

# 組込み向けファイルシステム GR-FILE

## 第 1.31 版

2020年 3 月

対応 GR-FILE バージョン : 1.30

株式会社グレースシステム

**[注意事項]**

- すべての著作権は、株式会社グレープシステムにあります。
- 本ドキュメントの内容の一部または全部を無断で転載、複写、複製する事を禁じます。
- 本製品の仕様は予告なく変更される事があります。
- 本ドキュメントに記載されている会社名、製品名は各社の商標または登録商標です。

Copyright (C) 2003 - 2020 Grape Systems, Inc. All Rights Reserved

## はじめに

本書は、組込みシステムでファイルアクセスを実現するためのミドルウェア「**GR-FILE**」について記述します。特に、**GR-FILE** では、PC とのデータ交換が可能な FAT ファイルシステムをサポートします。

## 改訂履歴

Rev.	日付	改訂内容
1.00	2003 年 09 月	初版
1.00a	2003/12/29	①chdir/grp_fs_chdir にカレントディレクトリを設定しなかった場合の記述追加 ②grp_fs_get_mnt_by_name の使用形式の記述が grp_fs_get_mnt_by_dev となっていたミスを修正 ③FAT ファイルシステムで同時にオープン可能なファイル数を制限している内部グローバル変数を公開し、grp_fs_init の記述に変更方法を追記。また、同公開に伴い、再コンパイルにより変更可能なパラメータ一覧から FAT_MAX_OPEN を削除 ④fopen インタフェースの説明の記載漏れを追加
1.10	2004/8	① ファイル名称キャッシュ機能を追加 (GRP_FS_FNAME_CACHE、GRP_FS_FAT_CACHE_BY_GET_DIRENT オプション) ②FAT フォーマット機能/パーティションの設定変更機能追加 (SD カード固有のサンプルライブラリ関数も追加) ③デバイス直接制御インタフェース追加 ④RAM ディスク機能を追加 (GRP_FS_RAM_DISK オプション) ⑤POSIX opendir/closedir/readdir インタフェース追加 ⑥μITRON のサンプルフロー追加 ⑦grp_fs_lookup_dev をファイルシステム依存関数向けインタフェースから <b>GR-FILE</b> 固有のアプリケーションインタフェースに変更 ⑧マウント時のオプションとして GRP_FS_NO_UPD_ACCTIME、GRP_FS_NO_MNT_FLAG、GRP_FS_NO_CRT_ACCTIME を追加 ⑨キャッシュパラメータのデフォルト値 GRP_FS_FBLK_SHIFT、GRP_FS_DBLK_SHIFT、GRP_FS_DBLK_CNT を変更 ⑩構成定義定数 GRP_FS_MAX_DEV を削除 (変数化) ⑪ファイルシステム依存処理関数の create 処理を一部変更 ⑫ファイルシステム依存処理関数の match_comp 処理にファイル名称キャッシュの purge パラメータ追加 ⑬リリース CD の <b>GR-VOS</b> のソースディレクトリを <b>GR-FILE</b> 内のソースディレクトリから <b>GR-FILE</b> のソースディレクトリと同一レベルに変更したことに対応し、ソースディレクトリ構成の記述を変更

1. 11	2004/12	<p>①mount 時のフリークラスタ数カウント処理の高速化のためのパラメータの追加 Version 1.11 では、FAT のフリークラスタの検索処理を最適化し、さらに、mount 時のフリークラスタ数カウント処理に別の I/O バッファを使用し、高速化を行なっているため、その I/O バッファのサイズパラメータの記述を追加</p> <p>②GRP_FS_FAT_NO_DIR_SIZE_INFO オプションの追加 コンパイルオプションとして、GRP_FS_FAT_NO_DIR_SIZE_INFO を追加し、ディレクトリの場合、grp_fs_get_dirent でわざわざディレクトリのサイズ情報を計算せず、0 で返すことを可能化</p> <p>③grp_fs_exec_dev_io 関数のパラメータの追加変更 ①の変更に対応し、ファイルシステム依存関数向け <b>GR-FILE</b> 関数の 1 つ grp_fs_exec_dev_io にパラメータを 1 つ追加</p> <p>③ドライブ型マウントのドライブ名称記述ミスの修正 ドライバ型マウントのドライブ名称について、アルファベット一文字ではなく、任意の文字列が可能であるが、最後にコロンが必要であることの記述漏れを修正</p> <p>④FAT12/16 フォーマット時のデフォルトルートディレクトリエントリ数の変更 <b>GR-FILE</b> でフォーマットする場合の FAT12/FAT16 に対するデフォルトのルートディレクトリエントリ数を、性能上の問題から 256/512 にそれぞれ変更</p> <p>⑤FAT ファイルの最大オープン数設定の削除 クローズ後もオープンファイルキャッシュに残っている間、各 FAT ファイルシステム毎のファイルの最大オープン数を制限する変数 grp_fat_max_open_cnt 分しかオープンできない問題があったため、同変数、および、同変数の初期値の define FAT_MAX_OPEN を削除し、個々のファイルシステムではなく、<b>GR-FILE</b> 全体での最大オープンファイル数制限だけに変更。 また、同変更に対応し、fat_BPB_t の構造体の ptOpenFree フィールドを削除</p>
1. 11a	2005/1	<p>①サイズ 0 ファイルの扱い変更に伴う変更</p> <ul style="list-style-type: none"> <li>・ファイルシステム管理構造体 grp_fs_info に、ファイル ID による操作ロックのための uiFsBusyFid を追加し、関連するステータスビットを usStatus フィールドに追加</li> <li>・ファイルシステム依存関数向けの <b>GR-FILE</b> の共通関数に grp_fs_block_file_op_by_id、grp_fs_unblock_file_op_by_id、grp_fs_change_fid を追加</li> <li>・FAT 依存のオープン管理情報構造体 fat_open_info のメンバを追加/変更し、同管理情報テーブルを管理するための管理構造体 fat_open_info_ctl の記述を追加</li> </ul> <p>②readdir の記述に FAT 依存の記述を追加</p>
1. 11b	2005/2	<p>① 非 32 ビットシステム対応のためのタイプ変更</p> <ul style="list-style-type: none"> <li>・read、write、fread、fwrite、grp_fs_read、grp_fs_write、grp_fs_get_error、grp_mem_alloc、grp_fs_copyin、grp_fs_copyout、grp_fs_read_t、grp_fs_write_t、grp_fs_read_buf、grp_fs_write_buf のリターン値やパラメータのタイプを変更</li> <li>・非 32 ビットシステム向け移植手順の記述を追加</li> </ul>

1. 11. doc_ rev3	2006/11	<ul style="list-style-type: none"> <li>① ソースのリビジョンとドキュメントのリビジョン体系を区別するため、本ドキュメントのリビジョン名称形式変更 (a,b,c,... ⇒ .doc_rev1,2,3,...)</li> <li>② open/grp_fs_open の iMode の説明補足追加</li> <li>③ getcwd/grp_fs_getcwd の、カレントディレクトリが設定されていない場合の仕様の記述間違いを修正</li> <li>④ fopen の、"b" 指定時の仕様の記述間違いを修正</li> <li>⑤ ftell の "a","a+" モード時の動作記述を追加</li> <li>⑥ grp_fs_dir_t 構造体の説明、chmod、creat、mkdir、open、stat、grp_fs_chmod、grp_fs_creat、grp_fs_open、grp_fs_stat、pfnCreate 関数の記述において、FAT ファイルシステムの uiProtect 情報の記述が一部不正確ところを正しく修正</li> <li>⑦ grp_fs_mnt_info_t 構造体の説明のボリューム名関係のフィールド名間違いを修正</li> <li>⑧ grp_mem_free_to_vl_pool の関数説明のタイトルと関数名の不一致を修正</li> <li>⑨ grp_time_localtime、grp_time_mktime に標準時刻のトータル秒であることの記述を追加</li> <li>⑩ grp_fs_mount のエラー番号一覧の GRP_FS_ERR_NEED_CHECK の記載漏れを修正</li> <li>⑪ ファイルシステム依存関数に pfnSync 処理を追加し、GR-FILE 1.11.f において unmount 時以外でも FAT32 の残りクラスタのヒント情報の書き戻しをサポート</li> <li>⑫ 上記に関連し、grp_fs_sync に GRP_FS_SYNC_HINT オプションを追加</li> <li>⑬ 上記に関連し、fat_BPB_t に uiFreeHint フィールド追加</li> <li>⑭ 巻末のサポート連絡先を gr@support.grape.co.jp に変更</li> </ul>
1. 11. doc_ rev4	2008/1	<ul style="list-style-type: none"> <li>① grp_fat_format_param_t 構造体の説明に uiAlign、uiOption、uiFatSec、uiNotUsed、uiAdjust を追加</li> <li>② ファイル管理情報構造体名を修正(grp_file_t → grp_fs_file_t)</li> <li>③ ファイル管理情報構造体メンバ ucFsFBlkShift を削除</li> <li>④ grp_fs_ctl_t 構造体の説明に iWaitCnt を追加</li> <li>⑤ 4 章のインタフェースの説明にフック関数インタフェースを追加</li> <li>⑥ getcwd/grp_fs_getcwd の、カレントディレクトリが設定されていない場合の仕様の記述間違いを修正</li> <li>⑦ grp_fat_format で使用する、フォーマットパラメータのボリュームラベルで使用できる文字、文字長の説明を追加</li> <li>⑧ getcwd/grp_fs_getcwd/readdir/grp_fs_get_dirent の、実際のコンポーネント長が、<b>GR-FILE</b> の config パラメータ GRP_FS_MAX_COMP より長い場合の説明を追加</li> <li>⑨ ファイルシステム依存関数向けの <b>GR-FILE</b> フック関数を追加</li> <li>⑩ 4 章のインタフェース説明に合わせ、デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数の関数説明をファイルシステム依存関数向けの <b>GR-FILE</b> フック関数の後ろへ移動</li> <li>⑪ サンプルフローに <b>GR-FILE</b> フック関数の記述を追加</li> <li>⑫ 強制アンマウント時のファイルのクローズ操作の必要性、強制アンマウントされた場合に返る GRP_FS_ERR_SHOULD_CLOSE のリターン値の記述を grp_fs_unmount、grp_fs_get_dirent、grp_fs_read、grp_fs_write、grp_fs_truncate、ftruncate、read、write の説明に追加</li> </ul>

1. 20	2008/7	<ul style="list-style-type: none"> <li>①表紙に対応する <b>GR-FILE</b> のバージョンを表記</li> <li>②コンパイルオプションに、<b>GR-FILE</b> 用多国語オプション(別売)を有効にする <b>GRP_FS_MULTI_LANGUAGE</b> の記述を追加</li> <li>③コンパイルオプションが、<b>grp_fs_sysdef.h</b> に定義されたことを追加</li> <li>④環境依存のコンパイルオプション「<b>M32R</b>」、「<b>GRP_VOS</b>」、「<b>ITRON</b>」、「<b>THREADX</b>」、「<b>T_KERNEL</b>」を削除</li> <li>⑤ディレクトリ構成/ファイルの追加により、全章に渡ってディレクトリ/ファイル名を変更</li> <li>⑥<b>GR-FILE</b> の使用するメモリープールは、環境/使用方法により、CPU の非キャッシュ領域へ確保する必要がある旨、<b>grp_mem_vl_init/grp_mem_init_vl_pool</b> に追記</li> <li>⑦RAM ディスク機能の説明で、セクタサイズが 512 バイト固定の説明になっていたところを、設定されたサイズでの説明に変更</li> <li>⑧<b>fat_BPB_t</b> 構造体メンバ「<b>uiDBlkStart</b>」の説明が不十分だった所を、説明を追記</li> <li>⑨変更可能パラメータ「<b>GRP_FS_MAX_FSTYPE</b>」は使用していない為、削除</li> <li>⑩<b>lseek</b>、<b>fseek</b>、<b>grp_fs_lseek</b> の説明で、ファイルサイズを超えた場合の説明が誤っていたのを修正。正しくは <b>write</b> 時のみエラーが発生</li> <li>⑪SD カードのフォーマットサンプルコードは、ご要望のあった場合のみ提供する旨追記</li> <li>⑫ボリュームラベルに使用できる文字の説明に、空白が使えることを追記</li> <li>⑬何れかのタスクが <b>GR-FILE</b> 内で実行中の場合、アンマウントが失敗する事を、<b>grp_fs_unmount</b> 関数に追記</li> <li>⑭ダイレクト I/O を使用する際に、機器によっては奇数バイトアクセスができないことを追記</li> <li>⑮誤字、誤記修正</li> </ul>
1. 21	2010/10	<ul style="list-style-type: none"> <li>①コンパイルオプションに、<b>GR-FILE</b> で使用する ROM 量の削減レベルを設定する <b>GRP_FS_MINIMIZE_LEVEL</b> の記述を追加</li> <li>②誤字、誤記修正</li> </ul>
1. 22	2010/11	<ul style="list-style-type: none"> <li>①マウント情報を返す構造体「<b>grp_fs_mnt_info_t</b>」の、状態を表すメンバ「<b>uiStatus</b>」の説明が不十分だった所を、説明を追記</li> <li>②ロングファイル名でファイルを作成した際に生成するショート名の生成方法を選択するコンパイルオプション <b>GRP_FS_FAT_TRY_NO_NUM_SHORT</b> を追加</li> <li>③SDHC カードのフォーマット説明を追加</li> <li>④フォーマットパラメータの <b>uiSafetyGap</b> 値を 16 から 10 へ変更</li> <li>⑤<b>grp_fs_get_current_time</b> でエラー発生時に、<b>piTime</b> に 0 を返すように追記</li> <li>⑥<b>grp_stdio_io_stdin</b>、<b>grp_stdio_io_stdout</b> に <b>NULL</b> を設定した場合に不正呼び出しが発生する可能性があることを追記</li> <li>⑦「3.2 標準+固有アプリケーションインタフェース」に各インタフェースの比較表を追記</li> <li>⑧表目次、図目次を追記</li> <li>⑨誤字、誤記、判り難い説明を修正</li> </ul>

1. 23	2011/06	<p>①アクセス中以外のデバイスのアンマウントを可能にする機能を追加</p> <p>②単一ディレクトリに似たようなロングファイル名が多数ある場合に、ショートファイル名の生成速度を改善する機能を追加</p> <p>③ファイルの作成/書き込み/名称変更時に ARCHIVE 属性をセットする機能を追加</p> <p>④上記①～③の機能を有効/無効にするコンパイルスイッチ「GRP_FS_ASYNC_UNMOUNT」、「GRP_FS_FAST_MAKE_SNAME」、「GRP_FS_UPDATE_ARCHIVE」を追加</p> <p>⑤②の機能に切り替える閾値定義「GRP_FS_MAKE_SNAME_THRESHOLD」を追加</p> <p>⑥grp_fs_mount()の説明にライトプロテクト状態の場合の説明を追加</p> <p>⑦grp_fs_ioctl_dev()、grp_fs_dev_ioctl_t の定義にメディアのライトプロテクト状態取得定義を追加定義</p>
1. 24	2012/06	<p>①ファイルハンドル構造体(grp_fs_fhdl_t)の説明に GRP_FS_OPEN_PARENT、GRP_FS_OPEN_DIRECT_IO を追加</p> <p>②grp_fs_mount()のリターン値に GRP_FS_ERR_EXIST を追加</p>
1. 25	2015/03	<p>①RAM ディスク機能の説明でファイル名が誤っていたのを修正</p> <p>②ソースファイルの構成に mdep_vos2.xx を追加</p> <p>③ソースファイルの構成にサンプルを追加</p> <p>④全体的に句読点、カンマを修正</p>
1. 30	2016/04	<p>①コンパイルスイッチ「GRP_FS_ENABLE_OVER_2G」を追加</p> <p>②符号なしオフセット型として「grp_uioffset_t」を追加</p> <p>③ファイルサイズを保持するメンバ変数の型を符号付 32 ビットから符号無し 32 ビットに変更し名称を「iSize」から「uiSize」に変更</p> <p>④オフセットを保持するメンバ変数の型を符号付 32 ビットから符号無し 32 ビットに変更し名称を「iOffset」から「uiOffset」に変更</p> <p>⑤4G-1 バイトに対応する以下の Seek 系 API を追加 grp_fs_lseek4G()、lseek4G()、fseek4G()、ftell4G() 4G-1 バイト対応 API と従来の API を混在使用した場合の注意点を各 API に追記</p> <p>⑥readdir()のパラメータ説明に「GRP_FS_ENABLE_OVER_2G」を定義した場合の変更点を追記</p>
1. 31	2020/03	<p>①OS 抽象化インタフェースに grp_fs_to_upper、grp_fs_cmp_fname を追加</p> <p>②chdir、grp_fs_chdir に強制アンマウントを行った際の説明を追記</p> <p>③grp_fs_unmount に強制アンマウントを行った際にカレントディレクトリを無効化する必要があることを追記</p> <p>④コンパイルオプションに GRP_USB の説明を追記</p>

## 目次

1. 概要 .....	1
1.1 <b>GR-FILE</b> の位置づけ .....	1
1.2 機能・特徴概要.....	4
2. 前提条件・制限 .....	6
3. 機能詳細 .....	7
3.1 FAT ファイルシステムサポート .....	7
3.2 標準+固有アプリケーションインタフェース .....	8
3.2.1 POSIX 互換の I/O インタフェース.....	9
3.2.2 C 言語標準 I/O インタフェース (オプション) .....	10
3.2.3 <b>GR-FILE</b> 固有インタフェース .....	12
3.2.4 デバイス直接制御インタフェース .....	13
3.3 マルチタスク I/O.....	14
3.3.1 並列ファイル I/O.....	14
3.3.2 OS 非依存のファイル情報管理.....	16
3.4 日本語 (多国語) ファイル名サポート .....	17
3.4.1 ショートファイル名 .....	17
3.4.2 ロングファイル名 .....	18
3.4.3 ファイル名の長さ制限 .....	18
3.5 ドライブ型+階層化 mount .....	19
3.5.1 ドライブ型 mount.....	19
3.5.2 階層化 mount.....	20
3.5.3 パーティション分割されたメディアアクセスのためのデバイスネーミング規則 .....	21
3.6 ファイルデータとファイル管理情報のキャッシング .....	24
3.6.1 ファイルデータキャッシュ .....	25
3.6.2 ファイル管理ブロックキャッシュ .....	26
3.6.3 オープンファイルキャッシュ (オープン中のファイル管理情報) .....	26
3.6.4 フリーブロックキャッシュ .....	26
3.6.5 ファイル名称キャッシュ .....	27
3.7 キャッシュの write 制御 .....	28
3.7.1 write through 方式 .....	28
3.7.2 each close 方式.....	29
3.7.3 last close 方式 .....	30
3.7.4 unmount 方式 .....	31
3.8 アプリケーションバッファとメディア間の直接/連続ブロック I/O (ダイレクト I/O 機能) .....	32
3.9 ファイルシステム依存部の分離 .....	34
3.10 OS 依存処理部の分離 .....	35
3.11 メディアの挿抜対応.....	36
3.11.1 メディアの挿抜処理の概要 .....	36



3.1.1.2	メディアの正常挿抜対応 .....	39
3.1.1.3	メディアの異常挿抜対応 .....	39
3.1.1.4	不当メディア/ファイルシステムへの対応 .....	40
3.1.2	メディアのフォーマットとパーティションの設定 .....	41
3.1.2.1	フォーマット機能 .....	41
3.1.2.2	パーティションの設定/変更 .....	46
3.1.3	RAM ディスク機能 .....	49
3.1.4	各種パラメータの設定・変更、および、コンパイルオプション .....	51
3.1.4.1	実行時に変更可能なパラメータ .....	51
3.1.4.2	再コンパイルにより変更可能なパラメータ .....	56
3.1.4.3	コンパイルオプション .....	58
4.	インタフェース .....	60
4.1	エラー番号 .....	61
4.2	データタイプの定義 .....	62
4.2.1	基本データタイプの定義 .....	62
4.2.2	アプリケーションインタフェース関連の構造体 .....	63
4.2.3	ファイルシステム非依存部の管理構造体 .....	67
4.2.4	FAT ファイルシステム依存部の管理構造体 .....	73
4.3	POSIX 互換アプリケーションインタフェース .....	76
4.3.1	chdir .....	78
4.3.2	chmod .....	79
4.3.3	close .....	80
4.3.4	closedir .....	81
4.3.5	creat .....	82
4.3.6	ftruncate .....	84
4.3.7	getcwd .....	85
4.3.8	lseek .....	86
4.3.9	lseek4G .....	87
4.3.10	mkdir .....	89
4.3.11	open .....	91
4.3.12	opendir .....	93
4.3.13	read .....	94
4.3.14	readdir .....	96
4.3.15	rename .....	98
4.3.16	rmdir .....	99
4.3.17	stat .....	100
4.3.18	stat4G .....	102
4.3.19	sync .....	104
4.3.20	unlink .....	105
4.3.21	utimes .....	106

4.3.2 2	write .....	107
4.4	C 言語標準アプリケーション I/O インタフェース .....	108
4.4.1	clearerr .....	110
4.4.2	fclose .....	111
4.4.3	feof.....	112
4.4.4	ferror .....	113
4.4.5	fflush .....	114
4.4.6	fgetc / getc .....	115
4.4.7	fgets.....	116
4.4.8	fileno .....	117
4.4.9	fopen.....	118
4.4.1 0	fprintf / vfprintf .....	119
4.4.1 1	fputc / putc .....	120
4.4.1 2	fread .....	121
4.4.1 3	fseek .....	122
4.4.1 4	fseek4G.....	123
4.4.1 5	ftell .....	124
4.4.1 6	ftell4G.....	125
4.4.1 7	fwrite .....	126
4.4.1 8	getchar .....	127
4.4.1 9	putchar.....	128
4.4.2 0	rewind .....	129
4.4.2 1	ungetc.....	130
4.5	<b>GR-FILE</b> 固有アプリケーション I/O インタフェース .....	131
4.5.1	grp_fat_find_type .....	133
4.5.2	grp_fat_format .....	135
4.5.3	grp_fat_format_sd (参考ライブラリ) .....	138
4.5.4	grp_fs_chdir.....	140
4.5.5	grp_fs_check_fs_dev.....	141
4.5.6	grp_fs_check_volume .....	143
4.5.7	grp_fs_chmod.....	145
4.5.8	grp_fs_close .....	146
4.5.9	grp_fs_closedir.....	147
4.5.1 0	grp_fs_create.....	148
4.5.1 1	grp_fs_err.....	150
4.5.1 2	grp_fs_get_attr .....	151
4.5.1 3	grp_fs_get_cwd .....	152
4.5.1 4	grp_fs_get_dirent.....	153
4.5.1 5	grp_fs_get_error .....	157
4.5.1 6	grp_fs_get_mnt.....	159

4.5.1 7	grp_fs_get_mnt_by_dev .....	160
4.5.1 8	grp_fs_get_mnt_by_name .....	161
4.5.1 9	grp_fs_init .....	162
4.5.2 0	grp_fs_invalidate_fs_dev .....	163
4.5.2 1	grp_fs_lookup_dev .....	164
4.5.2 2	grp_fs_lseek .....	165
4.5.2 3	grp_fs_lseek4G .....	166
4.5.2 4	grp_fs_mount .....	168
4.5.2 5	grp_fs_open .....	172
4.5.2 6	grp_fs_opendir .....	174
4.5.2 7	grp_fs_read .....	175
4.5.2 8	grp_fs_readdir .....	176
4.5.2 9	grp_fs_read_part .....	178
4.5.3 0	grp_fs_rename .....	179
4.5.3 1	grp_fs_set_attr .....	180
4.5.3 2	grp_fs_stat .....	181
4.5.3 3	grp_fs_sync .....	183
4.5.3 4	grp_fs_task_free_all_env .....	184
4.5.3 5	grp_fs_task_free_env .....	185
4.5.3 6	grp_fs_task_free_env_by_id .....	186
4.5.3 7	grp_fs_truncate .....	187
4.5.3 8	grp_fs_unlink .....	188
4.5.3 9	grp_fs_unmount .....	189
4.5.4 0	grp_fs_utimes .....	191
4.5.4 1	grp_fs_write .....	192
4.5.4 2	grp_fs_write_part .....	194
4.6	デバイス直接制御アプリケーションインタフェース .....	196
4.6.1	grp_fs_open_dev .....	197
4.6.2	grp_fs_close_dev .....	199
4.6.3	grp_fs_read_dev .....	200
4.6.4	grp_fs_write_dev .....	201
4.6.5	grp_fs_ioctl_dev .....	202
4.7	ファイルシステム抽象化インタフェース .....	204
4.7.1	grp_fs_open_root_t *pfnOpenRoot .....	205
4.7.2	grp_fs_mount_t *pfnMount .....	206
4.7.3	grp_fs_umount_t *pfnUmount .....	207
4.7.4	grp_fs_open_t *pfnOpen .....	208
4.7.5	grp_fs_close_t *pfnClose .....	210
4.7.6	grp_fs_read_t *pfnRead .....	211
4.7.7	grp_fs_write_t *pfnWrite .....	213

4.7.8	grp_fs_create_t *pfnCreate.....	215
4.7.9	grp_fs_unlink_t *pfnUnlink .....	217
4.7.10	grp_fs_rename_t *pfnRename .....	219
4.7.11	grp_fs_get_attr_t *pfnGetAttr .....	220
4.7.12	grp_fs_set_attr_t *pfnSetAttr.....	221
4.7.13	grp_fs_truncate_t *pfnTruncate .....	222
4.7.14	grp_fs_get_dirent_t *pfnGetDirEnt .....	223
4.7.15	grp_fs_match_comp_t *pfnMatchComp .....	225
4.7.16	grp_fs_check_volume_t *pfnCheckVolume .....	226
4.7.17	grp_fs_sync_t *pfnSync .....	227
4.8	OS 抽象化インタフェース.....	228
4.8.1	grp_fs_create_sem.....	230
4.8.2	grp_fs_get_sem.....	231
4.8.3	grp_fs_release_sem .....	232
4.8.4	grp_fs_get_taskid .....	233
4.8.5	grp_fs_copyin.....	234
4.8.6	grp_fs_copyout.....	235
4.8.7	grp_fs_get_str.....	236
4.8.8	grp_fs_get_current_time.....	237
4.8.9	grp_fs_printf.....	238
4.8.10	grp_fs_char_cnt .....	239
4.8.11	grp_fs_char_to_unicode.....	240
4.8.12	grp_fs_unicode_to_char.....	241
4.8.13	grp_fs_cmp_fname.....	242
4.8.14	grp_fs_to_upper .....	243
4.8.15	grp_mem_alloc.....	244
4.8.16	grp_mem_free .....	245
4.8.17	grp_fs_inform_io_err .....	246
4.8.18	grp_stdio_io_stdin .....	248
4.8.19	grp_stdio_io_stdout .....	249
4.9	デバイスドライバインタフェース .....	250
4.9.1	grp_fs_dev_open_t *pfnOpen.....	251
4.9.2	grp_fs_dev_close_t *pfnClose .....	253
4.9.3	grp_fs_dev_read_t *pfnRead .....	254
4.9.4	grp_fs_dev_write_t *pfnWrite .....	255
4.9.5	grp_fs_dev_ioctl_t *pfnIoctl.....	256
4.10	ファイルシステム依存関数向け <b>GR-FILE</b> フック関数.....	258
4.10.1	fat_interrupt_lookup .....	259
4.11	ファイルシステム依存関数向けの <b>GR-FILE</b> 関数.....	261
4.11.1	grp_fs_block_buf_mod/grp_fs_unblock_buf_mod .....	262

4.1.1.2	grp_fs_block_file_op/grp_fs_unblock_file_op .....	263
4.1.1.3	grp_fs_block_file_op_by_id/grp_fs_unblock_file_op_by_id .....	264
4.1.1.4	grp_fs_block_fs_mod/grp_fs_unblock_fs_mod .....	265
4.1.1.5	grp_fs_buf_fill_end .....	266
4.1.1.6	grp_fs_change_fid .....	267
4.1.1.7	grp_fs_check_dev_busy .....	268
4.1.1.8	grp_fs_check_io_status .....	269
4.1.1.9	grp_fs_check_mnt_dev .....	271
4.1.1.10	grp_fs_close_file .....	272
4.1.1.11	grp_fs_exec_dev_io .....	273
4.1.1.12	grp_fs_file_open_common .....	274
4.1.1.13	grp_fs_get_mount_root_attr .....	276
4.1.1.14	grp_fs_get_path_comp .....	277
4.1.1.15	grp_fs_lookup_buf .....	278
4.1.1.16	grp_fs_lookup_file_ctl .....	280
4.1.1.17	grp_fs_lookup_fname_cache .....	281
4.1.1.18	grp_fs_make_sname_another_method .....	282
4.1.1.19	grp_fs_purge_fname_cache_by_dev .....	283
4.1.1.20	grp_fs_read_buf .....	284
4.1.1.21	grp_fs_set_access_time .....	285
4.1.1.22	grp_fs_set_fname_cache .....	286
4.1.1.23	grp_fs_unref_buf .....	287
4.1.1.24	grp_fs_wait_io .....	288
4.1.1.25	grp_fs_write_buf .....	289
4.1.2	デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数 .....	290
4.1.2.1	grp_char_sjis_cnt .....	291
4.1.2.2	grp_char_sjis_to_unicode .....	292
4.1.2.3	grp_char_unicode_to_sjis .....	293
4.1.2.4	grp_fs_get_part .....	294
4.1.2.5	grp_mem_vl_init / grp_mem_init_vl_pool .....	296
4.1.2.6	grp_mem_vl_add / grp_mem_add_vl_pool .....	297
4.1.2.7	grp_mem_vl_alloc / grp_mem_alloc_from_vl_pool .....	298
4.1.2.8	grp_mem_vl_free / grp_mem_free_to_vl_pool .....	299
4.1.2.9	grp_time_localtime .....	300
4.1.2.10	grp_time_mktime .....	301
4.1.2.11	grp_time_set_base_year .....	302
4.1.2.12	grp_time_set_time_diff .....	303
4.1.2.13	grp_time_get_config .....	304
5.	サンプルフロー .....	305
6.	ソースファイルの構成と <b>GR-FILE</b> ライブラリの構築・使用方法 .....	309

6.1	ソースファイルの構成.....	309
6.2	<b>GR-FILE</b> の構築・使用方法 .....	312

## 表目次

表 1-1	<b>GR-FILE</b> およびその周りのコンポーネントの概要 .....	2
表 1-2	<b>GR-FILE</b> のインタフェース概要.....	3
表 1-3	<b>GR-FILE</b> の機能・特徴一覧.....	4
表 2-1	<b>GR-FILE</b> の前提条件・制限.....	6
表 3-1	インタフェース比較.....	8
表 3-2	オープンファイル/カレントディレクトリのクローズ/無効化処理.....	16
表 3-3	ファイル名の長さの制限.....	18
表 3-4	デバイス番号操作のためのマクロ/define.....	23
表 3-5	メディアの挿抜状態.....	37
表 3-6	メディアの各挿抜処理の概要 .....	38
表 3-7	フォーマット情報パラメータの詳細 .....	42
表 3-8	フォーマットパラメータのデフォルト値/自動計算で使用するグローバル変数 .....	43
表 3-9	自動計算時のメディア/パーティションサイズとクラスタサイズ/FAT タイプとの対応関係	43
表 3-10	メディア情報パラメータの詳細 .....	44
表 3-11	パーティション情報 <code>grp_fs_dk_part_t</code> の詳細.....	47
表 3-12	実行時に変更可能な <code>grp_fs_param</code> の詳細 .....	51
表 3-13	FAT 固有のパラメータ変数.....	52
表 3-14	<code>grp_fat_cluster_limit_tbl[]</code> の詳細.....	53
表 3-15	FAT 規格で適用可能なトータルサイズとクラスタサイズ/FAT タイプの関係概要 .....	54
表 3-16	<code>grp_fat_format_cfg</code> の詳細 .....	54
表 3-17	<code>grp_fat_default[]</code> の詳細.....	55
表 3-18	<code>grp_fs_default_part_type</code> の詳細 .....	55
表 3-19	再コンパイルにより変更可能なパラメータ（ファイルシステム非依存部） .....	56
表 3-20	再コンパイルにより変更可能なパラメータ（FAT ファイルシステム依存部） .....	56
表 3-21	再コンパイルにより変更可能なパラメータ（C 言語標準 I/O インタフェース） .....	56
表 3-22	再コンパイルで変更可能なフォーマット/パーティション設定関連のパラメータ .....	57
表 3-23	再コンパイルで変更可能なショートファイル名生成関連のパラメータ .....	57
表 3-24	コンパイルオプション.....	58
表 4-1	エラー番号一覧.....	61
表 4-2	基本データタイプ定義一覧.....	62
表 4-3	POSIX 互換アプリケーションインタフェース一覧.....	76
表 4-4	POSIX 互換アプリケーション向け 4G 対応アプリケーションインタフェース一覧.....	77
表 4-5	C 言語標準 I/O インタフェース一覧.....	108
表 4-6	C 言語標準アプリケーション向け 4G 対応 I/O インタフェース一覧 .....	108
表 4-7	<b>GR-FILE</b> 固有 I/O インタフェース一覧.....	131
表 4-8	デバイス直接制御アプリケーションインタフェース一覧.....	196
表 4-9	ファイルシステムテーブルエントリの構造体 <code>grp_fs_type_tbl_t</code> の内容 .....	204
表 4-10	ファイルシステム依存処理関数テーブルの構造体 <code>grp_fs_op_t</code> の内容 .....	204
表 4-11	OS 抽象化インタフェース一覧 .....	228

表 4-1 2	<b>GR-FILE</b> で前提とする標準ライブラリ関数.....	228
表 4-1 3	デバイスドライバテーブルエントリの構造体 <code>grp_fs_dev_tbl_t</code> の内容.....	250
表 4-1 4	デバイスドライバの I/O 関数テーブル構造体 <code>grp_fs_dev_op_t</code> 内容 .....	250
表 4-1 5	ファイルシステム依存関数向けの <b>GR-FILE</b> フック関数.....	258
表 4-1 6	ファイルシステム依存関数向けの <b>GR-FILE</b> 関数.....	261
表 4-1 7	OS/プラットフォーム依存関数向けサポートライブラリ関数一覧 .....	290
表 6-1	ソースファイルの構成 .....	309

## 図目次

図 1-1	<b>GR-FILE</b> の位置づけ .....	1
図 3-1	POSIX 互換インタフェース名称変換.....	9
図 3-2	C 言語標準の I/O インタフェース名称変換.....	10
図 3-3	マルチタスクによる並列 I/O .....	14
図 3-4	キャッシュレベルの複数タスク向けの処理機能（領域管理情報） .....	15
図 3-5	ドライブ型 mount の例 .....	19
図 3-6	階層化 mount の例.....	20
図 3-7	デバイス/ドライブ/パーティションの対応関係.....	21
図 3-8	<b>GR-FILE</b> のキャッシング機能 .....	24
図 3-9	write through 方式 .....	28
図 3-1 0	each close 方式 .....	29
図 3-1 1	last close 方式 .....	30
図 3-1 2	unmount 方式 .....	31
図 3-1 3	ダイレクト I/O 機能の read 処理のイメージ .....	32
図 3-1 4	ダイレクト I/O 機能の write 処理のイメージ.....	33
図 3-1 5	ファイル依存処理部の分離 .....	34
図 3-1 6	OS 依存処理部の分離.....	35
図 3-1 7	メディアの挿抜処理方法.....	36
図 3-1 8	メディアの挿抜処理の流れ .....	37
図 5-1	<b>GR-FILE</b> を用いたシステムの概略フロー例（ $\mu$ ITRON OS の場合） .....	305
図 5-2	<b>GR-FILE</b> を用いたシステムの概略フロー例（OS 抽象化インタフェース <b>GR-VOS</b> の場合） .....	306
図 5-3	メディア挿抜処理の概略フローの一例.....	307



## 1. 概要

**GR-FILE** は、組込みシステムでファイルアクセスを実現するためのミドルウェアです。本章では、まず、**GR-FILE** の位置づけ、および、機能・特徴概要について説明します。

### 1.1 GR-FILE の位置づけ

**GR-FILE** は、ディスクやメモ리카ード上に構築されたファイルシステム内のファイルをアクセスするためのミドルウェアです。**GR-FILE** は、図 1-1 に示しますように、ディスクやメモ리카ードを物理ブロック単位で I/O するデバイスドライバと、アプリケーションプログラムの間に位置します。

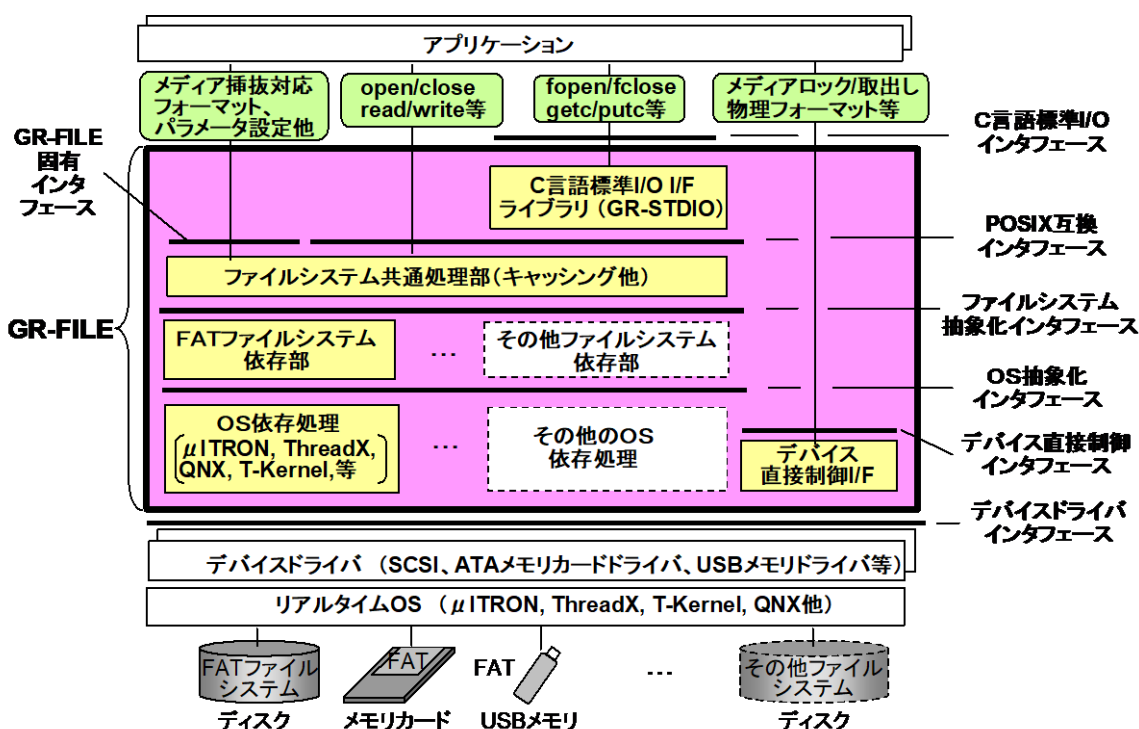


図 1-1 GR-FILE の位置づけ

**GR-FILE** は、ファイルシステム依存部を分離し、様々なファイルシステムをサポートできるようになっています。本バージョンの **GR-FILE** は、ファイルシステムとして、PC 等で利用されている FAT ファイルシステムをサポートしています。また、OS 依存部分を分離し、様々なリアルタイム OS に簡単に対応可能な形になっています。μITRON 準拠の OS に対応した依存コードを提供可能です。さらに、OS レス環境にも適用可能です。

アプリケーションインタフェースとしては、POSIX 準拠の open/close/read/write 等を使ったインタフェースを提供しますと共に、fopen/fclose/getc/putc 等を使った C 言語の標準 I/O インタフェースの利用が可能です。また、メディアの挿抜処理対応、フォーマット、キャッシングサイズ等の各種パラメータ設定等の **GR-FILE** 固有のインタフェース、メディアの取出し/ロック/アンロック、物理フォーマット機能等のデバイス固有で提供された機能を利用するためのデバイス直接制御インタフェースも提供します。

**GR-FILE** は、100% ANSI C 言語で記述されており、組込みアプリケーションとリンクして利用頂くことができます。

表 1-1 に **GR-FILE** とその周りの各コンポーネントの概要を示します。また、表 1-2 に **GR-FILE** が提供または利用するインタフェースの概要を示します。

