INTERFACE CO., LTD.

mini MIDDLEWARE SERIES

mini-FILE

初版　　2007 年 12 月 24 日
第二版　　2009 年 5 月 4 日

User's Manual

インターフェイス株式会社

ご注意

1．本製品及び本書はインターフェイス株式会社の著作物です。
したがって、本製品及び本書の一部または全部を無断で複製、複写、転載、改変することは法律で禁じられています。

2．本製品及び本書の内容については、改良のために予告なく変更することがございます。

3．本製品を運用した結果の他への影響については、責任を負いかねますのでご了承ください。

4．本製品は、外国為替及び外国貿易法の規定により戦略物資等輸出規制品に該当する場合があります。
国外に持ち出す場合には、日本国政府の輸出許可申請などの手続きが必要となる場合があります。

5．本製品は、医療機器、航空宇宙機器など人命にかかわる設備や機器での使用は考慮しておりません。
これらの設備や機器に本製品を使用され、本製品の故障により、人身事故、火災などが発生しても、弊社では責任を負いかねます。

6．本製品は国内仕様です。本製品を国外で使用された場合の不具合などについては対応できませんのであらかじめご了承ください。

・Windows2000,WindowsXP は、米国 Microsoft Corporation の米国およびその他の国における登録商標です。

【目次】

1 はじめに

このたびは mini-FLE をお買い上げいただき誠にありがとうございます。

2 本書について

本マニュアルは、mini-FILE の概要および取り扱いについて述べたものです。
本書は以下の章から構成されています。

なお，本書の内容は予告なしに変更することがあります。本書の内容について，ご不明な点がありましたら当社へお問合せください。

3 製品内容

3.1 概要

本 mini-FILE は FAT12 / FAT16 / FAT32、VFAT に対応しており、組み込みシステム上で、FAT 型ファイルシステムを実現します。

本製品は､当社 mini ミドルウェアシリーズの USB ホストストレージスタック「mini-USBHostSTR」と組み合わせる事で、USB ストレージデバイス(USB メモリなど)へのファイルの読み出し／書き込みを実現することが可能です。

また、本製品のサンプルドライバとして、Windows 環境下（A ドライブ ; FD）でのドライバも提供しており、アプリケーション開発をスムーズに実現することが可能となっております。

3.2 フォルダ構成

mini-FILE が提供するファイルが格納されているディレクトリの構成は次の通りです。

```
+【Doc】                                マニュアル（本書を含む）
+【Src】
  +【ITFLib】                           弊社共通ライブラリ
    +--【Include】                      各種設定定義ファイル
        :（以下省略 ;「共通ライブラリ User's Manual」も合わせて参照下さい）
        :
  +【miniFile】
    +--【Include】                      mini-FILE ヘッダファイル
    +--【Library】                      mini-FILE ライブラリ格納用フォルダ
        +--【Object】                   mini-FILE オブジェクト格納用フォルダ
    +--【Sample】
        +--【Application】              サンプルアプリケーションソースファイル
        +--【Driver】                   サンプルドライバ
    +--【Source】                       mini-FILE ソースファイル
        +--【Depend】                   ドライバソース格納用フォルダ
        +--【Include】                  mini-FILE 内部定義ヘッダファイル
        +--【Io】                       mini-FILE API ソースファイル
```

3.3 ファイル構成

フォルダ名	説明	ファイル名
¥Doc	マニュアル	mini-FILE User's Manual.pdf 共通ライブラリ User's Manual.pdf
¥Src¥ITFLib		
		以下省略
¥Src¥miniFile		
¥	修正履歴ファイル	修正履歴(mini-FILE).txt
	HEW プロジェクトファイル	miniFILE_IFBUSBR1.hwp
¥Include	mini-FILE ヘッダファイル	itffileconfig.h fileresource.h itffile.h itffileerror.h DriverIF.h
¥Source	mini-FILE ソースファイル	BUF.C CLOSE.C COMMON.C DEV.C DISK.C DISKINIT.C FAT.C FINDPATH.C FORMAT.C GETPATH.C MKDIR.C MOUNT.C OPEN.C RCNV_TBL1.C RCNV_TBL2.C READ.C UNICNV.C VFAT.C WRITE.C
¥Source¥Depend	時間ドライバソースファイル	DRV_TIME.C ※1
	同期/非同期用下位ソースファイル	WRAPDRIVER.C WRAPDRIVER.H

※1.「¥Src¥miniFile¥Sample¥Driver」内のファイルと同一になっております。

¥Source¥Include	mini-FILE 内部定義ヘッダファイル	FSYS_PROTO.H FSYS_STRUCT.H FSYS_VFAT.H rcnv_tbl.h ucnv_tbl.h
¥Source¥Io	mini-FILE API ソースファイル	APPLIIF.C
¥Sample¥Application	サンプルアプリケーション	sample.c
	Windows プロジェクトファイル (Visual C++ Ver6.0)	sample.dsw sample.dsp
¥Sample¥Driver	時間ドライバソースファイル	DRV_TIME.C
	Windows 用サンプルドライバ	DRV_WIN.C

注意）

「¥Src¥miniFile¥Sample¥Driver」にのみ「DRV_WIN.C」が格納されていますので、Windows 環境下で動作させたい場合は、お客様自身で、「¥Src¥miniFile¥Source¥Depend」にコピーして頂く必要があります。

