するデバイスと以下の RTOS の組み合わせに対応 ・FreeRTOS ・RI600V4
4.02	2020.9.10	-	・以下の記憶メディアに対しフォーマット機能（f_mkfs）を追加 ・SD ・USB

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。