表 1-1 GR-FILE およびその周りのコンポーネントの概要

#	コンポーネント	機能概要・要件
1	アプリケーション	<ul style="list-style-type: none"> <li>組込みシステムの各種アプリケーションプログラム</li> <li>アプリケーションプログラムは <b>GR-FILE</b> が提供する C 言語標準 I/O インタフェース、POSIX 互換インタフェース、<b>GR-FILE</b> 固有インタフェース、デバイス直接制御インタフェースを使い、ファイル I/O を実現</li> <li>特別なシステムアプリケーションにおいて、メディアの挿抜処理を実現。本挿抜処理では、<b>GR-FILE</b> が提供するインタフェースを使用し、デバイスドライバからのメディアの挿抜通知等を受け、ファイルシステムの自動 mount/unmount 処理等を実行</li> </ul>
2	C 言語標準 I/O インタフェースライブラリ	<ul style="list-style-type: none"> <li>fopen/fclose/getc/putc 等の C 言語の標準 I/O インタフェースをアプリケーションに対して提供</li> <li>本ライブラリは、<b>GR-FILE</b> のオプションとして提供</li> </ul>
3	ファイルシステム共通処理部	<ul style="list-style-type: none"> <li>ファイルシステム非依存のアプリケーションインタフェースを提供               <ul style="list-style-type: none"> <li>①POSIX 互換インタフェース(open/close/read/write 等)</li> <li>②<b>GR-FILE</b> 固有インタフェース(メディア挿抜対応処理、パラメータ設定等)</li> </ul> </li> <li>ファイルデータやファイル管理情報のキャッシング処理機能を提供</li> <li><b>GR-FILE</b> で規定したファイルシステム抽象化インタフェースを用い、ファイルシステム依存処理を実行(ファイルシステムテーブルに設定されたインタフェース関数を実行)</li> </ul>
4	ファイルシステム依存処理部 (FAT ファイルシステム依存部等)	<ul style="list-style-type: none"> <li>ファイルシステムに依存した処理を実行</li> <li>各ファイルシステム依存部は、<b>GR-FILE</b> で規定したファイルシステム抽象化インタフェースを提供し、同関数をファイルシステムテーブルに登録することで、ファイルシステム共通処理部からコールされる</li> <li>本バージョンでは、FAT ファイルシステムをサポート</li> <li>その他のファイルシステムを利用する場合は、<b>GR-FILE</b> で規定したファイルシステム抽象化インタフェースを提供する同ファイルシステム依存処理部の作成が必要</li> </ul>
5	OS 依存処理部	<ul style="list-style-type: none"> <li>セマフォア、メモリ割当て/解放、現在時刻取得、タスク ID の取得、タスク固有空間とのデータ授受等の OS/プラットフォーム依存の処理を実行</li> <li>本バージョンでは、<math>\mu</math>ITRON、VOS の OS 依存部をサポート</li> <li>その他の OS (含む OS レス) では、<b>GR-FILE</b> で規定した OS 抽象化インタフェースを提供する OS 依存処理部の作成が必要</li> <li>OS レス環境等のために簡易メモリ管理ライブラリを <b>GR-FILE</b> で提供</li> </ul>
6	デバイスドライバ	<ul style="list-style-type: none"> <li><b>GR-FILE</b> で規定したデバイスドライバインタフェース(open/close/物理ブロック単位の read/write)に従い、ディスクやメモ리카ード等のメディアへの物理ブロック単位での I/O 機能を <b>GR-FILE</b> に対して提供</li> <li>メディアの挿抜処理を行うシステムアプリケーションに対し、メディアの挿抜等のイベント通知を実行</li> <li>オプションで、メディアの取出し/ロック、物理フォーマット等、デバイス固有の制御機能を提供</li> <li>本バージョンでは、RAM 上の仮想ディスクを使った <b>GR-FILE</b> とのインタフェース関数のサンプルを提供</li> </ul>
7	リアルタイム OS	<ul style="list-style-type: none"> <li>セマフォア、メモリ割当て/解放機能、現在時刻取得、タスク ID 取得等の機能を <b>GR-FILE</b> に提供</li> <li>可変長のメモリ割当て/解放機能がない場合は、<b>GR-FILE</b> が提供する簡易メモリ管理ライブラリを利用可能</li> <li>OS レスまたはシングルタスク環境では、セマフォア、タスク ID 取得機能はダミー処理で可</li> </ul>

注) # 2、# 3、# 4、# 5 が **GR-FILE** で提供するコンポーネントです。その他は、**GR-FILE** とインタフェースを持つコンポーネントに対して、**GR-FILE** が前提する機能の概要を示しています。

表 1-2 GR-FILE のインタフェース概要

#	インタフェース	形態	概要
1	POSIX 互換 インタフェース	提供	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> からアプリケーションに対するインタフェース</li> <li>・ open/close/read/write 等 LINUX/Windows で提供されている POSIX 互換 I/O インタフェース相当</li> <li>・ 実インタフェースは、"grp_fs_XXX"、"GRP_FS_XXX"等の名称を持ち、"include/grp_fs_conv.h"をインクルードすることで、POSIX 互換のインタフェース名でアクセス可能</li> <li>・ コンパイルオプション GRP_FS_MINIMIZE_LEVEL を 1 または 2 に設定した場合、使用不可</li> </ul>
2	C 言語標準 I/O インタフェース	提供	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> からアプリケーションに対するインタフェース</li> <li>・ <b>GR-FILE</b> のオプションライブラリ GR-STDIO により提供</li> <li>・ C 言語標準 I/O の fopen/fclose/getc/putc 等のインタフェースを提供</li> <li>・ 実インタフェースは、"grp_stdio_XXX" 等の名称を持ち、&lt;stdio.h&gt;の代わりに"include/grp_stdio.h"をインクルードすることで、C 言語標準 I/O 互換のインタフェース名でアクセス可能</li> <li>・ コンパイルオプション GRP_FS_MINIMIZE_LEVEL を 1 または 2 に設定した場合、使用不可</li> </ul>
3	<b>GR-FILE</b> 固有 インタフェース	提供	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> からアプリケーションに対するインタフェース</li> <li>・ <b>GR-FILE</b> のメディアの挿抜対応処理インタフェースや、パラメータ設定等のインタフェースを提供</li> <li>・ コンパイルオプション GRP_FS_MINIMIZE_LEVEL を 1 または 2 に設定した場合、特殊な関数は使用不可（使用可否については表 4-7 参照）</li> </ul>
4	デバイス直接制御 インタフェース	提供	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> からアプリケーションに対するインタフェース</li> <li>・ メディアの取出し/ロック、物理フォーマット等、デバイス固有で提供された機能等をアプリケーションから利用するインタフェースを提供</li> </ul>
5	ファイルシステム 抽象化インタフェース	内部 I/F	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> 内のファイルシステム共通処理部とファイルシステム依存部間の内部インタフェース</li> <li>・ 同インタフェースに従い、新たなファイルシステムに対応したファイルシステム依存処理関数をファイルシステムテーブルに登録することで、FAT 以外のファイルシステムもアクセスが可能となる</li> </ul>
6	OS 抽象化 インタフェース	内部 I/F	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> と OS/プラットフォーム依存部分のインタフェース</li> <li>・ セマフォア、メモリ割当て/解放、現在時刻取得、タスク ID の取得、タスク固有空間とのデータの授受等</li> <li>・ <math>\mu</math> ITRON 準拠の OS については、同インタフェースに対応した OS 依存部を <b>GR-FILE</b> でサポート</li> <li>・ その他の OS については、同インタフェースに準拠した OS 依存部の作成が必要</li> </ul>
7	デバイスドライバ インタフェース	利用	<ul style="list-style-type: none"> <li>・ <b>GR-FILE</b> がデバイスドライバに対して規定したインタフェース</li> <li>・ デバイスの open/close/物理ブロック単位の read/write のインタフェースを規定</li> <li>・ 本インタフェースに従ったデバイスドライバの作成が必要</li> <li>・ 本バージョンでは、<math>\mu</math> iTron のデバイスドライバを使った <b>GR-FILE</b> とのインタフェース関数のサンプルを提供</li> <li>・ なお、メディアの挿抜に伴うファイルシステムの自動 mount/unmount 処理は、本デバイスドライバインタフェースでは規定しない。同処理については、デバイスドライバでメディアの挿抜を検出して、同イベントを挿抜処理用のシステムアプリケーションに伝え、同システムアプリケーションが、<b>GR-FILE</b> が提供する#3 のインタフェースを使用して実現する</li> <li>・ <b>GR-FILE</b> では、挿抜処理用のシステムアプリケーションのサンプルを提供</li> <li>・ #6 のデバイス直接制御インタフェース等で利用する、デバイス固有の制御機能をオプションでサポート</li> </ul>

## 1.2 機能・特徴概要

表 1-3 に **GR-FILE** の機能・特徴を示します。

表 1-3 GR-FILE の機能・特徴一覧

#	項目	機能・特徴	効果
1	FAT ファイルシステムサポート	・ FAT12/16/32、ロングファイル名をサポート	・ PC とのデータ交換が可能
2	標準+固有アプリケーションインタフェース	<ul style="list-style-type: none"> <li>・ POSIX 準拠 I/O インタフェース open/close/read/write 等</li> <li>・ C 言語標準 I/O インタフェース(オプション) fopen/fclose/getc/putc 等</li> <li>・ <b>GR-FILE</b> 固有インタフェース メディア抜き取り対応処理、フォーマット/パーティション設定、パラメータ設定等</li> <li>・ デバイス直接制御インタフェース メディアの取出し/ロック/アンロック、物理フォーマット等、デバイス固有機能の利用インタフェース</li> </ul>	<ul style="list-style-type: none"> <li>・ LINUX/Windows 用ソースの利用、同 OS 上でのテストが可能</li> <li>・ アプリケーション開発が容易</li> <li>・ メディア、プラットフォームに応じた最適処理が可能</li> <li>・ デバイス固有機能利用が可能</li> </ul>
3	マルチタスク I/O	・ 複数タスクによる同時 I/O をサポート	・ マルチタスク環境で利用可能
4	日本語（多国語）ファイル名対応	<ul style="list-style-type: none"> <li>・ Shift-JIS ファイル名をサポート</li> <li>・ 他の日本語コード、多国語の対応も容易</li> <li>・ Shift-JIS 用コードの取外しも可能</li> </ul>	<ul style="list-style-type: none"> <li>・ 日本語システムに即適用可能</li> <li>・ 多国語システムにも対応可能</li> <li>・ コードサイズの削減が可能</li> </ul>
5	ドライブ型+階層化 mount	・ ドライブ毎に個別のファイルシステムとする方法に加え、ファイルパス上に別ファイルシステムを接続する階層型 mount をサポート	・ 複数デバイスを 1 つの論理ファイルシステムに見せ、各デバイスサイズの制限を排除可能
6	データと管理情報のキャッシング（高速化）	・ ファイルデータとファイル管理ブロックの情報を分離してキャッシュし、さらに、オープン中のファイル情報、フリーブロック、ファイル名情報も別途キャッシング	・ 大容量ファイルの I/O に対しても、ファイル領域割当て情報をキャッシュ上に維持でき、高速 I/O が可能
7	キャッシュの write 制御（信頼性）	<ul style="list-style-type: none"> <li>・ メディアの特性に応じ、キャッシュ上の最新情報のメディアへの反映方式を選択可能</li> <li>(1)write through 方式 各 write 時に変更をメディアに反映</li> <li>(2)each close 方式 ファイル close 時にメディアに反映</li> <li>(3)last close 方式 当該ファイルシステムの最後のファイルに対する close 時に反映</li> <li>(4)unmount 方式 メディア unmount 時にメディアに反映</li> </ul>	<ul style="list-style-type: none"> <li>・ メディアの特性に応じて、性能と信頼性をバランスよく実現可能</li> <li>(例 1) 内蔵ハードディスク 取外しができないため、性能を重視し、unmount 方式を採用</li> <li>(例 2) メモリカード 不意の取外しがあるため、信頼性を考え、各アプリケーション終了時に確実にデータの反映を行う each close 方式を採用</li> </ul>
8	アプリケーションバッファとメディア間の直接/連続ブロック I/O	・ キャッシュにないデータに対し、キャッシュバッファを使用せず、アプリケーションバッファとメディア間で直接、かつ、連続したブロックを一括して I/O するモードを選択可能	・ 大容量データで、頻繁には read しないファイルと、少量データで頻繁に read するファイルを区別して高速な I/O を実現可能。
9	ファイルシステム依存部の分離	・ キャッシング等のファイルシステムに共通な処理と、ファイルシステム依存処理部を分離	・ 同一インタフェースで、様々なファイルシステムをサポート可能
10	OS 依存処理部の分離	<ul style="list-style-type: none"> <li>・ OS/プラットフォーム依存処理部分をファイルシステム処理から分離</li> <li>・ <math>\mu</math>ITRON 準拠の OS の依存処理をサポート</li> </ul>	<ul style="list-style-type: none"> <li>・ 様々な OS への移植が容易</li> <li>・ OS レス環境下での利用も可能</li> </ul>
11	メディアの挿抜対応	<ul style="list-style-type: none"> <li>・ キャッシュ write 制御により反映契機を制御</li> <li>・ 不意の取外し以降の I/O 抑止機能を提供</li> <li>・ メディア再挿入時のボリューム名チェック機能を提供</li> <li>・ メディアへの反映ができなくなったキャッシュデータの読出し機能を提供</li> </ul>	・ メモリカード等で、不意の挿抜が予想されるメディアに対し、メディア上のデータの整合性確保、間違ったメディアへの書込みの防止が可能

12	メディアの フォーマット/ パーティション 設定	<ul style="list-style-type: none"><li>・メディアサイズに応じて FAT12/16/32 を自動的に選択しメディアのフォーマットが可能</li><li>・FAT タイプ、クラスタサイズの明示指定も可</li><li>・パーティションの設定変更も可能</li></ul>	<ul style="list-style-type: none"><li>・自動簡易フォーマットとカスタマイズフォーマットの両方が可能</li><li>・パーティションのフォーマットに加え、パーティション設定が可能</li></ul>
13	RAM ディスク 機能の提供	<ul style="list-style-type: none"><li>・特別なデバイスメディアの 1 つとしてメモリ上の仮想ディスク機能を提供</li></ul>	<ul style="list-style-type: none"><li>・メモリ上に高速なファイルシステムを構築可能</li></ul>
14	各種パラメータの 設定・変更	<ul style="list-style-type: none"><li>・キャッシュブロックサイズ、キャッシュブロック数、同時オープンファイル数等の各種パラメータを実行時の初期化処理で設定・変更が可能</li></ul>	<ul style="list-style-type: none"><li>・ターゲットシステムの構成に応じ、同ターゲットシステムに適した各種パラメータを実行時に設定可能</li></ul>



## 2. 前提条件・制限

**GR-FILE** で仮定している前提条件・制限等を、表 2-1 に示します。

表 2-1 GR-FILE の前提条件・制限

#	項目	前提条件・制限
1	ファイルのクローズ、カレントディレクトリの無効化処理	<ul style="list-style-type: none"> <li>• <b>GR-FILE</b> は、タスク毎にオープン中のファイルとカレントディレクトリを管理し、ファイルのオープン/カレントディレクトリ設定時に同管理情報を生成して、設定します。しかし、タスクが消滅しても、同タスクがオープンしたファイルのクローズ/カレントディレクトリの無効化は自動的には行いません。</li> <li>• ファイルのオープン、あるいは、カレントディレクトリの設定をした場合は、必ず、同処理を行ったアプリケーションが責任を持ってクローズまたは無効化関数をコールするか、あるいは、別のアプリケーションまたは OS でタスクの消滅を検知し、同タスクのオープンファイル、カレントディレクトリの無効化関数をコールして下さい。(「マルチタスク I/O」の節参照)</li> <li>• クローズ処理/無効化処理が行われない場合、メディアの unmount 処理が I/O 処理中としてビジーエラーとなったり、管理テーブルが不足して、オープン処理がエラーとなる場合があります。</li> </ul>
2	マルチタスク環境のサポート	<ul style="list-style-type: none"> <li>• <b>GR-FILE</b> では、<b>GR-FILE</b> のコードおよびデータがタスク間でシェアされることを前提として、マルチタスクによる同時 I/O をサポートしています。</li> <li>• 動作形態としては、サブシステムとしてのファイルシステムではなく、アプリケーションに <b>GR-FILE</b> をリンクする形を仮定しています。</li> <li>• サブシステムのような形で、空間が異なるタスク間で <b>GR-FILE</b> を共用する場合は、システムコールで <b>GR-FILE</b> のアプリケーションインタフェースに結びつける処理や、タスク空間渡りのデータの授受処理等を実装する必要があります。なお、タスク間渡りのデータ授受は <b>GR-FILE</b> でも考慮し、同インタフェースを規定しています。</li> </ul>
3	ファイルのプロテクション	<ul style="list-style-type: none"> <li>• アプリケーションインタフェースとしては、POSIX に準拠し、オーナー/グループ/その他利用者によるファイルプロテクションの設定インタフェースを提供していますが、FAT には利用者の概念がないため、利用者毎のプロテクションの設定はできません。</li> <li>• 現行バージョンは、ファイルシステム共通処理部で、利用者の概念をサポートしておらず、利用者毎でのファイルプロテクションはサポートできません。</li> </ul>
4	メディアの挿抜処理	<ul style="list-style-type: none"> <li>• メディアの挿抜に伴うファイルシステムの自動 mount/unmount 等の挿抜処理は、デバイスドライバがメディアの挿抜検出機能を持ち、挿抜処理用の特別なシステムアプリケーションがデバイスドライバから挿抜通知を受けて行うことを前提としています。</li> <li>• <b>GR-FILE</b> は、mount/unmount 機能、メディア再挿入時のボリューム名チェック機能等、この挿抜処理用のシステムアプリケーションのためのインタフェースを提供しています。</li> <li>• <b>GR-FILE</b> が提供するこれらのメディア挿抜処理用のインタフェースは、割込みの延長ではなく、タスクの環境下でコールされることを前提とします。</li> </ul>
5	ファイルシステムの整合性チェックプログラム	<ul style="list-style-type: none"> <li>• ファイルシステムのフォーマット機能は提供しますが、ファイルシステムの整合性チェックプログラムは、本バージョンでは、提供しません。</li> <li>• ファイルシステムの整合性チェックは、PC 等で行うようにして下さい。</li> </ul>
6	デバイス I/O タイミング	<ul style="list-style-type: none"> <li>• アプリケーションがデータを write しても、キャッシング機能により、その時点ではメディアに書込まず、即座にはメディアに反映されないケースがあります。</li> <li>• また、キャッシングにより、更新順にメディアに反映されとは限りません。</li> <li>• あるタスクで read 要求を出した場合でも、read 時に使用するキャッシュバッファを確保するため、同バッファに保持された別のキャッシュデータを書戻すためのメディアへの write が、同タスクの延長で行われることがあります。</li> <li>• read 要求でも、アクセス日時情報の更新のため、メディアへの write が発生することがあります。(FAT では同情報が日単位のためアクセス日が異なる場合)</li> </ul>
7	標準ライブラリ	<ul style="list-style-type: none"> <li>• str(n)cmp、memcpy、memmove、(v)s(n)printf 等の標準ライブラリ関数はターゲットプラットフォームで提供されていることを前提とします。</li> </ul>

### 3. 機能詳細

本章では、表 1-3 で示した **GR-FILE** の各機能、特徴について説明します。

#### 3.1 FAT ファイルシステムサポート

**GR-FILE** は、ファイルシステム依存部を分離し、いろいろなファイルシステムをサポートできるように作られています。本バージョンでは、ファイルシステムとして、PC 等で利用されている FAT ファイルシステムをサポートします。

具体的には、以下の FAT ファイルシステムをサポートします。

- (1) FAT12
- (2) FAT16
- (3) FAT32

また、ファイル名は、Windows 等でサポートされている可変長のロングファイル名をサポートし、英小文字や日本語（多国語）を含んだファイル名、長いファイル名の利用が可能です。ただし、コンパイルオプション **GRP\_FS\_MINIMIZE\_LEVEL** を 2 に設定した場合、ロングファイル名は未サポートとなります。ファイル名に関する仕様の詳細については、「3.4 日本語（多国語）ファイル名サポート」の節を参照下さい。

なお、ファイルシステムの修復、ボリューム名の変更は、サポートしておりません。ファイルシステム整合性チェック・修復、ボリューム名変更は、PC 等で行って下さい。

FAT ファイルシステム自体の詳細仕様については、本書の範囲外のため、市販の概説書等をご参考下さい。

### 3.2 標準+固有アプリケーションインタフェース

**GR-FILE** では、標準のアプリケーションインタフェースである POSIX 互換の I/O インタフェース、および、C 言語標準 I/O インタフェース（オプション）と、**GR-FILE** 固有の機能を利用するための **GR-FILE** 固有インタフェース、および、デバイス固有の機能を利用するためのデバイス直接制御インタフェースをサポートしています。これにより、他システムからの移植性や、開発の容易性を高めるとともに、**GR-FILE** 固有の特徴ある制御や木目細やかな制御やデバイス固有の機能の利用を可能としています。なお、各インタフェースの詳細につきましては、「インタフェース」の章をご参照下さい。

なお、**GR-FILE** でサポートする POSIX 互換、および、C 言語標準 I/O インタフェースを用いたファイル I/O では、Windows のような改行コードの復帰+改行コードへの変換は一切行いません。すなわち、Windows でいう、バイナリモードのみをサポートしています。

以下に **GR-FILE** の持つ 3 つのインタフェースについて、特徴・注意点の比較を表 3-1 インタフェース比較に示します。

表 3-1 インタフェース比較

#	インタフェース名	特徴	注意点
1	POSIX 互換の I/O インタフェース	<ul style="list-style-type: none"> <li>アプリケーションは、POSIX 互換の表記が可能</li> <li>POSIX 表記を、<b>GR-FILE</b> 固有インタフェースに変換しているの、C 言語標準 I/O インタフェースに比べ、パフォーマンス面で有利</li> </ul>	<ul style="list-style-type: none"> <li>エラーコードは <b>GR-FILE</b> のコードになる</li> <li>POSIX 表記を、<b>GR-FILE</b> 固有インタフェースに置き換える為のヘッダーファイル” grp_fs_conv.h” をインクルードする必要がある</li> <li>マウントなど、POSIX に無い機能は <b>GR-FILE</b> 固有インタフェースを使用する必要がある</li> </ul>
2	C 言語標準 I/O インタフェース	<ul style="list-style-type: none"> <li>C 言語標準の表記が可能</li> </ul>	<ul style="list-style-type: none"> <li>C 言語標準の表記を、<b>GR-FILE</b> 固有インタフェースに置き換える為のヘッダーファイル” grp_stdio.h” をインクルードする必要がある</li> <li>内部的には <b>GR-FILE</b> 固有インタフェースを使用しているため、違いを吸収する処理が行われる。その為、他のインタフェースに比べパフォーマンスが出難い。また、この処理に別途バッファを必要とする</li> <li><b>GR-FILE</b> のダイレクト I/O 機能は使用できない</li> <li>POSIX 互換の I/O インタフェース、<b>GR-FILE</b> 固有インタフェースとの混在使用は出来ない</li> </ul>
3	<b>GR-FILE</b> 固有インタフェース	<ul style="list-style-type: none"> <li><b>GR-FILE</b> が持つ全ての機能が使用可能</li> <li><b>GR-FILE</b> を使用する上での最大のパフォーマンスが期待できる</li> </ul>	<ul style="list-style-type: none"> <li><b>GR-FILE</b> 固有の名称の為、他のプラットフォームへの移植時は注意が必要</li> </ul>



### 3.2.1 POSIX 互換の I/O インタフェース

open/close/read/write 等、POSIX で規定された I/O インタフェースをサポートしています。従いまして、Linux や UNIX システム、Windows 上で動作しているアプリケーションのファイル I/O 処理を殆ど変更なく、そのまま組込みシステム上で動作させることが可能です。また、新規アプリケーションを作成する場合も、Linux や Windows システム上のファイルを使ってテストし、そのまま、組込みシステムを持っていくことが可能です。ただし、コンパイルオプション `GRP_FS_MINIMIZE_LEVEL` を 1 または 2 に設定した場合、POSIX 互換の I/O インタフェースは使用できません。

#### (1) インタフェース名称

POSIX 互換の I/O 関数名、I/O 定数名を使用し、アプリケーションの作成が可能です。但し、実際には、“`grp_fs_open`” や “`GRP_FS_O_RDONLY`” 等、POSIX 互換の名称に “`grp_fs_`”、“`GRP_FS_`” を付加した形の関数を使って実現しています。図 3-1 に示しますように、アプリケーションプログラムで Windows の “`io.h`” や Linux の “`fcntl.h`” 等の代わりに “`include`” ディレクトリ下にある “`grp_fs_conv.h`” をインクルードして頂くことで、POSIX 互換名称で記述した I/O 関数名、I/O 定数名を、**GR-FILE** の実際の名称に変換し、POSIX 互換インタフェースでのプログラミングを可能としています。


<pre>#include "grp_fs_conv.h"  int open_file(const char *file) {     int fd;      fd = open(file, O_RDONLY, 0);     ... }</pre>	<p>変換</p> 	<pre>...  int open_file(const char *file) {     int fd;      fd = grp_fs_open(file, GRP_FS_O_RDONLY, 0);     ... }</pre>
---	---	--

図 3-1 POSIX 互換インタフェース名称変換

このインクルードファイルによる変換機能を利用しますと、Windows 環境等でのテストを行う場合に、ターゲットプログラムのソースファイルだけに “`include`” ディレクトリ下にある “`grp_fs_conv.h`” をインクルードすることで、動作ログ出力等のテスト用のプログラムは Windows のファイル等に出力するなど、ターゲットとテスト環境の混在を実現することができます。

#### (2) リターン値

I/O 関数からの処理成功時のリターン値は、POSIX に準拠した値を返します。但し、エラー時は -1 ではなく、エラー内容に応じた **GR-FILE** 固有の負値のエラー番号を返します。各エラー番号に対しては、**GR-FILE** 固有のシンボリックな名称が “`grp_fs_if.h`” に `define` されています。なお、エラーには、固有のエラー情報もあるため、POSIX で使用されるシンボリックな名称との変換は用意しておりません。

従いまして、I/O 関数の処理結果がエラーかどうかの判定は、-1 に等しいかどうかで行うのではなく、負値かどうかで判定頂く必要があります。また、エラー番号に基づき、詳細なエラー処理を行う場合は、`errno` という変数ではなく、I/O 関数からのリターン値を使い、“`grp_fs_if.h`” で定義された **GR-FILE** 固有のシンボリックなエラー名称を使用して行って下さい。

### 3.2.2 C 言語標準 I/O インタフェース（オプション）

fopen/fclose/getc/putc 等の C 言語の標準 I/O インタフェースをオプションとしてライブラリの形で提供します。このインタフェースは、I/O 要求をライブラリレベルでバッファリングする機能を持っていますので、getc/putc を使い 1 文字ずつ処理するアプリケーションプログラムでも、実際のファイルへの read/write 回数が減り、性能を維持することが可能です。

すなわち、バッファリングを意識せずにプログラムが書けるため、アプリケーションプログラムの作成が容易です。しかも、C 言語標準の I/O インタフェースですので、他システムからの移植性が高く、Linux や Windows 等、様々なシステム上でテストが可能です。ただし、コンパイルオプション GRP\_FS\_MINIMIZE\_LEVEL を 1 または 2 に設定した場合、C 言語の標準 I/O インタフェースは使用できません。

#### （1）インタフェース名称

C 言語標準の I/O 関数名を利用し、アプリケーションの作成が可能です。但し、実際には、"grp\_stdio\_fopen" 等、概ね C 言語標準の I/O 関数名に "grp\_stdio\_" を付加した形の関数を使って実現しています。図 3-2 に示しますように、アプリケーションプログラムで <stdio.h> の代わりに、"include" ディレクトリ下にある "grp\_stdio.h" をインクルードして頂くことで、C 言語標準の I/O 関数名を、**GR-FILE** の実際の名称に変換し、C 言語標準の I/O インタフェースでのプログラミングを可能としています。


<pre>#include "grp_stdio.h"  int open_file(const char *file) {     FILE *fp     fp = fopen(file, "w");     ... }</pre>	変換 	<pre>...  int open_file(const char *file) {     FILE *fp;     fp = grp_stdio_fopen(file, "w");     ... }</pre>
--	---	--

図 3-2 C 言語標準の I/O インタフェース名称変換

このインクルードファイルによる変換機能を利用しますと、Windows 等でのテストを行う場合に、ターゲットプログラムのソースファイルだけ <stdio.h> に代えて "grp\_stdio.h" をインクルードすることで、動作ログ出力等のテスト用のプログラムは Windows のファイル等に出力するなど、ターゲットとテスト環境の混在を実現することができます。

#### （2）リターン値

I/O 関数からのリターン値は、C 言語の標準 I/O 関数の仕様に準拠した値を返します。例えば、I/O 関数の処理がエラーの場合、リターン値として -1 を返します。最後に発生したエラーの詳細なエラー番号は、ferror により知ることが可能です。この ferror により返ってくる値は、上記、POSIX 互換 I/O インタフェースで返ってくる **GR-FILE** 固有の負値のエラー番号です。各エラー番号に対しては、**GR-FILE** 固有のシンボリックな名称が "grp\_fs\_if.h" に define されていますので、プログラムでエラー番号を参照される場合は、同ファイルに define されているシンボリックな名称をお使い下さい。

### (3) 標準入出力

**GRP-FILE** では、標準入出力も同様に扱えるように、C 言語の標準 I/O と同様に、既定義の FILE 構造体へのポインタ変数 `stdin`、`stdout`、`stderr`（実際の変数名称は、`grp_stdio_stdin`、`grp_stdio_stdout`、`grp_stdio_stderr`）を用意しています。

但し、標準入出力を実現するためには、標準入出力のためのプラットフォーム依存関数を定義し、以下の関数ポインタ変数に、定義したプラットフォーム依存関数を登録する必要があります。本関数ポインタ変数を登録しないと、`stdin`、`stdout`、`stderr` に対する入出力要求は、すべてエラーとなります。

<code>grp_stdio_io_stdin</code>	<code>stdin</code> に対応したプラットフォーム依存の入力関数へのポインタ変数
<code>grp_stdio_io_stdout</code>	<code>stdout/stderr</code> に対応したプラットフォーム依存の出力関数へのポインタ変数 ( <code>stdout</code> と <code>stderr</code> の区別はなく、本変数に登録された関数がコールされます)

標準入出力のためのプラットフォーム依存関数のインタフェースについては、「4 インタフェース」の章の「4.8 OS 抽象化インタフェース」の節を参照下さい。

### 3.2.3 GR-FILE 固有インタフェース

上記標準の I/O インタフェースに加え、メディア挿抜処理を実現するための mount/unmount 機能、ボリューム名チェック機能、反映不能となったキャッシュデータの読出し機能や、ファイルシステム依存の属性パラメータ指定（例えば、FAT ファイルシステムの隠しファイル属性指定）、さらに、メディアのフォーマットやパーティションの設定変更機能、**GR-FILE** の各種パラメータの設定・変更機能等の **GR-FILE** 固有のインタフェースを提供します。ただし、コンパイルオプション `GRP_FS_MINIMIZE_LEVEL` を 1 または 2 に設定した場合、特殊な **GR-FILE** 固有インタフェースは使用できません。使用可否については表 4-5 を参照して下さい。

上記標準の I/O インタフェースのインタフェース名称の説明で示しましたように、**GR-FILE** で提供します外部インタフェース名称は、すべて、`"grp_fs_"` または `"grp_stdio_"` で始まります。

これらの外部インタフェースには、ほぼ、1 対 1 に標準の I/O インタフェースに対応する場合もあれば、パラメータ等を追加して標準の I/O インタフェースを拡張したもの、標準の I/O インタフェースには全く規定されていないものがあります。

広義には、POSIX 互換インタフェースを実現するために 1 対 1 に対応した関数も含め、`"grp_fs_"` で始まる外部インタフェースすべてが、**GR-FILE** 固有インタフェースです。狭義には、標準の I/O インタフェースを拡張し、POSIX で規定されていないタイプのファイルシステムをサポートするためのインタフェースや、メディア挿抜対応等の POSIX で規定されていない機能をサポートするインタフェースが、**GR-FILE** 固有のインタフェースです。

### 3.2.4 デバイス直接制御インタフェース

メディアの自動取出し/ロック/アンロック機能、物理フォーマット機能など、デバイス固有で提供される機能をアプリケーションから利用できるようにするためのインタフェースを提供します。

このインタフェースは、**GR-FILE** のデバイスドライバインタフェースに薄皮を被せてアプリケーションインタフェースとして見せているもので、上記、デバイス固有機能を利用するための `grp_fs_ioctl_dev` に加え、メディア直接のセクタ単位の `read/write` 等を行うための `grp_fs_open_dev`、`grp_fs_close_dev`、`grp_fs_read_dev`、`grp_fs_write_dev` の利用も可能です。但し、**GR-FILE** によるファイルシステムとしてのアクセスとの排他は行っていないので、メディアをファイルシステムとして `mount` していない間や、ファイルシステムとしてアクセスされている領域外のエリアに限定して `I/O` を行って下さい。ファイルシステムとして `mount` 中にファイルシステムとしてアクセスされている領域内のデータを変更した場合は、動作の保証ができません。

また、デバイス固有の機能につきましては、`grp_fs_ioctl_dev` という汎用的なインタフェースの枠組みと、一部の機能については、具体的なパラメータを **GR-FILE** で規定していますが、その他の機能の具体的なパラメータについては、各デバイスで提供する機能に依存します。既に、**GR-FILE** で規定しているデバイス固有機能につきましても、必ずしも各デバイスでサポートされているとは限らず、サポートするか否かはデバイス依存です。指定した機能がサポートされていない場合は、`grp_fs_ioctl_dev` のリターン値として、`GRP_FS_ERR_NOT_SUPPORT` が返ります。

### 3.3 マルチタスク I/O

**GR-FILE** は、シングルトaskによるファイルの読書きだけでなく、複数のタスクによるファイルの読書きをサポートしています。また、仮想的に 1 タスクからなる OS レスの環境でも利用可能です。

#### 3.3.1 並列ファイル I/O

**GR-FILE** は、キャッシュに要求のファイルのデータがない場合、あるいは、メディアへのキャッシュの書戻しが必要となった場合、I/O 要求を出したアプリケーションタスクの延長で、デバイス I/O を実行します。このデバイス I/O は同期型の実行モデルを前提としており、デバイス I/O を開始すると、同 I/O が終了するまで、同アプリケーションタスクの実行はブロックされます。

**GR-FILE** では、その間を利用し、他のアプリケーションタスクからの I/O 要求を並列に処理します。例えば、図 3-3 に示しますように、タスク A がキャッシュにない file1 のあるデータの read 終了待ちをしている間に、タスク B がキャッシュ上にある file2 のあるデータに対して read 要求を出しますと、同要求が並列に処理され、I/O 待ち中に並行した処理が可能です。さらに、タスク A が read 終了待ちで、タスク B もキャッシュにない、file2 のあるデータに対して read 要求出したようなケースでは、タスク B も、デバイス I/O を実行し、タスク A と同様に read 終了待ちとなります。

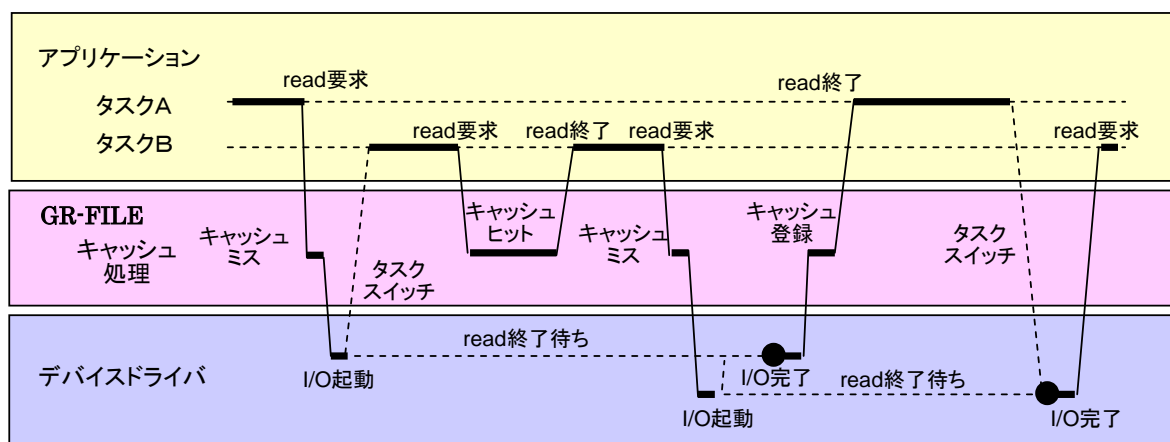


図 3-3 マルチタスクによる並列 I/O

すなわち、キャッシュを用いたファイル I/O と、実際のデバイス I/O を伴うファイル I/O とが並列に行えますし、実際のデバイス I/O を伴うファイル I/O 間でも、複数のメディア等への I/O で、デバイスドライバが実 I/O を並列に実行可能であれば、並列に処理することが可能です。逆に、**GR-FILE** では、デバイスドライバが、複数のタスクからの I/O 要求に対し、並列に I/O 可能であれば、並列に I/O を実行し、並列 I/O が不可の場合は、キューイングやシリアルライズを行い、1 つずつ順に I/O 要求を実行する機能を持つことを前提としています。

また、**GR-FILE** では、キャッシュレベルでも複数タスクに対する処理機能を持っています。**GR-FILE** では、各ファイルのデータだけでなく、複数のファイルの管理情報を保持したファイル管理ブロックもキャッシュとして保持しています。例えば、図 3-4 に示しますように、タスク A であるファイルの I/O 処理を行うため、メディアからの同ファイルの管理情報を含むファイル管理ブロックの read を行い、その read 待ち中に、タスク B の処理でも同じファイル管理ブロックの情報が必要となった場合を考えます。この場合、タスク B から、同管理ブロックの情報に対する read 要求をデバイスドライバに対して出すのではなく、タスク A が同情報の read 中であることを検出し、タスク A の read が終了して、read されたデータがキャッシュに登録されるのを待ってから、タスク B は登録されたキャッシュを使って処理を行います。

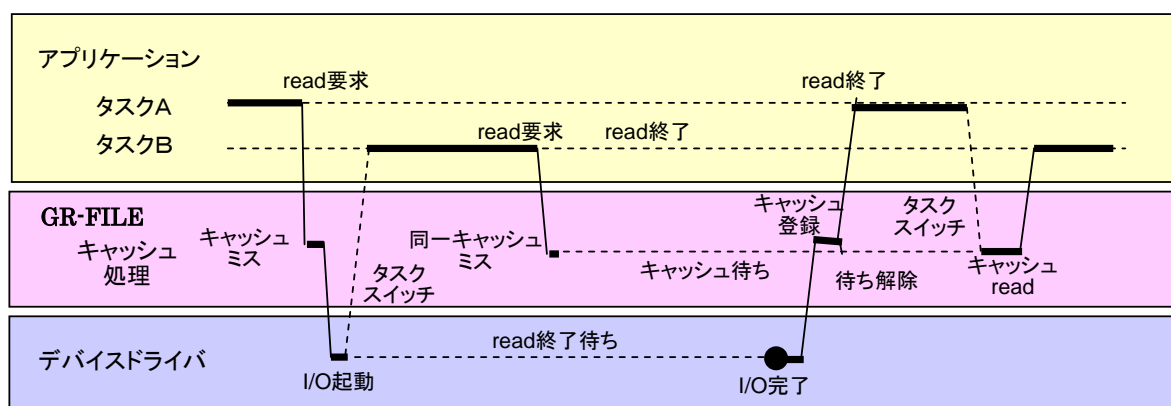


図 3-4 キャッシュレベルの複数タスク向けの処理機能（領域管理情報）

なお、**GR-FILE** では、データの不整合を防止するため、同一ファイルに対する複数のタスクからの I/O 要求に対しては、ファイル単位で、シリアルライズ処理を行っています。あるタスクがあるファイルに対する I/O 待ちの状態では、他のタスクからの同一ファイルに対する I/O 要求は、別のブロックに対する I/O 要求であっても、前記タスクの I/O 処理（read/write 等）が終了するまで、待たされます。

## （2）タスク毎のファイル管理機能

**GR-FILE** では、タスク毎に、カレントディレクトリ、オープンしたファイルを管理しています。従いまして、アプリケーションタスク毎に別々のカレントディレクトリを設定することが可能です。

例えば、タスク A、タスク B、それぞれで、カレントディレクトリを `"/usr"`、および、`"/src"` に設定し、それぞれのタスクで、`"file1"` というファイルをオープンしたとしますと、タスク A では、`"/usr/file1"`、タスク B では、`"/src/file1"` をそれぞれオープンしたことと等しい形になります。

また、カレントディレクトリもパス名を記憶しているのではなく、カレントディレクトリをオープンした状態でキャッシングし、カレントディレクトリ下のファイルのアクセスを高速化しています。

なお、各タスクでオープンしたファイル、および、各タスクで設定したカレントディレクトリは、各タスクの責任、あるいは、同タスクの消滅を監視し、その後始末をするタスクまたは OS の責任で、クローズ/無効化をする必要があります。タスク消滅時にクローズ/無効化せずに放置しておきますと、ファイル/ディレクトリがオープンしたままの状態となり、メディアを取外し時等のファイルシステムの `unmount` 処理がビジーエラーとなります。また、そのようなファイルが累積しますと、オープンファイルの管理ブロックが不足し、新たなファイルをオープンできない状態になります。



表 3-2 に、各タスクのオープンファイルのクローズ、カレントディレクトリの無効化の方法を示します。

表 3-2 オープンファイル/カレントディレクトリのクローズ/無効化処理

#	対象 タスク	クローズ/無効化対象	クローズ/無効化方法
1	自タスク	各オープンファイル	・ オープン時に使用したインタフェースに対応した <code>grp_fs_close</code> 、 <code>close</code> 、 <code>fclose</code> を利用
2	自タスク	カレントディレクトリ	・ カレントディレクトリパラメータとして、NULL パスまたは長さ 0 のパス名を指定して <code>grp_chdir</code> または <code>chdir</code> をコール
3	自タスク	全オープンファイル カレントディレクトリ	・ <code>grp_fs_task_free_env</code> をコール
4	他タスク	全オープンファイル カレントディレクトリ	・ タスク ID を指定して、 <code>grp_fs_task_free_env_by_id</code> をコール
5	全タスク	全オープンファイル カレントディレクトリ	・ <code>grp_fs_task_free_all_env</code> をコール