3.4 導入

＜提供サンプル環境その１＞
動作環境；IFB-USBR1 (当社作成、評価ボード)
CPU；H8S/2218(ルネサス社製)
メディア；USB ストレージデバイス(USB メモリ)
OS；なし
コンパイラ；

High-performance Embedded Workshop	Version 4.02.00
H8S,H8/300 Series C/C++ Compiler	Version 6.01.03
H8S,H8/300 Series C/C++ Library Generator	Version 2.01.01
H8S,H8/300 Series Assembler	Version 6.01.01
Optimizing Linkage Editor	Version 9.02.00

備考；「mini-USBHostSTR」側のマニュアルと合わせて参照下さい。
また、サンプルとして提供している時間ドライバ(DRV_TIME.C)は、固定値
(2003 年 12 月 1 日 00 時 00 分 00 秒)を返すようになっておりますので、必要に応じて、
お客様ご自身で、ハードウェアリソース等から時間データを取得するような変更が
必要となります。

＜提供サンプル環境その２＞
動作環境；Windows2000、WindowsXP （要 Administrator 権限)
メディア；FD（A ドライブ）
コンパイラ；Microsoft Visual C++ Ver6.0
備考；上記環境下であれば、基本的には変更することなく動作致します。
提供サンプルドライバを変更することで、C ドライブ(HDD)等への読み書きも、
対応可能ですが、トラブル軽減のため、未対応で提供させて頂いております。
（CD イメージでは、上記サンプル環境その１がコンパイルされるようになっていますので、
Depend フォルダに「DRV_WIN.C」をコピーすればコンパイル動作可能となります）

3.4.1 コンパイルオプション

（1）共通コンフィグレーション

当社製の各製品共通のヘッダファイル(itfconfig.h）で定義されている設定を変更することで、お客様の環境にあった構成を構築することが可能となります。

（itfconfig.h には他のオプションもありますが、本 mini-FILE では不要な設定です）

名称	オプション	内容
エンディアン	__LITTLE_ENDIAN__	リトルエンディアン
	__BIG_ENDIAN__	ビックエンディアン
OS	__KERNEL_NORTI_V4__	NORTi Ver4 対応
	__KERNEL_TOPPERS__	Toppers 対応 ※1
	非指定	OS なし
OS 周期	ITF_MSEC	OS インターバルタイマー周期 ※2

※1．弊社での動作に関しては未確認となっております。

※2．OS 側で設定している同様の設定値を記述する必要があります。

（2）mini-FILE 専用コンフィグレーション

本 mini-FILE 専用のコンフィグレーション用ヘッダファイル(itffileconfig.h）で定義されている設定を変更することで、お客様の環境にあった構成を構築することが可能となります。

①VFAT 対応の設定

VFAT(ロングファイル名)対応の可否を選択出来ます。

「__VFAT_ASCII__」設定の場合は、漢字コードを非サポートとしたまま、ロングファイル名をサポートすることが出来ます。

なお、「__VFAT__」指定の場合は、約 100KB 近くの ROM 領域が必要となります。

オプション名	内容
__VFAT__	VFAT 対応
__VFAT_ASCII__	ASCII 文字での VFAT 対応
非指定	VFAT 未対応

②リムーバブルディスク(活線挿抜)の対応／非対応の設定
電源ＯＮ状態でのメディアの抜き挿しに対応するか否かの設定を行うことが出来ます。
USB ストレージデバイスに対応(mini-USBHostSTR と合わせて使用)する場合は
リムーバブルディスクへの対応とする必要があります。
なお、OS 使用時に「__REMOVABLE_DISK__」を設定するとタスク／メールボックス
／固定長メモリプールの OS 資源が必要となります。

オプション名	内容
__REMOVABLE_DISK__	リムーバブル対応
__UNREMOVABLE_DISK__	リムーバブル非対応

③同時オープンファイル数の設定
この設定を変更することで、同時オープンが可能なファイル数を制限することが出来ます。
設定値１に付き約１７０バイトの RAM 領域が必要となります。
ただし、下記「__FILE_PATH_LENGTH__」の設定値をデフォルト状態から増やしていると、
その分加算せされます。

オプション名	内容
__FILE_COUNT__	同時にオープン可能なファイル数

④キャッシュサイズの設定
この設定を変更することで、ディスクアクセスの回数が軽減されます、結果的に、
パフォーマンスを向上することが出来ます。
設定値１に付き約５２０バイトの RAM 領域が必要となります。

オプション名	内容
__CACHE_COUNT__	メディアアクセス軽減用のキャッシュサイズ指定

⑤ファイル／パス名の文字列長(フルパス)の上限の設定
f_open( )、f_mkdir( )における指定可能なファイル／パス名(フルパス指定)の最大の文字数を、
設定することが出来ます。
設定値１に付き３バイトの RAM 領域が必要となります。
また、設定値１に付き２バイト分のスタック領域が必要となりますので、注意して下さい。
なお、VFAT 未対応("__VFAT__"、"__VFAT_ASCII__"のいずれも指定しない状態)の場合は、
この設定値は、意味を持ちません。

オプション名	内容
__FILE_PATH_LENGTH__	ファイル/フォルダ名の文字列長の上限を指定

⑥遅延書き込みの設定

不要なディスクアクセスを軽減させるために、遅延書き込みを行う設定を
することが出来ます。
ただし、遅延書き込み命令用に特殊 API（f_flush( )）をコールすることが必要となります。