また、ファイルのオープン情報は、オープンしたタスクと関連づけて管理していますので、他タスクでオープンしたファイルのファイルハンドルを用いて I/O することはできません。但し、**GR-FILE** のオブジェクト作成時に、`GRP_FS_SHARE_OPEN` という define をつけてコンパイルしますと、タスク間でのファイルハンドルの共用が可能となります。

### 3.3.2 OS 非依存のファイル情報管理

**GR-FILE** では、上記で示しましたタスク毎のファイル管理機能を OS に依存しない形で提供しています。例えば、タスク毎の管理情報は、I/O 要求時に、自動的に要求タスクのタスク ID をベースに動的に生成します。また、表 3-2 に示しましたように、タスク ID を指定して、対象タスクのファイル管理情報をクリアする関数を提供していますので、OS のタスク管理機能と連動して、タスクの消滅時に同関数をコールすることで、OS のサブシステムとして、ファイル管理機能を簡単に付加することができます。

また、アプリケーションタスクと仮想空間が異なるケースも想定し、ファイルデータやパス名称等のデータのやりとりは、単純なメモリコピーで行えるという前提をおかず、アプリケーションタスク間とのデータ授受のためのインタフェースを規定して行っています。すなわち、多重仮想空間をサポートした OS で、本 **GR-FILE** をファイル管理サブシステムとして複数タスクで共用し、利用することも考慮した作りになっています。

なお、同時に I/O するタスクの最大数、オープンファイル数等のパラメータは、システム初期化時に変更することが可能です。（「3.1.4 各種パラメータの設定・変更、および、コンパイルオプション」の節参照）

FAT では、ファイル名に使用出来る文字に制限があります。以下に使用できる文字について説明します。



### 3.4 日本語（多国語）ファイル名サポート

**GR-FILE** の FAT ファイルシステムは、Windows 等でサポートされている可変長のロングファイル名をサポートし、英小文字や日本語（多国語）を含んだファイル名の利用を可能としています。特に、日本語については、シフト JIS に対応したコードを提供し、日本語システムへの適用を可能としています。

FAT では、ファイル名に使用出来る文字に制限があります。以下に使用できる文字について説明します。

#### 3.4.1 ショートファイル名

FAT ファイルシステムでは、すべてのファイルは、例えば、“FILE.TXT”のように、最大 8 バイトのベースファイル名と最大 3 バイトの拡張子をピリオドで区切った従来 MS-DOS 互換のショートファイル名を持ちます。**GR-FILE** は、この従来 MS-DOS の仕様に準拠したショートファイル名をサポートします。

ショートファイル名には、任意の文字が使えるのではなく、以下の文字のみが使用可能となっています。

- (a) 英大文字
- (b) 数字
- (c) \$ % ' - \_ @ ~ ` ! ( ) { } ^ # &
- (d) コード値 1 2 8 以上の日本語（他国語）文字

**GR-FILE** では、デフォルトで、シフト JIS 漢字（ひらがな、カタカナ、半角カタカナを含む）の処理コードを提供し、日本語としては、シフト JIS のファイル名称に対応可能となっています。

上記以外の文字を使用したファイル名、あるいは、ベースファイル名、識別子が規定のバイト数以内に収まらないファイル名は、後述するロングファイル名のディレクトリエントリに本当のファイル名称を保持し、同エントリに引続くディレクトリエントリに、ショートファイル名に変換した名称を保持します。

ロングファイル名からショートファイル名への変換は、概ね以下の形で行います。

- ① 英小文字は英大文字に変換する
  - ② ショートファイル名として許されない文字は、‘\_’に変換する
  - ③ 空白を取除く
  - ④ 最初のピリオドまでの 8 文字以内の文字と、ピリオド以降の 3 文字以内の文字を、それぞれ、ベースファイル名、拡張子とする
- ① 得られたショートファイル名が、既に存在する場合は、ベースファイル名称部の末尾の部分を“FILE~1.TXT”、“FILE~2.TXT”のように、“~数字”の形に置換し、ユニークなショートファイル名を見つける

### 3.4.2 ロングファイル名

上記で示しましたとおり、ショートファイル名の規定に合わないファイル名は、ロングファイル名としてロングファイル名のディレクトリエントリに保持します。但し、ロングファイル名でもすべての文字が許されるのではなく、上記ショートファイル名で利用可能な文字に加え、以下の文字がファイル名称として使用可能となっています。

英小文字, + , ; = [ ] . 空白

FAT ファイルシステムでは、ロングファイル名は、UNICODE に変換し、各ロングファイル名のディレクトリエントリに最大 13 文字（13 バイトではなく 13 文字）ずつ格納して保持します。（詳細は、市販の FAT ファイルシステムの概説書を参照）

**GR-FILE** では、デフォルトでシフト JIS と UNICODE の変換機能を提供し、シフト JIS のロングファイル名をサポートとしています。このロングファイル名の UNICODE 変換は、OS 抽象化インタフェースの一部として、分離された形で定義されていますので、他の日本語コードや多国語等、簡単に別のコード体系に対応させることが可能です。

なお、シフト JIS と UNICODE 間の変換は単純なアルゴリズムでは記述できず、シフト JIS・UNICODE 変換には、大きな変換テーブルを使って行っています。従いまして、日本語ファイル名を必要としないシステムでは、ロングファイル名の UNICODE 変換処理から、シフト JIS・UNICODE 変換を取り外すことで、コードサイズを小さくすることができます。これは、コンパイルオプション **GRP\_FS\_MINIMIZE\_LEVEL** を 2 に設定することで、ロングファイル名を未サポートにすることが可能です。

### 3.4.3 ファイル名の長さ制限

ファイル名には長さの制限があります。この制限値は、“include” ディレクトリ下の “grp\_fs\_param.h” に記述されています。その設定値を表 3-3 に示します。この値は、変更可能ですが、変更した場合は、再コンパイルが必要です。

表 3-3 ファイル名の長さの制限

#	define 名称	意味	GR-FILE の設定値	Windows の最大値
1	GRP_FS_MAX_PATH	ファイルパス名の最大長 (NULL を含む)	256 (バイト)	260 文字
2	GRP_FS_MAX_COMP	ロングファイル名の最大長 (NULL を含む)	128 (バイト)	255 文字 + NULL

なお、**GR-FILE** では、ファイル名の区切り文字は、‘/’、‘¥’ のいずれでも OK です。但し、URL 等の互換性を考え、‘/’ の利用を推奨します。

### 3.5 ドライブ型+階層化 mount

**GR-FILE** では、USB メモリやメモリーカードリーダーといった各デバイスに対応して **GR-FILE** のデバイスドライバテーブルに各デバイスの I/O 関数とデバイスタイプ名称を登録しておくことで、USB メモリやメモリーカード等のメディア上のデータをファイルシステムとしてアクセスすることを可能としています。このメディア上のデータをファイルシステムとしてアクセス可能とする処理が mount 処理です。

**GR-FILE** では、Windows のように各メディアをデバイスドライブとして、個別のファイルシステムとして見せるドライブ型 mount 機能だけでなく、**LINUX** 等のように、あるファイルシステムのファイルパス下に、別のファイルシステムを接続する階層化 mount 機能をサポートしています。ドライブ型 mount と階層化 mount は、混在した形で利用することも可能です。

また、1つのメディアを論理的なパーティションに分割し、各パーティションを個別のファイルシステムとしてアクセスすることも可能としています。指定したパーティションのアクセスは、デバイスドライバが提供する機能ですが、パーティション指定のためのネーミング規則を **GR-FILE** で規定し、さらに、PC 等で使用されている標準のパーティションテーブルのアクセス関数を **GR-FILE** で提供しています。

#### 3.5.1 ドライブ型 mount

Windows のように、各メディアをデバイスドライブとし、個別のファイルシステムのようにアクセスすることが可能です。例えば、図 3-5 に示しますように、ハードディスクは “C:”、メモリーカードは “D:” のように、各メディアにアルファベットのドライブ名を対応づけ、ドライブ名に続けて、各メディア内のファイルのパスを指定して、ファイルアクセスを実現します。

ドライブ名は、各メディアを挿着時等の mount 処理で実行する `grp_fs_mount` 関数のマウント先パス名パラメータで指定します。ドライブ名は、“アルファベット:” という形式だけでなく、“USB:” 等のようにコロンで終われば、任意の文字列を割当てることが可能です。


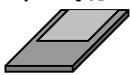

ドライブ名称例	“C:”	“D:”	“USB:”
メディア			
ファイルシステム の構成例	<pre> graph TD     I1["/"] --- tmp     I1 --- system     system --- f1     system --- fn           </pre>	<pre> graph TD     I2["/"] --- work     I2 --- save     save --- s1     save --- sm           </pre>	<pre> graph TD     I3["/"] --- my_save     my_save --- m1     my_save --- mx           </pre>
ファイル名称 の指定例	C:/system/f1	D:/save/s1	USB:/my_save/m1

図 3-5 ドライブ型 mount の例

### 3.5.2 階層化 mount

LINUX 等のように、あるメディアのファイルシステムのファイルパスの下に、別のファイルシステムを接続することが可能です。例えば、図 3-6 に示しますように、ハードディスクをルートファイルシステムとし、メモ리카ードをハードディスクの “/mnt/card” 下に、USB メモリを “/mnt/usb” 下に接続して、3 つのファイルシステムを仮想的に 1 つのファイルシステムのように見せかけることが可能です。

この階層化 mount 機能を使いますと、メディア一杯になった場合に、別のメディアを追加し、仮想的に 1 つの大きなメディアに見せかけることが可能となります。

なお、新たなファイルシステムを他のファイルシステムのどこに接続するかは、ドライブ型 mount と同様に、各メディアを挿着時等の mount 処理で実行する `grp_fs_mount` 関数のマウント先パス名パラメータで指定します。

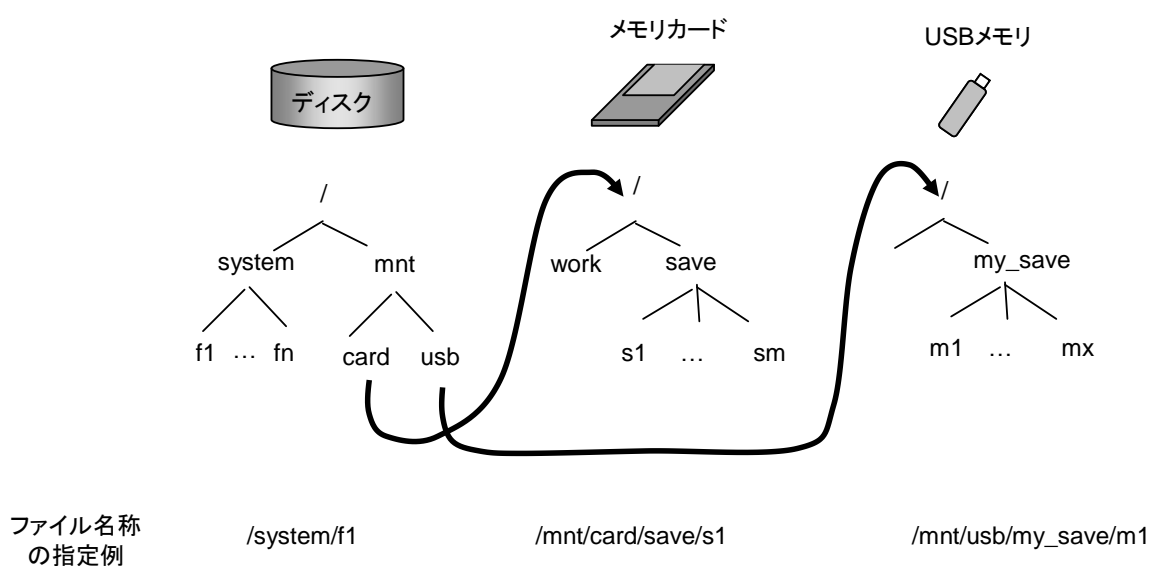


図 3-6 階層化 mount の例

### 3.5.3 パーティション分割されたメディアアクセスのためのデバイスネーミング規則

パーティション分割されたメディアのファイルシステムアクセスを可能とするため、パーティション指定のためのデバイスのネーミング規則を規定しています。

**GR-FILE** では、図 3-7 に示しますように、各デバイスの I/O 関数とデバイスタイプ名称を **GR-FILE** のデバイスドライバテーブルに予め登録し、**mount** 処理時に指定するデバイス名称により、各デバイスの I/O 関数との対応をとっています。さらに、同じ種類のデバイスでも複数のドライブや、1つのドライブのメディアが複数のパーティションからなっている場合に、デバイステーブルに登録された名称に続き 0~15 のサブデバイス番号と、a~d(※1)の英文字 1 字で a を基点としてパーティション番号を指定することで、ドライブやパーティションの指定を可能としています。例えば、USB デバイス用のデバイスドライバが“USB”というデバイスタイプ名称で **GR-FILE** のデバイスドライバテーブルに登録されている場合、デバイス名称を“USB0a”と指定すれば、USB の 0 番デバイスの、先頭パーティションを指し示すことができます。パーティション番号を表す英文字を省略した場合は、パーティションとして a を指定したと解釈します。サブデバイス番号も省略した場合は、サブデバイス番号として 0 を指定したものと解釈します。

※1 規定上は、a~o の 15 個のパーティションを指定できますが、**GR-FILE** で提供しているパーティションテーブルのアクセスライブラリ `grp_fs_get_part` 等は、4 つの基本パーティションのみのサポートですので、実質は、a~d の 4 文字のみが指定可能です。

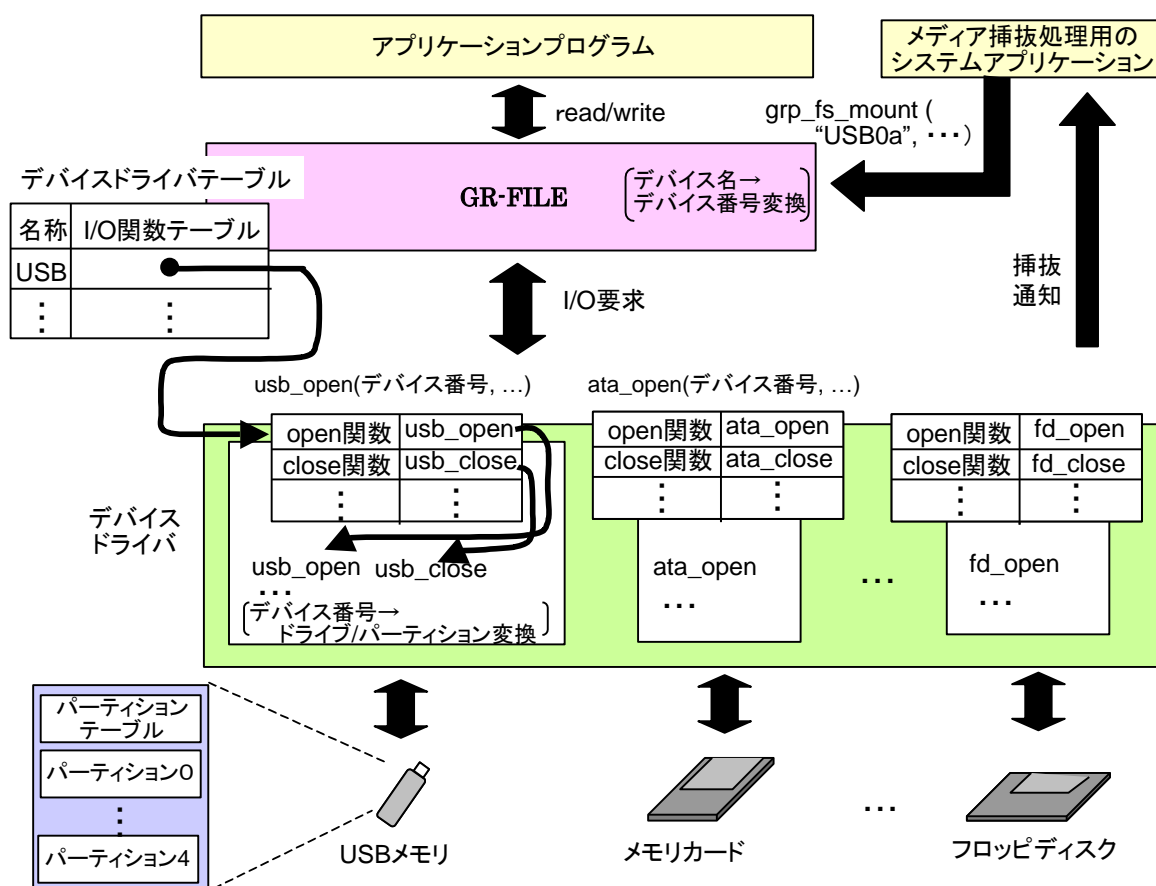


図 3-7 デバイス/ドライブ/パーティションの対応関係

**GR-FILE** のバージョン 1.0X では、明示的なパーティション記述の規定がなく、サブデバイス番号指定に 0~255 の指定を可能とし、その値をどのように実際のサブデバイス番号とパーティション番号として解釈するかは、デバイス依存としていました。しかし、バージョン 1.10 から、上記のとおり、明示的にサブデバイス番号とパーティション番号を分離して記述するように変更しました。

さらに、バージョン 1.10 からは、パーティションの再設定やパーティションレスでのフォーマットを可能とするため、現在メディア上にあるパーティション情報を無視し、パーティションレスの形でメディアをアクセスする手段を提供しています。具体的には、“USB0\*”のように、パーティション番号を表す文字 a~d の代わりに、\*を指定することで行います。

mount 関数等での対象デバイスメディア指定は、上記規則に従ったデバイス名称により行いますが、デバイスドライバに対しては、指定したデバイス名称がデバイス番号に変換されて渡されます。デバイスタイプ名称に対応したデバイステーブルのエントリ番号をメジャーデバイス番号、パーティション番号文字 ‘a’ パーティション番号とした場合、デバイス番号は、以下の値となります。

$$\text{＜メジャーデバイス番号＞} \times 256 + \text{＜パーティション番号＞} \times 16 + \text{＜サブデバイス番号＞}$$

なお、上記パーティションレスアクセスのために、パーティション番号文字として\*を指定した場合は、パーティションレスアクセスのための特別な番号 15 がパーティション番号として使用されます。

上記のとおり、サブデバイス番号を下位 4 ビットに、パーティション番号を下位 8 ビットの上位 4 ビットに持ってくるようにしていますので、パーティション番号文字を省略した名称では、下位 8 ビットをサブデバイス番号とするバージョン 1.0X と互換性があります。従いまして、もともとパーティションレスや 1 パーティションのデバイスでは、1.0X 対応のデバイスドライバをそのまま 1.10 以降のドライバとして使用することも可能です。ただ、1.10 で追加された形式によるサブデバイス、パーティションの扱いや、パーティションの再設定や既存パーティションを無視したパーティションレスでのフォーマット機能を利用する場合は、1.10 のデバイス番号のエンコード規則に従い、サブデバイス番号やパーティション番号を処理するようにすると共に、\*指定によるパーティションレスアクセスに対しては、既存のパーティション情報を無視するようにデバイスドライバを一部修正する必要があります。

なお、デバイスドライバでは、表 3-4 に示すマクロや define を利用して、デバイス番号を操作するようにして下さい。これらのマクロ/define は、“include” 下の “grp\_fs\_dev\_io\_if.h” に記述されています。

表 3-4 デバイス番号操作のためのマクロ/define

#	マクロ/define 名	状態の説明	バージョン 1.0X での有無
1	GRP_FS_DEV_MAJOR(dev)	デバイス番号からメジャーデバイス番号の取得	あり
2	GRP_FS_DEV_MINOR(dev)	デバイス番号から下位 8 ビットのマイナーデバイス番号の取得	あり
3	GRP_FS_DEV_SUBID(dev)	デバイス番号からサブデバイス番号の取得	なし
4	GRP_FS_DEV_PART(dev)	デバイス番号からパーティション番号の取得	なし
5	GRP_FS_DEV_RAW_PART	パーティションレスアクセス用のパーティション番号	なし
6	GRP_FS_DEV_NO (major, minor)	メジャー/マイナー番号からデバイス番号の構築	あり
7	GRP_FS_DEV_MK_MINOR (sub_id, part)	サブデバイス番号、パーティション番号からマイナーデバイス番号の作成	なし
8	GRP_FS_DEV_MATCH_SUB (dev1, dev2)	メジャー/サブデバイス番号の一致判定	なし



### 3.6 ファイルデータとファイル管理情報のキャッシング

**GR-FILE** では、メディアに格納されている、各ファイルのデータや、複数のファイルの管理情報を格納したファイル管理ブロックをメモリ上にキャッシングし、メディアへの I/O 回数を大幅に減らしています。

(図 3-8 参照) しかも、ファイルデータのキャッシュとファイル管理ブロックのキャッシュを分離して管理していますので、動画ファイル等の大容量ファイルをアクセスしても、ファイル管理ブロックキャッシュのデータが追い出されず、高速なファイルアクセスが可能です。

さらに、オープン中のファイルに対しては、上記ファイル管理ブロックキャッシュから、同ファイルのファイル管理情報を取出し、オープンファイルキャッシュ（オープン中のファイル管理情報）として別途保持しています。特に、オープンファイルキャッシュには、現在の **read** 位置から数ブロック分のブロック割り当て情報を保持し、ファイル管理ブロックへのアクセス回数を減らしています。また、新規データ用に割り当て可能なフリーブロックのリストもフリーブロックキャッシュとして保持し、**write** 時のフリーブロックサーチのためファイル管理ブロックへアクセス回数を減らしています。

また、バージョン 1.10 からは、コンパイルオプションで、ファイル名称キャッシュをサポートし、アクセスしたファイルのファイル名称と、オープンファイルキャッシュの対応関係をキャッシュとして保持します。これにより、1 ディレクトリ内に数多くのファイルが存在しても、最近アクセスしたファイルであれば、ディレクトリ内を 1 つずつ順にサーチすることなく、高速なファイルのオープンが可能です。

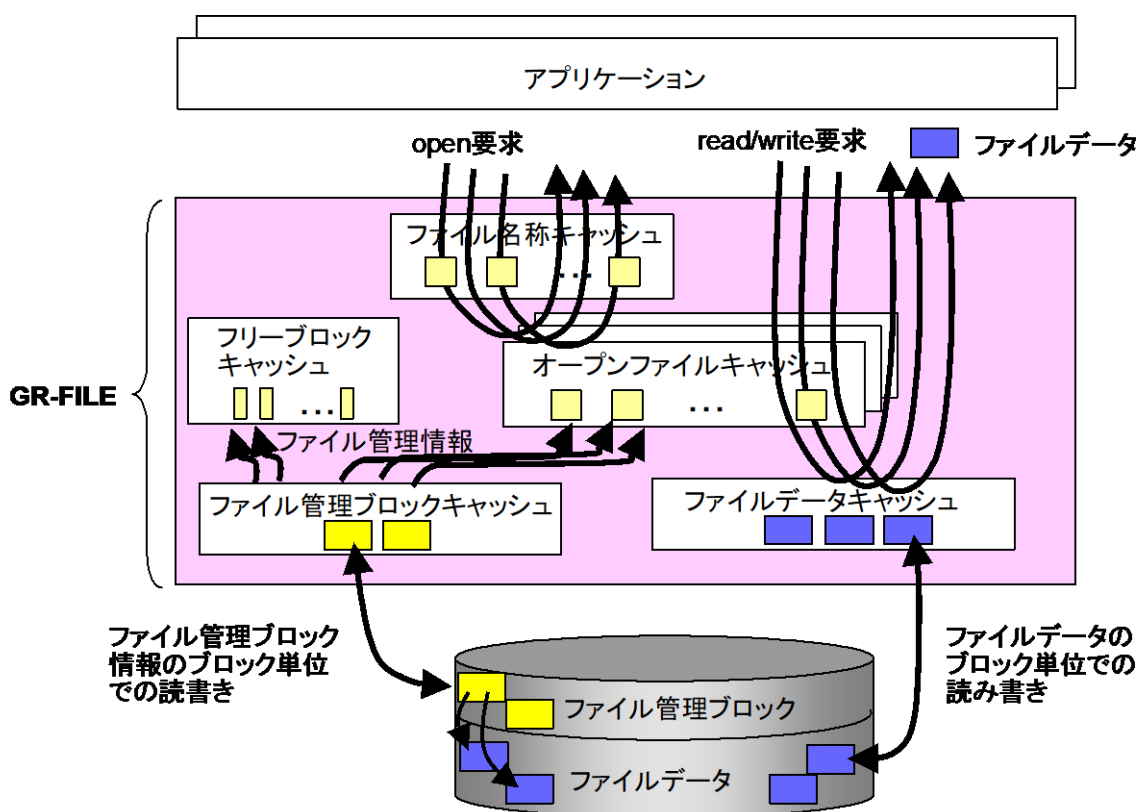


図 3-8 GR-FILE のキャッシング機能



### 3.6.1 ファイルデータキャッシュ

メディアから I/O した各ファイルデータのキャッシュです。各データは、対象のファイルシステムで規定されたファイルシステムのブロックサイズ、または、ファイルデータキャッシュバッファのブロックサイズの小さい方を単位としてキャッシュします。

アプリケーションプログラムから **read** 要求を受けますと、**GR-FILE** は、まず対象のデータブロックがファイルデータキャッシュ内にあるかどうかを調べます。もし、対象のデータブロックがファイルデータキャッシュにない場合は、実際にメディアからそのデータを読み出し、ファイルデータキャッシュに読み込んでからそのデータをアプリケーションプログラムに返します。すでに、ファイルデータキャッシュ内に存在する場合は、メディアからの読み出しを行わず、キャッシュされたデータをアプリケーションプログラムに返します。

アプリケーションプログラムから **write** 要求を受けた場合も、まず対象のデータブロックがファイルデータキャッシュ内にあるかどうか調べます。その結果、キャッシュ内に存在した場合は、要求されたデータを同キャッシュに書込みます。データキャッシュ内に無かった場合は、1 ブロック丸ごと書換えでなければ、メディアから対応するブロックをキャッシュに読み込んでから、要求されたデータを同キャッシュに書込みます。1 ブロック丸ごと書換えの場合は、フリーなキャッシュを割当て、要求されたデータを同キャッシュに書込みます。

**Write** 要求の場合、キャッシュに書込んだ更新結果はメディアに反映する必要があります。しかし、メディアへの書込み回数を減らして性能を上げるため、上記いずれのケースでも、即座に更新結果をメディアに反映するとは限りません。更新をどのようなタイミングでメディアに反映するかは、後述するキャッシュの **write** 制御に従います。

なお、ファイルデータキャッシュの個数は固定ですので、アプリケーションプログラムからの **read** や **write** 要求を処理する際に、フリーなキャッシュを割当てるためには、別のブロック用に割当てられたキャッシュを再利用する必要があります。その際、そのキャッシュがメディアに反映されていない更新情報を保持している場合、同キャッシュの更新をメディアに反映するための書込みが必要となります。このキャッシュのメディアへの反映のための書込みは、キャッシュの再利用を引き起こした **read/write** 要求を出したアプリケーションプログラムの延長で行います。

上記、ファイルデータキャッシュの個数やブロックサイズの最大値は、システム初期化時に変更することが可能です。（「各種パラメータの設定・変更」の節参照）

なお、ファイルデータキャッシュのブロックサイズの最大値は、利用するメディアの物理ブロックサイズ（セクタサイズ）より大きくなければいけません。

### 3.6.2 ファイル管理ブロックキャッシュ

一般にファイルシステムでは、各ファイルの管理情報や各ファイルのデータやフリーブロックがどこに格納されているか等の情報を集めて、ファイル管理ブロックとしてファイルシステムの一部に保持しています。このメディアに格納されたファイル管理ブロックをキャッシュとして保持したものが、ファイルブロックキャッシュです。**GR-FILE** は、アプリケーションプログラムからの read/write 要求に対し、このファイル管理ブロックキャッシュを参照し、対象のファイルデータブロックの場所等を求めて、同要求を処理しています。

各ファイル管理ブロックキャッシュは、固定のブロックサイズを持ち、複数のファイル管理情報や、フリーブロックの情報が含まれています。**GR-FILE** では、この中から、オープン中のファイルの管理情報と、フリーブロックの情報を取出し、さらに、後述するオープンファイルキャッシュや、フリーブロックキャッシュとして保持しています。

なお、ファイル管理ブロックキャッシュも、ファイルデータキャッシュと同様に個数が固定のため、同様のキャッシュの再利用処理が発生します。

ファイル管理ブロックキャッシュの個数とブロックサイズは、システム初期化時に変更することが可能です。（「3.14 各種パラメータの設定・変更、および、コンパイルオプション」の節参照）

### 3.6.3 オープンファイルキャッシュ（オープン中のファイル管理情報）

**GR-FILE** では、オープン中のファイルのアクセスを速くするため、オープン中のファイルに対し、同ファイルの管理情報をオープンファイルキャッシュとして保持しています。このオープンファイルキャッシュには、ファイルのデータブロックがどこに格納されているかの場所情報（ファイルブロックマップ情報）を、現在の read 位置から数ブロック分、保持しています。オープンファイルキャッシュは、上記ファイル管理ブロックキャッシュから読み込み作成します。

なお、オープンファイルキャッシュ自身はファイルシステム非依存ですが、オープンファイルキャッシュにいくつのファイルブロックマップ情報を保持するかは、ファイルシステム依存です。**FAT** ファイルシステムの場合は、“grp\_fat\_param.h” に各ファイルのファイルブロックマップ情報の個数が “FAT\_MAP\_CNT” という define で定義されています。ファイルブロックマップ情報の個数を変更するには、同 define を変更し、再コンパイルする必要があります。

### 3.6.4 フリーブロックキャッシュ

**GR-FILE** では、write 時のフリーブロックの割当てを速くするため、ファイルシステムのフリーブロックのリストをフリーブロックキャッシュとして保持しています。

このフリーブロックキャッシュは、上記ファイル管理ブロックキャッシュから読み込み作成します。

ファイルフリーブロックキャッシュは、ファイルシステムに依存した形で管理しています、**FAT** ファイルシステムの場合は、“grp\_fat\_param.h” にフリーブロックキャッシュの個数が “FAT\_FREE\_TBL” という define で定義されています。フリーブロックキャッシュの個数を変更するには、同 define を変更し、再コンパイルする必要があります。

### 3.6.5 ファイル名称キャッシュ

**GR-FILE** では、バージョン 1.10 より、ファイルオープン処理を高速化のため、コンパイルオプションにてファイル名称キャッシュ機能をサポートし、オープンしたファイル名称とオープンファイルキャッシュの対応関係をキャッシュとして保持します。

通常、ファイルのオープン時には、対象ディレクトリ内にある各ファイルと 1 つずつ名称を比較し、オープン対象のファイルを見つける必要があります。従って、ディレクトリ内に多数のファイルが存在する場合、メディアからディレクトリデータを読み込みやファイル名の比較処理が多数発生し、処理に時間を要する結果となります。また、ファイル名指定が、ルートディレクトリからの絶対パス指定の場合、ファイル名パス内の各ディレクトリに対して、ファイル名検索が必要なため、同様の問題が発生します。そこで、ファイル名称キャッシュ機能を提供することで、前にアクセスしたことがあるファイルで、まだ、キャッシュ上に残っているファイルに対しては、ディレクトリ内のファイル名の逐次検索を省略させ、高速なオープンを可能とします。

ただ、ディレクトリの逐次検索の前に、常にファイル名称キャッシュの検索処理を行う形となりますので、アクセスしたことのないファイル等をオープンする際には、逆に多少遅くなります。そこで、本ファイル名称キャッシュは、コンパイルオプションとしてサポートします。本ファイル名キャッシュ機能を使用する場合は、コンパイルオプション **GRP\_FS\_FNAME\_CACHE** を指定して **GR-FILE** ライブラリを構築して下さい。また、**GRP\_FS\_FAT\_CACHE\_BY\_GET\_DIRENT** オプションを指定しますと、ディレクトリエントリ情報の取得関数 `grp_fs_get_dirent` で取得した最後のディレクトリエントリ情報も、ファイル名称キャッシュとして保持します。

コンパイルオプション **GRP\_FS\_FNAME\_CACHE** を指定して **GR-FILE** ライブラリを構築した場合、ファイル名称キャッシュの個数とそのハッシュバケット数を、システム初期化時に変更することが可能です。（「3.14 各種パラメータの設定・変更、および、コンパイルオプション」の節参照）

### 3.7 キャッシュの write 制御

前節で説明しましたように、ファイル I/O を高速化するため、**GR-FILE** では、ファイルデータやファイル管理ブロックをメモリ上にキャッシングしています。**GR-FILE** では、キャッシュで性能を高め、一方でキャッシュとメディア上のデータの不整合による問題の発生を抑えるため、これらのキャッシュにキャッシュされた最新データのメディアへの反映のタイミングを、メディアの特性に応じて選択を可能としています。具体的には、以下の 4 つの方式をメディアの特性や処理内容等に応じて `mount` 処理時に、`grp_fs_mount` のモードパラメータにより選択できます。

- (1) `write through` 方式 (各 `write` 要求時方式)
- (2) `each close` 方式 (ファイルクローズ時方式)
- (3) `last close` 方式 (ファイルの最終クローズ時方式)
- (4) `unmount` 方式 (`unmount` 時方式)

なお、上記方式で選択されたタイミング以外でも、アプリケーションプログラムやデーモンプログラムで `sync` 関数を用いることで、任意のタイミングでキャッシュをメディアに反映することが可能です。

#### 3.7.1 write through 方式

アプリケーションプログラムからの更新要求時に、更新されたキャッシュブロックを、同アプリケーションタスクの延長で即座にメディアに反映します。本方式では、図 3-9 に示しますように、アプリケーションプログラムが更新要求を出しますと、要求のあったデータを一旦キャッシュに書き込み、更新されたキャッシュをブロック単位でメディアに書き込んで、書き込みの終了を待つて、アプリケーションプログラムに制御を返します。

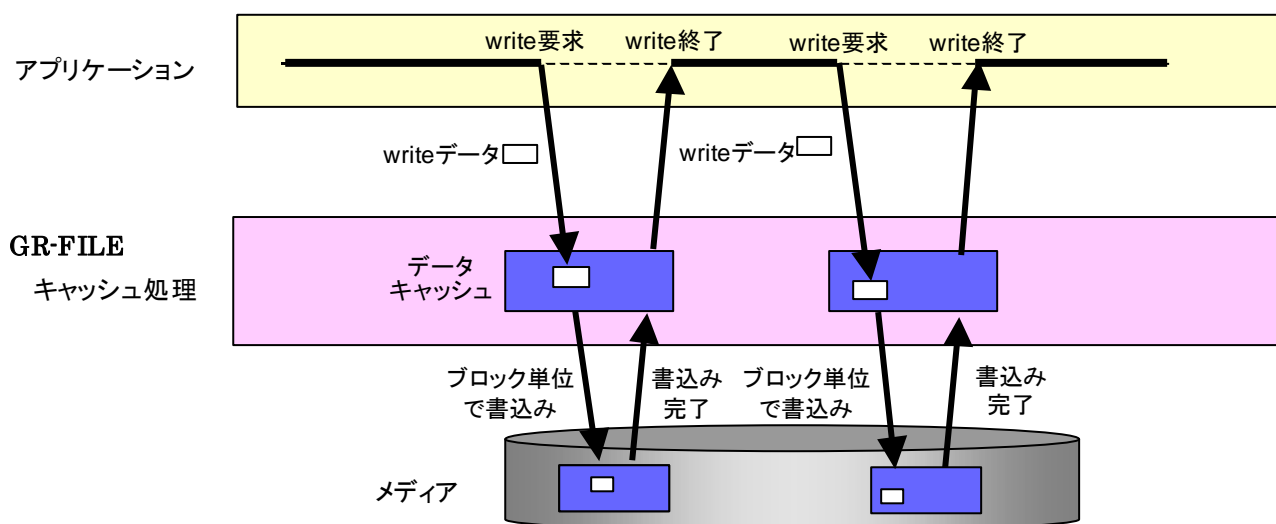


図 3-9 write through 方式

本方式では、アプリケーションプログラムからの更新要求毎に、メディアへの反映が行われるため、キャッシュとメディアの不整合を極力防ぐことができますが、I/O が頻繁に発生し、性能が大幅に低下します。従いまして、本方式は、更新回数が少なく、整合性を確実に維持する必要がある場合に使用します。

### 3.7.2 each close 方式

各アプリケーションプログラムがファイルをクローズした際に、未反映のキャッシュの内容をメディアに反映します。本方式では、図 3-10 に示しますように、ファイルがオープンされた後、同ファイルがクローズされるまでに発生したアプリケーションプログラムからの更新要求に対して、キャッシュに余裕がある限り、キャッシュ上のデータのみを更新し、メディアへの反映は、close されるまで遅延させます。

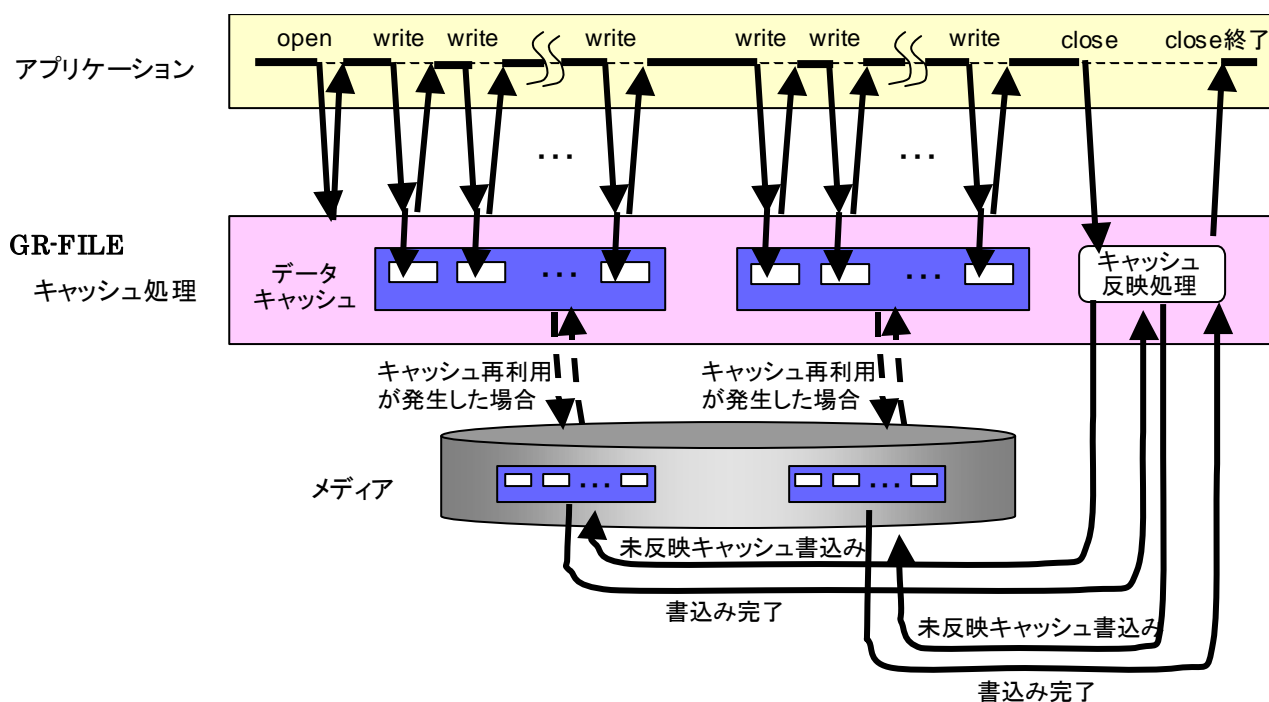


図 3-10 each close 方式

本方式は、各ファイルの `close` 時に `sync` をコールしたのと同等の効果を持ち、各アプリケーションの処理中はキャッシュを利用して高速な I/O を可能とし、処理の完了に連動してメディアとの整合性を取る方式です。従いまして、メモ리카ード等、取外される可能性のあるメディアで、処理中は、アプリケーションプログラムが利用者に I/O 中であることを何らかの形で示し、利用者が処理中には、取外す可能性が低い場合に最適な方式です。

### 3.7.3 last close 方式

システム全体でオープンされているファイルが無くなった時点で、未反映のキャッシュの内容をメディアに反映します。本方式では、図 3-11 に示しますように、システム全体で、ファイルがオープンされた後、全てのファイルがクローズされるまでに発生したアプリケーションプログラムからの更新要求に対して、キャッシュに余裕がある限り、キャッシュ上のデータのみを更新し、メディアへの反映は、全てのファイルが close されるまで遅延させます。

なお、カレントディレクトリも、オープンされたファイルの 1 つとして解釈し、カレントディレクトリがそのメディアに設定されている間は、最終 close とは、解釈されません。逆に、カレントディレクトリを無効化した時点で、同メディアに対して他のファイルがオープンされていない場合は、同カレントディレクトリの無効化を最終 close として、メディアへの反映処理が行われます。