オプション名	内容
未指定	通常動作
__HIGH_PERFORMANCE__	遅延書き込みモード指定

（3）OS 資源コンフィグレーション

本 mini-FILE が使用している OS 資源に関して、リソース用ヘッダファイル（fileresource.h）で
定義されている設定を変更することで、お客様の環境にあった構成を構築することが
可能となります。

名称	オプション	内容
セマフォＩＤ (API 排他制御用)	ITFFILE_SEMID	本ＩＤ番号による生成
	非指定	ＩＤ自動割り当てによる生成
メールボックスＩＤ (活栓挿抜監視用)	ITFFILE_MBXID	本ＩＤ番号による生成
	非指定	ＩＤ自動割り当てによる生成
メモリプールＩＤ (活栓挿抜監視用)	ITFFILE_MPFID	本ＩＤ番号による生成
	非指定	ＩＤ自動割り当てによる生成
タスクＩＤ (活栓挿抜監視用)	ITFFILE_TSKID	本ＩＤ番号による生成
	非指定	ＩＤ自動割り当てによる生成
タスクプライオリティ (活栓挿抜監視用)	REMVABLE_PORIORTY	タスクの優先度　※1
メモリブロック数 (活栓挿抜監視用)	MEMORY_BLOCK_COUNT	メモリブロック数　※2

※1．活栓挿抜監視用のタスクの優先順位を指定します。
優先順位を高く(値としては小さく)することで、メディア挿抜時の応答速度が
早くなります。
また、ドライバ側にもタスクが存在する場合は、ドライバ側のタスクより優先度を
低くする必要があります。

※2．メディア抜き挿しのイベントを保留できる数になります。
連続的に抜き挿しを行う場合は増やす必要があります。

（4）OS 資源

本 mini-FILE は、OS(NORTi Ver4)使用時に、呼び出すシステムコールは、
全て抽象化したコードにしてあります。このため、他の OS の移行を容易に実現することが
可能になっており、お客様で同様の処理を作成していただく事で、他の OS にも対応することが
容易となっています。
以下に実際に、本 mini-FILE が使用している OS 抽象化システムコールを明記致します。

OS 資源	抽象化 OS 資源	備考
セマフォ	OSAPI_Cre_Sem OSAPI_Acre_Sem OSAPI_Del_Sem OSAPI_Wai_Sem OSAPI_Sig_Sem	OS 使用時は無条件で必要となる資源です。 iTRON 仕様のセマフォと同等の処理を実装する必要があります。
メールボックス	OSAPI_Cre_Mbx OSAPI_Acre_Mbx OSAPI_Del_Mbx OSAPI_Snd_Mbx OSAPI_Rcv_Mbx	“__REMOVABLE_DISK__”を指定した際には、必要となる資源です。 iTRON 仕様のメールボックスと同等の処理を実装する必要があります。
メモリプール	OSAPI_Cre_Mpf OSAPI_Acre_Mpf OSAPI_Del_Mpf OSAPI_Pget_Mpf OSAPI_Rel_Mpf	“__REMOVABLE_DISK__”を指定した際には、必要となる資源です。 iTRON 仕様の固定長メモリプールと同等の処理を実装する必要があります。
タスク	OSAPI_Cre_Tsk OSAPI_Acre_Tsk OSAPI_Del_Tsk OSAPI_Ext_Tsk OSAPI_Wup_Tsk OSAPI_Tslp_Tsk	“__REMOVABLE_DISK__”を指定した際には、必要となる資源です。 iTRON 仕様のタスクと同等の処理を実装する必要があります。

（5）その他のコンフィグレーション

本 mini-FILE は、コンフィグレーション用ヘッダファイル(itffileconfig.h）で定義されている設定以外に、以下の設定を用意しています。
これらの設定を有効にしたい場合は、お客様ご自身で、コンパイルオプション等で設定して頂く必要があります。

①メモリ領域の動的割り当ての設定

mini-FILE 内部で使用するメモリ空間(ほぼ全て)を、上位アプリケーションから割り当てられた領域を使用するように、設定することが出来ます。
具体的には、初期化時(ITF_FileSystemInit( )にて指定)に、上位アプリケーションで用意したメモリ空間を指定することで、mini-FILE はそれ以降の処理をこのメモリ空間を使って動作します。その後、終了時(ITF_FileSystemEnd( ))に解放します。
このため、常時ファイルシステムを使用しないようなシステムでは、
ファイルシステムを使用しない期間のメモリ領域を上位アプリケーションと共有することが可能となります。
なお、上位アプリケーション側で、用意しなければいけないメモリ空間は、“itffile.h”で定義されている「FILE_MEMORY_SIZE」バイトです。この領域分が確保されたメモリ空間を初期化関数(ITF_FileSystemInit( ))に指定する必要があります。
また、初期化時(ITF_FileSystemInit( ))から終了時(ITF_FileSystemEnd( ))までの間は、このメモリ空間へのアクセスが保証(他の処理よりアクセスされないなど)されていることとします。

オプション名	内容
未指定	静的メモリ空間を使用
__DYNAMIC_MEMORY__	動的メモリ空間を使用

②連続書き込みモード選択

同一のフォルダに連続的にファイルを作成する場合に、不要なファイル検索等を極力排除し、高速に書き込みを行えることを実現しております。
通常のファイルシステムは、ファイルの作成時には、毎回、同一のファイル名が存在しないことを確認した上で、空き領域を検索し、そこにファイルを作成すると言う処理になっております。このため、単純に連続的なファイル書き込みをする場合でも、ファイル数が増えると、処理時間が増えてしまうと言う問題があります。
この機能は、これらの処理を上位側の管理のもとに省略することで、高速な書き込みを実現しております。
使用方法／注意事項など、詳細は、「処理能力の向上に関して」の章を合わせてご確認下さい。