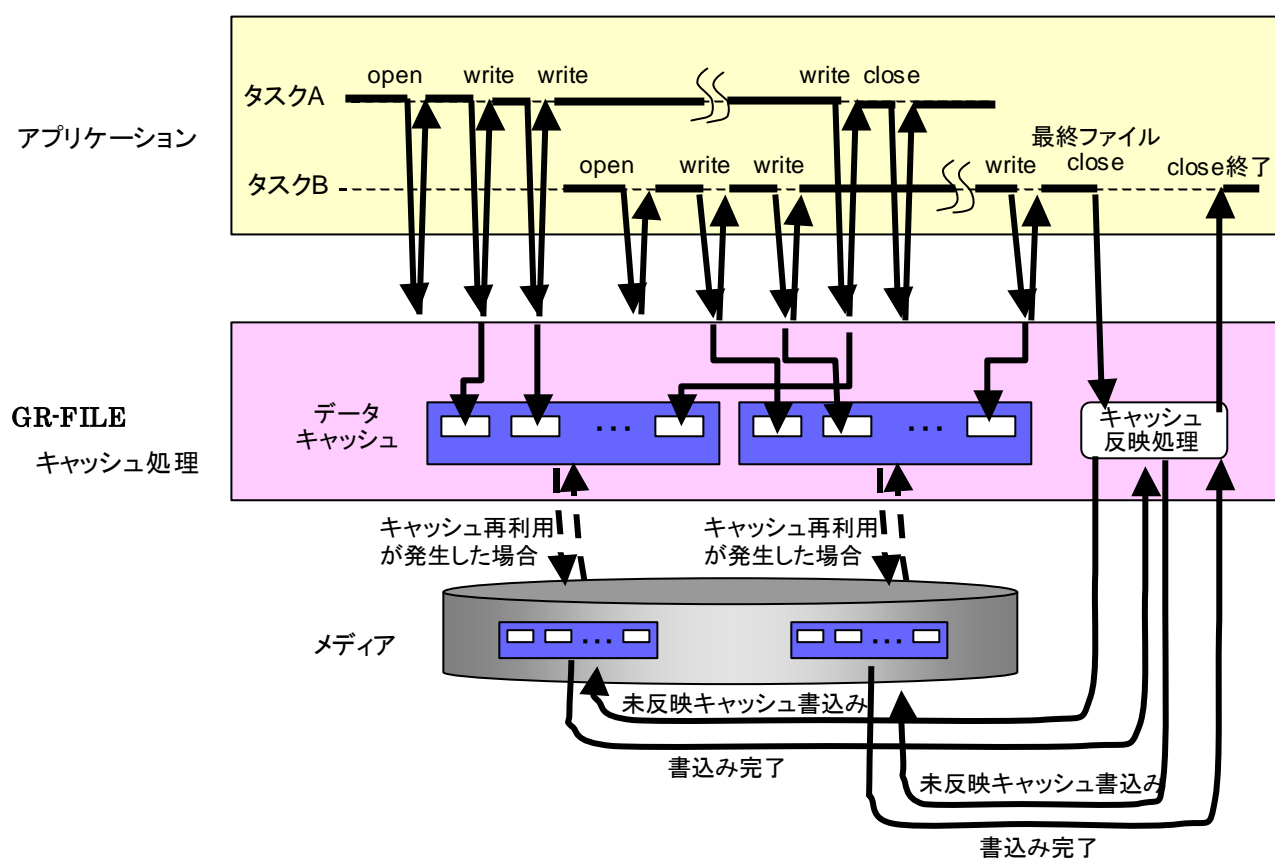


図 3-11 last close 方式

本方式は、システム全体での最後のファイルの close 時に sync をコールしたのと同等の効果を持ち、各アプリケーションの処理中はキャッシュを利用して高速な I/O を可能とし、すべての処理の完了に連動してメディアとの整合性を取る方式です。従いまして、each close 方式と同様に、メモ리카ード等、取外される可能性のあるメディアで、処理中は、アプリケーションプログラムが利用者に I/O 中であることを何らかの形で示し、利用者が処理中には、取外す可能性が低い場合に最適な方式です。Each close 方式との違いは、各ファイル毎ではなく、システム全体での最終 close ということですので、本方式は、1 つ、または、複数のタスクで複数のファイルを同時にオープンして処理する場合に適した方式です。

### 3.7.4 unmount 方式

可能な限りキャッシュ内で処理し、メディアの unmount 処理に未反映のキャッシュの内容をメディアに反映します。本方式では、図 3-12 に示しますように、メディアに対して mount 処理を行った以降、同メディアの unmount 処理を行うまでに発生したアプリケーションプログラムからの更新要求に対して、キャッシュに余裕がある限り、キャッシュ上のデータのみを更新し、メディアへの反映は、unmount 処理が行われるまで可能な限り遅延させます。

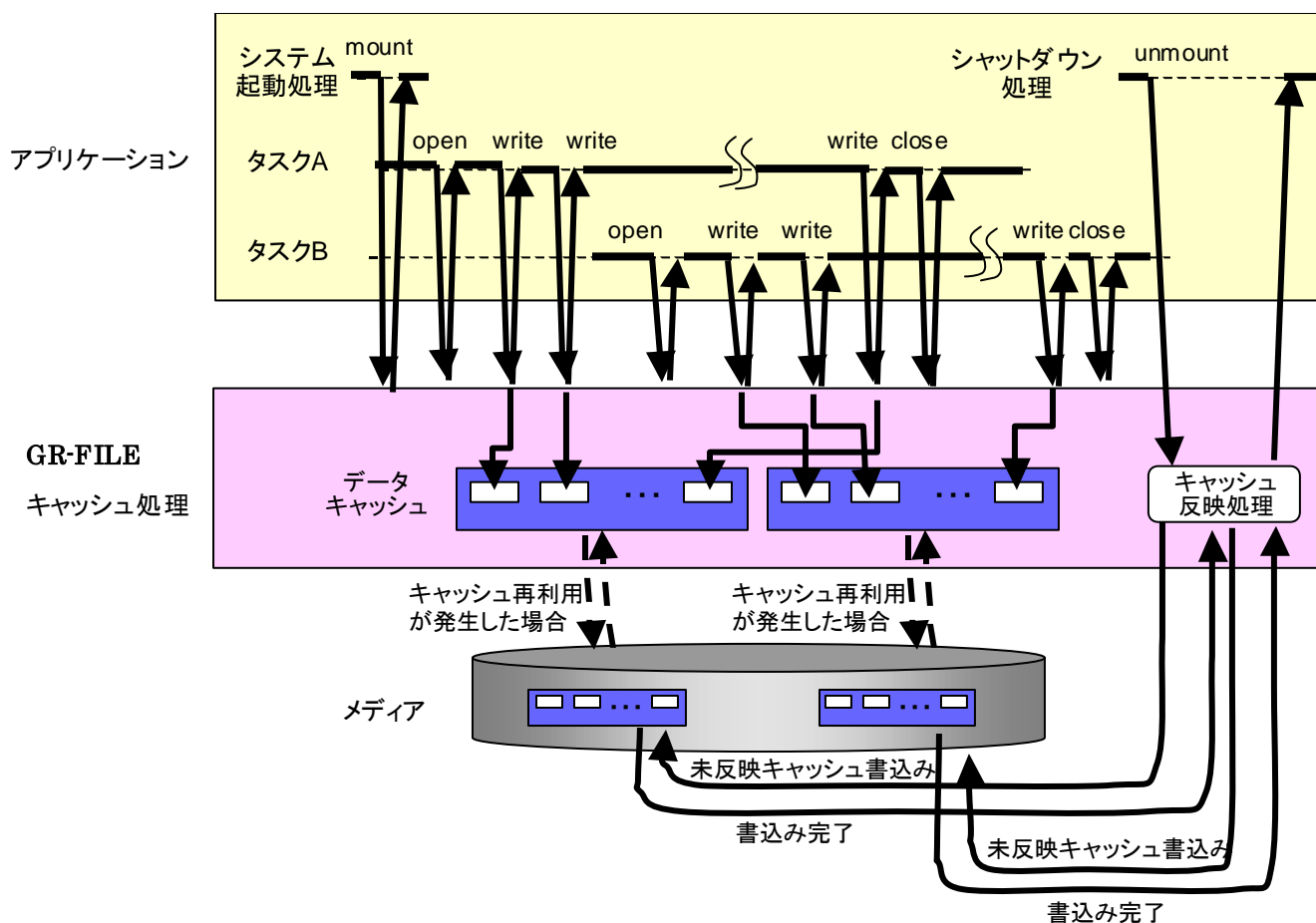


図 3-12 unmount 方式

本方式は、可能な限りキャッシュで処理し、性能を追求する方式です。一方、unmount 以前にメディアの取外しや、システムダウンが発生しますと、メディア上のデータがファイルシステムとして不整合な状態に陥る可能性があります。

従いまして、本方式は、不意のメディアの取外しが発生しない、内蔵のハードディスクや処理中は取外しを物理的にロックできるメディアに対して使用するのに適した方式です。但し、システムダウン等で不整合が発生する可能性が高くなりますので、デーモンプログラム等で sync を定期的にコールし、不整合となる可能性を低くすることが必要です。



### 3.8 アプリケーションバッファとメディア間の直接/連続ブロック I/O（ダイレクト I/O 機能）

上記で説明しました **GR-FILE** のキャッシング機能は、ファイル管理ブロック情報やディレクトリ情報、そして、頻繁に read するファイル等に対する I/O については、メディアへの I/O 回数を削減でき、キャッシング機能なしに比べ、大幅な性能向上が期待できます。しかし、写真データ等、一回で読書きするバイト数が大きく、しかも、たまにしか read しないようなファイルに対しては、上記キャッシング機能は、キャッシュバッファにデータを一旦コピーするオーバーヘッドがあり、また、メディアとキャッシュバッファとの I/O がキャッシュブロック単位となるため、I/O 回数が増え、かえって遅くなるケースもあります。

そこで、**GR-FILE** では、ファイルオープン時のオプション指定により、同ファイルに対する I/O 要求に対しては、キャッシュ上に対応するデータがなければ、キャッシュバッファを可能な限り使用せず、アプリケーションの指定したバッファとメディアとの間で直接 I/O を行い、しかもメディア上で連続したブロックは一括して I/O を行なう「ダイレクト I/O 機能」をサポートしています。

具体的には、「ダイレクト I/O 機能」を使用する場合、open または grp\_fs\_open 時のオープンモードパラメータとして、GRP\_FS\_O\_DIRECT\_IO を指定します。

図 3-1 3 は、GRP\_FS\_O\_DIRECT\_IO を指定した場合の **GR-FILE** の FAT ファイルシステムでの read 処理のイメージを示します。この例では、キャッシュブロックのサイズが 512 バイトで、6000 バイトからなるファイルの先頭 5000 バイトを read する例を示しています。また、同ファイルは、先頭の 4 ブロックと、後の 8 ブロックがそれぞれメディア上で連続に割当てられおり、先頭から 6 ブロック目のデータがキャッシュとして残っている状態であるとします。

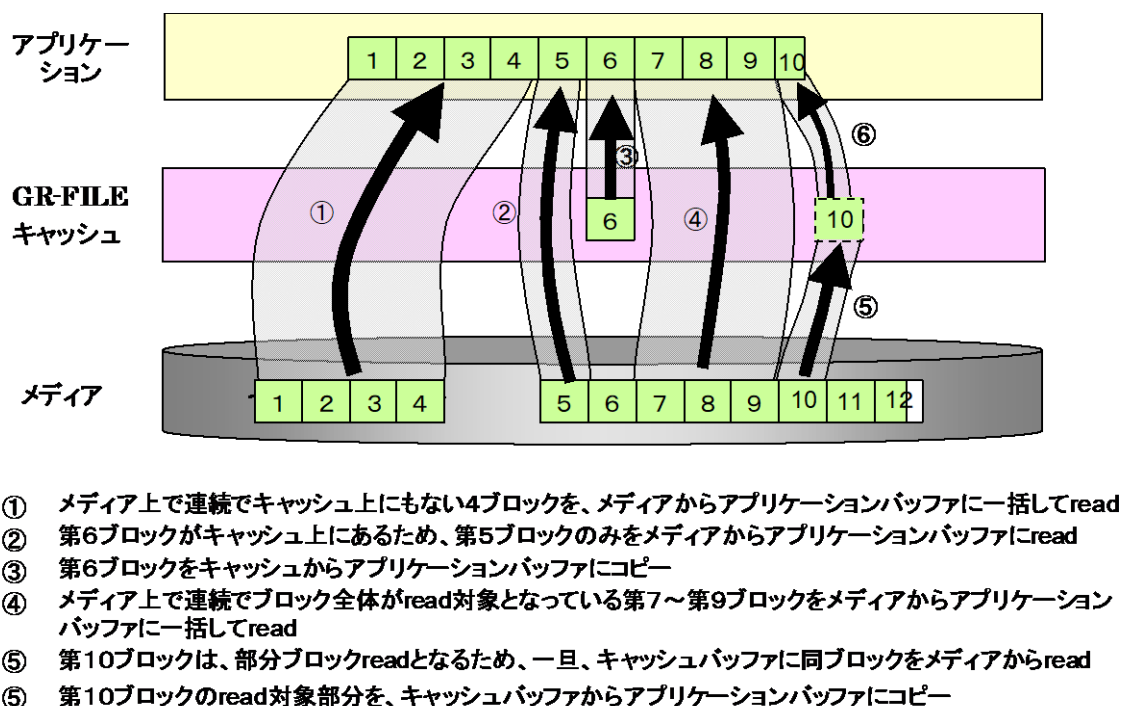


図 3-1 3 ダイレクト I/O 機能の read 処理のイメージ



また、図 3-1 4 に、上記と同一のファイルに対する先頭 5000 バイトの write 要求時の処理イメージを示します。この例でも第 6 ブロックだけがキャッシュ上にあるとします。

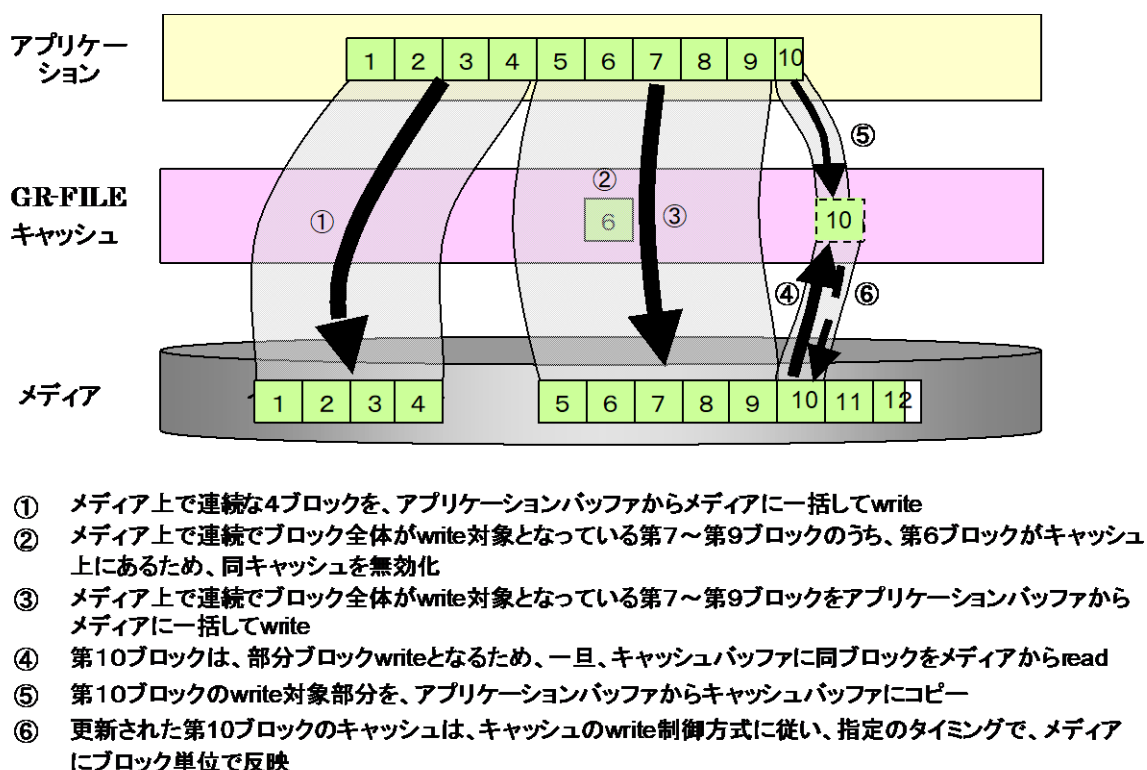


図 3-1 4 ダイレクト I/O 機能の write 処理のイメージ

以上に示しましたように、ダイレクト I/O 機能を用いますと、キャッシュバッファへのコピーを省略して、アプリケーションバッファとメディアとの間で、直接、かつ、メディア上で連続したブロックを一括して I/O することができます。

なお、ダイレクト I/O 機能は、オプション機能で、ファイルシステム依存部がサポートしている場合のみ、GRP\_FS\_O\_DIRECT\_IO 指定が有効です。**GR-FILE** の FAT ファイルシステムの場合は、シングル空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合に、GRP\_FS\_O\_DIRECT\_IO 指定が有効となります。

#### ※ 直接/連続ブロック I/O (ダイレクト I/O 機能) 使用時の注意点

デバイスドライバへ転送要求を行う際に、バッファのアドレスを指定しますが、デバイスドライバや、使用する機器によっては奇数アドレスが使用できない場合があります。

1つのファイルに対し、奇数バイトの転送を行った後のダイレクト I/O では、キャッシュバッファとアプリケーションバッファの両方を使用する為、奇数アドレスに対する転送要求が発生する事があります。

このような場合は、奇数バイトのアクセスを行わなくするか、ダイレクト I/O を使用しないで下さい。

### 3.9 ファイルシステム依存部の分離

**GR-FILE** では、アプリケーションインタフェース処理やキャッシング処理等のファイルシステムに共通した処理と、実際のファイルシステムの構造に依存したファイルシステム依存処理部を分離した構造になっています。具体的は、図 3-15 に示しますように、ファイルシステム共通部とファイルシステム依存部の間のファイルシステム抽象化インタフェースを規定し、規定された各ファイルシステムのインタフェース関数をファイルシステムテーブルに登録することで、ファイルシステム共通部から各ファイルシステムに依存した処理の実行を可能としています。

各メディアをどのタイプのファイルシステムとしてアクセスするかは、メディアの **mount** 処理時に、ファイルシステムテーブルに登録されたファイルシステムタイプ名称を **grp\_fs\_mount** 関数のパラメータとして指定することで行います。デフォルトでは、**FAT** ファイルシステム用の処理関数が、“fat” というファイルシステムタイプ名称で登録されています。

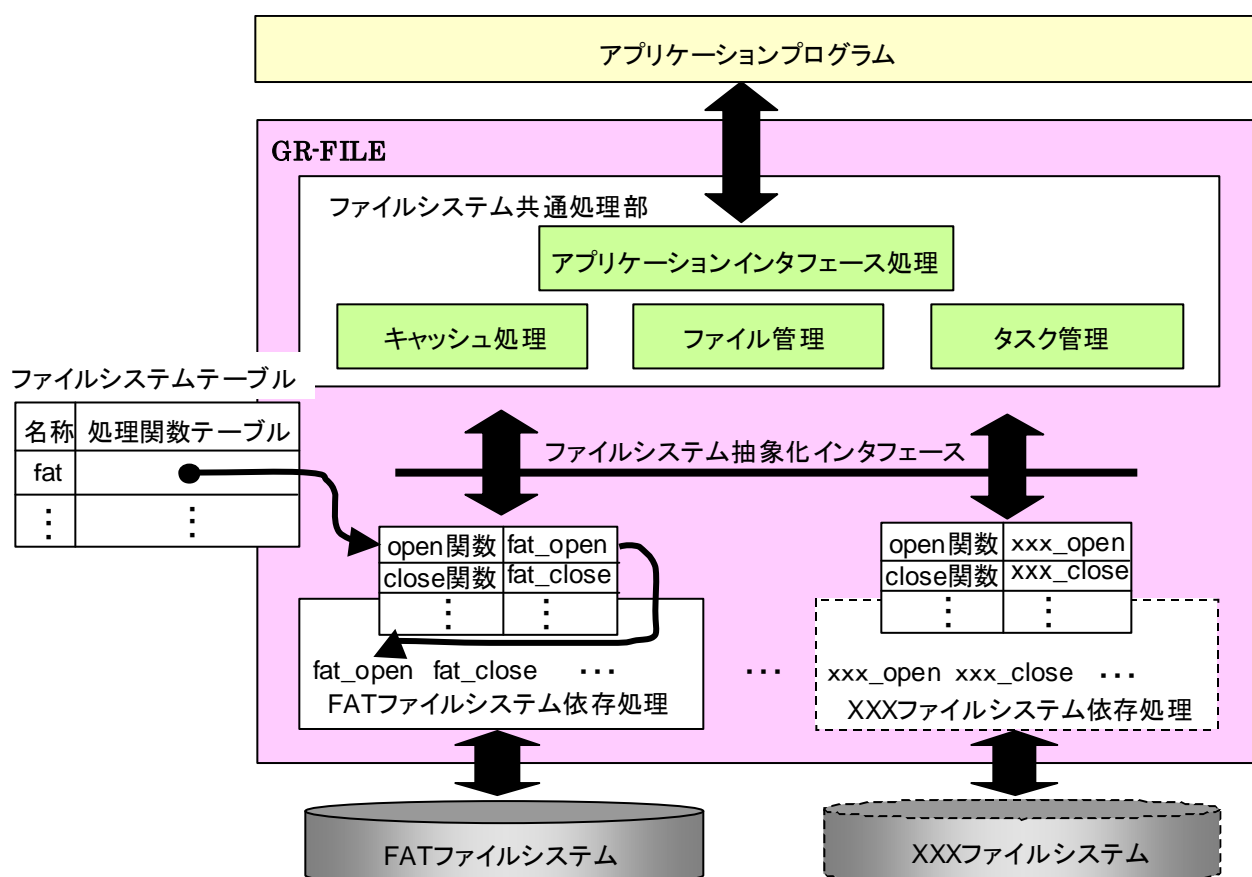


図 3-15 ファイル依存処理部の分離

これにより、様々なファイルシステムに対する I/O を、共通のアプリケーションインタフェースで実現すること可能とすると共に、新たなファイルシステムをサポートするのに必要なプログラムの作成量を削減し、新たなファイルシステムのサポートを容易にします。

ファイルシステム抽象化インタフェースの詳細については、「4 インタフェース」の章を参照下さい。

### 3.10 OS 依存処理部の分離

**GR-FILE** では、OS やプラットフォーム依存処理部を分離した構造になっています。具体的は、図 3-16 に示しますように、排他処理、タスク情報の取得処理、メモリ管理、タスク空間のコピー処理、時間処理、文字コード処理等の OS やプラットフォームに依存する処理のインタフェースを OS 抽象化インタフェースとして規定し、同インタフェースを使って、OS 非依存処理部から、OS 依存処理部をコールする形になっています。

なお、現行バージョンでは、VOS、および、 $\mu$ ITRON 準拠システムの、OS 依存処理部のサンプルコードは提供可能です。

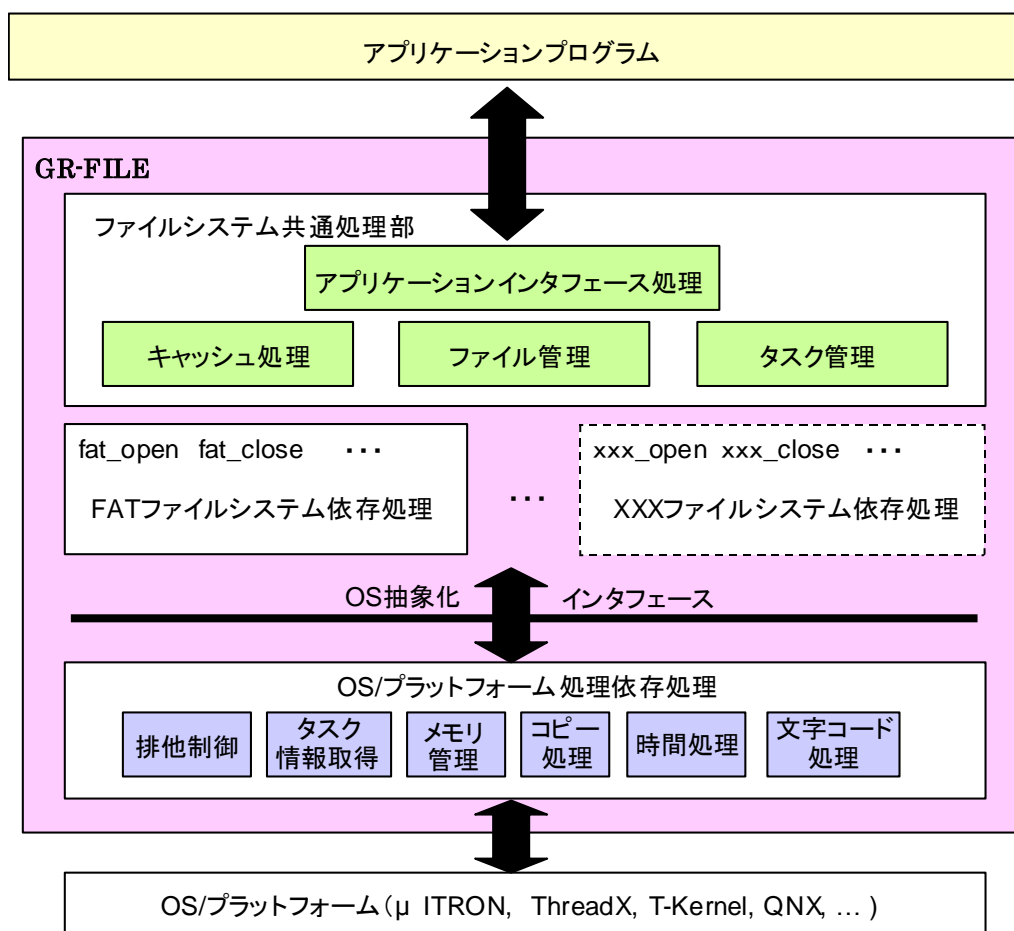


図 3-16 OS 依存処理部の分離

これにより、様々な OS、プラットフォームのサポートを容易にしています。

OS 抽象化インタフェースの詳細については、「4 インタフェース」の章を参照下さい。

### 3.1.1 メディアの挿抜対応

**GR-FILE** では、メモリカード、USB メモリ等、メディアの挿抜に対応した機能を提供しています。

#### 3.1.1.1 メディアの挿抜処理の概要

メディアの挿抜処理は、図 3-17 に示しますように、デバイスドライバがメディアの挿抜検出機能を持ち、同デバイスドライバからの挿抜通知を受けて、挿抜処理用のシステムアプリケーションプログラムが **GR-FILE** が提供する機能を利用して実現します。デバイスドライバの挿抜検出機能は、割込み等によりメディアの挿抜を検出し、メディアの挿抜時に、メディアの挿抜を挿抜処理用のシステムアプリケーションに通知します。挿抜処理用のシステムアプリケーションは、デバイスドライバからのメディアの挿抜通知や利用者からの要求を受け、利用者への通知/応答等、利用者とのインタラクションを取りながら、**GR-FILE** が提供する機能を利用してメディアの挿抜処理を行います。

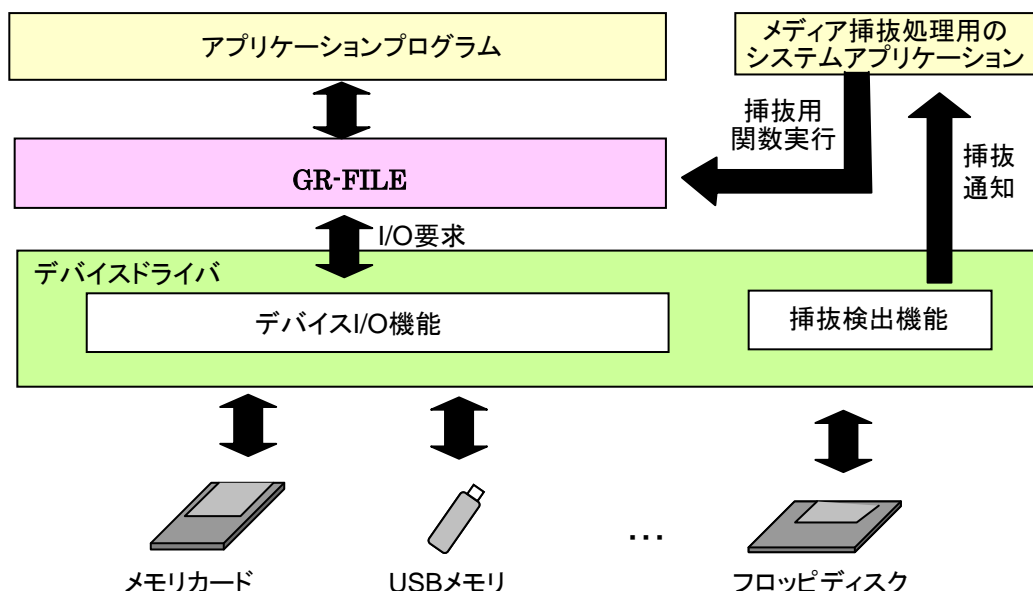


図 3-17 メディアの挿抜処理方法

図 3-18 に、挿抜処理用のシステムアプリケーションにおける、挿抜処理の状態遷移と処理の流れを示します。**GR-FILE** は、この挿抜処理用のシステムアプリケーションが行うこれらの処理をサポートするための関数群を提供しています。図 3-18 の四角に囲まれた番号は、処理状態を示し、表 3-5 に、各状態の概要を示します。丸に囲まれた番号は処理を示し、表 3-6 に各処理の概要を示します。

なお、**GR-FILE** が提供するメディアの挿抜処理向けの関数群は、すべて、タスク環境下で実行されることを前提としており、割込みの延長では使用できません。従いまして、メディアの挿抜処理は、図 3-17 に示しましたように、割込みの延長ではなく、アプリケーションタスクで行う必要があります。

メディアの挿抜処理の概略フローについては、「5 サンプルフロー」を参照下さい。

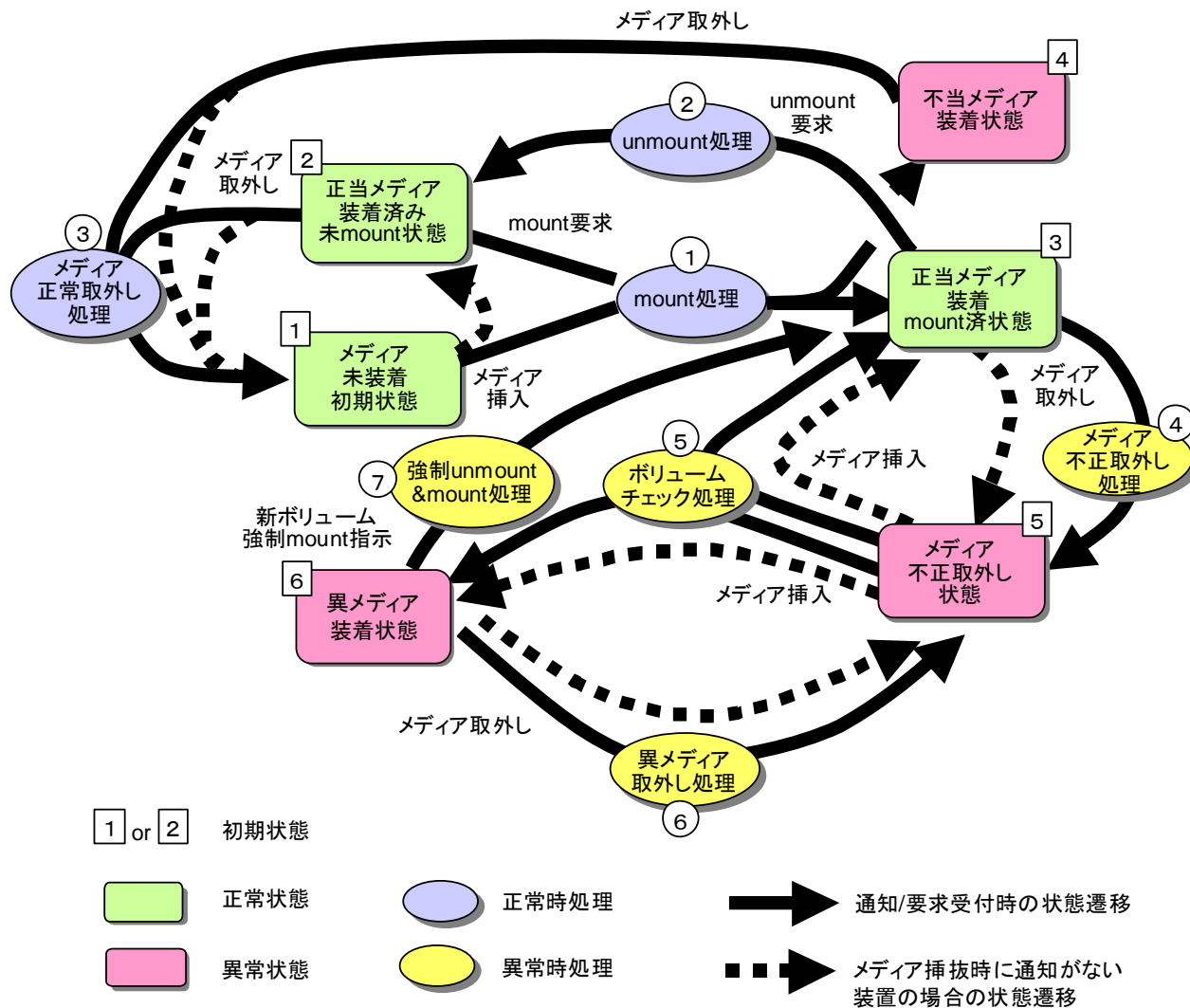


図 3-18 メディアの挿抜処理の流れ

表 3-5 メディアの挿抜状態

#	状態名	状態の説明
1	メディア未装着初期状態	・メモ리카ード等のメディアが装置に装着されておらず、以前、メディアが装着されていた場合も、同メディアが正常に unmount された後に取外れた状態
2	正当メディア装着済み未 mount 状態	・固定ハードディスク等の常設のメディアで mount 処理がされていない状態 ・フロッピー等、メディアの挿入が検出できない装置で、メディアが挿入されているが、mount 処理がなされていない状態
3	正当メディア装着 mount 済状態	・整合性のあるファイルシステムが入ったメディアが挿入/装着され、mount 処理されて、ファイルシステムとしてアクセス可能となった状態
4	不当メディア装着状態	・ファイルシステムとして正しくないメディアが装着され、あるいは、mount 処理中に I/O エラーが発生し、mount 処理が失敗した状態
5	メディア不正取外し状態	・正常に mount されたメディアを unmount せずに取外した状態
6	異メディア装着状態	・正常に mount されたメディアを unmount せずに取外し、取外したメディアと異なるメディアを挿入した状態

表 3-6 メディアの各挿抜処理の概要

#	処理名	処理の内容
1	mount 処理	<ul style="list-style-type: none"> <li>メディアの挿入通知、あるいは、利用者/システム起動時の mount 要求を受け、挿入/装着されたメディアをファイルシステムとしてアクセス可能とする。</li> <li>mount 関数 <code>grp_fs_mount</code> を使用して実現する。</li> <li>以前 unmount 処理を行わずにメディアが取外され、同メディアに反映できず、退避しておいたキャッシュセーブがある場合は、mount 前に、<code>grp_fs_check_volume</code> をコールし、取外された同メディアかどうかをチェックする。同メディアの場合は、退避してあったキャッシュデータをデバイスドライバの I/O 関数を使い、メディアに反映してから mount する。</li> </ul>
2	unmount 処理	<ul style="list-style-type: none"> <li>利用者、あるいは、システム停止時の unmount 要求をうけ、キャッシュをメディアに反映し、メディアを取外し可能な状態とする。</li> <li>unmount 関数 <code>grp_fs_unmount</code> を使用して実現する。</li> <li>なお、ソフト的なメディア取出しボタンを持ち、同ボタンの押下により、メディア挿抜処理用のシステムアプリケーションが通知を受け、その通知に対する処理として、unmount を実行する形もある。この場合、unmount 処理により、同システムアプリケーションがハード的なメディアの取出しを実行して、「メディア未装着初期状態」に自動的に遷移する。</li> </ul>
3	メディア 正常取外し処理	<ul style="list-style-type: none"> <li>正常に ummount されたメディアの取外し通知に対する処理。</li> <li>正常に unmount されているので、本処理では、特に処理は行わない。</li> </ul>
4	メディア 不正取外し処理	<ul style="list-style-type: none"> <li>unmount されずにデバイスからメディアが取外された場合の処理。</li> <li>利用者に、不正な取外しであることを警告し、再挿入を促す。</li> <li>さらに、<code>grp_fs_invalidate_fs_dev</code> 関数をコールし、以降の同デバイスに対する I/O を無効化し、すべてエラーで返すように設定する。</li> </ul>
5	ボリューム チェック処理	<ul style="list-style-type: none"> <li>unmount されずにメディアが取外された状態で、再度メディアが挿入された際に行う処理。</li> <li><code>grp_fs_check_fs_dev</code> 関数をコールし、前回 mount されていたメディアかどうかをチェックする。前回、mount されていたメディアの場合は、#4 で行った同デバイスに対する I/O 無効化を解除し、ファイルアクセスを続行可能とする。</li> <li>違うメディアの場合は、メディアが異なることを利用者に通知する。</li> </ul>
6	異メディア 取外し処理	<ul style="list-style-type: none"> <li>unmount されずにメディアが取外され、再度メディアが挿入されたメディアが異なっていた場合で、同メディアの取外し通知を受けた際の処理。</li> <li>#4 で同デバイスに対する I/O 無効化処理を行っているので、本処理では、単に通知を無視し、特別な処理を行わない。</li> </ul>
7	強制 unmount 処理	<ul style="list-style-type: none"> <li>unmount されずにメディアが取外され、再度挿入されたメディアが異なっていた場合で、利用者から要求で、前メディアを強制的に unmount し、現在のメディアのファイルシステムに対するアクセスを可能とする場合の処理。</li> <li><code>grp_fs_get_error</code> 関数を <code>GRP_FS_GE_DIRTY   GRP_FS_GE_RELEASE</code> オプションでコールし、未反映の前メディアのキャッシュを退避する。</li> <li>さらに、強制 unmount オプションで unmount 処理を実行し、新メディアを mount 処理可能な状態にする。</li> <li>最後に、mount 処理を実行して、挿入されたメディアをアクセス可能な状態にする。</li> <li>なお、<code>grp_fs_get_error</code> により退避した未反映のキャッシュデータは、前メディアの mount 処理にて、書戻しを行う。</li> </ul>



### 3.1 1.2 メディアの正常挿抜対応

上記概要で示しましたとおり、メディアの挿入により、同メディアのファイルシステムのアクセスを可能とし、メディアの取外し前にキャッシュされた更新結果をメディアに反映するために、**GR-FILE** では、`mount/unmount` 関数（正式名称は `grp_fs_mount/grp_fs_unmount`）を提供しています。

`mount` 関数 `grp_fs_mount` は、メディアの挿入通知、あるいは、システム起動時等を契機としてコールし、メディアに格納されたファイルシステムのアクセスを可能とする関数です。`mount` 関数では、ファイルシステムのタイプを指定したり、メディアの特性に応じて、キャッシュの反映タイミングを指定することが可能です。なお、**FAT** ファイルシステム以外のファイルシステムタイプを指定するには、同ファイルシステムに依存した処理関数を **GR-FILE** に組み込む必要があります。

`unmount` 関数 `grp_fs_unmount` は、メディアを取外す前に必ず実行しなければならない関数で、同関数を実行することで、未反映の更新結果のキャッシュデータをメディアに反映し、メディアを取外し可能な状態とします。

### 3.1 1.3 メディアの異常挿抜対応

正常に `mount` 処理をしたメディアを `unmount` 処理をせずに取外した場合の処理を支援する機能として、以下の機能を提供しています。

#### （1）I/O 抑止機能

`unmount` 処理をせずにメディアを取外し、同メディアとは違うメディアを間違って挿入しますと、間違って挿入したメディアにデータを書込み、挿入したメディアのデータを壊してしまう可能性があります。そこで、**GR-FILE** では、この問題を防止するため、特定のデバイスに対する I/O を抑止する関数 `grp_fs_invalidate_fs_dev` を提供しています。本関数を実行しますと、同デバイスに対する以降のファイル I/O 要求をエラーで返します。但し、キャッシュ上のあるデータに対する `read` は、エラーなしで継続して実行可能です。なお、I/O エラー時のプラットフォーム依存フック関数変数 `grp_fs_inform_io_err` を設定することで、I/O 要求に対しエラーで返すのではなく、ユーザに通知し、メディアの挿入を待ってから同 I/O 要求を再実行することも可能です。

本関数は、上記概要で示しましたとおり、`unmount` 処理されず、メディアが取外された際に実行します。本関数により、設定された I/O 抑止設定は、不意に取外されたメディアが再度挿入された際に、`grp_fs_check_fs_dev` を実行することで、リセットされます。逆に、取外されたメディアと同じボリューム名、シリアル番号を持つメディアが再挿入され、かつ、`grp_fs_check_fs_dev` により同じメディアであることが確認されるまで、`grp_fs_invalidate_fs_dev` により設定された I/O 抑止設定は継続します。