オプション名	内容
未指定	通常動作
__SEQUENCE_WRITE__	連続書き込みモード

③非同期型(ポーリング)API 選択

本 mini-File には、非同期型(ポーリング)の API に対応しており、ファイル操作の API で、待ちが発生するような処理(USB の応答待ち)等で、いったん API を終了させることで、長期間処理を占有し続けないような機能を実現しております。

なお、この__ASYNCHRONOUS_API__指定を行った際には、“f_asynchronous( )”と言う API を合わせて使う必要があります。

この API は、各 API(ITF_FileSystemInit( )、ITF_FileSystemEnd( )は対象外)で、いったん終了したその続きの処理を行わせるために、メイン処理(非割り込みハンドラ)から、周期的に呼び出す必要があり、各 API の処理が終了(FILES_DRIVER_INCOMPLETE 以外の戻り値が返るまで)するまで、本関数を呼び続ける必要があります。

オプション名	内容
未指定	通常動作
__ASYNCHRONOUS_API__	非同期型 API

なお、本オプションと「__DYNAMIC_MEMORY__」オプションを同時に指定することは、出来ません。(コンパイル時にエラーメッセージを出すようになっております)

（6）タスク優先度

本 mini-FILE は、OS 使用時には以下に示すタスクが存在し、デフォルトのタスク優先度が設定されています。

ユーザーアプリケーションタスク(mini-FILE の API を使用するタスク)としては、このタスクの優先度より、低く(値としては大きく)する必要があります。

タスク名称	定義名	タスク優先度
リムーバブルディスク(__REMOVABLE_DISK__)指定時		
メディア挿抜イベント監視タスク	FILE_RESOURCE_TASK_PRIORITY	5

（7）スタック領域

スタックサイズの指定は、コンパイル時に指定する方法や、明示的にはコンパイル時に指定しない方法など、コンパイル方法などによって異なりますが、現在の mini-FILE が最低限必要とするスタックサイズは、以下の通りとなっております。

割り込み処理や実装方法などやコンパイラによっては、必要となるスタックサイズは、異なりますので、注意して下さい。

スタックサイズ	デフォルト値
約 520 バイト＋__FILE_PATH_LENGTH__×2	約 680

3.4.2 コンパイル環境

本 mini-FILE をコンパイルするにあたって以下の設定が必要となります。

（1）インクルードパス
提供時のフォルダ名で以下の５つのパスをインクルードパスとして、コンパイラへの設定が必要となります。

- src¥itflib¥include
- src¥minifile¥include
- src¥minifile¥Source¥Include
- src¥minifile¥Source¥Depend（ドライバの構成によります）
- その他 OS 用ヘッダファイルパス

お客様で、フォルダ構成を変えた場合は、同等のフォルダに対して、設定が必要となります。

（2）ヘッダファイル
本 mini-FILE をご使用になる際は、「itffile.h」（src¥minifile¥include)を、ユーザーアプリケーションは、インクルードする必要があります。

3.5 mini-FILE独自仕様

本 mini-FILE は、スムーズに開発が出来るように、ANSI のファイル操作 API をイメージして実装されております。
ただし、mini-FILE では独自に以下のような仕様(制限事項)を設けることで、処理の単純化、より軽量化を図っております。

（1）フルパス指定のみの対応
本 mini-FILE は、カレントディレクトリと言う概念がありません。そのため、ファイル／フォルダ操作時には、フルパス指定(ルートディレクトリから指定)で行う必要があります。
具体的には、f_open( )、f_mkdir( )がこの対象となります。
例えば、ルートディレクトリ直下に、「TEST」と言うフォルダが存在していて、
このフォルダ内の「12345678.TXT」と言うファイルを示したい時は、以下のように指定します。

“TEST¥12345678.TXT”
　もしくは
“¥TEST¥12345678.TXT”

（2）単一ドライブのみの対応
本 mini-FILE は、複数ドライブには対応をしておりません。このため、ファイル／フォルダを操作する際には、ドライブ名を指定する必要がありません。
具体的には、f_open( )、f_mkdir( )がこの対象となります。
例えば、以下の「×」で示されるような指定をする必要はなく、「○」で示されたように指定して下さい。

× ； “C:¥TEST¥12345678.TXT”
○ ； “TEST¥12345678.TXT” もしくは “¥TEST¥12345678.TXT”

ただし、API によっては、互換性(ITF-FILE との)のためにドライブ指定するようなパラメータが存在するものがあります。
具体的には、disk_mount( )、disk_unmount( )、f_format( )、f_flush( )になります。
これらの関数に渡されるドライブ名は、実際にはパラメータチェックの対象からも外されており、本 mini-FILE では意味を持たないものとなっておりますので、「0」などの固定値を渡して頂ければ、問題ありません。

また、デバイスの挿抜時に上位アプリケーション側へ通知する手段として、コールバック関数(disk_mount( )にて登録)を使う手法をとっておりますが、このコールバック関数にも第二引数のドライブ名が渡される仕様となっておりますが、これも同様に意味を持たないもの(0 固定)となっております。
また、第一引数のデバイス番号も同様に意味を持たないもの(1 固定)となっております。