#### （2）ボリューム名チェック機能

`unmount` 処理をせずにメディアが取外された場合、再度、メディアが挿入された際に、同じメディアが挿入されたかどうかをチェックし、同じメディアの場合にファイルアクセスを再開する必要があります。そこで、**GR-FILE** では、そのための関数 `grp_fs_check_fs_dev` を提供しています。

本関数は、メディアが再挿入された際に実行します。本関数を実行しますと、同メディアのボリューム名とシリアル番号をチェックし、`unmount` されずに取外されたメディアと同じかどうかを比較します。比較の結果、同じであることが判明した場合は、メディア取外された際に実行した `grp_fs_invalidate_fs_dev` により設定された I/O 抑止設定をリセットして、同メディアに対するファイルアクセスを再開し可能とし



ます。

また、後述するメディアへの反映ができなくなったキャッシュデータの読出し機能により退避したキャッシュデータを `mount` 時に書戻す際、退避したメディアかどうかを判定するためのボリューム名チェック関数 `grp_fs_check_volume` も提供しています。

### (3) 反映不能となったキャッシュデータの読出し機能

`unmount` 処理をせずにメディアが取外され、警告表示にも関わらず、同メディアが再挿入されず、更新結果がメディアに反映できない場合が考えられます。また、メディアの不良等で、一部データの書込みがどうしてもできないケースも考えられます。このような場合、反映できないデータがキャッシュ上に残り、新たなメディアの挿入ができなくなります。

そこで、**GR-FILE** では、反映不能となったキャッシュデータを読出し、退避するための関数 `grp_fs_get_error` を提供しています。本関数を実行しますと、反映不能となったキャッシュデータのメディア上の位置情報とデータとをペアにして読出し、同キャッシュ領域を解放することができます。`write` エラーとなったキャッシュデータのみを読出し、同キャッシュ領域を解放する場合は、`GRP_FS_GE_RELEASE` オプションで `grp_fs_get_error` を実行します。メディアの取外しで、反映不能となったすべての更新されたキャッシュデータを読出し、同キャッシュ領域を解放する場合は、`GRP_FS_GE_RELEASE` に加え、`GRP_FS_GE_DIRTY` を指定して `grp_fs_get_error` を実行します。

なお、`grp_fs_get_error` は、キャッシュ情報の読出しと同キャッシュ領域の解放を行うだけで、`unmount` 処理自体は行いません。`unmount` 処理を行い、新しいメディアを `mount` 可能な状態にするためには、`grp_fs_unmount` を強制 `unmount` オプション `GRP_FS_FORCE_UMOUNT` 付で実行する必要があります。

また、`grp_fs_get_error` で退避したキャッシュデータは、`mount` 時等に、`grp_fs_check_volume` 関数を使って挿入されたメディアをチェックし、退避したデータに対応したメディアであると判明した場合に、利用者に確認した後、デバイスドライバの I/O 関数を使って書戻す必要があります。

#### 3.1 1.4 不当メディア/ファイルシステムへの対応

**GR-FILE** では、`mount` 時に、簡単なファイルシステムの整合性チェックを行い、間違ったメディアや壊れたファイルシステムに対しては、`mount` をエラーで返し、メディアに対し、間違ったデータを書込まないようにしています。

例えば、`FAT` ファイルシステムでは、ファイルシステムの先頭ブロックにあるマジックナンバー等をチェックし、正しい `FAT` ファイルシステムでない場合、`GRP_FS_ERR_FS` というエラーを返します。

また、`FAT16/FAT32` では、正しく `unmount` 処理されずに取外されたメディアの場合、`unmount` 処理がされていないというフラグがメディアに記録されていますので、`GRP_FS_ERR_NEED_CHECK` というエラーを返し、ファイルシステムの整合性チェックが必要であることを知らせます。この場合、**GR-FILE** では、ファイルシステムの整合性チェックプログラムは、提供していませんので、`PC` 等で同メディアのファイルの整合性チェックを行うことが必要です。但し、`GRP_FS_FORCE_MOUNT`、または、`GRP_FS_ROONLY` オプションを指定して `grp_fs_mount` をコールし、強制マウント、あるいは、読出しのみの形で、無理やりに `mount` することも可能です。

### 3.1.2 メディアのフォーマットとパーティションの設定

**GR-FILE** では、バージョン 1.10 以降、メディアのフォーマットとパーティションの設定変更機能をサポートしています。

#### 3.1.2.1 フォーマット機能

**GR-FILE** では、FAT12/16/32 すべての FAT タイプファイルシステムのフォーマット作成が可能です。さらに、メディア/パーティションのサイズから自動的にクラスタサイズ、FAT タイプを選択してフォーマットを行う自動簡易フォーマット機能に加え、明示的にクラスタサイズや FAT タイプを指定して、カスタマイズしたフォーマットも可能です。

**GR-FILE** が提供するフォーマットインタフェースは、下記のとおり、フォーマットを行うメディアパーティションのデバイス名に加え、フォーマット時に使用する I/O バッファ領域の情報、フォーマットのパラメータ情報を示すフォーマット情報パラメータ、メディアの情報を示すメディア情報パラメータを指定して行います。

```
int grp_fat_format(
    const char          *pcDev,      /* [IN] メディア/パーティションのデバイス名 */
    grp_fat_format_param_t *ptParam, /* [IN/OUT] フォーマット情報パラメータ */
    grp_fs_media_info_t  *ptMedia); /* [IN/OUT] メディア情報パラメータ */
```

`pcDev` で指定するデバイス名は、「3.5.3 パーティション分割されたメディアアクセスのためのデバイスネーミング規則」で説明しましたネーミング規則に従い、対象のメディア/パーティションを指定します。本関数は、すでに必要なパーティション設定が済んでいるという前提をしており、`pcDev` パラメータには、通常、“USB0a” や “USB0” のように、パーティション番号文字省略を含め、パーティション付のデバイス名を指定します。パーティションの設定/変更が必要な場合は、本関数を実行する前に、予め `grp_fs_write_part` を使用し、パーティションの設定/変更を行なっておく必要があります。また、既にパーティション設定がされているメディアにパーティションレスでフォーマットする場合は、“USB0\*” のように、パーティションレス形式のデバイス名を指定します。

`ptParam` は、FAT タイプやクラスタサイズなど表 3-7 に示す各種フォーマットパラメータ情報を指定します。表 3-7 に示す各パラメータは、それぞれ 0 を指定することで、呼び出し側では具体的な値を指定せず、自動計算あるいはデフォルトの値によるフォーマットが可能です。すなわち、`ptParam` で示した構造体のすべてのフィールドを 0 にしてコールしますと、自動設定によるフォーマットが可能です。逆に、各フィールドに所望の値を設定してコールしますと、FAT タイプやクラスタサイズをカスタマイズした形でフォーマットが可能です。但し、メディア/パーティションのトータルサイズにより、適用できる FAT タイプ/クラスタサイズには FAT 規格上制限がありますので、適用範囲外のパラメータを指定した場合は、GRP\_FS\_ERR\_BAD\_PARAM エラーとなります。

`ptParam` には、本関数への入力パラメータとなるフィールドだけでなく、本関数実行結果の出力フィールドもあります。また、入力フィールドも、本関数で自動設定したり、値を補正したりするケースがありますので、フォーマットが成功した場合には、入力フィールドにも、結果の情報が設定されて返ります。

なお、ptParam に NULL を指定した場合は、結果の設定情報を得ることはできませんが、全フィールドに 0 を指定したのと同じ効果となります。

表 3-7 フォーマット情報パラメータの詳細

#	フィールド名	意味	備考
1	ucFatType (入力/出力)	<ul style="list-style-type: none"> <li>FAT タイプ</li> <li>GRP_FAT_TYPE_12(1) ... FAT12</li> <li>GRP_FAT_TYPE_16(2) ... FAT16</li> <li>GRP_FAT_TYPE_32(3) ... FAT32</li> </ul>	<ul style="list-style-type: none"> <li>0 を指定した場合、メディア/パーティションサイズ、および、クラスタサイズより自動計算されます</li> </ul>
2	aucVolLab[11] (入力/出力)	<ul style="list-style-type: none"> <li>ボリューム名称</li> </ul>	<ul style="list-style-type: none"> <li>先頭バイトが 0x00 の場合、“NO NAME” が設定されます</li> <li>ボリューム名称は 11 バイトで設定します、11 バイトより短い場合は、スペースで埋められます。</li> <li>11 バイトで設定した場合は、末尾に'¥0' は設定できませんのでご注意ください</li> </ul>
3	uiClstSec (入力/出力)	<ul style="list-style-type: none"> <li>クラスタサイズ (クラスタ当たりのセクタ数)</li> </ul>	<ul style="list-style-type: none"> <li>0 を指定した場合、メディア/パーティションサイズより、grp_fat_cluster_limit_tbl を用いて決定されます</li> </ul>
4	uiRootCnt (入力/出力)	<ul style="list-style-type: none"> <li>root ディレクトリのエントリ数</li> </ul>	<ul style="list-style-type: none"> <li>0 を指定した場合、FAT タイプ毎のデフォルト値を定めた grp_fat_default の情報により設定されます</li> </ul>
5	uiRsvSecCnt (入力/出力)	<ul style="list-style-type: none"> <li>FAT 情報領域の開始位置セクタオフセット</li> </ul>	<ul style="list-style-type: none"> <li>0 を指定した場合、FAT タイプ毎のデフォルト値を定めた grp_fat_default の情報により設定されます</li> </ul>
6	uiAlign (入力)	<ul style="list-style-type: none"> <li>データクラスタ領域の境界調整セクタ値 (SD 等で使用)</li> </ul>	<ul style="list-style-type: none"> <li>0 を指定した場合、クラスタサイズ境界とします。その他の場合、指定した値の倍数でデータクラスタ領域を位置づけます</li> </ul>
7	uiOption (入力)	<ul style="list-style-type: none"> <li>フォーマットオプション</li> <li>GRP_FAT_NO_CRT_ACC_TIME</li> <li>GRP_FAT_ADJ_BY_START (SD 等で使用)</li> </ul>	<ul style="list-style-type: none"> <li>作成/アクセス時刻記録なし FAT で作成</li> <li>uiAlign による境界調整を uiRsvSecCnt でなく、パーティション開始位置で調整します</li> </ul>
8	uiClst (出力)	<ul style="list-style-type: none"> <li>トータルクラスタ数</li> </ul>	<ul style="list-style-type: none"> <li>出力パラメータ</li> </ul>
9	uiFatSec (出力)	<ul style="list-style-type: none"> <li>1 つの FAT 情報領域のセクタ数</li> </ul>	<ul style="list-style-type: none"> <li>出力パラメータ</li> </ul>
10	aucVolSer[4] (出力)	<ul style="list-style-type: none"> <li>自動的にアサインしたボリュームシリアル番号(現在時刻値)</li> </ul>	<ul style="list-style-type: none"> <li>出力パラメータ</li> </ul>
11	uiNotUsed (出力)	<ul style="list-style-type: none"> <li>ファイルシステムとして管理していないセクタ数</li> </ul>	<ul style="list-style-type: none"> <li>出力パラメータ</li> </ul>
12	uiAdjust (出力)	<ul style="list-style-type: none"> <li>uiAlign の境界調整による調整セクタ数 (SD 等ではこの値を使い、パーティションの開始位置を調整します)</li> </ul>	<ul style="list-style-type: none"> <li>出力パラメータ</li> </ul>

表 3-7 に示しましたように、各フィールドに 0 を指定した場合、あるグローバル変数を参照して、それぞれのデフォルト値や値の自動計算が行われます。従いまして、そのグローバル変数値を起動時に変更することで、デフォルト/自動設定動作のカスタマイズも可能です。表 3-8 にフォーマット関連のデフォルト値や自動計算の動作を左右するグローバル変数の一覧を示します。

表 3-8 フォーマットパラメータのデフォルト値/自動計算で使用するグローバル変数

#	変数名	意味
1	grp_fat_cluster_limit_tbl [6]	<ul style="list-style-type: none"> <li>・ クラスタサイズの自動計算で使用するテーブル</li> <li>・ それぞれ、512、1K、2K、4K、8K、16K のクラスタサイズを使用するメディア/パーティションサイズの上限值（単位 512 バイトセクタ）を規定する</li> </ul>
2	grp_fat_format_cfg	<ul style="list-style-type: none"> <li>・ フォーマットを行う際の以下の設定を保持 <ul style="list-style-type: none"> <li>・ BPB 領域に保持する OEM 名称</li> <li>・ FAT12/16/32 の誤認識を防ぐためのトータルクラスタの閾値前後の未使用クラスタ数情報</li> <li>・ フォーマット時に使用する I/O バッファサイズ</li> </ul> </li> </ul>
3	grp_fat_default[3]	<ul style="list-style-type: none"> <li>・ FAT12/16/32 それぞれの、デフォルトパラメータ値を保持</li> <li>・ 保持するパラメータは以下の通り <ul style="list-style-type: none"> <li>・ root ディレクトリエントリ数</li> <li>・ FAT 情報の開始セクタ数オフセット</li> <li>・ ヘッド数</li> <li>・ sector/track</li> </ul> </li> </ul>

特に、grp\_fat\_cluster\_limit\_tbl は、メディア/パーティションのサイズからクラスタサイズを決定し、その情報を基に FAT タイプを自動計算することになりますので、使用目的に応じて、必要なカスタマイズを行って下さい。それぞれの上記グローバル変数の詳細については、「3.1 4 各種パラメータの設定・変更、および、コンパイルオプション」の節を参照下さい。

なお、デフォルトでのメディア/パーティションサイズと、クラスタサイズ、FAT タイプとの関係は、表 3-9 に示す対応関係となっています。

表 3-9 自動計算時のメディア/パーティションサイズとクラスタサイズ/FAT タイプとの対応関係

#	メディア/パーティションのトータルサイズ	クラスタサイズ	FAT タイプ
1	約 ～2MB (～4166 (512 バイトセクタ))	512 バイト	FAT12
2	約 2MB～4.1MB (4167～8399 (512 バイトセクタ))	512 バイト	FAT16
3	約 4.1MB～16MB (8400～32767 (512 バイトセクタ))	1KB	FAT16
4	約 16MB～128MB (32768～262143 (512 バイトセクタ))	2KB	FAT16
5	約 128MB～256MB (262144～524287 (512 バイトセクタ))	4KB	FAT16
6	約 256MB～512MB (524288～1049713 (512 バイトセクタ))	8KB	FAT16
7	約 512MB～ (1049714～ (512 バイトセクタ))	8KB	FAT32

最後のパラメータの **ptMedia** は、メディア情報を示すパラメータです。表 3-10 に示しますように、**pcDev** で指定した対象のメディア/パーティションのトータルサイズ、セクタサイズ、開始オフセット、ヘッドやトラックあたりのセクタ数、メディアタイプ識別情報などを示します。

表 3-10 メディア情報パラメータの詳細

#	フィールド名	意味	備考
1	uiStartSec	・対象パーティションの開始位置セクタ（入力/出力）	・0 を指定した場合、デバイスドライバの <b>ioctl</b> または <b>open</b> 関数で得られる値を使用します
2	uiTotalSec	・対象メディア/パーティションのトータルセクタ数（入力/出力）	・同上
3	usTrkSec	・対象メディアのトラックあたりのセクタ数（入力/出力）	・0 を指定した場合、デバイスドライバの <b>ioctl</b> 関数、または、FAT タイプ毎のデフォルト値を定めた <b>grp_fat_default</b> の情報により設定されます。なお、2HD、2DD フロッピーサイズのメディア/パーティション場合は、 <b>grp_fat_default</b> を使用せず、同タイプの標準値を使用します。
4	usHead	・対象メディアのヘッド数（入力/出力）	・同上
5	iSecShift	・対象メディアセクタサイズのシフトカウント値（入力/出力）	・0 を指定した場合、デバイスドライバの <b>ioctl</b> または <b>open</b> 関数で得られる値を使用します
6	ucMediaType	・FAT0 に格納されるメディアタイプの識別情報（入力/出力）	・0 を指定した場合、デバイスドライバの <b>ioctl</b> 関数で得られる値、または、固定ディスクの値である <b>0xf8</b> を使用します。なお、2HD、2DD フロッピーサイズのメディア/パーティションの場合は、 <b>ioctl</b> で得られなければ、それぞれ <b>0xf0</b> 、 <b>0xf9</b> を使用します。

**ptParam** のフォーマットパラメータ情報と同様に **ptMedia** の各フィールドに対して 0 を指定しますと、それぞれ、デバイスドライバの **open** 関数、**ioctl** 関数により得られる値、あるいは、デフォルト値を使用します。本来、デバイスドライバの **open** 関数、**ioctl** 関数により、必須のメディア情報は得られますので、アプリケーションプログラムは、**ptMedia** の各フィールドを 0 に設定してコールする形が一般的です。逆に、0 以外の値を指定しますと、デバイスドライバの **open** 関数、**ioctl** 関数で得られる値やデフォルトの値より優先して使用され、カスタマイズしたフォーマットが可能です。

**ptMedia** の各フィールドは、本フォーマット関数に対する入力でもあり同時に、フォーマット成功時には、実際に使用した値が各フィールドに戻されます。**ptParam** と同様、**ptMedia** 自身を **NULL** で指定することも可能であり、**NULL** を指定した場合は、実際に使用されたメディア情報を得ることはできませんが、各フィールドに対し、0 を指定した場合と同じ効果を持ちます。

**ptMedia** 情報の自動取得処理で使用されるデバイスドライバの **ioctl** 関数は、バージョン 1.10 から新たに追加された **GR-FILE** とデバイスドライバとの間のインタフェースで、デバイスの様々な I/O 制御を行うための関数です。この **ioctl** では、このメディア情報の取得や、メディアの取出し/ロック/アンロック制御、物理フォーマット等、デバイス固有の機能の制御を行います。デバイスドライバでの **ioctl** 関数のサポートは必ずしも必須ではなく、なしでもフォーマットは可能です。但し、フォーマットには、パーティションの開始位置セクタ情報、トータルセクタ情報、セクタサイズのシフト情報は必要ですので、デバイスドライバで **ioctl** 関数はサポートしていなくても、デバイスドライバの **open** 関数では、これらの必須情報を 0 ではなく、正しい値を設定して返す必要があります。例えば、**ioctl** 関数がサポートされておらず、**open** 関数でもトータルセクタ情報が 0 として返ってきている場合は、トータルセクタ情報を指示せず、



`grp_fat_format` を実行しますと、`GRP_FS_ERR_BAD_PARAM` エラーとなります。

`grp_fat_format` は、対象メディア/パーティションが `mount` されていない状態で実行する必要があります。 `mount` 中に本関数を実行した場合は、`GRP_FS_ERR_BUSY` エラーとなります。

`grp_fat_format` 関係のインタフェース定義は、“include” ディレクトリ下の“`grp_fat_format.h`”、“`grp_fs_dev_io_if.h`”に記述されています。

なお、本関数で提供するフォーマット機能では、必要最小限の FAT フォーマットデータのみを書き込み、物理フォーマットや、メディア/パーティション全体にわたるデータの上書き/消去、不良セクタの検出、不良クラスタ登録は行いません。物理フォーマットが必要な場合は、`grp_fs_ioctl_dev` の物理フォーマット機能を使って行って下さい。但し、デバイスドライバが物理フォーマット機能をサポートしていることが前提です。メディア/パーティション全体にわたるデータの上書き/消去、不良セクタの検出、不良クラスタ登録については、今回のバージョンではサポートしません。

また、SD/SDHC 用のフォーマット関数(`grp_fat_format_sd`)は、SD Card ライセンスを保有されていて、ご要望がある場合のみ、ご提供しております。`grp_fat_format_sd` では、SD 規格推奨のフォーマットパラメータにてフォーマットを行なうため、メディアサイズに対応したデフォルトの FAT タイプやクラスタサイズ等は、上記表 3-8 とは異なります。`grp_fat_format_sd` を用いたフォーマットに関しては、同関数の詳細説明と SD 規格の規格書をご参照下さい。

### 3.1 2.2 パーティションの設定/変更

**GR-FILE** では、メディアに対して、パーティション情報を設定・変更する機能を提供しています。

**GR-FILE** では、1つのメディアに対して、PC 等で利用されている 4つの基本パーティションの設定が可能です。なお、パーティションをさらにパーティションに分割し、5つ以上のパーティションの構築を可能とする論理パーティションについては、本バージョンではサポートしていません。

パーティション情報の読み出し、設定・変更は、以下の関数を用いて行います。

```
int  grp_fs_read_part(                /* パーティション情報の読み出し */
    const char          *pcDev,       /* [IN] メディア/パーティションのデバイス名 */
    grp_fs_dk_part_t    *ptPart);    /* [OUT] 読み出したパーティション情報 */

int  grp_fs_write_part(              /* パーティション情報の書き込み */
    const char          *pcDev,       /* [IN] メディア/パーティションのデバイス名 */
    int                 iAuto,        /* [IN] 自動による 1パーティション設定指示 */
    grp_fs_dk_part_t    *ptPart);    /* [IN/OUT] 書き込むパーティション情報 */
```

pcDev で指定するデバイス名は、パーティション情報の読み出し/書き込みのメディアの指定で、「3.5.3 パーティション分割されたメディアアクセスのためのデバイスネーミング規則」で説明しましたネーミング規則に従い行います。パーティションの読み出し/書き込みの場合は、そのメディアの先頭部分から読み書きをしなければならないため、“USB0\*”のように、pcDev にパーティションレス形式のデバイス名指定を行います。”USB0” “USB0a” といった、通常のパーティション指定のデバイス名を指定した場合は、GRP\_FS\_ERR\_BAD\_DEV のエラーとなります。

iAuto は、本来なら ptPart で示された領域に詳細なパーティション情報を設定して指定するところを、自動またはデフォルトの値で対象メディア全体を 1パーティションで設定するかどうかを指定します。このパラメータ値が 0 以外の場合、自動設定を行います。なお、0 以外を指定した場合、詳細なパーティション情報を予め ptPart で示された領域に設定しておく必要はありませんが、領域としては、4 エントリ分を確保し、その領域のアドレスを ptPart パラメータとして渡すことが必要です。パーティション設定が正常に終了した場合、ptPart で示された領域には、設定されたパーティション情報が設定されて返ります。

ptPart は、読み書きする 4つの基本パーティション情報を格納あるいは同情報が設定された領域のアドレスを指定します。表 3-11 に、各パーティション情報エントリの構造を示します。



表 3-1 1 パーティション情報 grp\_fs\_dk\_part\_t の詳細

#	フィールド名	意味
1	ucActive	<ul style="list-style-type: none"> <li>・アクティブパーティションかどうかの区別</li> <li>GRP_FS_PART_ACT            0x80    アクティブ</li> <li>GRP_FS_PART_NACT        0x00    アクティブでない</li> </ul>
2	ucPartType	<ul style="list-style-type: none"> <li>・パーティションタイプ番号（下記 define 値参照）</li> <li>GRP_FS_PART_NULL        0x00    NULL パーティション</li> <li>GRP_FS_PART_FAT12        0x01    FAT12</li> <li>GRP_FS_PART_FAT16_L32    0x04    FAT16 &lt; 32MB</li> <li>GRP_FS_PART_EXT         0x05    拡張パーティション</li> <li>GRP_FS_PART_FAT16_H32    0x06    FAT16 &gt; 32MB</li> <li>GRP_FS_PART_NTFS        0x07    NT ファイルシステム</li> <li>GRP_FS_PART_FAT32_CHS    0x0b    FAT32 シリンダアドレス方式</li> <li>GRP_FS_PART_FAT32_LBA    0x0c    FAT32 セクタアドレス方式</li> <li>GRP_FS_PART_FAT16_LBA    0x0e    FAT16 セクタアドレス方式</li> <li>GRP_FS_PART_EXT_LBA      0x0f    拡張パーティション セクタアドレス方式</li> <li>GRP_FS_PART_FAT32_HCHS   0x1b    隠しパーティション FAT32 シリンダアドレス方式</li> <li>GRP_FS_PART_FAT32_HLBA   0x1c    隠しパーティション FAT32 セクタアドレス方式</li> <li>GRP_FS_PART_FAT16_HLBA   0x1e    隠しパーティション FAT16 セクタアドレス方式</li> <li>GRP_FS_PART_LINUX_SW    0x82    LINUX スワップ/Solaris</li> <li>GRP_FS_PART_LINUX        0x83    LINUX パーティション</li> <li>GRP_FS_PART_LINUX_EXT    0x85    LINUX 拡張パーティション</li> <li>GRP_FS_PART_FREE_BSD     0xa5    FreeBSD パーティション</li> </ul>
3	tStartCHS	<ul style="list-style-type: none"> <li>・パーティション開始位置のシリンダ/ヘッド/トラック内セクタ情報（下記構造体）</li> <li>ucHead    ヘッド番号（0～0xff）</li> <li>ucSec      トラック内セクタ番号（1～0x3f）</li> <li>usCyl      シリンダ番号（0～0x3ff）</li> </ul>
4	tEndCHS	<ul style="list-style-type: none"> <li>・パーティション終了位置のシリンダ/ヘッド/トラック内セクタ情報（下記構造体）</li> <li>ucHead    ヘッド番号（0～0xff）</li> <li>ucSec      トラック内セクタ番号（1～0x3f）</li> <li>usCyl      シリンダ番号（0～0x3ff）</li> </ul>
5	uiStartSec	・パーティション開始位置のセクタ情報（セクタ数で指定）
6	uiSecCnt	・パーティションのサイズ（トータルセクタカウント）

なお、パーティション情報に設定する対象メディアのパーティションの開始位置やトータルセクタ情報は、**GR-FILE** がアプリケーションプログラムに対して提供するデバイス直接制御インタフェースの `grp_fs_open_dev` や、`grp_fs_ioctl_dev` の `GRP_FS_DEV_CTL_GET_MEDIA` を利用して得ることができます。但し、デバイスによっては、`grp_fs_dev_ioctl` の `GRP_FS_DEV_CTL_GET_MEDIA` をサポートしていなかったり、`grp_fs_open_dev` でも、トータルセクタ数を 0 として返すデバイスもあります。

`tStartCHS`、`tEndCHS` は、大きなメディアでは、値がおさまりきらず、参考値にすぎないケースもありますが、設定する値は、`tStartCHS <= tEndCHS` となるようにする必要があります。 `tStartCHS`、`tEndCHS` で許された範囲以外の値を指定した場合、または、`tStartCHS <= tEndCHS` とならない情報で `grp_fs_write_part` を実行した場合は、`GRP_FS_ERR_BAD_PARAM` エラーとなります。

`iAuto` に 0 以外を指定した場合、デバイスドライバの `open` 関数、`ioctl` 関数を使用して、開始位置セクタ情報やトータルセクタ数情報等を得て、パーティション情報の自動設定を行います。 `open` 関数、`ioctl`

関数いずれでも、トータルセクタ数情報等が得られなかった場合は、GRP\_FS\_ERR\_BAD\_PARAM エラーとなります。

また、iAuto に 0 以外を指定した場合で、デバイスの ioctl 関数により、メディアのヘッド数やトラック当たりのセクタ情報が得られなかった場合は、2HD、2DD フロッピーサイズのメディアについては、標準 2HD、2DD のヘッド数、トラック当たりのセクタ数情報を使用し、その他の場合は、メディアのヘッド数やトラック当たりのセクタ数情報として、それぞれ以下の値をデフォルトとして使用します。

GRP_FS_PART_TRK_SEC	32	トラック当たりのセクタ数
GRP_FS_PART_HEAD	128	ヘッド数

iAuto に 0 以外を指定した場合のパーティションタイプ番号は、対象メディアのトータルセクタ数からデフォルトの自動計算により求められる FAT タイプ種別に従い、grp\_fs\_default\_part\_type というテーブルを参照して、設定されます。grp\_fs\_default\_part\_type テーブルの初期値は、以下のとおりです。この値を起動時に変更し、カスタマイズすることも可能です。

FAT12	GRP_FS_PART_FAT12	(0x01)
FAT16 (< 32MB)	GRP_FS_PART_FAT16_L32	(0x04)
FAT16 (>=32MB)	GRP_FS_PART_FAT16_H32	(0x06)
FAT32	GRP_FS_PART_FAT32_LBA	(0x0c)

iAuto に 0 を指定し、パーティション情報を明示的に指定する場合も、フォーマット関数の一環として、トータルセクタ数から FAT タイプ種別を求める関数 grp\_fat\_find\_type も提供していますので、同関数を使用して、FAT タイプを求め、適切なパーティションタイプ情報を求めることが可能です。

パーティション設定関連のインタフェース定義は、"include" ディレクトリ下の"grp\_fs\_disk\_part.h"に記述されています。また、メディア情報取得のための grp\_fs\_open\_dev、grp\_fs\_ioctl\_dev 等のインタフェース定義は、"grp\_fs\_dev\_io\_if.h"に、FAT タイプ取得のための grp\_fat\_find\_type は、"grp\_fat\_format.h"に定義されています。

### 3.1.3 RAM ディスク機能

**GR-FILE** では、バージョン 1.10 より、特別なデバイスドライバのサンプルとして、メモリ上に仮想的なディスクを構築する RAM ディスク機能を提供します。本 RAM ディスク上にファイルシステムを構築することで、メモリ上で高速なファイルシステムを実現できます。

RAM ディスクを使った高速なファイルシステムを実現は、以下の手順で行います。

#### (1) RAM ディスクのデバイステーブルへの登録

RAM ディスクを **GR-FILE** からアクセスできるように、“grp\_fs\_dev\_sw\_tbl.c” に記述されている **GR-FILE** のデバイスのコンフィグレーションテーブル `grp_fs_dev_tbl[ ]` に、RAM ディスク用のアクセス関数リスト `grp_fs_dev_op_ram` を登録しておきます。

#### (2) RAM ディスクドライブの設定

メモリ上のどこの領域を RAM ディスクとして使用するか `grp_fs_dev_io_ram_init` 関数を使って行います。同関数のパラメータは以下のとおりです。

```
int  grp_fs_dev_io_ram_init(
    int          iDiskNo,          /* RAM ディスクのドライブ番号 */
    grp_uchar_t  *pucStart,        /* RAM ディスクのスタートアドレス */
    grp_uint32_t uiSecCount);      /* RAM ディスクのサイズ (セクタサイズ単位) */
```

`iDiskNo` で、設定する RAM ディスクのドライブ番号を指定します。デフォルトでは、0～3 の値の最大 4 つのドライブを定義できます。範囲外の値を指定すると `GRP_FS_ERR_BAD_DEV` エラーとなります。また、`grp_fs_dev_io_ram_init` で既に定義されたドライブ番号を指定すると、`GRP_FS_ERR_BUSY` エラーとなります。

`pucStart`、`uiSecCount` は、RAM ディスクとして使用するメモリ領域の開始位置とサイズです。サイズは、セクタサイズ単位のセクタカウントで指定します。

RAM ディスクのセクタサイズは、`GRP_FS_DEV_RAM_SECSFT` にセクタサイズのシフト量として、定義されています。(例:セクタサイズ 512 バイトでは、9 と定義)

ご提供時は、`GRP_FS_DEV_RAM_SECSFT` は 9 (512 バイト) と定義されています。

本関数の定義は、“sample/base/grp\_fs\_dev\_io\_ram.h” に記述されています。

#### (3) ファイルシステムの作成

`grp_fat_format` を使用し、RAM ディスクをフォーマットしてファイルシステムを作成します。

#### (4) ファイルシステムのマウント

`grp_fs_mount` を使用し、フォーマットした RAM ディスクをマウントし、ファイルシステムとしてアクセスするようにします。

なお、RAM ディスクでは、スペースを節減するため、パーティションは使用しません。RAM ディスクのデバイスを指定する場合は、パーティション番号文字を省略 (第 0 パーティションと同じ)、“mem0”、

“mem1”、“mem2”、“mem3” のような形式でデバイス名称を指定して下さい。

また、RAM ディスクのソースは、GRP\_FS\_RAM\_DISK の #ifdef が入っていますので、使用する場合は、コンパイルオプションの GRP\_FS\_RAM\_DISK を指定してコンパイルして下さい。

### 3.1.4 各種パラメータの設定・変更、および、コンパイルオプション

**GR-FILE** では、各種パラメータを、実行時の初期化処理、あるいは、**define** を変更し、再コンパイルすることで設定・変更することが可能です。また、コンパイルオプションにより、ある機能を組込んだり、外したりすることができます。

#### 3.1.4.1 実行時に変更可能なパラメータ

##### (1) ファイルシステム I/O 関連のパラメータ

キャッシュブロックサイズ、キャッシュブロック数、同時オープンファイル数等のパラメータは実行時の初期化処理において設定・変更が可能です。これらのパラメータは、**"grp\_fs\_param\_t"** というタイプを持つグローバル構造体変数 **"grp\_fs\_param"** の各フィールドに値を設定し、初期化処理関数 **grp\_fs\_init** を実行することで行います。

表 3-1 2 に、**"grp\_fs\_param"** に設定する各種パラメータの一覧を示します。

表 3-1 2 実行時に変更可能な **grp\_fs\_param** の詳細

#	grp_fs_param のフィールド名	パラメータの内容	初期値定数名 (初期値)
1	sMount	・同時に mount するファイルシステムの最大数 ・最小値は 1	GRP_FS_MAX_MOUNT (8)
2	sTask	・同時にファイルシステムをアクセスするタスクの最大数 ・最小値は 1	GRP_FS_MAX_TASK (8)
3	ucFBlkShift	・ファイル管理ブロックキャッシュのブロックサイズのシフト値 ・本値により得られるキャッシュのブロックサイズは、メディアの物理ブロック（セクタ）サイズの倍数でなければならない	GRP_FS_FBLK_SHIFT (11)
4	ucDBlkShift	・ファイルデータキャッシュのブロックサイズのシフト値 ・本値により得られるキャッシュのブロックサイズは、メディアの物理ブロック（セクタ）サイズの倍数でなければならない（特に MO の 2 KB セクタ等を扱う場合注意）	GRP_FS_DBLK_SHIFT (11)
5	uiFBlkCnt	・ファイル管理ブロックキャッシュのブロック数 ・最小値は 2	GRP_FS_FBLK_CNT (4)
6	uiDBlkCnt	・ファイルデータキャッシュのブロック数 ・最小値は 2	GRP_FS_DBLK_CNT (16)
7	uiBHashCnt	・キャッシュブロックのハッシングバケットの数 ・1 以上の 2 のべき乗値でなければならない	GRP_FS_BLK_NHASH (64)
8	uiFileCnt	・同時にオープン可能なファイルの最大数 ・最小値は 1	GRP_FS_MAX_FILE (16)
9	uiFHashCnt	・オープン中のファイルのハッシングバケット数 ・1 以上の 2 のべき乗値でなければならない	GRP_FS_FILE_NHASH (16)
10	uiFhdlCnt	・同時にオープン中のファイルハンドルの最大数 ・最小値は 1（uiFileCnt 以上）	GRP_FS_MAX_FHDL (16)
11	uiFnameCache Cnt	・ファイル名称キャッシュの最大数 ・GRP_FS_FNAME_CACHE オプション指定時のみ指定可	GRP_FS_FNAME_CACHE_CNT (GRP_FS_MAX_FILE*2)
12	UiFnameHash Cnt	・ファイル名称キャッシュのハッシングバケットの数 ・GRP_FS_FNAME_CACHE オプション指定時のみ指定可	GRP_FS_FNAME_HASH_CNT (GRP_FS_FILE_NHASH*2)

なお、`grp_fs_param_t` の構造体は”`grp_fs/base`”ディレクトリ下にある”`grp_fs_cfg.h`”に定義されています。初期値定数名の `define` は、”`grp_fs/include`”ディレクトリ下にある”`grp_fs_param.h`”に定義されています。`grp_fs_param_t` 構造体には、上記以外に `uiFBlkSize`、`uiDBlkSize` というフィールドが定義されていますが、これらのフィールドは、`grp_fs_init` で自動的に計算され設定されます。

## (2) FAT 固有のパラメータ変数

実行時に変更可能な FAT 固有のパラメータ変数を表 3-13 に示します。なお、本変数の値を変更する場合は、最初の `mount` までに行なう必要があります。

表 3-13 FAT 固有のパラメータ変数

#	変数名	パラメータの内容	初期値定数名 (初期値)
1	<code>grp_fs_fat_cnt_buf_sz</code>	<ul style="list-style-type: none"> <li>• <code>mount</code> 時のフリークラスタ数のカウント処理を高速化するための I/O バッファの大きさ</li> <li>• 本サイズで指定したサイズのバッファが取れなかった場合は、通常のファイル管理情報キャッシュのバッファを使って、カウント処理が行なわれる</li> </ul>	<code>FAT_CNT_BUF_SZ</code> (0x8000)

`FAT_CNT_BUF_SZ` は、”`grp_fs/include/grp_fat_param.h`” に定義されています。なお、`grp_fs_fat_cnt_buf_sz` は、`grp_int32_t` の型を持つグローバル変数として、`fat.c` に定義されていますが、特にヘッダファイルには、宣言されていませんので、同変数の値を別のファイルから変更する場合は、`extern` 宣言して使用する必要があります。

## (3) フォーマット/パーティション設定関連のパラメータ

フォーマット/パーティション設定関連のパラメータの変更は、起動時に `grp_fat_cluster_limit_tbl[ ]`、`grp_fat_format_cfg`、`grp_fat_default[ ]`、`grp_fs_default_part_type` の値を変更することで可能です。

(a) `grp_fat_cluster_limit_tbl[ ]`

フォーマット対象のメディア/パーティションのトータルサイズからクラスタサイズを自動計算する際に使用するテーブルです。それぞれのエントリは順に、512、1KB、2KB、4KB、8kB、16KB サイズのクラスタを使用するメディア/パーティションのトータルサイズの上限值を定義しています。単位は 512 バイトセクタです。値が 0 の場合は、無限大を意味します。

クラスタサイズを自動計算する場合、`grp_fat_cluster_limit_tbl` テーブルの先頭から順に検索し、トータルサイズ がエントリのトータルサイズリミットより小さいか、等しいエントリを探します。エントリが見つかった場合は、エントリに対応したクラスタサイズを使用します。見つけれなかった場合は、32KB クラスタを使用します。表 3-14 に、`grp_fat_cluster_limit_tbl` の詳細を示します。

表 3-14 `grp_fat_cluster_limit_tbl[ ]` の詳細

#	エントリ番号	内容	初期値定数名 (初期値(512 バイトセクタ値))
1	[0]	・ 512 バイトクラスタのトータルサイズリミット	GRP_FAT_512_CLST_LIMIT ( $4200 \times 2 \div 4.1\text{MB}$ )
2	[1]	・ 1 KB クラスタのトータルサイズリミット	GRP_FAT_1K_CLST_LIMIT ( $0x4000 \times 2 = 16\text{MB}$ )
3	[2]	・ 2KB クラスタのトータルサイズリミット	GRP_FAT_2K_CLST_LIMIT ( $0x20000 \times 2 = 128\text{MB}$ )
4	[3]	・ 4KB クラスタのトータルサイズリミット	GRP_FAT_4K_CLST_LIMIT ( $0x40000 \times 2 = 256\text{MB}$ )
5	[4]	・ 8KB クラスタのトータルサイズリミット	GRP_FAT_8K_CLST_LIMIT (0 = 無限大)
6	[5]	・ 16KB クラスタのトータルサイズリミット ・ 本値以上のトータルサイズは、32KB クラスタとなる	GRP_FAT_16K_CLST_LIMIT (0 = 無限大)

「初期定数名欄」の「 $\times 2$ 」の表記は、「 $\times 2$ 」の左辺値を 1KB 表現にするための計算式です。例えば「GRP\_FAT\_1K\_CLST\_LIMIT」では、「 $0x4000(16384)\text{セクタ} \times 2$ 」で 16MB となります。

なお、FAT の規格上、任意のサイズのメディア/パーティションに、任意の FAT タイプのファイルシステムを構築できるのではなく、トータルクラスタ数により、FAT タイプが一意的に決まるため、メディア/パーティションのサイズに応じて、適用できる FAT タイプ、クラスタサイズの範囲に制限あります。FAT の規格上適用可能なトータルサイズと、クラスタサイズ/FAT タイプの大体の関係を表 3-15 に示します。(正確な値は、各種パラメータにより異なります)



表 3-15 FAT 規格で適用可能なトータルサイズとクラスタサイズ/FAT タイプの関係概要

#	クラスタ サイズ	トータルサイズ		
		FAT12	FAT16	FAT32
1	512	～ 2MB	2MB ～ 32MB	32MB ～
2	1KB	～ 4MB	4MB ～ 64MB	64MB ～
3	2KB	～ 8MB	8MB ～ 128MB	128MB ～
4	4KB	～ 16MB	16MB ～ 256MB	256MB ～
5	8KB	～ 32MB	32MB ～ 512MB	512MB ～
6	16KB	～ 64MB	64MB ～ 1GB	1GB ～
7	32KB	～ 128MB	128MB ～ 2GB	2GB ～

## (b) grp\_fat\_format\_cfg

BIOS Parameter Block(BPB)に設定するフォーマットの OEM 名称と、FAT12/16/32 の誤認識を防ぐためのトータルクラスタ閾値前後のクラスタギャップ値を保持します。FAT の規格では、トータルクラスタ数により FAT タイプが一意的に決まるため、FAT タイプを識別する閾値付近のトータルクラスタ数を使用すると、FAT タイプを誤認識するファイルシステム管理ソフトが存在する可能性があり、閾値付近のトータルクラスタ数の使用は推奨されていません。そこで、**GR-FILE** では、grp\_fat\_format\_cfg のクラスタ閾値ギャップ情報を使って、使用しない閾値前後のクラスタ数を定義しています。表 3-16 に、grp\_fat\_format\_cfg の詳細を示します。

表 3-16 grp\_fat\_format\_cfg の詳細

#	フィールド名	内容	初期値定数名 (初期値)
1	aucOEMName[8]	・ BPB に設定するフォーマットの OEM 名称	GRP_FAT_OEM_NAME ("GR-FILE")
2	uiSafetyGap	・ FAT12/16/32 の誤認識を防ぐための、トータルクラスタ閾値前後のクラスタギャップ値	GRP_FAT_SAFETY_GAP (10)
3	iBufSize	・ フォーマット時に使用する I/O バッファのサイズ	GRP_FAT_IO_BUF_SZ (4096)

## (c) grp\_fat\_default[]

FAT12/16/32 それぞれのルートディレクトリエントリ数、FAT 情報の開始位置セクタ数、トラック当たりのセクタ数、ヘッド数のデフォルト値を定義します。表 3-17 に、grp\_fat\_default の詳細を示します。

表 3-17 grp\_fat\_default[] の詳細

#	エントリ番号	内容	初期値定数名 (初期値)
1	[0]	FAT12 用のデフォルト値の定義	
		uiRootCnt ルートディレクトリエントリ数	GRP_FAT_12_ROOT_CNT (256)
		uiRsvSecCnt FAT 情報の開始位置 (セクタ数)	GRP_FAT_12_RSV_SEC (1)
		usTrkSec トラック当たりのセクタ数	GRP_FAT_12_TRK_SECS (18)
		usHead ヘッド数	GRP_FAT_12_HEADS (2)
2	[1]	FAT16 用のデフォルト値の定義	
		uiRootCnt ルートディレクトリエントリ数	GRP_FAT_16_ROOT_CNT (512)
		uiRsvSecCnt FAT 情報の開始位置 (セクタ数)	GRP_FAT_16_RSV_SEC (1)
		usTrkSec トラック当たりのセクタ数	GRP_FAT_16_TRK_SECS (32)
		usHead ヘッド数	GRP_FAT_16_HEADS (128)
3	[2]	FAT32 用のデフォルト値の定義	
		uiRootCnt ルートディレクトリエントリ数	GRP_FAT_32_ROOT_CNT (512)
		uiRsvSecCnt FAT 情報の開始位置 (セクタ数)	GRP_FAT_32_RSV_SEC (32)
		usTrkSec トラック当たりのセクタ数	GRP_FAT_32_TRK_SECS (32)
		usHead ヘッド数	GRP_FAT_32_HEADS (128)

## (d) grp\_fs\_default\_part\_type

メディア全体を自動で 1 パーティションとして設定する際に使用する、FAT タイプに応じたパーティションタイプ番号を定義しています。本変数の型は、grp\_fs\_default\_part\_type\_t という構造体です。表 3-18 に grp\_fs\_default\_part\_type の詳細を示します。

表 3-18 grp\_fs\_default\_part\_type の詳細

#	フィールド名	内容	初期値定数名 (初期値)
1	ucPartType12	FAT12 用のパーティションタイプ	GRP_FS_PART_FAT12 (0x01)
2	ucPartType16Small	FAT16 で 32MB 未満のメディア用のパーティションタイプ	GRP_FS_PART_FAT16_L32 (0x04)
3	ucPartType16Big	FAT16 で 32MB 以上のメディア用のパーティションタイプ	GRP_FS_PART_FAT16_H32 (0x06)
4	ucPartType32	FAT32 用のパーティションタイプ	GRP_FS_PART_FAT32_LBA (0x0c)

なお、(a) ~ (c) の変数は、”grp\_fs/include” ディレクトリ下にある ”grp\_fat\_format.h” に、(d) の変数は、”grp\_fs/include” ディレクトリ下にある ”grp\_fs\_disk\_part.h” にそれぞれ、型定義、初期値定数定義、変数の外部参照宣言がされています。

### 3.1 4.2 再コンパイルにより変更可能なパラメータ

#### (1) ファイルシステム I/O 関連のパラメータ

いくつかのパラメータは、**define** 値を変更し、再コンパイルすることで、変更が可能です。

表 3-1 9 に、ファイルシステム非依存部で、再コンパイルにより変更可能なパラメータの一覧を示します。  
また、表 3-2 0 に、FAT ファイルシステム依存部で、再コンパイルにより変更可能なパラメータの一覧を示します。

表 3-1 9 再コンパイルにより変更可能なパラメータ（ファイルシステム非依存部）

#	定数名	パラメータの内容	定義値
1	GRP_FS_MAX_PATH	・ ファイルパス名の最大長 (NULL を含む)	256
2	GRP_FS_MAX_COMP	・ ファイル名の各コンポーネントの最大長 (NULL を含む)	128
3	GRP_FS_DEV_NAME_LEN	・ デバイス名の最大長 (NULL を含む)	16
4	GRP_FS_TYPE_LEN	・ ファイルシステムタイプ名の最大長 (NULL を含む)	16
5	GRP_FS_MOUNT_COMP	・ mount 先名の各コンポーネントの最大長 (NULL を含む)	16
6	GRP_FS_MOUNT_PATH	・ mount 先パス名の最大長 (NULL を含む)	64
7	GRP_FS_VOL_NAME_LEN	・ ボリューム名称の最大長	16
8	GRP_FS_DIR_NEST	・ ディレクトリ階層の最大ネスト数	64

注) これらの定義は、"include"ディレクトリ下の "grp\_fs\_param.h" に定義されています。

表 3-2 0 再コンパイルにより変更可能なパラメータ（FAT ファイルシステム依存部）

#	定数名	パラメータの内容	定義値
1	FAT_BLK_SHIFT	・ FAT セクタサイズの 2 のべき乗数	9
2	FAT_MAP_CNT	・ オープン中の各 FAT ファイルの領域情報のキャッシュ数	4
3	FAT_FREE_TBL	・ フリーブロックキャッシュの数	16
4	FAT_COMP_SZ	・ FAT ファイル名の各コンポーネントの最大長 (NULL を含む)	256
5	FAT_COMP_CHCNT	・ FAT ファイル名の各コンポーネントの最大文字数 (NULL を含む)	128

注) これらの定義は、"include"ディレクトリ下の "grp\_fat\_param.h" に定義されています。

また、C 言語標準 I/O インタフェースで再コンパイルにより変更可能なパラメータの一覧を、表 3-2 1 に示します。

表 3-2 1 再コンパイルにより変更可能なパラメータ（C 言語標準 I/O インタフェース）

#	定数名	パラメータの内容	定義値
1	GRP_STDIO_BUF	・ 各ファイルの I/O バッファのサイズ	1024
2	GRP_STDIO_PBUF	・ 書式付 write 処理時に使用する展開バッファのサイズ	512
3	GRP_STDIO_CRT_PROT	・ ファイル作成時の保護モード	S_IRUSR S_IWUSR  S_IRGRP S_IROTH

注) これらの定義は、"include"ディレクトリ下の "grp\_stdio.h" に定義されています。

## (2) フォーマット/パーティション設定関連のパラメータ

表 3-2 2 に、再コンパイルで変更可能なフォーマット/パーティション設定関連のパラメータを示します。

表 3-2 2 再コンパイルで変更可能なフォーマット/パーティション設定関連のパラメータ

#	定数名	パラメータの内容	定義値
1	GRP_FAT_AREA_CNT	・ FAT 情報エリアの数	2
2	GRP_FAT_FSINFO_SEC	・ FAT32 時の FSINO セクタのセクタオフセット	1
3	GRP_FAT_BACKUP_BPB	・ FAT32 時の BPB ブロックのバックアップを格納する領域のセクタオフセット	6

注) これらの定義は、"grp\_fs/include" ディレクトリ下にある "grp\_fat\_format.h" に定義されています。

## (3) ショートファイル名生成関連のパラメータ

表 3-2 3 に、再コンパイルで変更可能なショートファイル名生成関連のパラメータを示します。

表 3-2 3 再コンパイルで変更可能なショートファイル名生成関連のパラメータ

#	定数名	パラメータの内容	定義値
1	GRP_FS_MAKE_SNAME_THRESHOLD	・ ショート名が何回重複した場合に生成方法を変更するかの閾値	5

注) これらの定義は、"grp\_fs/base" ディレクトリ下にある "grp\_fs.h" に定義されています。

## 3.14.3 コンパイルオプション

**GR-FILE**、および、その関連ライブラリを構築する際に、ある機能を入れたり、外したり、あるいは、あるプラットフォーム向けにするためのコンパイルオプションを表 3-24 に示します。

コンパイルオプションは、「grp\_fs\_sysdef.h」に定義されています。統合開発環境やコンパイラへの指定は必要ありません。

表 3-24 コンパイルオプション

#	コンパイル オプション	意味	備考
1	GRP_FS	・ <b>GR-FILE</b> 用のライブラリを生成するためのコンパイルオプション	・必須オプション
2	GRP_FS_SHARE_OPEN	・タスク間でファイルハンドルを共用するためのオプション ・ファイルシステム非依存処理の grp_fs.c 用のオプション	・オプション
3	GRP_FS_FAT_DIRECT_IO	・FAT ファイルシステムでダイレクト I/O 機能を有効とするためのオプション ・FAT 依存処理の fat.c 用のオプション ・アプリケーションタスクと <b>GR-FILE</b> がアドレス空間を共用しているシステム向けのオプション	・オプション
4	GRP_FS_FNAME_CACHE	・ファイル名称キャッシュを使用するためのオプション	・オプション
5	GRP_FS_FAT_CACHE_BY_GET_DIRENT	・ディレクトリエントリ情報の取得関数 grp_fs_get_dirent で取得した最後のディレクトリエントリ情報をファイル名称キャッシュとして保持するオプション	・オプション
6	GRP_FS_FAT_NO_DIR_SIZE_INFO	・ディレクトリの場合、grp_fs_get_dirent で、わざわざ同ディレクトリのサイズ情報を計算せず、0 のまま返すためのオプション	・オプション
7	GRP_FS_RAM_DISK	・RAM ディスクを使用するためのオプション	・オプション
8	GRP_FS_TRACE	・I/O トレース機能の追加のためのオプション ・ <b>GR-FILE</b> の行った処理をトレース出来るが、理解するには <b>GR-FILE</b> の内部構造の熟知が必要 ・grp_fs_trace.c は、LINUX 用の例となっており、他のシステムでは、インクルードファイルや初期化処理の変更が必要	・オプション
9	GRP_MEM	・malloc/free → grp_mem_alloc/grp_mem_free 変換のためのオプション ・malloc/free 関数がないシステムで使用 ・grp_fs_conv.h に上記変換定義あり ・grp_stdio ライブラリ構築時にも必要であれば定義要	・オプション (通常使用)
10	NON_POSIX	・非 POSIX 準拠プラットフォーム用のプラットフォーム依存 (mdep_xxx 下) コード生成用オプション ・可変長メモリ管理機能 (grp_mem.c) と、時刻設定/取得関数 (grp_time_set.c/grp_time_get.c) で使用 ・可変長メモリ管理機能は、VOS 環境で使用 ・非 POSIX 準拠プラットフォームでは、本オプションを指定し、かつ、依存コードを作成要。また、時刻設定/取得関数の依存コードを作成要。 ・本オプションなしで構築すれば、POSIX 準拠可変長メモリ管理関数、時刻設定/取得関数が使用される	・NON_POSIX 用

11	WIN32	<ul style="list-style-type: none"> <li>• Windows テスト用のコンパイルオプション</li> </ul>	• Windows 用
12	GRP_FS_MULTI_LANGUAGE	<ul style="list-style-type: none"> <li>• <b>GR-FILE</b> での多国語オプション(別売)を使用可能にするためのコンパイルオプション</li> <li>• 本コンパイルオプションを指定した場合、<b>GRP_FS_FNAME_CACHE</b> と <b>GRP_FS_FAT_CACHE_BY_GET_DIRENT</b> のコンパイルオプションは無効となる</li> </ul>	• オプション
13	GRP_FS_MINIMIZE_LEVEL	<ul style="list-style-type: none"> <li>• <b>GR-FILE</b> の ROM 削減レベル設定オプション 0 : ROM 削減なし 1 : POSIX 互換、C 言語標準 I/O、特殊な <b>GR-FILE</b> 固有インタフェース使用不可 2 : 1 の内容に加えてロングファイル名未サポート</li> </ul>	• オプション (通常は 0)
14	GRP_FS_FAT_TRY_NO_NUM_SHORT	<ul style="list-style-type: none"> <li>• ロングファイル名を指定した場合に自動生成されるショート名の生成方法を指定</li> <li>• 本オプションを指定すると、Version1.21 までの生成方法でショート名を生成</li> </ul>	• オプション
15	GRP_FS_ASYNC_UNMOUNT	<ul style="list-style-type: none"> <li>• アクセスを行っていないデバイスのアンマウントを可能とします。</li> <li>• 本オプションを指定しない場合の動作は Version1.22 までと同様の動作となります。</li> </ul>	• オプション
16	GRP_FS_UPDATE_ARCHIVE	<ul style="list-style-type: none"> <li>• Create、Write、Rename を行った場合に ARCHIVE 属性をセットします。</li> </ul>	• オプション
17	GRP_FS_FAST_MAKE_SNAME	<ul style="list-style-type: none"> <li>• 単一ディレクトリに同じような名称のロングファイル名が存在した場合のショート名生成方法を変更します。</li> <li>• 変更後のショート名生成処理は各依存ソースファイルで行います。</li> </ul>	• オプション
18	GRP_FS_ENABLE_OVER_2G	<ul style="list-style-type: none"> <li>• 2G バイト以上、4G-1 バイト以下のファイルサイズサポートを有効にします。本コンパイルオプションを有効にした場合、いくつかのアプリケーション I/O インタフェースの追加、戻り値、構造体メンバ名、使用構造体の変更が行われます。</li> </ul> <p><b>【注意事項】</b> 4G-1 バイト対応の Seek 系 API、Tell 系 API と未対応の Seek 系 API、Tell 系 API を組み合わせて使用することは想定外の動作（読書き位置の自動補正）の原因となりますのでご注意ください。 本オプションを定義すると構造体メンバ名、使用構造体の変更が行われるため、以下の API で互換性がなくなります。</p> <p>readdir() 、 grp_fs_get_attr() 、 grp_fs_get_dirent() 、 grp_fs_readdir() 、 grp_fs_stat()</p> <p>また、ファイルサイズを保持するメンバ変数名が符号なし 32 ビットに変更されるため iSize から uiSize に変更されます。アプリケーションにも影響がありますのでご注意ください。</p>	• オプション
19	GRP_USB	<ul style="list-style-type: none"> <li>• USB マスストレージ統合キットを使用する場合に定義します。</li> </ul>	• オプション

## 4. インタフェース

第 1 章で示しましたとおり、**GR-FILE** では、以下のインタフェースを提供/規定しています。

### (I) アプリケーションより使用するインタフェース

アプリケーションよりファイルシステムを利用する為の以下の関数群を提供します。

- (1) POSIX 互換インタフェース
- (2) C 言語標準 I/O インタフェース
- (3) **GR-FILE** 固有インタフェース (メディア挿抜処理インタフェースを含む)
- (4) デバイス直接制御インタフェース

### (II) その他のファイルシステムを **GR-FILE** でサポートする際の内部インタフェース

その他のファイルシステムをサポートする為、以下の内部インタフェース用関数群を提供します。

- (5) ファイルシステム抽象化インタフェース (ファイルシステム依存部とのインタフェース)

### (III) **GR-FILE** をターゲット環境で動作させる為、ポーティングの必要なインタフェース

**GR-FILE** をターゲットシステム上で動作させる為に必要な、以下のポーティング対象関数群を提供します。

- (6) OS 抽象化インタフェース (OS/プラットフォーム依存部とのインタフェース)
- (7) デバイスドライバインタフェース (デバイスドライバとのインタフェース)
- (8) ファイルシステム依存関数向けの **GR-FILE** フック関数

### (IV) ポーティングで使用可能なライブラリ関数

**GR-FILE** は、デバイスドライバや OS/プラットフォーム依存部、さらに、その他のファイルシステム依存部の実現をサポートするため、以下の関数群を提供しています。

- (9) ファイルシステム依存関数向けの **GR-FILE** 関数
- (10) デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数

以下、本章で、これらのインタフェースに共通なエラー番号、データタイプの定義を説明し、上記各インタフェースの詳細について説明します。



## 4.1 エラー番号

**GR-FILE** の関数で返す各エラー番号は、負の値で、“grp\_fs/include” ディレクトリ下にある “grp\_fs\_if.h” で定義されており、GRP\_FS\_ERR(num) というマクロを用いて定義されています。エラー番号の値は、-17921 ～ -17945 の値をとります。表 4-1 に、**GR-FILE** のエラー番号の一覧を示します。

表 4-1 エラー番号一覧

#	エラー番号名	値	意味
1	GRP_FS_ERR_IO	-17921	・ I/O エラーが発生した
2	GRP_FS_ERR_FHDL	-17922	・ ファイルハンドルが正しくない
3	GRP_FS_ERR_FS	-17923	・ ファイルシステムの構造が正しくない
4	GRP_FS_ERR_NOMEM	-17924	・ メモリやリソースが足りず、処理が行えない *1
5	GRP_FS_ERR_NEED_CHECK	-17925	・ 正常に unmount されていないファイルシステムを mount しようとした (PC 等でファイルシステムの整合性チェックが必要)
6	GRP_FS_ERR_BAD_DEV	-17926	・ 指定したデバイス名、または、デバイス番号が正しくない
7	GRP_FS_ERR_PERMIT	-17927	・ ファイル属性や利用者保護属性により、要求された処理が許されていない。
8	GRP_FS_ERR_BAD_FS_NAME	-17928	・ ファイルシステムタイプ名称が正しくない
9	GRP_FS_ERR_TOO_MANY	-17929	・ 処理中のファイル等の数がシステムの規定値を超えていて、処理できない
10	GRP_FS_ERR_EXIST	-17930	・ 生成しようとしたファイルは既に存在している
11	GRP_FS_ERR_BUSY	-17931	・ オープン中のファイルや、処理中の他タスクがあり、要求の処理が実行できない
12	GRP_FS_ERR_TOO_LONG	-17932	・ 指定したパス名が長すぎる
13	GRP_FS_ERR_NOT_FOUND	-17933	・ 指定したファイル等が見つからない
14	GRP_FS_ERR_BAD_MODE	-17934	・ 指定したモードが正しくない
15	GRP_FS_ERR_SHOULD_CLOSE	-17935	・ 本ファイルハンドルまたはカレントディレクトリによるファイルアクセスは無効化されている (もやは、アクセスできないので、close すべきである)
16	GRP_FS_ERR_BAD_OFF	-17936	・ ファイルオフセットが正しくない
17	GRP_FS_ERR_NO_SPACE	-17937	・ ファイル領域が不足し、新たなファイルブロックを確保できない
18	GRP_FS_ERR_BAD_NAME	-17938	・ ファイル名等の名前の記述に不当な文字等が含まれ正しくない
19	GRP_FS_ERR_BAD_DIR	-17939	・ 指定したディレクトリが正しくない (ディレクトリでない)
20	GRP_FS_ERR_BAD_TYPE	-17940	・ ファイルタイプの値が正しくない
21	GRP_FS_ERR_XFS	-17941	・ 別のファイルシステム内への rename を行おうとした
22	GRP_FS_ERR_BAD_PARAM	-17942	・ パラメータが正しくない
23	GRP_FS_ERR_TOO_BIG	-17943	・ ファイルシステムのブロックサイズが <b>GR-FILE</b> の設定値より大きい
24	GRP_FS_ERR_SEM	-17944	・ セマフォアの作成/取得処理で失敗した
25	GRP_FS_ERR_NOT_SUPPORT	-17945	・ 指定した処理はサポートされていない

\*1) メディアを unmount 処理せずに取出し、キャッシュバッファ不足となっている場合は、多くの関数が本エラーとなります。その場合、取出したメディアを再挿入して、挿抜処理で grp\_fs\_check\_fs\_dev を実行するか、grp\_fs\_get\_error により、反映不能となったキャッシュバッファを讀出してバッファを解放して下さい。

## 4.2 データタイプの定義

### 4.2.1 基本データタイプの定義

**GR-FILE** で使用される基本データタイプの定義を表 4-2 に示します。

表 4-2 基本データタイプ定義一覧

#	タイプ名称	定義内容 (一例)	意味
1	grp_uchar_t	unsigned char	符号なし char 型整数/文字
2	grp_ushort_t	unsigned short	符号なし short 型整数
3	grp_uint_t	unsigned int	符号なし int 型整数
4	grp_uint8_t	unsigned char	符号なし 1 バイト整数/文字
5	grp_uint16_t	unsigned short	符号なし 2 バイト整数
6	grp_uint32_t	unsigned int	符号なし 4 バイト整数
7	grp_int8_t	char	符号付き 1 バイト整数/文字
8	grp_int16_t	short	符号付き 2 バイト整数
9	grp_int32_t	int	符号付き 4 バイト整数
10	grp_uisize_t	grp_uint32_t	符号なしサイズ
11	grp_isize_t	grp_int32_t	符号付きサイズ
12	grp_ioffset_t	grp_int32_t	符号付きオフセット値
13	grp_uioffset_t	grp_uint32_t	符号なしオフセット値 コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合のみ定義
14	grp_fs_sem_t	プラットフォーム依存	セマフォ ID
15	grp_fs_task_t	プラットフォーム依存	タスク ID

- ・#1～#12 は、“include” ディレクトリ下の “grp\_types.h” に定義されています。
- ・#13、#14 は、“mdep\_XXX / include” ディレクトリ下の “grp\_fs\_mdep\_types.h” に、ターゲットのプラットフォームに依存して、定義する必要があります。

## 4.2.2 アプリケーションインタフェース関連の構造体

**GR-FILE** のアプリケーションインタフェースで使用する構造体のタイプ定義を以下に示します。

(1) ～ (3) の定義は、”include” ディレクトリ下の ”grp\_fs\_if.h” に定義されています。

(1) grp\_fs\_dir\_ent\_t (ディレクトリエントリ情報)

grp\_fs\_get\_dirent、grp\_fs\_get\_attr、grp\_fs\_set\_attr 等で使用されるディレクトリエントリ情報です。

#	構造体メンバ名	タイプ	意味
1	iDev	int	デバイス番号 (3.5.3 節 参照)
2	uiFid	grp_uint32_t	ファイル ID
3	pucName	grp_uchar_t *	ファイル名称 (格納領域へのポインタをコール前に設定)
4	sNameSize	short	コール時 ファイル領域のサイズ リターン時 ファイル名称の長さ
5	ucType	grp_uchar_t	ファイルの種別 (下記括弧内は、FAT の場合の意味) GRP_FS_FILE_FILE 1 通常ファイル GRP_FS_FILE_DIR 2 ディレクトリ GRP_FS_FILE_LINK 3 ファイル別名 (ロングファイル名) GRP_FS_FILE_OTHER 4 その他ファイル (ボリューム名)
6	uiProtect	grp_uint32_t	ファイルの保護属性 GRP_FS_PROT_RUSR 0400 オーナ read 可能 GRP_FS_PROT_WUSR 0200 オーナ write 可能 GRP_FS_PROT_XUSR 0100 オーナ実行可能 GRP_FS_PROT_RGRP 0040 グループ read 可能 GRP_FS_PROT_WGRP 0020 グループ write 可能 GRP_FS_PROT_XGRP 0010 グループ実行可能 GRP_FS_PROT_ROTH 0004 他ユーザ read 可能 GRP_FS_PROT_WOTH 0002 他ユーザ write 可能 GRP_FS_PROT_XOTH 0001 他ユーザ実行可能 ・ FAT ファイルシステムでは、ユーザの概念が無い為、各ビットは オーナビット値の指定で統一される ・ 但し、隠しファイルの場合は、「グループ」、「他のユーザ」のビ ットは自動的に 0 に設定される。また、実行可否の概念も無い為、 設定時の「実行可能」ビットは無視される。「実行可能」ビットは、 ファイル名のサフィックスが “EXE”、“COM”、“DLL”、“BAT” の場 合、自動的に 1 に設定され、その他の場合は、0 に設定される
7	iSize	grp_isize_t	ファイルサイズ コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を無効に した場合のみ定義
8	uiSize	grp_uisize_t	ファイルサイズ コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効に した場合のみ定義
9	iCTime	grp_int32_t	ファイル作成時刻 (1970/1/1 からのトータル秒)
10	iMTime	grp_int32_t	ファイル更新時刻 (1970/1/1 からのトータル秒)
11	iATime	grp_int32_t	ファイルアクセス時刻 (1970/1/1 からのトータル秒) 但し FAT の場合、日単位の精度しかない。
12	uiAttr	grp_uint32_t	ファイルシステム依存ファイル属性情報 FAT の場合 (FAT で規定されたファイル属性値) FAT_ATTR_RDONLY 0x01 read only FAT_ATTR_HIDDEN 0x02 隠しファイル FAT_ATTR_SYSTEM 0x04 システムファイル FAT_ATTR_VOLID 0x08 ボリューム名 FAT_ATTR_DIR 0x10 ディレクトリ FAT_ATTR_ARCHIVE 0x20 バックアップ要 FAT_ATTR_LONG 0x0f ロングファイル名

13	uiMisc	grp_uint32_t	その他のファイルシステム依存情報 FAT の場合、grp_fs_get_dirent 関数からのリターンで、ロングファイル名エントリの情報に限り、対応するショートネームのハッシュ値が入っています。
14	uiStart	grp_uint32_t	本ディレクトリエントリの開始オフセット
15	uiEnd	grp_uint32_t	本ディレクトリエントリの終了オフセット

注) #11 の FAT\_ATTR\_XXX は、"fat.h" に定義されています。

## (2) grp\_fs\_mnt\_info\_t (mount 情報)

grp\_fs\_get\_mnt、grp\_fs\_get\_mnt\_by\_dev、grp\_fs\_get\_mnt\_by\_name 等で使用される mount 情報です。

#	構造体メンバ名	タイプ	意味
1	iDev	int	mount されているメディアのデバイス番号 (3.5.3 節 参照)
2	iParentDev	int	上記デバイスの mount 先ディレクトリのデバイス番号
3	uiStatus	grp_uint32_t	mount ステータス GRP_FS_MSTAT_RDONLY 0x0001 read only mount GRP_FS_MSTAT_DAY_ACCTIME 0x0002 日単位のアクセス時刻記録 GRP_FS_MSTAT_NO_UPD_ACCTIME 0x0004 アクセス時刻記録のメディアへの反映抑止 GRP_FS_MSTAT_NO_MNT_FLAG 0x0008 メディアへのマウント中フラグの書き込み抑止 GRP_FS_MSTAT_NO_CRT_ACCTIME 0x1000 メディア上の作成/アクセス時刻記録なし GRP_FS_MSTAT_DEV_INV 0x0400 I/O 抑止状態 GRP_FS_MSTAT_SYNC_ALL 0x0010 write through 反映 GRP_FS_MSTAT_SYNC_FL_CLOSE 0x0020 each close 反映 GRP_FS_MSTAT_SYNC_FS_CLOSE 0x0040 last clos 反映
4	acDevName	char []	デバイス名称
5	aucPath	grp_uchar_t[]	mount 先のディレクトリのパス名
6	acFsType	char []	ファイルシステムタイプ
7	usFsSubType	grp_ushort_t	ファイルシステム依存のファイルシステムのサブタイプ番号 FAT の場合、FAT12/16/32 に対応し、12、16、32 の値が入る
8	usVolNameLen	grp_ushort_t	ファイルシステムのボリューム名の長さ
9	aucVolName	grp_uchar_t[]	ファイルシステムのボリューム名
10	uiVolSerNo	grp_uint32_t	ファイルシステムボリュームのシリアル番号
11	uiFsBlkSize	grp_uint32_t	ファイルシステムのブロック (クラスタ) サイズ
12	uiFsBlkCnt	grp_uint32_t	ファイルシステムのトータルブロック数
13	uiFsFileCnt	grp_uint32_t	ファイルシステムの最大ファイル数
14	uiFsFreeBlk	grp_uint32_t	ファイルシステムのフリーブロック数
15	uiFsFreeFile	grp_uint32_t	ファイルシステムのフリーファイル数
16	uiFBufSize	grp_uint32_t	ファイル管理ブロックキャッシュのバッファサイズ
17	uiDBufSize	grp_uint32_t	ファイルデータキャッシュのバッファサイズ
18	uiClusterSize	grp_uint32_t	ファイルクラスターのサイズ
19	uiFBufOff	grp_uint32_t	メディア内のファイル管理ブロックの開始オフセット
20	uiDBufOff	grp_uint32_t	メディア内のファイルデータブロックの開始オフセット
21	uiDevOff	grp_uint32_t	デバイス内の開始オフセット

## (3) grp\_fs\_err\_binfo\_t (反映不能キャッシュ情報)

grp\_fs\_get\_error で使用されるメディアに反映不能となったキャッシュ情報です。

#	構造体メンバ名	タイプ	意味
1	iDev	int	メディアのデバイス番号 (3.5.3 節 参照)
2	uiBlk	grp_uint32_t	反映不能となったキャッシュのブロック番号
3	ucBufType	grp_uchar_t	キャッシュのタイプ GRP_FS_EI_BUF_FILE 1 ファイル管理ブロックキャッシュ GRP_FS_EI_BUF_DATA 2 ファイルデータキャッシュ
4	ucBlkShift	grp_uchar_t	キャッシュバッファのブロックサイズの 2 のべき乗数値
5	uiBlkOff	grp_uint32_t	メディア内の同タイプのデータブロックの開始オフセット (メディア内オフセット = ucBlkOff + (uiBlk << ucBlkShift))
6	uiSize	grp_uint32_t	キャッシュデータのサイズ

## (4) grp\_fat\_format\_param\_t (フォーマットパラメータ情報)

grp\_fat\_format で使用されるフォーマットパラメータ情報です。本定義は、"include" ディレクトリ下の "grp\_fat\_format.h" に記述されています。

#	構造体メンバ名	タイプ	意味
1	ucFatType	grp_uchar_t	・ FAT タイプ (入力/出力) GRP_FAT_TYPE_12(1) ... FAT12 GRP_FAT_TYPE_16(2) ... FAT16 GRP_FAT_TYPE_32(3) ... FAT32
2	aucVolLab	grp_uchar_t[]	・ ボリューム名称 (入力/出力) ・ ボリューム名称は 11 バイトで設定 ・ 11 バイトより短い場合は、スペースで埋められる 11 バイトで設定する場合は、末尾に '¥0' は設定できない (aucVolLab は 11 バイトの為)
3	uiClstSec	grp_uint32_t	・ クラスタサイズ (クラスタ当たりのセクタ数) (入力/出力)
4	uiRootCnt	grp_uint32_t	・ root ディレクトリのエントリ数 (入力/出力)
5	uiRsvSecCnt	grp_uint32_t	・ FAT 情報領域の開始位置セクタオフセット (入力/出力)
6	uiAlign	grp_uint32_t	・ 0 を指定した場合、クラスタサイズ境界とする ・ その他の場合、指定した値の倍数でデータクラスタ領域を位置づける
7	uiOption	grp_uint32_t	・ フォーマットオプション GRP_FAT_NO_CRT_ACC_TIME (0x00000001) 作成/アクセス時刻記録なし FAT で作成 GRP_FAT_ADJ_BY_START (0x00000002) uiAlign による境界調整を uiRsvSecCnt でなく、パーティション開始位置で調整
8	uiClst	grp_uint32_t	・ トータルクラスタ数 (出力)
9	uiFatSec	grp_uint32_t	・ FAT の使用するセクタ数
10	aucVolSer	grp_uchar_t[]	・ 自動的にアサインしたボリュームシリアル番号(現在時刻値) (出力)
11	uiNotUsed	grp_uint32_t	・ 未使用セクタ数
12	uiAdjust	grp_uint32_t	・ uiRsvSecCnt 調整後の差分セクタ数

表中の「入力」の項目は、grp\_fat\_format へ指定できるパラメータです。「出力」の項目は grp\_fat\_format を実行後、結果として値が設定される項目です。

## (5) grp\_fs\_media\_info\_t (メディア情報)

grp\_fat\_format、grp\_fs\_dev\_ioctl の機能番号「GRP\_FS\_DEV\_CTL\_GET\_MEDIA」で使用するメディア情報です。本定義は、”include” ディレクトリ下の ”grp\_fs\_dev\_io\_if.h” に記述されています。

#	構造体メンバ名	タイプ	意味
1	uiStartSec	grp_uint32_t	・対象パーティションの開始位置セクタ (入力/出力)
2	uiTotalSec	grp_uint32_t	・対象メディア/パーティションのトータルセクタ数 (入力/出力)
3	usTrkSec	grp_uint16_t	・対象メディアのトラック当たりのセクタ数 (入力/出力)
4	usHead	grp_uint16_t	・対象メディアのヘッド数 (入力/出力)
5	iSecShift	int	・対象メディアセクタサイズのシフトカウント値 (入力/出力)
6	ucMediaType	grp_uchar_t	・FAT0 に格納されるメディアタイプの識別情報 (入力/出力)

表中の「入力」の項目は、grp\_exfat\_format、grp\_fat\_format へ指定できるパラメータです。「出力」の項目は grp\_exfat\_format、grp\_fat\_format、grp\_fs\_dev\_ioctl を実行後、結果として値が設定される項目です。

## (6) grp\_fs\_dk\_part\_t (パーティション情報)

grp\_fs\_read\_part、grp\_fs\_write\_part で使用するパーティション情報です。本定義は、”include” ディレクトリ下の ”grp\_fs\_disk\_part.h” に記述されています。

#	構造体メンバ名	タイプ	意味
1	ucActive	grp_uchar_t	・アクティブパーティションかどうかの区別 GRP_FS_PART_ACT    0x80    アクティブ GRP_FS_PART_NACT   0x00    アクティブでない
2	ucPartType	grp_uchar_t	・パーティションタイプ番号
3	tStartCHS	grp_fs_dk_chs_t	・パーティション開始位置のシリンダ/ヘッド/トラック内セクタ情報
4	tEndCHS	grp_fs_dk_chs_t	・パーティション終了位置のシリンダ/ヘッド/トラック内セクタ情報
5	uiStartSec	grp_uint32_t	・パーティション開始位置のセクタ情報 (セクタ数で指定)
6	uiSecCnt	grp_uint32_t	・パーティションのサイズ (トータルセクタカウント)

## grp\_fs\_dk\_chs\_t 型の tStartCHS、tEndCHS の詳細

#	構造体メンバ名	タイプ	意味
1	ucHead	grp_uchar_t	・ヘッド番号 (0~0xff)
2	ucSec	grp_uchar_t	・トラック内セクタ番号 (1~0x3f)
3	usCyl	grp_fs_dk_chs_t	・シリンダ番号 (0~0x3ff)

## (7) grp\_fs\_dev\_io\_info\_t (デバイス I/O 制御情報)

grp\_fs\_open\_dev のリターン情報として使用されるデバイス I/O 制御情報です。本定義は、”include” ディレクトリ下の ”grp\_fs\_dev\_io\_if.h” に記述されています

#	構造体メンバ名	タイプ	意味
1	iHandle	grp_int32_t	・デバイス I/O ハンドル
2	uiOff	grp_uint32_t	・開始位置オフセット (単位: ブロック (セクタ))
3	uiSize	grp_uint32_t	・トータルサイズ (単位: ブロック (セクタ))
4	iSzShift	int	・ブロック (セクタ) サイズのシフト値



## 4.2.3 ファイルシステム非依存部の管理構造体

ファイルシステム非依存部の管理構造体のタイプ定義を以下に示します。これらの定義は、”grp\_fs.h”に定義されています。

## (1) grp\_fs\_info\_t (ファイルシステム情報)

mount された各ファイルシステムの情報を保持します。

#	構造体メンバ名	タイプ	意味
1	iDev	int	デバイス番号 (3.5.3 節 参照)
2	usStatus	grp_ushort_t	ファイルシステムの状態 GRP_FS_STAT_RDONLY 0x0001 read only mount GRP_FS_STAT_DAY_ACCTIME 0x0002 メディア上は日単位のアクセス時刻記録 GRP_FS_STAT_NO_UPD_ACCTIME 0x0004 メディアへのアクセス時刻記録反映抑止 GRP_FS_STAT_NO_MNT_FLAG 0x0008 mount 中フラグ無 GRP_FS_STAT_NO_CRT_ACCTIME 0x1000 メディア上の作成/アクセス時刻記録無 GRP_FS_STAT_SYNC_ALL 0x0010 write through 反映 GRP_FS_STAT_SYNC_FL_CLOSE 0x0020 each close 反映 GRP_FS_STAT_SYNC_FS_CLOSE 0x0040 last close 方式反映 GRP_FS_STAT_MOD 0x0100 更新中 GRP_FS_STAT_WAITMOD 0x0200 更新待ち中 GRP_FS_STAT_DEV_INV 0x0400 I/O 抑止状態 GRP_FS_STAT_BUSY_FID 0x0800 file ID ロック中 GRP_FS_STAT_WAIT_BUSY_FID 0x8000 file ID ロック待ち中
3	pucName	grp_uchar_t *	ファイル名称 (格納領域へのポインタをコール前に設定)
4	sPathLen	short	ファイルシステムの mount 先ディレクトリパス名の長さ
5	iFsRef	grp_int32_t	ファイルシステムを参照している処理の数 例えば、ファイルを開くと+1 され、ファイルを閉じると-1 される
6	iFsOpen	grp_int32_t	オープン中のファイル数
7	ptFsOp	grp_fs_op_t *	ファイルシステム依存関数一覧を示したファイルシステムテーブルエントリへのポインタ
8	ptFsNest	grp_fs_info_t *	本ファイルシステムのディレクトリに mount されているファイルシステムのリスト
9	ptFsOtherFwd ptFsOtherBwd	grp_fs_info_t *	本ファイルシステムと同じファイルシステムに mount されているファイルシステムリストの前後エントリへのポインタ
10	aucPath	grp_uchar_t[]	mount 先ディレクトリパス名
11	ptFsParent	grp_fs_info_t *	mount 先ディレクトリ情報へのポインタ
12	iDevHandle	grp_int32_t	メディアをアクセスするための open ハンドル
13	uiDevSize	grp_uint32_t	メディアの最大ブロック番号
14	uiDevOff	grp_uint32_t	メディア内の開始オフセット
15	ucDevBlkShift	grp_uchar_t	メディア I/O のための物理ブロックサイズのシフト値
16	ucFsFBlkShift	grp_uchar_t	ファイル管理ブロックキャッシュのブロックサイズのシフト値
17	ucFsDBlkShift	grp_uchar_t	ファイルデータキャッシュのブロックサイズのシフト値
18	ucFsCBlkShift	grp_uchar_t	ファイルシステムのクラスタサイズのシフト値
19	uiFsFBlkOff	grp_uint32_t	メディア内のファイル管理ブロックの開始オフセット
20	uiFsDBlkOff	grp_uint32_t	メディア内のファイルデータブロックの開始オフセット
21	uiFsFreeBlk	grp_uint32_t	フリーブロック数
22	uiFsFreeFile	grp_uint32_t	フリーファイル数
23	uiFsBlkCnt	grp_uint32_t	メディアの最大ブロック数
24	uiFsFileCnt	grp_uint32_t	メディア内の最大ファイル数
25	uiVolSerNo	grp_uint32_t	ファイルシステムボリュームのシリアル番号



26	aucVolName	grp_uchar_t[]	ファイルシステムのボリューム名
27	usVolNameLen	grp_ushort_t	ファイルシステムのボリューム名の長さ
28	usFsSubType	grp_ushort_t	ファイルシステム依存のファイルシステムのサブタイプ番号
29	pvFsInfo	void *	ファイルシステム依存情報領域へのポインタ
30	uiFsBusyFid	grp_uint32_t	ファイル操作ロック中のファイル ID