4 API仕様

mini-FILE の API 仕様に関して以降に解説します。

4.1 初期化／終了関数

ITF_FileSystemInit

機　能　ライブラリ初期化

形　式　**ISTATUS ITF_FileSystemInit(void)**

ISTATUS ITF_FileSystemInit(uint8_t *mem)　（__DYNAMIC_MEMORY__指定時）
mem　　：mini-FILE が使用するメモリ領域の先頭ポインタ

解　説　ライブラリの初期化を行います。
本関数は、他のどの API よりも優先して呼び出す必要があります。
“__DYNAMIC_MEMORY__”指定時は、「FILE_MEMORY_SIZE」バイト分のメモリ領域が存在するポインタを指定する必要があります。
（合わせて「コンパイルオプション」の章もご確認下さい）
なお、このメモリ領域を使用して、mini-FILE は動作致します。

戻　値　**0　　：正常終了**
その他　　：エラーコード一覧参照

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet=ITF_FileSystemInit();
    if(theRet!=0){
        // エラー処理
    }
}
```

ITF_FileSystemEnd

機　能　ライブラリ終了処理

形　式　**void ITF_FileSystemEnd(void)**

解　説　ライブラリの終了処理を行います。
なお、オープン中のファイルがある場合は、事前にクローズ(f_close( ))する必要があります。
また、マウントされている場合は、事前にアンマウント(disk_unmount( ))する必要があります。

戻　値　なし

例　ライブラリ利用の終了時に一度だけ Call します

```
int main(void)
{
      :
    f_close( fp );
    disk_unmount( 1,0 );
    ITF_FileSystemEnd();
}
```

disk_mount

機　能　ディスク装置マウント処理

形　式　ISTATUS disk_mount(DiskDriver pDriver,int8_t deviceno,int8_t letter,
EventDriver pCallback, int8_t disk)

pDriver　：ドライバ関数ポインタ
deviceno　：デバイス識別番号（１～）
letter　：ドライブレター指定（’A’～’Z’　or　-1=自動アサイン）
pCallback　：マウント完了通知用コールバック関数ポインタ
（本関数の引数；ドライバ識別番号、ドライブレター、挿(1)抜(0)状態 ）
disk　：リムーバブルディスク指定
（DISK_UNREMOVABLE；固定ディスク、
DISK_REMOVABLE；リムーバブルディスク）

解　説　ディスク装置のマウント処理を行います。
deviceno、*letter* は、他のバージョンのファイルシステム(ITF-FILE)との互換用に存在するパラメータとなっており、本ファイルシステムでは、意味を持ちません。
これは、本ルーチンに登録されたコールバック関数の第一、第二引数に関しても同様のことが言えます。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）

戻　値　0　：正常終了
その他　：エラーコード一覧参照

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet=disk_mount( pDriver, 1, -1, pCallback, DISK_REMOVABLE);
    if(theRet!=0){
        // エラー処理
    }
}
```

disk_unmount

機　能　ディスク装置アンマウント処理

形　式　ISTATUS disk_unmount(int8_t deviceno,int8_t letter)
deviceno　　：デバイス識別番号（0 or 1～）
　0 指定時は「letter」のドライブのみアンマウント
　1～指定時は「letter」に関わらず同一のデバイス識別番号のドライブをアンマウント
letter　　：ドライブレター　（デバイス識別番号が 0 指定の時のみ有効）

解　説　ディスク装置のアンマウント処理を行います。
なお、オープン中のファイルがある場合は、事前にクローズ(f_close( ))する必要があり、mini-FILE の終了処理(ITF_FileSystemEnd( ))を行う場合は、事前に本 API でアンマウントしておく必要があります。
また、*deviceno*、*letter* は、他のバージョンのファイルシステム(ITF-FILE)との互換用に存在するパラメータとなっており、本ファイルシステムでは、意味を持ちません。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）

戻　値　0　　：正常終了
その他　　：エラーコード一覧参照

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet=disk_unmount( 1, 0 );
    if(theRet!=0){
        // エラー処理
    }
}
```

4.2 ディレクトリ操作関数

f_mkdir

機　能　新規ディレクトリ作成

形　式　ISTATUS f_mkdir(uint8_t* path)
path　　：作成ディレクトリパス名

解　説　指定した名前の新規ディレクトリを作成します。
パスの指定書式は、フルパス指定となります。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）

戻　値　0　　：正常終了
その他　　：エラーコード一覧参照

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet=f_mkdir("new_dir");
    if(theRet!=0){
        // エラー処理
    }
}
```

4.3 ファイル操作

f_open

機　能　ファイルのオープン・クリエイト

形　式　**ISTATUS f_open(EXTFCB ** fcb, uint8_t *path, int32_t flag, int32_t pmode)**

fcb	**：ユーザー指定ファイル管理領域ポインタのポインタ**
path	**：オープンするファイルのパス名**
flag	**：オープンするファイルの操作指定**
pmode	**：flag が O_CREAT ならば属性指定**