## (2) grp\_fs\_file\_t (オープン中のファイル管理情報)

オープンされた各ファイルの管理情報を保持します。

#	構造体メンバ名	タイプ	意味
1	ucType	grp_uchar_t	ファイルの種別 (下記括弧内は、FAT の場合の意味) GRP_FS_FILE_FILE      1 通常ファイル GRP_FS_FILE_DIR        2 ディレクトリ GRP_FS_FILE_LINK       3 ファイル別名 (ロングファイル名) GRP_FS_FILE_OTHER      4 その他ファイル (ボリューム名)
2	usStatus	grp_ushort_t	ファイルの状態 GRP_FS_FSTAT_BUSY      0x0001 処理中 GRP_FS_FSTAT_UPD_ETIME 0x0002 アクセス時刻更新 GRP_FS_FSTAT_UPD_MTIME 0x0004 更新時刻更新 GRP_FS_FSTAT_UPD_ATTR 0x0008 ファイル属性情報更新 GRP_FS_FSTAT_ROOT      0x0010 ルートファイル GRP_FS_FSTAT_MOUNT     0x0020 mount ポイント GRP_FS_FSTAT_NO_UPD_TIME 0x0040 時刻更新抑止 GRP_FS_FSTAT_INVALID   0x0100 無効管理情報 GRP_FS_FSTAT_WR_LOCK   0x0200 書き込み禁止 GRP_FS_FSTAT_WR_WAIT   0x4000 書き込み待ち GRP_FS_FSTAT_WAIT      0x8000 処理待ちあり
3	uiProtect	grp_uint32_t	ファイルの保護属性 (詳細は、grp_fs_dirent_t の uiProtect 参照)
4	iRefCnt	int	処理中の数
5	iDev	int	デバイス番号 (3.5.3 節 参照)
6	uiFid	grp_uint32_t	ファイル ID
7	iSize	grp_isize_t	ファイルサイズ コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を無効にした場合のみ定義
8	uiSize	grp_uisize_t	ファイルサイズ コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合のみ定義
9	iCTime	grp_int32_t	ファイル作成時刻 (1970/1/1 からのトータル秒)
10	iMTime	grp_int32_t	ファイル更新時刻 (1970/1/1 からのトータル秒)
11	iATime	grp_int32_t	ファイルアクセス時刻 (1970/1/1 からのトータル秒) 但し FAT の場合、日単位の精度しかない。
12	uiAttr	grp_uint32_t	ファイルシステム依存ファイル属性情報 FAT の場合 (FAT で規定されたファイル属性値) FAT_ATTR_RDONLY      0x01    read only FAT_ATTR_HIDDEN       0x02    隠しファイル FAT_ATTR_SYSTEM       0x04    システムファイル FAT_ATTR_VOLID        0x08    ボリューム名 FAT_ATTR_DIR           0x10    ディレクトリ FAT_ATTR_ARCHIVE      0x20    バックアップ要 FAT_ATTR_LONG          0x0f    ロングファイル名
13	uiMapFBlk	grp_uint32_t	ファイル領域割当て情報リストの先頭のブロック番号
14	uiMapCnt	grp_uint32_t	ファイル領域割当て情報リストの数
15	puiMap	grp_uint32_t *	ファイル領域割当て情報リスト (リスト領域へのポインタ)
16	ptFs	grp_fs_info_t *	同ファイルのファイルシステム情報へのポインタ

17	pvFileInfo	void *	ファイルシステム依存のファイル関連情報領域へのポインタ (FAT の場合、同ファイルのディレクトリ情報)
18	ptListFwd ptListBwd	grp_fs_file_t *	ファイル管理情報のリストの前後エントリへのポインタ (LRU(Least Recently Used)リスト)
19	ptHashFwd ptHashBwd	grp_fs_file_t *	ファイル管理情報のハッシュリストの前後エントリへのポインタ
20	pptHashTop	grp_fs_file_t**	ファイル管理情報のハッシュリストの先頭バケットへのポインタ

(3) grp\_fs\_fhdl\_t (ファイルハンドル)

オープンされたファイルをアクセスするためのファイルハンドル情報を保持します。

#	構造体メンバ名	タイプ	意味
1	sHdlId	short	ファイルハンドルの ID
2	usMode	grp_ushort_t	ファイルのオープン状態 GRP_FS_OPEN_EXEC            0x0001    実行可能 GRP_FS_OPEN_WRITE        0x0002    write 可能 GRP_FS_OPEN_READ        0x0004    read 可能 GRP_FS_OPEN_APPEND      0x0008    append モード GRP_FS_OPEN_PARENT      0x1000    親ディレクトリ GRP_FS_OPEN_DIRECT_IO   0x2000    ダイレクト IO
3	ptFile	grp_fs_file_t * (grp_fs_fhdl_t *)	ファイル管理情報へのポインタ (フリーエントリの場合は、次のフリーエントリへのポインタ)
4	iOffset	grp_ioffset_t	I/O 位置を示すファイル内オフセット値
5	ptTask	grp_fs_task_ctl_t *	本ファイルをオープンしたタスクのタスク管理情報へのポインタ

(4) grp\_fs\_buf\_t (キャッシュバッファ情報)

メディアに I/O するためのキャッシュバッファの情報を保持します。

#	構造体メンバ名	タイプ	意味
1	iDev	int	デバイス番号 (3.5.3節 参照)
2	iRefCnt	int	本バッファの参照数
3	usStatus	grp_ushort_t	キャッシュバッファの状態 GRP_FS_BSTAT_FBUF 0x0001 ファイル管理ブロックキャッシュ GRP_FS_BSTAT_DBUF 0x0002 ファイルデータキャッシュ GRP_FS_BSTAT_SYNC 0x0010 write through モード GRP_FS_BSTAT_TSYNC 0x0020 一時 write through モード GRP_FS_BSTAT_FILL 0x0100 データ読込み中 GRP_FS_BSTAT_MOD 0x0200 データ更新中 GRP_FS_BSTAT_DIRTY 0x0400 更新未反映 GRP_FS_BSTAT_WFAIL 0x0800 書込み失敗 GRP_FS_BSTAT_WAITMOD 0x4000 データ更新終了待ちあり GRP_FS_BSTAT_WAIT 0x8000 バッファ待ちあり
4	ptFs	grp_fs_info_t	同キャッシュのファイルシステム情報へのポインタ
5	uiBlk	grp_uint32_t	同キャッシュのブロック番号
6	iSize	grp_int32_t	キャッシュのデータサイズ
7	pucData	grp_uchar_t	キャッシュバッファ領域へのポインタ
8	ptListFwd ptListBwd	grp_fs_buf_t *	キャッシュバッファリストの前後のエントリへのポインタ (LRU リスト)
9	ptHashFwd ptHashBwd	grp_fs_buf_t *	キャッシュバッファのハッシュリストの前後のエントリへのポインタ
10	pptHashTop	grp_fs_buf_t**	キャッシュバッファのハッシュリストの先頭のバケットへのポインタ

## (5) grp\_fs\_bio\_t (バッファ I/O 情報)

キャッシュバッファを介したバッファリング I/O のための制御情報(I/O 結果)を保持します。

#	構造体メンバ名	タイプ	意味
1	ptBuf	grp_fs_buf_t *	I/O 結果を保持したキャッシュバッファ情報へのポインタ
2	pucData	grp_uchar_t *	I/O 結果のデータを保持した領域へのポインタ
3	uiSize	grp_uint32_t	I/O できたデータのサイズ
4	uiBlk	grp_uint32_t	I/O したブロックのブロック番号

## (6) grp\_fs\_task\_ctl\_t (GR-FILE 固有のタスク管理情報)

GR-FILE を介してファイルをアクセス中のタスクの管理情報を保持します。

#	構造体メンバ名	タイプ	意味
1	tTaskId	grp_fs_task_t	タスク ID
2	tTaskSem	grp_fs_sem_t	タスク管理情報の排他制御のためのセマフォ ID
3	pvWaitRsc	void *	処理終了待ち中のリソース
4	iOpenCnt	int	同タスクでオープン中のファイルの数
5	ptCurDir	grp_fs_file_t *	カレントディレクトリのファイル管理情報へのポインタ
6	ptTaskListFwd ptTaskListBwd	grp_fs_task_ctl_t *	タスク管理情報リストの前後のエントリへのポインタ

## (7) grp\_fs\_ctl\_t (GR-FILE 全体の管理情報)

GR-FILE システム全体の管理情報を保持します。

#	構造体メンバ名	タイプ	意味
1	tFsSem	grp_fs_sem_t	GR-FILE 処理全体の排他制御のためのセマフォ ID
2	iFsStat	int	GR-FILE の状態 GRP_FS_CSTAT_BUSY 0x0001 処理中 GRP_FS_CSTAT_FBWAIT 0x0100 ファイル管理ブロックキャッシュバッファの空き待ち GRP_FS_CSTAT_DBWAIT 0x0200 ファイルデータキャッシュバッファの空き待ち GRP_FS_CSTAT_WAIT 0x0400 mount/unmount 待ち
3	iFsWaitCnt	int	GR-FILE 内部で処理待ち中のタスク数
4	ptFsFree	grp_fs_info_t *	フリーなファイルシステム情報のリストへのポインタ
5	ptFsMnt	grp_fs_info_t *	ルートレベルに mount されたファイルシステムのファイルシステム情報リストへのポインタ
6	ptFsDefault	grp_fs_info_t *	デフォルトファイルシステムのファイルシステム情報へのポインタ
7	ptFileFwd ptFileBwd	grp_fs_file_t *	ファイル管理情報の LRU リストの先頭/最終エントリへのポインタ
8	ptDBufFwd ptDBufBwd	grp_fs_buf_t *	ファイルデータキャッシュバッファの LRU リストの先頭/最終エントリへのポインタ
9	ptFBufFwd ptFBufBwd	grp_fs_buf_t *	ファイル管理ブロックキャッシュバッファの LRU リストの先頭/最終エントリへのポインタ
10	ptTask	grp_fs_task_ctl_t *	アクティブなタスク管理情報のリストへのポインタ
11	ptTaskFree	grp_fs_task_ctl_t *	フリーなタスク管理情報のリストへのポインタ
12	ptFhdlTbl	grp_fs_fhdl_t *	ファイルハンドル情報テーブルの先頭エントリへのポインタ
13	ptFhdlFree	grp_fs_fhdl_t *	フリーなファイルハンドル情報リストへのポインタ
14	pptBufHash	grp_fs_buf_t **	キャッシュバッファハッシュバケットの先頭エントリへのポインタ

15	pptFileHash	grp_fs_file_t **	ファイル情報ハッシュバケットの先頭エントリへのポインタ
16	pvTblMem	void *	システム起動時に確保した管理情報用領域の先頭アドレス
17	pvBufMem	void *	システム起動時に確保したキャッシュバッファ領域の先頭アドレス

## (8) grp\_fs\_fname\_cache\_t (ファイル名称キャッシュ情報)

ファイル名称キャッシュ情報を保持します。

#	構造体メンバ名	タイプ	意味
1	iDev	int	デバイス番号 (3.5.3 節 参照)
2	uiDirFid	grp_uint32_t	親ディレクトリのファイル ID
3	uiFid	grp_uint32_t	ファイル ID
4	sNameLen	short	ファイル名称の長さ
5	sNameBufLen	short	ファイル名称バッファのサイズ
6	pucName	grp_uchar_t *	ファイル名称バッファ
7	ptAlias	grp_fs_fname_cache_t *	別名がある場合のファイル名称キャッシュエントリへのポインタ
8	ptListFwd ptListBwd	grp_fs_fname_cache_t *	ファイル名称キャッシュの LRU リスト
9	ptHashFwd ptHashBwd	grp_fs_fname_cache_t *	ファイル名称キャッシュのハッシュリスト
10	pptHashTop	grp_fs_fname_cache_t *	ファイル名称キャッシュのハッシュバケットへのポインタ

## 4.2.4 FAT ファイルシステム依存部の管理構造体

FAT ファイルシステム依存部の管理構造体のタイプ定義を以下に示します。これらの定義は、"fat.h" に定義されています。

## (1) fat\_BPB\_t (FAT ファイルシステムのブートパラメータブロック情報)

FAT のブートパラメータブロックの情報を保持します。

#	構造体メンバ名	タイプ	意味
1	aucJumpBoot	grp_uchar_t[]	ブートプログラムへのジャンプコード
2	aucOEMName	grp_uchar_t[]	OEM 名
3	ucFatCnt	grp_uchar_t	FAT 管理領域の数
4	ucMedia	grp_uchar_t	メディアのタイプ
5	aucFsVersion	grp_uchar_t[]	ファイルシステムのバージョン
6	ucSecPerClst	grp_uchar_t	クラスタ当りのセクタ数
7	usBytePerSec	grp_ushort_t	セクタ当りのバイト数
8	usRsvSecCnt	grp_ushort_t	予約セクタのセクタ数
9	usRootEntCnt	grp_ushort_t	ルートディレクトリのディレクトリエントリ数
10	usSecPerTrk	grp_ushort_t	トラック当りのセクタ数
11	usNumHeads	grp_ushort_t	ヘッドの数
12	usExtFlags	grp_ushort_t	拡張フラグ
13	usFsInfo	grp_ushort_t	トータルクラスタ数とフリークラスタのヒント情報を保持したエリアのセクタ番号
14	usBackupBootSec	grp_ushort_t	ブートパラメータブロックのバックアップのセクタ番号
15	uiFatSz	grp_uint32_t	FAT 管理領域のセクタ数
16	uiTotalSec	grp_uint32_t	トータルセクタ数
17	uiHidenSec	grp_uint32_t	隠しセクタの数
18	uiRootClst	grp_uint32_t	ルートディレクトリのクラスタ番号
19	ucDrvNum	grp_uchar_t	ドライブ番号
20	ucBootSig	grp_uchar_t	boot 識別情報 (0x29)
21	aucVolSer	grp_uchar_t[]	ボリュームのシリアル番号
22	aucVolLab	grp_uchar_t[]	ボリューム名
23	aucFsType Name	grp_uchar_t[]	ファイルシステムタイプ名称
24	usBPBSig	grp_ushort_t	ブートパラメータブロック識別情報 (0xaa55)
25	uiStatus	grp_uint32_t	ファイルシステムの状態 FAT_STAT_RDONLY      0x00000001   read only FAT_STAT_SYNC_ALL    0x00000002   write through 方式 FAT_STAT_IO_ERR      0x00000004   write エラー FAT_STAT_FREE_CHKED 0x00000008   フリークラスタチェック済 FAT_STAT_MOD        0x00010000   更新処理中 FAT_STAT_WAIT_MOD    0x00020000   更新処理終了待ちあり
26	uiMaxClst	grp_uint32_t	最大クラスター番号
27	uiNextFree	grp_uint32_t	フリークラスタのヒント
28	uiFreeHint	grp_uint32_t	FAT32 の残りクラスタのヒント情報として現在メディアに格納されている値
29	uiEOC	grp_uint32_t	クラスタリストの終わりの値
30	uiEOF	grp_uint32_t	EOF と判定するためのクラスタ値の境界値
31	uiBadC	grp_uint32_t	不良クラスタ用のクラスタ値
32	iRootCTime	grp_int32_t	ルートディレクトリの作成時刻
33	iRootMTime	grp_int32_t	ルートディレクトリの更新時刻

34	iRootATime	grp_int32_t	ルートディレクトリのアクセス時刻
35	iFsType	int	ファイルシステムタイプ FAT_TYPE_12 1 FAT12 FAT_TYPE_16 2 FAT16 FAT_TYPE_32 3 FAT32
36	iClstShift	int	クラスタサイズの 2 のべき乗数値
37	iDBlkShift	int	ファイルデータキャッシュのブロックサイズの 2 のべき乗数値
38	iFBlkShift	int	ファイル管理ブロック (FAT 管理領域) キャッシュのブロックサイズの 2 のべき乗数値
39	iDBlkClstShift	int	クラスタ当りのファイルデータキャッシュブロック数の 2 のべき乗数値
40	uiClstSize	grp_uint32_t	クラスタサイズ
41	uiDBlkSize	grp_uint32_t	ファイルデータキャッシュのブロックサイズ
42	uiFBlkSize	grp_uint32_t	ファイル管理ブロック (FAT 管理領域) キャッシュのブロックサイズ
43	uiDBlkStart	grp_uint32_t	FAT 領域の終わりを基点とした、ファイルデータブロックのメディア内での開始位置 キャッシュブロックサイズ単位のルートディレクトリのサイズが入る FAT32 では 0 固定
44	uiFatStart	grp_uint32_t	ファイル管理ブロック (FAT 管理領域) のメディア内での開始位置
45	iByteSecShift	int	セクタ当りのバイト数の 2 のべき乗数値
46	iFreeTblCnt	int	フリークラスタテーブルに載っているフリークラスタの数
47	iFreeGetIdx	int	フリークラスタテーブルからのフリークラスタの取出しのためのインデクス
48	iFreePutIdx	int	フリークラスタテーブルへのフリークラスタ登録のためのインデクス
49	auiFreeTbl	grp_uint32_t[]	フリークラスタテーブル

## (2) fat\_open\_info\_t (FAT 依存のオープンファイル管理情報)

オープン中のファイルについて、FAT ファイルシステム依存のオープンファイル管理情報を保持します。  
具体的には、オープン中のファイルのディレクトリ関連情報を保持します。

#	構造体メンバ名	タイプ	意味
1	iDev	int	デバイス番号
2	uiUniqFid	grp_uint32_t	サイズ 0 ファイルの場合の一時的なユニークなファイル ID
3	uiDirFid	grp_uint32_t	同ファイルのディレクトリエントリを含むディレクトリファイルのファイル ID
4	uiDirBlk	grp_uint32_t	同ファイルのディレクトリエントリのブロック番号
5	uiDirStart	grp_uint32_t	同ファイルのディレクトリエントリのディレクトリファイルファイル内での開始オフセット
6	uiDirEnd	grp_uint32_t	同ファイルのディレクトリエントリのディレクトリファイルファイル内での終了オフセット
7	uiMap	grp_uint32_t[]	ファイル領域情報キャッシュリスト領域へのポインタ
8	ptSz0Fwd ptSz0Bwd	fat_open_info_t *	サイズ 0 ファイルのリスト、または、 FAT 依存のオープンファイル管理情報のフリーリスト



## (3) fat\_open\_info\_ctl\_t (FAT 依存のオープンファイル管理情報の管理情報)

上記 FAT ファイルシステム依存のオープンファイル管理情報 fat\_open\_info\_t を管理するための情報を保持します。

#	構造体メンバ名	タイプ	意味
1	ptOpenInfo	fat_open_info_t *	FAT ファイルシステム依存のオープンファイル管理情報テーブルの先頭アドレス
2	ptOpenFree	fat_open_info_t *	FAT ファイルシステム依存のオープンファイル管理情報テーブルのフリーリスト (ptSz0Fwd フィールドを利用)
3	ptSize0List	fat_open_info_ *	オープン中のサイズ 0 ファイルのオープンファイル管理情報リスト

### 4.3 POSIX 互換アプリケーションインタフェース

3. 2 節で説明しましたとおり、**GR-FILE** では、Windows の “io.h” や LINUX の “fcntl.h” 等の代わりに “include” ディレクトリ下にある “grp\_fs\_conv.h” をアプリケーションプログラムでインクルードすることで、POSIX 互換名称で記述した I/O 関数名、I/O 定数名を、**GR-FILE** の実際の名称に変換し、POSIX 互換インタフェースでのプログラミングを可能としています。ただし、コンパイルオプション **GRP\_FS\_MINIMIZE\_LEVEL** を 1 または 2 に設定した場合、POSIX 互換のアプリケーションインタフェースは使用できません。

なお、前述のとおり、エラー時のリターン値が異なる他、関数によっては、サポートしていないオプションがあります。

POSIX 互換のアプリケーションインタフェースの一覧を表 4-3 に示します。

表 4-3 POSIX 互換アプリケーションインタフェース一覧

#	関数名	機能	関連実関数名
1	chdir	・カレントディレクトリの設定、および、無効化	grp_fs_chdir
2	chmod	・ファイル保護モードの設定・変更	grp_fs_chmod
3	close	・ファイルのクローズ	grp_fs_close
4	closedir*1	・opendir でオープンしたディレクトリのクローズ	grp_fs_closedir
5	creat	・ファイルの作成	grp_fs_open
6	ftruncate	・指定サイズ以降のファイル領域の解放	grp_fs_ftruncate
7	getcwd	・カレントディレクトリ名の取得	grp_fs_get_cwd
8	lseek	・ファイルの読書き位置の設定・変更	grp_fs_lseek
9	mkdir	・ディレクトリの作成	grp_fs_create
10	open	・ファイルのオープン	grp_fs_open
11	opendir*1	・readdir でディレクトリエントリ読み出しを行うためのディレクトリのオープン	grp_fs_opendir
12	read	・ファイルの読み込み	grp_fs_read
13	readdir*1*2	・ディレクトリエントリの読み出し	grp_fs_readdir
14	rename	・ファイル/ディレクトリ名称の変更	grp_fs_rename
15	rmdir	・ディレクトリの削除	grp_fs_unlink
16	stat	・ファイル属性情報の取得 コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合は stat を grp_fs_stat0 に変換するのではなく stat0 API が定義され使用されます。	grp_fs_stat
17	sync	・未反映キャッシュデータのメディアへの書戻し	grp_fs_sync
18	unlink	・ファイルの削除	grp_fs_unlink
19	utimes	・ファイル/ディレクトリのアクセス・更新時刻情報の変更	grp_fs_utimes
20	write	・ファイルの書き込み	grp_fs_write

\*1 これらの関数の型宣言は、“include” ディレクトリ下の “grp\_fs\_readdir.h” に含まれています。

\*2 コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、メンバ変数 d\_size の型が符号なし 32 ビットに変更されます。

表 4-4 に示す POSIX 互換アプリケーション向け 4G 対応アプリケーションインタフェースはコンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に定義される POSIX 互換アプリケーション向けの 4G-1 バイト対応アプリケーションインタフェースです。

表 4-4 POSIX 互換アプリケーション向け 4G 対応アプリケーションインタフェース一覧

#	関数名	機能	関連実関数名
1	lseek4G	・ファイルの読書き位置の設定・変更 4G 対応版 コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合のみ定義	grp_fs_lseek4G

また、“grp\_fs\_conv.h”では、“GRP\_MEM”を define して include しますと、以下のメモリ取得・解放関数名変換も同時に行われます。

- (1) malloc → grp\_mem\_alloc
- (2) free → grp\_mem\_free

これらのメモリ取得・解放関数につきましては、単純な名称変換で、文法・機能とも、標準のインタフェースと同じです。

## 4.3.1 chdir

【機能概要】 カレントディレクトリの設定

【使用形式】

```
#include "grp_fs_conv.h"
int  chdir(const char  *pcDir) ;
```

【パラメータ】

pcDir	入力	カレントディレクトリのパス名 or  NULL
-------	----	-------------------------

【機能詳細】

コールしたタスクのカレントディレクトリを pcDir で指定したパス名に設定します。pcDir に NULL または、長さ 0 の文字列 (“”) を指定した場合は、コールしたタスクのカレントディレクトリを無効化します。

なお、chdir を用いてカレントディレクトリを設定した場合は、ファイルシステムを unmount する前に、すべてのタスクのカレントディレクトリを同ファイルシステム外に設定するか、無効化する必要があります。unmount しようとするファイルシステム上にカレントディレクトリを持つタスクが存在しますと、GRP\_FS\_ERR\_BUSY のエラーで、unmount 処理がエラーとなります。無効化する前に強制 unmount を行った場合でも、カレントディレクトリを無効化する必要があります。無効化を行わない場合、mount 後の処理が正常に行えない場合があります。

また、カレントディレクトリを設定しない状態、あるいは、無効化した状態でも、**GR-FILE** の各処理関数は使用可能です。その場合、ファイル名の指定はフルパスで行なう必要があります。

【リターン値】

0	正常終了
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したディレクトリが存在しない
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスクの数がシステムの上限值を超えた
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_chdir((grp_uchar_t *)pcDir)
```

## 4.3.2 chmod

【機能概要】 ファイルの保護モードの設定・変更

【使用形式】

```
#include "grp_fs_conv.h"
int  chmod(const char *  pcPath,  grp_uint32_t  uiProt) ;
```

【パラメータ】

pcPath	入力	ファイル保護モードを設定・変更するファイル/ディレクトリ名
uiProt	入力	設定するファイルの保護モード（以下のモードの bit or で指定）
S_IRUSR	0400	ファイルオーナー read 可能
S_IWUSR	0200	ファイルオーナー write 可能
S_IXUSR	0100	ファイルオーナー実行可能
S_IRGRP	0040	ファイルオーナーグループ内 read 可能
S_IWGRP	0020	ファイルオーナーグループ内 write 可能
S_IXGRP	0010	ファイルオーナーグループ内実行可能
S_IROTH	0004	他のユーザ read 可能
S_IWOTH	0002	他のユーザ write 可能
S_IXOTH	0001	他のユーザ実行可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが EXE”、“COM”、“DLL”、“BAT” の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、S\_IRUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pcPath で指定したファイル/ディレクトリの保護モードを uiProt に従い設定・変更します。

なお、FAT ファイルシステムの隠しファイル属性等、ファイルシステム固有の属性を設定する場合は、grp\_fs\_set\_attr を用いて行って下さい。

【リターン値】

0	正常終了
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_PERMIT	親ディレクトリに対する write が許可されていない
GRP_FS_ERR_BAD_PARAM	pcPath で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_chmod((grp_uchar_t *)pcPath, uiProt)
```

## 4.3.3 close

【機能概要】 ファイルのクローズ

【使用形式】

```
#include "grp_fs_conv.h"
int close(int iFhdl);
```

【パラメータ】

iFhdl	入力	open 関数等で返ってきたファイルハンドル
-------	----	------------------------

【機能詳細】

open 関数等でオープンしたファイルをクローズします。

なお、タスク毎にオープンしたファイルが管理されていますので、ファイルをオープンした場合は、オープンしたタスクが本 close 関数を実行して必ずクローズ処理をするか、あるいは、タスクの消滅時に、OS または、別のタスクが、消滅したタスクのオープンファイルの無効化処理を grp\_fs\_task\_free\_env\_by\_id 等を使用して行って下さい。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_close(iFd)
```

#### 4.3.4 closedir

【機能概要】 opendir でオープンしたディレクトリのクローズ

【使用形式】

```
#include "grp_fs_readdir.h"
int closedir(DIR *ptDirHdl);
```

【パラメータ】

ptDirHdl	入力	opendir 関数等で返ってきたファイルハンドル
----------	----	---------------------------

【機能詳細】

opendir 関数でオープンしたディレクトリファイルをクローズします。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_closedir(ptDirHdl)
```



## 4.3.5 creat

【機能概要】 ファイルの作成

【使用形式】

```
#include "grp_fs_conv.h"
int creat(const char *pcFile, grp_uint32_t uiProt);
```

【パラメータ】

pcFile	入力	作成するファイルのパス名
uiProt	入力	ファイルの保護モード
S_IRUSR	0400	ファイルオーナー read 可能
S_IWUSR	0200	ファイルオーナー write 可能
S_IXUSR	0100	ファイルオーナー実行可能
S_IRGRP	0040	ファイルオーナーグループ内 read 可能
S_IWGRP	0020	ファイルオーナーグループ内 write 可能
S_IXGRP	0010	ファイルオーナーグループ内実行可能
S_IROTH	0004	他のユーザ read 可能
S_IWOTH	0002	他のユーザ write 可能
S_IXOTH	0001	他のユーザ実行可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、ファイル名のサフィックスが EXE”、“COM”、“DLL”、“BAT” の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、S\_IRUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pcFile で指定したファイルを uiProt で指定した保護モードで作成し、write 処理のためのファイルハンドルを返します。指定したファイルが既に存在する場合は、同ファイルのデータを削除し、ファイルを空の状態にします。

なお、FAT ファイルシステムの隠しファイル属性等、ファイルシステム固有の属性でファイルを作成する場合は、grp\_fs\_create を用いて行って下さい。

【リターン値】

0 または 正值	write 処理のためのファイルハンドル
GRP_FS_ERR_PERMIT	親ディレクトリまたは同ファイルに対して write 権限がない
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスク数がシステムの規定値を超えた
GRP_FS_ERR_NOT_FOUND	指定したパス中のディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcFile で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 【変換後の実際の実行形式】

```
grp_fs_open((const grp_uchar_t *) pcFile,  
            GRP_FS_O_CREAT|GRP_FS_O_TRUNC|GRP_FS_O_WRONLY, uiProt)
```

## 4.3.6 ftruncate

【機能概要】 指定サイズ以降のファイル領域の解放

【使用形式】

```
#include "grp_fs_conv.h"
int ftruncate(int iFd, grp_uint32_t uiOffset);
```

【パラメータ】

iFd	入力	open 関数等で返ってきたファイルハンドル
uiOffset	入力	ファイル領域を解放する開始オフセット

【機能詳細】

iFd で指定されたファイルのファイルサイズを、uiOffset で指定されたサイズに切り詰めます。

切り詰められたファイルのファイルサイズは uiOffset となります。uiOffset 以降の領域は解放されます。

uiOffset にファイルサイズ以上の値を指定した場合はエラーとなります。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	同ファイルに対して <b>write</b> 権限がない
GRP_FS_ERR_BAD_OFF	uiOffset の値が正しくない（ファイルサイズを超えている）
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

【変換後の実際の実行形式】

```
grp_fs_truncate(iFd, uiOffset)
```

## 4.3.7 getcwd

【機能概要】 カレントディレクトリ名の取得

【使用形式】

```
#include "grp_fs_conv.h"
char *getcwd(char *pcBuf, int iSize);
```

【パラメータ】

pcBuf	出力	取得したカレントディレクトリ名を格納する領域のアドレス
iSize	入力	上記格納領域のサイズ

【機能詳細】

本関数をコールしたタスクのカレントディレクトリ名を pcBuf で指定した領域に返し、pcBuf で指定した値をそのままリターン値として返します。

なお、カレントディレクトリが設定されていない場合は、既にマウントされているファイルシステムがあれば、最初にマウントされたファイルシステムのルートへのパスを返します。マウントされているファイルシステムがない場合は、NULL を返します。また、各コンポーネント長が、**GR-FILE** の config パラメータ GRP\_FS\_MAX\_COMP-1 よりも長い場合は、GRP\_FS\_MAX\_COMP-1 の長さで切り捨てられ、pcBuf に返されます。

iSize で指定したサイズ内にカレントディレクトリ情報が収まらない場合や、その他のエラーが発生した場合も、リターン値として NULL を返します。

【リターン値】

NULL	カレントディレクトリが設定されておらず、かつ、マウントされているファイルシステムがない、あるいは、iSize で指定したサイズ内にカレントディレクトリ情報が収まらない、あるいは、その他エラーが発生した
その他 (pcBuf)	カレントディレクトリ名の入った領域へのポインタ

【変換後の実際の実行形式】

```
((grp_fs_get_cwd((grp_uchar_t *)pcBuf, iSize, '/') != 0)? NULL: (pcBuf))
```

## 4.3.8 lseek

【機能概要】 ファイルの読書き位置の設定・変更

【使用形式】

```
#include "grp_fs_conv.h"
grp_ioffset_t lseek(int iFd, grp_ioffset_t iOffset, int iMode);
```

【パラメータ】

iFd	入力	open 関数等で返ってきたファイルハンドル
iOffset	入力	ファイルの読書き位置情報
iMode	入力	ファイルの読書き位置情報の解釈方法
SEEK_SET		uiOffset で指定した値を読書き位置に設定
SEEK_CUR		現在のファイル読書き位置+iOffset の値を読書き位置に設定 (O_APPEND モードでオープンされている場合は、SEEK_END と同じ)
SEEK_END		現在のファイルサイズ+iOffset の値を読書き位置に設定

【機能詳細】

iFd で指定したファイルの read/write 時に使用するファイルの読書き位置を iOffset、iMode に従い設定し、設定した読書き位置を返します。

なお、現在のファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、write 時にエラーとなります。また、読書き位置が負値になった場合は、0 に自動補正します。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と lseek4G などの 4G 対応 API を混在して使用しないでください。lseek4G などで 2GB 以上のオフセットに移動したのちに本 API を使用し、オフセットが 2GB 以上になった場合、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0 または 正値	設定した読書き位置
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_BAD_PARAM	iMode パラメータが正しくない

【変換後の実際の実行形式】

```
grp_fs_lseek(iFd, iOffset, iMode)
```

## 4.3.9 lseek4G

【機能概要】 ファイルの読書き位置の設定・変更 4G 対応版

【使用形式】

```
#include "grp_fs_conv.h"
int lseek4G(int iFd, grp_uioffset_t uiOffset, int iMode,
            grp_uioffset_t *puiResultOffset);
```

【パラメータ】

iFd	入力	open 関数等で返ってきたファイルハンドル
uiOffset	入力	ファイルの読書き位置情報
iMode	入力	ファイルの読書き位置情報の解釈方法
SEEK_SET		uiOffset で指定した値を読書き位置に設定
SEEK_CUR		現在のファイル読書き位置+uiOffset の値を読書き位置に設定 (O_APPEND モードでオープンされている場合は、SEEK_END と同じ)
SEEK_END		現在のファイルサイズ+uiOffset の値を読書き位置に設定
SEEK_MINUS		読書き位置の算出の際に uiOffset を負値として扱う。 (本モードのみでの指定は不可。SEEK_CUR/SEEK_END と bit or で指定)
puiResultOffset	出力	設定後のファイルの読書き位置情報

【機能詳細】

本機能は、コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に使用できます。

iFd で指定したファイルの read/write 時に使用するファイルの読書き位置を uiOffset, iMode に従い設定し、puiResultOffset に設定した読書き位置を返します。

なお、現在のファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、read/write 時にエラーとなります。

また、読書き位置にオーバーフローもしくはアンダーフローが発生した場合には、エラーを返します。(この場合のオーバーフローは 4G-1 バイトを超えるオフセットを指定した場合を指し、アンダーフローは 0 バイトを下回るオフセットを指定した場合を指します)

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と lseek などの 4G 未対応 API を混在して使用しないでください。本 API で 2GB 以上のオフセットに移動したのちに lseek などを使用し、オフセットが 2GB 以上になった場合、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない

GRP\_FS\_ERR\_BAD\_PARAM    iMode パラメータが正しくない  
GRP\_FS\_ERR\_BAD\_OFF      uiOffset パラメータが正しくない

【変換後の実際の実行形式】

grp\_fs\_lseek4G(iFd, uiOffset, iMode, \*puiResultOffset)



## 4.3.1 0 mkdir

【機能概要】 ディレクトリの作成

【使用形式】

```
#include "grp_fs_conv.h"
int mkdir(const char *pcDir, grp_uint32_t uiProt);
```

【パラメータ】

pcDir	入力	作成するディレクトリのパス名
uiProt	入力	ディレクトリの保護モード
S_IRUSR	0400	オーナー read 可能
S_IWUSR	0200	オーナー write 可能
S_IXUSR	0100	オーナーサーチ可能
S_IRGRP	0040	グループ内 read 可能
S_IWGRP	0020	グループ内 write 可能
S_IXGRP	0010	グループ内サーチ可能
S_IROTH	0004	他のユーザ read 可能
S_IWOTH	0002	他のユーザ write 可能
S_IXOTH	0001	他のユーザサーチ可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、mkdir では、常に 1 に設定します。また、read 不可の概念がないため、S\_IRUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pcDir で指定したディレクトリを uiProt で指定した保護モードで作成します。

なお、FAT ファイルシステムの隠しファイル属性等、ファイルシステム固有の属性でファイルを作成する場合は、grp\_fs\_create を用いて行って下さい。

【リターン値】

0	作成成功
GRP_FS_ERR_PERMIT	親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	指定したパス中のディレクトリが存在しない
GRP_FS_ERR_EXIST	作成しようとしたディレクトリが既に存在する
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスク数がシステムの規定値を超えた
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcDir で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 【変換後の実際の実行形式】

```
grp_fs_create((const grp_uchar_t *) pcDir,  GRP_FS_FILE_DIR,  uiProt,  0)
```

## 4.3.1 1 open

【機能概要】 ファイルのオープン

【使用形式】

```
#include "grp_fs_conv.h"
int open(const char *pcFile, int iMode, grp_uint32_t uiProt);
```

【パラメータ】

pcFile	入力	オープン/生成するファイルのパス名
iMode	入力	オープンモード (O_APPEND 以下のモードは、bit or で指定可能)
O_RDONLY	0x0000	read のみのモードでオープン
O_WRONLY	0x0001	write のみのモードでオープン
O_RDWR	0x0002	read/write 両方が可能なモードでオープン
O_APPEND	0x0008	ファイルへの追加 write モードでオープン (write 要求毎に、常にファイルの最後に位置づけて処理)
O_CREAT	0x0200	存在しない場合、ファイルを生成
O_TRUNC	0x0400	オープン後、ファイルを空の状態にする
O_EXCL	0x0800	O_CREAT が指定され、かつ、既に同ファイルが存在する場合エラーで返す。
GRP_FS_O_DIRECT_IO	0x8000	キャッシュ上に対応するデータがない場合は、可能な限りキャッシュバッファを使用せず、アプリケーションバッファとメディアとの間で直接、かつ、連続したブロックで I/O を行う。
uiProt	入力	作成するファイルの保護モード (O_CREAT を指定した場合)
S_IRUSR	0400	ファイルオーナー read 可能
S_IWUSR	0200	ファイルオーナー write 可能
S_IXUSR	0100	ファイルオーナー実行可能
S_IRGRP	0040	ファイルオーナーグループ内 read 可能
S_IWGRP	0020	ファイルオーナーグループ内 write 可能
S_IXGRP	0010	ファイルオーナーグループ内実行可能
S_IROTH	0004	他のユーザ read 可能
S_IWOTH	0002	他のユーザ write 可能
S_IXOTH	0001	他のユーザ実行可能

FAT ファイルシステムの場合は、ユーザの概念がありません。その為「オーナー」、「グループ」、「他のユーザ」の保護モードはオーナービット値の指定で統一されます。この時、「グループ」、「他のユーザ」の指定は無視されます。但し、隠しファイルの場合は、「グループ」、「他のユーザ」ビットを自動的に 0 に設定します。さらに、FAT ファイルシステムには実行可否の概念がなく、情報も保持できないため、「実行可能」ビットを指定しても無視されます。「実行可能」ビットは、ファイル名のサフィックスが EXE、COM、DLL、BAT の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、GRP\_FS\_PROT\_RUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pcFile で指定したファイルをオープン/生成し、read/write 処理のためのファイルハンドルを返します。ファイル生成等のオープン時の処理モードや、read のみ、write のみ、read/write 両方等のオープン後の処理モードを iMode で指定します。iMode では、O\_RDONLY、O\_WRONLY、O\_RDWR

の何れか 1 つを指定して、read/write の区別を指定し、その他のモードは、オプションとして or の形で追加で指定します。O\_RDONLY、O\_WRONLY、O\_RDWR のいずれも指定しなかった場合は、O\_RDONLY が仮定されます。O\_APPEND を指定しても、O\_WRONLY、または、O\_RDWR を指定していなかった場合は、write できないことに注意して下さい。

また、iMode で O\_CREAT を指定した場合の生成するファイルの保護モードを uiProt で指定します。なお、iMode に O\_CREAT を指定しない場合も、uiProt を指定する必要がありますが、その場合、uiProt の値は、無視されます。

pcFile には、通常のファイル以外に、ディレクトリを指定することも可能です。但し、ディレクトリに対しては、O\_RDONLY モードでしかオープンできません。

GRP\_FS\_O\_DIRECT\_IO は、**GR-FILE** 固有のオプションで、キャッシュ上に対応するデータがない場合は、可能な限りキャッシュバッファを使用せず、アプリケーションの指定したバッファとメディアとの間で直接 I/O を行い、しかもメディア上で連続したブロックは一括して I/O を行なう指定です。本オプションは、一回の read/write 要求のサイズが大きく、頻繁には read しないようなファイルの I/O を高速に行うのに適したオプションです。なお、本オプションは、ファイルシステム依存部でサポートされている場合のみ有効です。**GR-FILE** の FAT ファイルシステムでは、シングル空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合に有効となります。

GRP\_FS\_O\_DIRECT\_IO には使用上の注意点があります。

詳しくは、本マニュアルの「アプリケーションバッファとメディア間の直接/連続ブロック I/O (ダイレクト I/O 機能)」を参照下さい。

#### 【リターン値】

0 または 正值	read/write 処理のためのファイルハンドル
GRP_FS_ERR_PERMIT	O_CREATE を指定した場合で、親ディレクトリまたは同ファイルに対して write 権限がない
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスク数がシステムの規定値を超えた
GRP_FS_ERR_NOT_FOUND	指定したパス中のディレクトリが存在しない
	O_CREAT 指定がなく、指定したファイルが存在しない
GRP_FS_ERR_EXIST	O_CREAT かつ O_EXCL 指定があり、指定したファイルが既に存在する
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcFile で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

#### 【変換後の実際の実行形式】

```
grp_fs_open((const grp_uchar_t *) pcFile, iMode, uiProt)
```

## 4.3.1 2 opendir

【機能概要】 `readdir` でディレクトリエントリ読み出しを行うためのディレクトリのオープン

【使用形式】

```
#include "grp_fs_readdir.h"
DIR *opendir(grp_uchar_t *pucPath);
```

【パラメータ】

<code>pucPath</code>	入力	オープンするディレクトリファイル名称
----------------------	----	--------------------

【機能詳細】

POSIX 互換インタフェースである `readdir` を使ってディレクトリエントリの読み出しを行うためのディレクトリファイルのオープンを行います。`pucPath` パラメータでオープンするディレクトリファイル名を指定します。

【リターン値】

NULL	オープン失敗
NULL 以外	<code>readdir</code> 、 <code>closedir</code> で使用するファイルハンドル情報

【変換後の実際の実行形式】

```
grp_fs_opendir(pucPath)
```

## 4.3.13 read

【機能概要】 ファイルの読み込み

【使用形式】

```
#include "grp_fs_conv.h"
grp_isize_t read(int iFd, char *pcBuf, grp_isize_t iSize);
```

【パラメータ】

iFd	入力	open 関数等で返ってきたファイルハンドル
pcBuf	出力	読込んだデータを格納する領域のアドレス
iSize	入力	読込むデータのサイズ

【機能詳細】

iFd で指定したファイルから、iSize バイト分データを読み込み、pcBuf で指定した領域に格納します。指定したファイルの現在の読書き位置から同ファイルの最後まで残りバイト数が iSize 分より少ない場合は、残りバイト数分だけを指定した領域に格納します。