解　説　既存ファイルのオープン、または新規作成を行います。
引数 *fcb* にはファイル管理領域ポインタのポインタを渡し、その後 f_read( )、f_write()、f_close( )で、使用します。
パスの指定書式は、フルパス指定となります。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）
引数 *flag* に関して、
O_CREAT(|O_RDWR) 指定したファイルが存在しない場合は、新規ファイルを作成し、既に存在する場合は、その内容を破棄して新規ファイルを作成して開きます。
O_CREAT|O_EXCL(|O_RDWR) 指定したファイルが既に存在する場合は、エラー値を返し、存在しない場合は、新規ファイルを作成して開きます。
O_APPEND(|O_RDWR) 既に存在するファイルを開く際に、ファイル ポインタをファイルの終端に移動したうえで、ファイルを開きます。
O_RDONLY 既に存在するファイルを開く際に、ファイルを読み取り専用で開きます。
O_WRONLY(|O_CREAT)(|O_APPEND) ファイルを書き込み専用で開きます。この際、O_CREAT を合わせて指定した場合は、新規ファイルを作成して開き、もしくは O_APPEND を合わせて指定した場合は、既に存在するファイルの書き込み位置を終端に移動して開きます。
引数 *pmode*:は *flag* で *O_CREAT* を指定したときのみ必要となります。
S_IREAD ファイルを読み取り専用ファイルとして作成します。
S_IREAD|S_IWRITE 読み取りおよび書き込みが出来るファイルを作成します。

戻　値

0	**：正常終了**
その他	**：エラーコード**

例

```
int main(void)
{
    ISTATUS theRet;
    EXTFCB *theFcb;
     :
theRet=f_open(&theFcb,"¥¥test.txt", O_CREAT|O_RDWR, S_IREAD|S_IWRITE);
    if(theRet!=0){
        // エラー処理
    }
}
```

f_close

機　能　オープンファイルクローズ

形　式　ISTATUS f_close(EXTFCB* fcb)
　　fcb　　：オープンされている fcb ポインタ

解　説　オープンしたファイルのクローズを行います。

戻　値　0　　：正常終了
　　その他　　：エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
    EXTFCB *theFcb;
     :
    theRet=f_open(&theFcb,"¥¥test.txt", O_CREAT|O_RDWR, S_IREAD|S_IWRITE
     :
    theRet=f_close(theFcb);
    if(theRet!=0){
        // エラー処理
    }
}
```

f_read

機　能　ファイルデータリード

形　式　ISTATUS f_read(EXTFCB* fcb, void* buffer, uint32_t　count, uint32_t* pcount)

fcb	: ユーザー管理 FCB ポインタ
buffer	: リードデータ格納ユーザーバッファ
count	: リードデータバイト数
pcount	: 実際リードしたバイト数格納ポインタ

解　説　データ読み出し処理を行います。
ファイルの読み出し位置が、既に終端にある場合(1 バイトも読みだせない)は、エラーコードとして、”FILES_END_OF_FILE”が返されます。

戻　値

0	: 正常終了
その他	: エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
    uint32_t theCount;
    EXTFCB *theFcb;
    uint8_t theBuffer[64];
     :
    theRet=f_open(&theFcb,“¥¥test.txt”, O_RDWR, S_IREAD|S_IWRITE);
     :
    theRet=f_read(theFcb, theBuffer, 64, &theCount);
    if(theRet!=0){
        // エラー処理
    }
}
```

f_write

機　能　ファイルデータライト

形　式　ISTATUS f_write (EXTFCB* fcb, void* buffer, uint32_t　count, uint32_t* pcount)

fcb	: ユーザー管理 FCB ポインタ
buffer	: ライトデータ格納ユーザーバッファ
count	: ライトデータバイト数
pcount	: 実際ライトしたバイト数格納ポインタ

解　説　データの書き込み処理を行います。

戻　値

0	: 正常終了
その他	: エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
    uint32_t theCount;
    EXTFCB *theFcb;
    uint8_t theBuffer[64];
     :
    theRet=f_open(&theFcb,"¥¥test.txt",O_CREAT|O_RDWR, _IREAD|S_IWRITE);
     :
    theRet=f_write(theFcb, theBuffer, 64, &theCount);
    if(theRet!=0){
        // エラー処理
    }
}
```

f_format

機　能　ディスクフォーマット

形　式　ISTATUS　f_format(int8_t letter)
letter　　：ドライブレター指定（’A’～’Z’）

解　説　指定ドライブのクイックフォーマットを行います。
なお、*letter* は、他のバージョンのファイルシステム(ITF-FILE)との互換用に存在するパラメータとなっており、本ファイルシステムでは、意味を持ちません。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）

戻　値　0　　：正常終了
その他　　：エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet= f_format( 'C' );
    if(theRet!=0){
        // エラー処理
    }
}
```

注意）物理／論理フォーマットには対応しておりません。

f_flush

機　能　遅延データ強制書き込み

形　式　**ISTATUS　f_flush(int8_t letter)**
letter　　：ドライブレター指定（’A’～’Z’）

解　説　__HIGH_PERFORMANCE__指定を行った際に、必要となる API です。
遅延書き込み状態となっているデータを強制的にメディアに書き込む処理です。
なお、*letter* は、他のバージョンのファイルシステム(ITF-FILE)との互換用に存在するパラメータとなっており、本ファイルシステムでは、意味を持ちません。
（合わせて「mini-FILE 独自仕様」の章もご確認下さい）

戻　値　**0　　：正常終了**
その他　　：エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet= f_flush( 'C' );
    if(theRet!=0){
        // エラー処理
    }
}
```

f_asynchronous

機　能　非同期用ポーリング関数

形　式　**ISTATUS　f_asynchronous(void)**

解　説　__ASYNCHRONOUS_API__指定を行った際に、必要となる API です。
各 API(ITF_FileSystemInit( )、ITF_FileSystemEnd( )は対象外)で待ち(USB の応答待ちなど)が発生するような処理の場合に、メイン処理(非割り込みハンドラ)から、周期的に呼び出すためのポーリング関数となります。
各 API の処理が終了(FILES_DRIVER_INCOMPLETE 以外の戻り値が返るまで)するまで、本関数を呼び続ける必要があります。
(処理が終了するより前に、違う API をさ呼び出した場合は、「FILES_DRIVER_BUSY」が返されます)
また、以下の API で渡される、各パラメータは処理が終了
(FILES_DRIVER_INCOMPLETE 以外の戻り値が返るまで)するまで、データの中身(指定アドレス先のデータ)を保持しておく必要があります。

f_mkdir( )　；作成ディレクトリパス名（**path*）
f_open( )　；オープンするファイルのパス名（**path*）
f_read( )　；リードデータ格納ユーザーバッファ（**buffer*）
　　実際リードしたバイト数格納ポインタ（**pcount*）
f_write( )　；ライトデータ格納ユーザーバッファ（**buffer*）
　　実際ライトしたバイト数格納ポインタ（**pcount*）

戻　値　**0**　：正常終了
その他　：エラーコード

例

```
int main(void)
{
    ISTATUS theRet;
     :
    theRet=f_mkdir("new_dir");
    while( theRet != FILES_DRIVER_INCOMPLETE ){
        //その他の処理
        theRet = f_asynchronous( );
    }
}
```

5 サンプルアプリケーション

5.1 概要

本製品には、当社 mini ミドルウェアシリーズの USB ホストストレージスタック「mini-USBHostSTR」と組み合わせた簡単なサンプルアプリケーションを同封してあります。

このサンプルアプリケーションは、以下のような動作を行います。

①USB ストレージデバイス(USB メモリ)の挿入を待つ。
②ルートディレクトリに「LongFileName_TestFolder」を作成する。
③ルートディレクトリに、「LongFileName_TestFile.bin」を作成する。(512 バイト)
④作成したファイルを読み出して、ベリファイする。

これは、「sample.c」ファイルで提供しております。
(「X:¥Src¥miniUSBHost¥Sample¥Application¥IFBUSBR1_FILE_ST.hws」にてコンパイル環境を構築できます)

6 処理能力の向上に関して

本章では、本 mini-FILE を、より高速に処理させるために有効な機能／手法に関して記載します。

（1）全体的なパフォーマンスの向上

本ファイルシステムは、メディアへのアクセス用にソフト的なキャッシュバッファを設けてあります。そのため、このキャッシュバッファを大きくすることで、全体的なパフォーマンスを向上することが可能です。特に同一のファイルに対してアクセスする(書き込みした後に読み出すなど)ようなシステムには、より有効的な機能となります。
なお、この設定方法に関しては、前述の「コンパイルオプション」の章の“__CACHE_COUNT__”に関する記載をご確認下さい。

（2）遅延書き込みによるパフォーマンス向上

上記に記載の通り、ソフト的なキャッシュバッファを設けてありますが、このキャッシュバッファをフラッシュするタイミングを API(f_flush())によって指定することで、遅延書き込みを実現し、より効果的な処理能力の向上を行うことが出来ます。特に同一のファイルに対してアクセスする(書き込みした後に読み出すなど)ようなシステムには、より有効的な機能となります。

ただし、実際にメディアへの書き込みを遅延するため、途中でデバイスが抜かれた場合に、壊れる可能性があるファイル／ディレクトリが通常時と比べると、広範囲に及ぶ可能性がありますので、ご注意が必要となります。
なお、この設定方法に関しは、前述の「コンパイルオプション」の章の“__HIGH_PERFORMANCE__”に関する記載をご確認下さい。

（3）連続書き込みモード

単純に新規作成でファイル作成を繰り返す(f_open() → f_write() → f_close() → f_open() →・・・)ようなシステム(データロガーなど)で、以下の条件(制限事項)を満たしていれば、本機能を使うことで、高速な処理を実現することが出来ます。

ただし、以下の条件を満たしていない状態で使用すると、既にメディア上に存在していたファイルを上書きしたり、書き込みをしたファイルが壊れたりする場合がありますので、ご注意下さい。

- 本機能の使用に際しては、コンパイルオプション“__SEQUENCE_WRITE__”を指定する必要があります。
- ファイルのオープン時には、“O_SEQUENCE”フラグ(f_open()の第三パラメータ)を指定する必要があります。(このフラグを指定しないと従来の動作のままです)
- この動作モードへ移行(f_open()を“O_SEQUENCE”指定でオープン)した後は、f_open()、f_write()、f_close()以外の API を呼び出すと、通常のモードへ移行します。
- 同一のフォルダに対するファイル操作のみを許容しております。
  一番最初の f_open()と同じフォルダに、二回目以降は書き込みを行います。
  (二回目以降の f_open()をフォルダ付きのファイル指定をするとエラーになります)
- この動作モード状態では、同時に複数のファイルをオープンしないで下さい。
- 書き込みを行うフォルダには、同一のファイル名を作成するような指定をしないで下さい。
- ディスクの状態が連続的に空き領域がある状態(フォーマット直後など)で、本機能を使用して下さい。

［ 使用例 ］

```
        :
    for(i=0;i<1000;i++){
        sprintf(srcfilename,"ABCDEFGHIJKLMNOPQRSTUVWXYZ_%03d",i);
        if((f_open(&fp,srcfilename,O_RDWR|O_CREAT|O_SEQUENCE,S_IREAD|S_IWRITE))==0){
            f_write(fp,buffer,1024,&count);
            f_close(fp);
        }
    }
        :
```

7 注意･制限事項

「mini-FILE 独自仕様」の章をご確認下さい。

8 一覧

8.1 公開構造体

シンボル名	型名	内容
	FILERTC	**日時データ構造体**
date	DOSDATE	日付データ(以下参照)
time	DOSTIME	時刻データ(以下参照)

シンボル名	型名	内容
	DOSDATE	**日付データ構造体**
day	uint8_t	日(1-31)
month	uint8_t	月(1-12)
year	uint16_t	年(1980-2107)
dayweek	uint8_t	曜日(0-6)未使用

シンボル名	型名	内容
	DOSTIME	**時刻データ構造体**
hour	uint8_t	時(0-23)
minute	uint8_t	分(0-59)
second	uint8_t	秒(0-59)
hsecond	uint8_t	1/100 秒(0-99)未使用

※. 各タイムスタンプは mini-FILE では以下のタイミングで更新されています。

①新規作成した時のファイルのタイムスタンプ(f_open( )を「O_CREAT」指定した時)

ファイル作成時刻(time_create)
最新アクセス時刻(time_access)
書き込み時刻(time_wirte)

②更新後のクローズしたファイルのタイムスタンプ(f_write( )後の f_colse( )した時)

最新アクセス時刻(time_access)
書き込み時刻(time_wirte)

③新規作成した時のディレクトリのタイムスタンプ(f_mkdir()した時)

ファイル作成時刻(time_create)
最新アクセス時刻(time_access)
書き込み時刻(time_wirte)

8.2 エラーコード

シンボル名	エラー番号	内容
	0	正常終了
初期化エラー（致命的エラー）		
FILES_SYSTEM_ERROR		初期資源確保のエラー
ドライバエラー		
FILES_DRIVER_NODISK		メディアが存在しない
FILES_DRIVER_MOUNTERROR		マウントの失敗
FILES_DRIVER_INCOMPLETE		処理継続中(非同期型)
FILES_DRIVER_BUSY		別処理(API)が継続中(非同期型)
FILES_DISK_ACCESS_ERROR		ディスクアクセスエラー ※1
ファイル／フォルダ操作エラー		
FILES_DISK_NOT_SPACE		ディスクフル
FILES_DISK_ACCESS_DENIED		アクセス拒否
FILES_DISK_INVALID_FORMAT		ディスクのフォーマット異常
FILES_PARAMETER_ERROR		パラメータエラー
FILES_PATHNAME_ERROR		パス指定異常
FILES_DRIVELETTER_OVER		ドライブレター割り当て可能範囲オーバー
FILES_SEEK_FILEOVER		ファイルサイズを超えたシーク
FILES_USED_DIR		削除ディレクトリはファイルが存在する
FILES_KILL_ROOT		ルートディレクトリを削除しようとした
FILES_KILL_CURRENT		カレントディレクトリを削除しようとした
FILES_PATH_ALREADY		パス／ファイルが既に存在する
FILES_PATH_NOTFIND		パス／ファイルが存在しない
FILES_END_OF_FILE		ファイルの終端の検出(f_read()のみ)

※1．ドライバからのエラー通知された場合は、"FILES_DRIVER_NODISK"以外のエラーは、このエラーでユーザーアプリケーションに通知します。

8.3 ファイル属性

シンボル名	内容
属性（f_open の第 3 パラメータにて指定）	
O_RDONLY	ファイルを読み取り専用で開きます。
O_WRONLY	ファイルを書き込み専用で開きます。
O_RDWR	読み出しおよび書き込み両用にファイルを開きます。
O_APPEND	書き込み操作が行われるたびに、事前にファイルポインタをファイルの終端に移動します。
O_CREAT	書き込み用に新しいファイルを作成して開きます。
O_EXCL	指定されたファイルが存在する場合は、エラー値を返します。
O_SEQUENCE	連続書き込みモード指定(__SEQUENCE_WRITE__指定時のみ)
permission（f_open の第 4 パラメータにて指定）	
S_IREAD	ファイルを読み取り専用に設定します。
S_IWRITE	ファイルの読み取りおよび書き込みを許可します。

8.4 リムーバブル／非リムーバブル指定

“disk_mount”の第５パラメータにて指定

シンボル名	内容
DISK_UNREMOVABLE	非リムーバブル指定
DISK_REMOVABLE	リムーバブル指定（活栓挿抜対応）

8.5 ユーザーアプリケーションへの挿抜通知指定

“disk_mount”のコールバック関数の第３パラメータにて指定

シンボル名	内容
EVENT_DISCONNECT	抜去イベント通知指定
EVENT_CONNECT	挿入イベント通知指定

8.6 API

ユーザ公開 API 一覧

関数名	内容
初期化／終了関数	
ITF_FileSystemInit	ライブラリ初期化
ITF_FileSystemEnd	ライブラリ終了処理
disk_mount	ディスクマウント
disk_unmount	ディスクアンマウント
ディレクトリ操作関数	
f_mkdir	新規ディレクトリ作成
ファイル操作関数	
f_open	ファイルのオープン・クリエイト
f_close	オープンファイルクローズ
f_read	リードデータ
f_write	ライトデータ
f_flush	遅延データ強制書き込み
f_format	クイックフォーマット
ファイル操作関数	
f_asynchronous	非同期用ポーリング関数

ご注意

■お問い合わせ

インターフェイス株式会社

〒190-0022　東京都立川市錦町２－２－１１

E-mail　support@itf.co.jp