リターン値としては、実際に読めたバイト数を返します。

なお、オープン時に GRP\_FS\_O\_DIRECT\_IO の指定がなかった場合は、キャッシュバッファを介して read 処理を行い、キャッシュ上にあれば、実際にメディアからは read せず、キャッシュからデータを返します。GRP\_FS\_O\_DIRECT\_IO 指定があり、ファイルシステム依存部で同オプションをサポートしている場合も、キャッシュ上にある場合の動作は同じですが、対象のデータがキャッシュ上にない場合は、可能な限りキャッシュバッファを使用せず、メディアからアプリケーションの指定したバッファに対し、直接連続ブロックで I/O を行います。

**GR-FILE** の FAT ファイルシステムでは、シングল空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファへの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続していれば、連続した単位で、一括してメディアから read します。

【リターン値】

0 または 正值	実際に読めたバイト数
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	read 可能なモードでオープンされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

## 【変換後の実際の実行形式】

```
grp_fs_read(iFd, (grp_uchar_t *)pcBuf, iSize)
```



## 4.3.1 4 readdir

【機能概要】 ディレクトリエントリの読み出し

【使用形式】

```
#include "grp_fs_readdir.h"
struct dirent *readdir(DIR *ptDirHdl);
```

【パラメータ】

ptDirHdl          入力                  opendir 関数で返ってきたファイルハンドル

【機能詳細】

ptDirHdl で指定したディレクトリファイルから、次の空でないディレクトリエントリを読み出し、読み出した情報へのポインタをリターン値として返します。返すディレクトリ情報の構造は、以下のとおりです。

```
struct dirent {
    int          d_dev;          /* デバイス番号 (3.5.3 節 参照)          */
    int          d_type;         /* ファイルタイプ                          */
                                /* DT_UNKNOWN (0)   (ボリュームラベル等)  */
                                /* DT_REG (1)   ファイル(ショート名称)      */
                                /* DT_DIR (2)   ディレクトリ(ショート名称)  */
                                /* DT_LINK (3)  リンクファイル(ロング名称) */
    grp_uint32_t d_ino;          /* ファイル番号                          */
    grp_int32_t  d_size          /* ファイルサイズ                        */
                                /* コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を無効にした場合の定義です。 */
                                /* 符号付 32 ビット変数型で定義されます。 */
    grp_uint32_t d_size          /* ファイルサイズ                        */
                                /* コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合の定義です。 */
                                /* 符号なし 32 ビット変数型で定義されます。 */
    int          d_reclen;       /* ファイル名の長さ                      */
    char         d_name[NAME_MAX]; /* ファイル名称バッファ                  */
};
```

FAT の場合、ロング名称エントリの d\_ino、d\_size フィールドは、0 が設定されています。同エントリに対応したファイルのファイル番号 (クラスタ番号)、ファイルサイズ情報は、次の readdir で得られる同ロング名称に対応したショート名称のエントリの d\_ino、d\_size フィールドの値を参照して下さい。

d\_name にはファイル名称が返されますが、ファイル名称が **GR-FILE** の config パラメータ GRP\_FS\_MAX\_COMP-1 よりも長い場合は、GRP\_FS\_MAX\_COMP-1 の長さで切り捨てられ、d\_name に返されます。

## 【リターン値】

NULL

最終まで読んで次の空でないエントリは存在しなかった、あるいは、エラーが発生した

NULL 以外

読み出したディレクトリ情報へのポインタ

## 【変換後の実際の実行形式】

`grp_fs_readdir(ptDirHdl)`

## 4.3.15 rename

【機能概要】 ファイル/ディレクトリ名称の変更

【使用形式】

```
#include "grp_fs_conv.h"
int  rename(const char  *pcOld,  const char  *pcNew);
```

【パラメータ】

pcOld	入力	名称を変更するファイル/ディレクトリのパス名
pcNew	入力	新しいファイル/ディレクトリ名称

【機能詳細】

pcOld で指定したファイル/ディレクトリの名称を pcNew で指定した名称に変更します。但し、pcOld と pcNew が異なるファイルシステム上にある場合は、エラーとなります。

【リターン値】

0	名称変更成功
GRP_FS_ERR_PERMIT	pcNew の親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	pcOld で指定したファイル/ディレクトリが存在しない pcNew 指定したファイル/ディレクトリのパス中のディレクトリが存在しない
GRP_FS_ERR_EXIST	pcNew で指定したファイル/ディレクトリが既に存在する
GRP_FS_ERR_XFS	pcOld と pcNew が異なるファイルシステム上にある
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcOld、pcNew で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_rename((const grp_uchar_t *) pcOld, ((const grp_uchar_t *) pcNew))
```

## 4.3.16 rmdir

【機能概要】 ディレクトリの削除

【使用形式】

```
#include "grp_fs_conv.h"
int rmdir(const char *pcDir);
```

【パラメータ】

pcDir	入力	削除するディレクトリのパス名
-------	----	----------------

【機能詳細】

pcDir で指定したディレクトリを削除します。削除するディレクトリにファイルやディレクトリが存在する場合はエラーとなり、削除することができません。また、“.”、“..”、アクセス中のファイルを削除しようとした場合も、エラーとなり削除することができません。

【リターン値】

0	削除成功
GRP_FS_ERR_PERMIT	親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	指定したディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcDir で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_BUSY	指定したディレクトリ下が空でない “.”、“..”、アクセス中のファイルを削除しようとした
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_unlink((const grp_uchar_t *) pcDir)
```

## 4.3.17 stat

【機能概要】 ファイル属性情報の取得

【使用形式】

```
#include "grp_fs_conv.h"
```

```
int stat(const char *pcPath, struct stat *ptStat);
```

【パラメータ】

pcPath            入力                      ファイル属性情報を取得したいファイル/ディレクトリのパス名

ptStat            出力                      ファイル属性情報を格納する領域のアドレス

```
struct stat {
```

```
    int          st_dev;            デバイス番号 (3.5.3 節 参照)
```

```
    grp_uint32_t st_ino;            ファイル ID
```

```
    grp_uint32_t st_mode;           保護モード/ファイルタイプ
```

```
    S_IRUSR    0400    ファイルオーナー read 可能
```

```
    S_IWUSR    0200    ファイルオーナー write 可能
```

```
    S_IXUSR    0100    ファイルオーナー実行可能
```

```
    S_IRGRP    0040    ファイルオーナーグループ内 read 可能
```

```
    S_IWGRP    0020    ファイルオーナーグループ内 write 可能
```

```
    S_IXGRP    0010    ファイルオーナーグループ内実行可能
```

```
    S_IROTH    0004    他のユーザ read 可能
```

```
    S_IWOTH    0002    他のユーザ write 可能
```

```
    S_IXOTH    0001    他のユーザ実行可能
```

```
    S_IFMT    0170000 ファイルタイプビットマスク
```

```
    S_IFDIR    0040000 ディレクトリファイル
```

```
    S_IFREG    0100000 通常ファイル
```

```
    S_IFLNK    0120000 リンクファイル(ロングファイル名)
```

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは、オーナービット値と同じです。隠しファイルの場合は、グループ、他のユーザのビットが 0 となります。さらに、FAT には実行可否の概念がなく、情報も保持できないため、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが "EXE"、"COM"、"DLL"、"BAT" の場合、1 で、その他の場合は、0 となります。また、read 不可の概念がないため、S\_IRUSR のビットは常に 1 となります。

```
    int          st_nlink;          リンクカウント (FAT の場合、常に 1)
```

```
    int          st_uid;            ファイルオーナーのユーザ ID (FAT の場合、常に 0)
```

```
    int          st_gid;            ファイルオーナーグループのグループ ID (FAT の場合、常に 0)
```

```
    grp_int32_t  st_atime;          最終アクセス時刻 (1970/1/1 からのトータル秒)
```

但し FAT の場合、精度は日単位

```
    grp_int32_t  st_mtime;          最終更新時刻 (1970/1/1 からのトータル秒)
```

```
    grp_int32_t  st_ctime          ファイルの生成時刻 (1970/1/1 からのトータル秒)
```

```
    grp_int32_t  st_size          ファイルサイズ
```

```
};
```

## 【機能詳細】

pcPath で指定したファイル/ディレクトリの属性情報を取得し、ptStat で指定された領域に格納します。

なお、st\_mode フィールドに格納されたファイルのタイプ情報判定のために、以下のマクロも定義されています。

```
#define S_ISDIR(iMode) (((iMode) & S_IFMT) == S_IFDIR)    /* ディレクトリ */
#define S_ISREG(iMode) (((iMode) & S_IFMT) == S_IFREG)    /* 通常ファイル */
#define S_ISLNK(iMode) (((iMode) & S_IFMT) == S_IFLNK)    /* リンクファイル */
```

## 【リターン値】

0	属性情報取得成功
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcPath で指定したパス名、ptStat が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 【変換後の実際の実行形式】

grp\_fs\_stat((const grp\_uchar\_t \*) pcPath, ptStat)

ただし、コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合は stat を grp\_fs\_stat0 に変換するのではなく、stat0 が定義され使用されます。また、st\_size は 2G-1 バイトに制限されます。

2G-1 バイト以上のファイルサイズを取得する場合は stat4G0 を使用してください。

## 4.3.18 stat4G

【機能概要】 ファイル属性情報の取得

【使用形式】

```
#include "grp_fs_conv.h"
int stat4G(const char *pcPath, grp_fs_stat_t *ptStat);
```

【パラメータ】

pcPath	入力	ファイル属性情報を取得したいファイル/ディレクトリのパス名
ptStat	出力	ファイル属性情報を格納する領域のアドレス

```
grp_fs_stat_t {
    int          st_dev;      デバイス番号 (3.5.3 節 参照)
    grp_uint32_t st_ino;      ファイル ID
    grp_uint32_t st_mode;     保護モード/ファイルタイプ
                                S_IRUSR   0400   ファイルオーナー read 可能
                                S_IWUSR   0200   ファイルオーナー write 可能
                                S_IXUSR   0100   ファイルオーナー実行可能
                                S_IRGRP   0040   ファイルオーナーグループ内 read 可能
                                S_IWGRP   0020   ファイルオーナーグループ内 write 可能
                                S_IXGRP   0010   ファイルオーナーグループ内実行可能
                                S_IROTH   0004   他のユーザ read 可能
                                S_IWOTH   0002   他のユーザ write 可能
                                S_IXOTH   0001   他のユーザ実行可能
                                S_IFMT    0170000 ファイルタイプビットマスク
                                S_IFDIR   0040000 ディレクトリファイル
                                S_IFREG   0100000 通常ファイル
                                S_IFLNK   0120000 リンクファイル(ロングファイル名)
                                但し、FAT ファイルシステムの場合は、ユーザの概念がない
                                ため、グループ、他のユーザのビットは、オーナービット値と同じ
                                です。隠しファイルの場合は、グループ、他のユーザのビットが
                                0 となります。さらに、FAT には実行可否の概念がなく、情報も
                                保持できないため、実行可否ビットは、ディレクトリ、または、
                                ファイル名のサフィックスが "EXE"、"COM"、"DLL"、"BAT" の
                                場合、1 で、その他の場合は、0 となります。また、read 不可
                                の概念がないため、S_IRUSR のビットは常に 1 となります。

    int          st_nlink;    リンクカウント (FAT の場合、常に 1)
    int          st_uid;      ファイルオーナーのユーザ ID (FAT の場合、常に 0)
    int          st_gid;      ファイルオーナーグループのグループ ID (FAT の場合、常に 0)
    grp_int32_t  st_atime;     最終アクセス時刻 (1970/1/1 からのトータル秒)
                                但し FAT の場合、精度は日単位
    grp_int32_t  st_mtime;     最終更新時刻 (1970/1/1 からのトータル秒)
    grp_int32_t  st_ctime     ファイルの生成時刻 (1970/1/1 からのトータル秒)
    grp_uint32_t st_size      ファイルサイズ
};
```

【機能詳細】



コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に使用できます。  
st\_size が符号なし 32 ビット定義の grp\_fs\_stat\_t になり、2G-1 バイトの制限がない以外は stat() と同じです。

**【リターン値】**

0	属性情報取得成功
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcPath で指定したパス名、ptStat が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

**【変換後の実際の実行形式】**

```
grp_fs_stat((const grp_uchar_t *) pcPath, ptStat)
```

#### 4.3.19 sync

【機能概要】 未反映キャッシュデータのメディアへの書戻し

【使用形式】

```
#include "grp_fs_conv.h"
int sync(void);
```

【パラメータ】

なし

【機能詳細】

未反映のキャッシュデータをメディアに書込み、メモリ上のキャッシュバッファにキャッシュされたデータとメディアとの整合性を確保します。

なお、`umount` 処理せずにメディアを取外したり、何らかの理由により一度 `write` エラーとなったキャッシュについては、本 `sync` では、メディアへの書戻しを行いません。これらの `write` エラーとなったキャッシュのメディアへの書戻しは、`grp_fs_sync` 関数を使用するか、メディアの再挿入後、`grp_fs_check_fs_dev` 関数で行って下さい。

【リターン値】

0	書戻し成功
GRP_FS_ERR_IO	I/O エラー

【変換後の実際の実行形式】

```
grp_fs_sync(0)
```

## 4.3.20 unlink

【機能概要】 ファイルの削除

【使用形式】

```
#include "grp_fs_conv.h"
int  unlink(const char  *pcFile) ;
```

【パラメータ】

pcFile	入力	削除するファイルのパス名
--------	----	--------------

【機能詳細】

pcFile で指定したファイルを削除します。pcFile で指定したファイルがディレクトリの場合は、rmdir と同じ削除する効果を持ちます。

【リターン値】

0	削除成功
GRP_FS_ERR_PERMIT	親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	指定したファイルが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcFile で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_BUSY	指定したファイルがディレクトリで、同ディレクトリが空でない “.”、“..”、アクセス中のファイルを削除しようとした
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_unlink((const grp_uchar_t *) pcFile)
```

## 4.3.2 1 utimes

【機能概要】 ファイル/ディレクトリのアクセス・更新時刻情報の変更

【使用形式】

```
#include "grp_fs_conv.h"
int utimes(const char * pcPath, struct timeval *ptTimes);
```

【パラメータ】

pcPath	入力	アクセス・更新時刻を変更するファイル/ディレクトリ名
ptTimes	入力	アクセス時刻 (ptTimes[0]) と更新時刻(ptTimes[1])情報が入った配列へのポインタ。それぞれの時刻情報の構造は以下のとおり

```
struct timeval {
    long    tv_sec;        1970/1/1 からのトータル秒
    long    tv_usec;      秒以下の値 (単位マイクロ秒)。但し、秒以下の情報は
                           ファイルシステム上の保持していないので、本値は
                           無視される。
};
```

なお、FAT のファイルシステムの場合は、日単位の精度でしかアクセス時刻情報をファイルシステム上に保持していないので、アクセス時刻情報は、日単位の精度で指定します。

【機能詳細】

pcPath で指定したファイル/ディレクトリのアクセス・更新時刻情報を ptTimes に従い変更します。

【リターン値】

0	設定成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_PERMIT	親ディレクトリに対する write が許可されていない
GRP_FS_ERR_BAD_PARAM	pcPath で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

【変換後の実際の実行形式】

```
grp_fs_utimes((grp_uchar_t *)pcPath, ptTimes)
```

## 4.3.2.2 write

【機能概要】 ファイルの書込み

【使用形式】

```
#include "grp_fs_conv.h"
grp_isize_t write(int iFd, char *pcBuf, grp_isize_t iSize);
```

【パラメータ】

iFd	入力	open 関数等で返ってきたファイルハンドル
pcBuf	入力	書込むデータの格納された領域のアドレス
iSize	入力	書込むデータのサイズ

【機能詳細】

pcBuf で指定した領域に格納されたデータを、iFd で指定したファイルに iSize バイト分書込みます。ファイルシステムの残り領域が iSize 分より少ない場合は、残り領域分だけ書込みます。

リターン値としては、実際に書けたバイト数を返します。

なお、オープン時に GRP\_FS\_O\_DIRECT\_IO の指定がなかった場合は、キャッシュバッファを介して write 処理を行ない、write 要求処理時にはキャッシュに書込むだけで、メディアへの反映は、キャッシュの write 制御方式に従ったタイミングで行います。

一方、GRP\_FS\_O\_DIRECT\_IO 指定があり、ファイルシステム依存部で同オプションをサポートしている場合は、可能な限りキャッシュバッファを使用せず、アプリケーションの指定したバッファからメディアに対し、直接連続ブロックで I/O を行います。この場合も、データ境界等の関係で、直接 I/O できない部分については、キャッシュバッファを介して I/O する形になります。

**GR-FILE** の FAT ファイルシステムでは、シングল空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファからの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続して確保できれば、連続した単位で、一括してメディアに対し write します。

【リターン値】

0 または 正值	実際に書けたバイト数
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	write 可能なモードでオープンされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

【変換後の実際の実行形式】

```
grp_fs_write(iFd, (grp_uchar_t *)pcBuf, iSize)
```

#### 4.4 C 言語標準アプリケーション I/O インタフェース

3. 2 節で説明しましたとおり、**GR-FILE** では、<stdio.h>の代わりに "include" ディレクトリ下にある "grp\_stdio.h" をアプリケーションプログラムでインクルードし、オプションの C 言語標準 I/O ライブラリをリンクすることで、C 言語標準インタフェースでのプログラミングを可能としています。ただし、コンパイルオプション GRP\_FS\_MINIMIZE\_LEVEL を 1 または 2 に設定した場合、C 言語標準のアプリケーション I/O インタフェースは使用できません。

表 4-5 に、**GR-FILE** でサポートしている C 言語標準 I/O インタフェースの一覧を示します。

表 4-5 C 言語標準 I/O インタフェース一覧

#	関数名	機能
1	clearerr	・エラー状態の解除
2	fclose	・ファイルのクローズ
3	feof	・ファイルの終わり判定
4	ferror	・エラー情報の取得
5	fflush	・I/O バッファ上のデータの書戻し/先読みデータのクリア
6	fgetc/ getc	・ファイルからの一文字 read
7	fgets	・ファイルからの一行 read
8	fileno	・ファイルハンドルの取得
9	fopen	・ファイルの open
10	fprintf/vfprintf	・ファイルへの書式付 write
11	fputc/putc	・ファイルへの一文字 write
12	fread	・ファイルからの read
13	fseek	・ファイルの読書き位置変更
14	ftell	・ファイルの読書き位置情報取得
15	fwrite	・ファイルへの write
16	getchar	・標準入力からの一文字 read
17	putchar	・標準出力への一文字 write
18	rewind	・ファイル読書き位置のファイル先頭への移動
19	ungetc	・入力の一文字ロールバック

表 4-6 に示す C 言語標準アプリケーション向け 4G 対応アプリケーションインタフェースはコンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に定義される C 言語標準アプリケーション向けの 4G-1 バイト対応 I/O インタフェースです。

表 4-6 C 言語標準アプリケーション向け 4G 対応 I/O インタフェース一覧

#	関数名	機能
1	fseek4G	・ファイルの読書き位置変更 4G 対応版 コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合のみ定義
2	ftell4G	・ファイルの読書き位置情報取得 4G 対応版 コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合のみ定義

これらのインタフェースは、"include" ディレクトリ下の "grp\_stdio.h" に宣言されており、多くの場合、マクロの形で定義されています。

また、**GR-FILE** では、標準入出力のために、既定義の変数として stdin、stdout、stderr を定義して用意します。(実際の変数名は、grp\_stdio\_stdin、grp\_stdio\_stdout、grp\_stdio\_stderr)

但し、3 章で述べましたとおり、標準入出力への I/O 要求は、プラットフォーム依存の標準入出力用の I/O 処理関数が、`grp_stdin_io_stdin`、`grp_stdio_io_stdout` という関数ポインタ変数に登録されている場合にのみ有効です。登録されていない場合は、標準入出力に対する I/O 要求（含む `getchar`、`putchar`）は、すべてエラーとなります。

標準入出力用のプラットフォーム依存 I/O 処理関数のインタフェースの詳細については、「OS 抽象化インタフェース」の節を参照下さい。

なお、本 C 言語標準 I/O インタフェース関数で使用する各ファイルの I/O バッファのサイズ `GRP_STDIO_BUF` と、書式付 `write` で使用する内部展開用のバッファのサイズ `GRP_STDIO_PBUF` は、“`grp_stdio.h`” に定義されています。本サイズを変更する場合は、本定義を変更し、C 言語標準 I/O ライブラリを再コンパイルする必要があります。



#### 4.4.1 clearerr

【機能概要】 エラー状態の解除

【使用形式】

```
#include "grp_stdio.h"
void clearerr(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイル I/O のエラー状態をクリアします。具体的には、内部的に記憶している EOF (End of File) 状態と、詳細エラー番号情報をクリアします。

【リターン値】

なし

#### 4.4.2 fclose

【機能概要】 ファイルのクローズ

【使用形式】

```
#include "grp_stdio.h"
int fclose(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

I/O バッファに保持したデータをファイルに書戻し、ptFile で指定したファイルにをクローズします。クローズ処理を成功した場合は 0 を、エラーの場合は -1 を返します。エラーの詳細は、**ferror** で取得することができます。

なお、I/O バッファの書戻しに失敗した場合等のエラーの場合でも、ファイルハンドルのクローズ処理、FILE 構造体の解放処理を行ない、本コール以降、ptFile の参照は行なえません。

【リターン値】

0	クローズ処理成功
-1	クローズ処理失敗

#### 4.4.3 feof

【機能概要】 ファイルの EOF（終わり）判定

【使用形式】

```
#include "grp_stdio.h"
int feof(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルがファイルの終わりまで読んだ状態 (EOF 状態) かどうか判定します。  
EOF 状態の場合は 0 以外を、そうでない場合は 0 を返します。なお、すでに、ファイルの終わりに到達していても、まだ、読出し要求を行わず、EOF を確認していない場合は、0 を返します。  
この EOF 状態は、clearerr でのみリセット可能です。

【リターン値】

0	EOF 状態ではない
0 以外	EOF 状態である

#### 4.4.4 ferror

【機能概要】 エラー情報の取得

【使用形式】

```
#include "grp_stdio.h"
int ferror(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルに対する I/O 要求で発生した最後のエラーのエラー番号を返します。  
本関数で返すエラー番号の詳細は、前述の「エラー番号」の節を参照下さい。

なお、最後に発生したエラーのエラー番号情報は、clearerr によりクリアすることができます。

【リターン値】

0	エラー情報なし
0 以外	最後のエラーのエラー番号

## 4.4.5 fflush

【機能概要】 I/O バッファ上のデータの書戻し/先読みデータのクリア

【使用形式】

```
#include "grp_stdio.h"
int fflush(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルの I/O バッファに、ファイルに未反映の更新されたデータがある場合、同データをファイルに書戻します。また、先読みデータが I/O バッファ上にある場合は、同データをクリアし、データの読出し要求があった場合に再度ファイルから読込むようにします。

エラーが発生した場合の詳細エラーは、**ferror** により取得することができます。

【リターン値】

0	処理成功
-1	書戻し処理失敗

## 4.4.6 fgetc / getc

【機能概要】 ファイルからの一文字 read

【使用形式】

```
#include "grp_stdio.h"
int fgetc(FILE *ptFile);
int getc(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルから一文字読み込み、得られたデータを返します。なお、ptFile で指定した FILE 構造体の I/O バッファに先読みされたデータが存在する場合は、実際には、ファイルからの I/O を行わず、同 I/O バッファ上にあるデータを返します。

ファイルの終わりまで達して返すべきデータがない場合、あるいは、エラーが発生した場合は、EOF を返します。返すべきデータがないか、エラーかの区別は、feof により判定することができます。エラーが発生した場合の詳細エラーは、ferror により取得することができます。

なお、fgetc と getc は、別名の関係で、機能は同じです。

【リターン値】

EOF	ファイルの終わりまで達して返すべきデータがない、 あるいは、エラーが発生した
その他	得られた 1 バイトのデータ

## 4.4.7 fgets

【機能概要】 ファイルからの一行 read

【使用形式】

```
#include "grp_stdio.h"
char *fgets(char *pcStr, int iSize, FILE *ptFile);
```

【パラメータ】

pcStr	出力	得られたデータを格納する領域のアドレス
iSize	入力	上記領域のサイズ
ptFile	入力	fopen で得られた FILE 構造体へのポインタ

【機能詳細】

ptFile で指定したファイルから、改行コードが得られるまでの一行分を読み込み、改行コードも含め、さらに NULL コードを最後に付加して、pcStr で指定した領域に格納します。NULL コードも含めて iSize バイトを超える場合は、改行コードが得られなくても、そこまでのデータを NULL 付で格納します。また、読み込み中にファイルの終わりまで達した場合は、そこまでのデータに NULL 付で格納します。但し、一文字も読み込めなかった場合は、バッファには何も格納せず、リターン値として NULL を返します。その他の場合は、pcStr で指定したアドレスをリターン値として返します。

【リターン値】

NULL	ファイルの終わりまで達した、あるいは、エラーで、返すべきデータが全くない、
その他	pcStr で指定したアドレス



## 4.4.8 fileno

【機能概要】 ファイルハンドルの取得

【使用形式】

```
#include "grp_stdio.h"
int fileno(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルのファイルハンドルを取得します。本関数で得られたファイルハンドルは、POSIX 互換インタフェースや **GR-FILE** 固有インタフェースなどで使用します。

なお、C 言語標準 I/O インタフェースは、バッファリング処理を行っていますので、fflush 処理等の適切な処理を行わず、本関数で取得したファイルハンドルを使用し、POSIX 互換インタフェースや **GR-FILE** 固有インタフェースと混在使用しますと、思った結果とならない場合があります。従いまして、C 言語標準 I/O インタフェースの動作を熟知していない場合は、混在使用は、避けて下さい。

【リターン値】

ファイルハンドル値

## 4.4.9 fopen

【機能概要】 ファイルの open

【使用形式】

```
#include "grp_stdio.h"
```

```
FILE *fopen(const char *pcPath, const char *pcMode);
```

【パラメータ】

pcPath	入力	オープンするファイルのパス名
pcMode	入力	オープンする際のモード
"r"	read-only でオープン。ファイルの読書き位置は、ファイルの先頭に設定。対象のファイルが存在しなかった場合は、エラーとなる。	
"w"	write-only でオープン。ファイルの読書き位置は、ファイルの先頭に設定。対象のファイルが存在しなかった場合、同ファイルを作成し、存在した場合は、同ファイルの内容を削除し、空の状態にする。	
"a"	write-only でオープン。ファイルの読書き位置は、ファイルの最後に設定。対象のファイルが存在しなかった場合、同ファイルを作成する。	
"r+"	read/write 用としてオープン。その他は、"r"と同じ。	
"w+"	read/write 用としてオープン。その他は、"w"と同じ。	
"a+"	read/write 用としてオープン。その他は、"a"と同じ。	

【機能詳細】

pcPath で指定したファイルをオープンし、C 言語標準 I/O インタフェースを使ったファイル I/O 操作を行なうための FILE 構造体へのポインタを返します。ファイルをオープンする際のモードは、pcMode パラメータで指定します。

なお、**GR-FILE** の C 言語標準 I/O インタフェース関数は、**LINUX** 等の標準 **POSIX** インタフェースに準拠し、**Windows** で行われているような、改行コード ('¥n') の書込みに対し、復帰コード ('¥r') の自動追加は行いません。すなわち、**GR-FILE** では、常に、**Windows** でいう "b" モードでオープンした状態となります。但し、本 **GR-FILE** では、pcMode の 2 バイト目以降は、'+' 以外無視する実装となっておりますので、意味はありませんが、"b" を pcMode の 2 バイト目以降に指定することは可能です。

【リターン値】

NULL	オープン失敗
NULL 以外	指定したファイルに対し、C 言語標準 I/O インタフェースを使ったファイル I/O 操作を行なうための FILE 構造体へのポインタ

## 4.4.1 O fprintf / vfprintf

【機能概要】 ファイルへの書式付 write

【使用形式】

```
#include "grp_stdio.h"
int fprintf(FILE *ptFile, const char *pcFormat, ...);
int vfprintf(FILE *ptFile, const char *pcFormat, va_list vap);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
pcFormat	入力	書式
...	入力	書式に対応したパラメータ
vap	入力	書式に対応したパラメータ

【機能詳細】

ptFile で指定したファイルに、pcFormat で指定した書式で、データを書込みます。

なお、本関数は、対象プラットフォームで提供される vsnprintf を使って実現しているため、pcFormat として指定可能な書式は、対象プラットフォームの vsnprintf の仕様によります。また、本関数では、スタック上に、GRP\_STDIO\_PBUF 分のバッファを確保し、一旦、同バッファ上にデータを展開して実現していますので、展開結果が GRP\_STDIO\_PBUF バイト (default の値は 512) を超える場合は、エラーとなります。

【リターン値】

0 または 正值	write したバイト数
-1	バッファ不足、または、write エラー

## 4.4.1 1 fputc / putc

【機能概要】 ファイルへの一文字 write

【使用形式】

```
#include "grp_stdio.h"
int fputc(int iChar, FILE *ptFile);
int putc(int iChar, FILE *ptFile);
```

【パラメータ】

iChar	入力	write する文字
ptFile	入力	fopen で得られた FILE 構造体へのポインタ

【機能詳細】

ptFile で指定したファイルに iChar で指定した文字を書込みます。なお、C 言語標準の I/O ライブラリでは、I/O バッファを使って I/O を行っていますので、バッファに余裕がある場合は、実際には、ファイルへの I/O を行わず、同 I/O バッファ上にデータを設定します。

なお、fputc と putc は、別名の関係で、機能は同じです。

【リターン値】

- 1	write エラー、
その他	iChar で指定したデータ

## 4.4.1 2 fread

【機能概要】 ファイルからの read

【使用形式】

```
#include "grp_stdio.h"
```

```
grp_isize_t fread(void *pvPtr, grp_isize_t iSize, grp_isize_t iCnt, FILE *ptFile);
```

【パラメータ】

pvPtr	出力	ファイルから読込んだデータを格納する領域のアドレス
iSize	入力	読込む各データオブジェクトのサイズ
iCnt	入力	読込むデータオブジェクトの数
ptFile	入力	fopen で得られた FILE 構造体へのポインタ

【機能詳細】

ptFile で指定したファイルから iSize バイトからなるデータオブジェクトを iCnt 個分読み、pvPtr で指定した領域に格納します。現在の読書き位置からファイルの終わりまでのバイト数が、iSize×iCnt に満たない場合は、完全に読めたデータオブジェクト分だけ格納し、そのデータオブジェクト数を返します。

【リターン値】

- 1	read エラー
その他	完全に読めたデータオブジェクト数

## 4.4.13 fseek

【機能概要】 ファイルの読書き位置の変更

【使用形式】

```
#include "grp_stdio.h"
int fseek(FILE *ptFile, long iOffset, int iMode);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
iOffset	入力	ファイルの読書き位置情報
iMode	入力	読書き位置情報の解釈方法
SEEK_SET		uiOffset で指定した値読書き位置に設定
SEEK_CUR		現在の読書き位置+iOffset の値を読書き位置に設定 ("a"または"a+"モードでオープンされている場合は、SEEK_END と同じ)
SEEK_END		現在のファイルサイズ+iOffset の値を読書き位置に設定

【機能詳細】

ptFile で指定したファイルの read/write 時に使用するファイルの読書き位置を iOffset、iMode に従い設定します。fseek は、lseek と異なり、変更が完了した場合、設定した読書き位置ではなく、0 を返します。

なお、現在のファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、write 時にエラーとなります。また、読書き位置が負値になった場合は、0 に自動補正します。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と fseek4G などの 4G 対応 API を混在して使用しないでください。fseek4G などで 2GB 以上のオフセットに移動したのちに本 API を使用し、オフセットが 2GB 以上になった場合、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0	変更完了
-1	変更失敗

## 4.4.14 fseek4G

【機能概要】 ファイルの読書き位置の変更 4G 対応版

【使用形式】

```
#include "grp_stdio.h"
int fseek4G(FILE *ptFile, unsigned long uiOffset, int iMode);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
iOffset	入力	ファイルの読書き位置情報
iMode	入力	読書き位置情報の解釈方法
SEEK_SET		uiOffset で指定した値読書き位置に設定
SEEK_CUR		現在の読書き位置+uiOffset の値を読書き位置に設定 (O_APPEND モードでオープンされている場合は、SEEK_END と同じ)
SEEK_END		現在のファイルサイズ+uiOffset の値を読書き位置に設定
SEEK_MINUS		読書き位置の算出の際に uiOffset を負値として扱う。 (本モードのみでの指定は不可。SEEK_CUR/SEEK_END と bit or で指定)

【機能詳細】

本機能は、コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に使用できます。

ptFile で指定したファイルの read/write 時に使用するファイルの読書き位置を uiOffset, iMode に従い設定します。fseek4G は、lseek4G と異なり、変更が完了した場合、設定した読書き位置ではなく、0 を返します。

なお、現在のファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、read/write 時にエラーとなります。

また、読書き位置にオーバーフローもしくはアンダーフローが発生した場合には、エラーとします。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と ftell などの 4G 未対応 API を混在して使用しないでください。本 API で 2GB 以上のオフセットに移動したのちに ftell などを使用すると、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0	変更完了
-1	変更失敗



## 4.4.15 ftell

【機能概要】 ファイル読書き位置情報の取得

【使用形式】

```
#include "grp_stdio.h"
long ftell(FILE *ptFile);
```

【パラメータ】

ptFile            入力            fopen で得られた FILE 構造体へのポインタ

【機能詳細】

ptFile で指定したファイルの読書き位置情報を取得し、リターン値として返します。なお、この値は、C 言語標準ライブラリの I/O バッファ上にバッファリングされたデータも含めた読書き位置情報ですので、lseek で得られる値とは違うことに注意して下さい。

また、“a”または“a+”モードでファイルがオープンされている場合は、ftell を実行しますと、ファイルの終わりの位置に移動し、その位置情報を返します。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と fseek4G などの 4G 対応 API を混在して使用しないでください。fseek4G などでは 2GB 以上のオフセットに移動したのちに本 API を使用すると、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0 または 正值	読書き位置情報
-1	取得失敗

## 4.4.16 ftell4G

【機能概要】 ファイル読書き位置情報の取得 4G 対応版

【使用形式】

```
#include "grp_stdio.h"
int ftell4G(FILE *ptFile, grp_uioffset_t *puiOffset);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
puiOffset	出力	ファイルの読書き位置情報

【機能詳細】

本機能は、コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に使用できます。

ptFile で指定したファイルの読書き位置情報を取得し、puiOffset に返します。なお、この値は、C 言語標準ライブラリの I/O バッファ上にバッファリングされたデータも含めた読書き位置情報ですので、lseek または lseek4G で得られる値とは違うことに注意して下さい。

【リターン値】

0	正常終了
-1	取得失敗

## 4.4.17 fwrite

【機能概要】 ファイルへの write

【使用形式】

```
#include "grp_stdio.h"
grp_isize_t fwrite(void *pvPtr, grp_isize_t iSize, grp_isize_t iCnt, FILE *ptFile);
```

【パラメータ】

pvPtr	入力	ファイルに書込むデータを格納した領域のアドレス
iSize	入力	読込む各データオブジェクトのサイズ
iCnt	入力	読込むデータオブジェクトの数
ptFile	入力	fopen で得られた FILE 構造体へのポインタ

【機能詳細】

pvPtr で指定した領域に格納された iSize バイトからなるデータオブジェクト iCnt 個分を、ptFile で指定したファイルに書込みます。実際に書けたデータオブジェクトが iCnt より少ない場合、完全 に書けたデータオブジェクト数を返します。

【リターン値】

- 1	write エラー
その他	完全 に書けたデータオブジェクト数

## 4.4.18 getchar

【機能概要】 標準入力からの一文字 read

【使用形式】

```
#include "grp_stdio.h"
int  getchar(void);
```

【パラメータ】

なし

【機能詳細】

標準入力 (stdin) から一文字読み、得られた文字の値を返します。

ファイルの終わりまで達して返すべきデータがない場合、あるいは、エラーが発生した場合は、EOF を返します。返すべきデータがないか、エラーかの区別は、feof により判定することができます。エラーが発生した場合の詳細エラーは、ferror により取得することができます。

なお、標準入力からの読みは、プラットフォーム依存の標準入力用の read 処理関数が定義され、標準入力用の処理関数ポインタ変数 grp\_stdio\_io\_stdin に登録されていないとエラーとなります。

【リターン値】

EOF	ファイルの終わりまで達して返すべきデータがない、 あるいは、エラーが発生した
その他	得られた 1 バイトのデータ

## 4.4.19 putchar

【機能概要】 ファイルへの一文字 write

【使用形式】

```
#include "grp_stdio.h"
int putchar(int iChar);
```

【パラメータ】

iChar	入力	write する文字
-------	----	------------

【機能詳細】

標準出力（stdout）に iChar で指定した文字を書込みます。

なお、標準出力への書込みは、プラットフォーム依存の標準出力用の write 処理関数が定義され、標準出力用の処理関数ポインタ変数 grp\_stdio\_io\_stdout に登録されていないとエラーとなります。

【リターン値】

- 1	write エラー、
その他	iChar で指定したデータ

## 4.4.2 0 rewind

【機能概要】 ファイル読書き位置のファイル先頭への移動

【使用形式】

```
#include "grp_stdio.h"
int  rewind(FILE *ptFile);
```

【パラメータ】

ptFile	入力	fopen で得られた FILE 構造体へのポインタ
--------	----	----------------------------

【機能詳細】

ptFile で指定したファイルの読書き位置情報をファイル先頭に設定します。

【リターン値】

0	設定成功
-1	設定失敗

## 4.4.2 1 ungetc

【機能概要】 入力の一文字ロールバック

【使用形式】

```
#include "grp_stdio.h"
int ungetc(int iChar, FILE *ptFile);
```

【パラメータ】

iChar	入力	戻す文字
ptFile	入力	fopen で得られた FILE 構造体へのポインタ

【機能詳細】

ptFile で指定したファイルから読出したデータを一文字戻します。戻す文字は、iChar で指定します。なお、同ファイルから読出していない場合や、直前に fflush 等を実行してしまっていた場合は、エラーとなります。

【リターン値】

-1	ロールバック失敗
その他	iChar (ロールバック成功)



#### 4.5 GR-FILE 固有アプリケーション I/O インタフェース

GR-FILE では、上記 POSIX 互換インタフェースや C 言語標準 I/O インタフェースのベースとなる関数、および、メディアの挿抜処理対応のインタフェースなどを、GR-FILE 固有のインタフェースとして提供しています。ただし、コンパイルオプション GRP\_FS\_MINIMIZE\_LEVEL を 1 または 2 に設定した場合、一部の GR-FILE 固有アプリケーション I/O インタフェースは使用できません。

表 4-7 に、GR-FILE 固有のアプリケーション向け I/O インタフェースの一覧を示します。これらのインタフェースの型宣言は、“include” ディレクトリ下の “grp\_fs\_if.h” に含まれています。

表 4-7 GR-FILE 固有 I/O インタフェース一覧

#	関数名	機能	備考	GRP_FS_MINIMIZE_LEVEL が 1 または 2 での使用可否
1	grp_fat_find_type *3	・メディア/パーティションのフォーマットパラメータ計算	フォーマット 関連	○
2	grp_fat_format *3	・メディア/パーティションの初期化	フォーマット 関連	○
3	grp_fat_format_sd *5	・SD カードのフォーマット (パーティション設定を含む)	参考ライブラリ	×
4	grp_fs_chdir	・カレントディレクトリの設定、および、無効化	POSIX 対応	○
5	grp_fs_check_fs_dev	・再挿入メディアのチェック	挿抜処理対応	×
6	grp_fs_check_volume	・未 mount メディアのボリューム名の取得	挿抜処理対応	×
7	grp_fs_chmod *1	・ファイル保護モードの設定・変更	POSIX 対応	×
8	grp_fs_close	・ファイルのクローズ	POSIX 対応	○
9	grp_fs_closedir *2	・grp_fs_opendir でオープンしたディレクトリのクローズ	POSIX 対応	×
10	grp_fs_create	・ファイル/ディレクトリの作成	mkdir 相当	○
11	grp_fs_err	・エラー番号/メッセージ変換	ライブラリ	×
12	grp_fs_get_attr	・ファイル属性情報の取得	stat 相当	○
13	grp_fs_get_cwd	・カレントディレクトリ名の取得	POSIX 対応 ライブラリ	×
14	grp_fs_get_dirent	・ディレクトリエントリ情報の取得	readdir、 _findnext 相当	○
15	grp_fs_get_error	・未反映キャッシュバッファの読出し・解放	挿抜処理対応	×
16	grp_fs_get_mnt	・全 mount 情報の取得	getmntinfo 相当	×
17	grp_fs_get_mnt by_dev	・デバイス番号による特定 mount 情報の取得	getmntinfo 相当	×
18	grp_fs_get_mnt by_name	・デバイス名称による特定 mount 情報の取得	getmntinfo 相当	○
19	grp_fs_init	・GR-FILE の初期化	初期化处理	○
20	grp_fs_invalidate _fs_dev	・メディアに対する I/O 抑止設定 (メディア不当取出し時)	挿抜処理対応	×
21	grp_fs_lookup_dev	・デバイス名称→デバイス番号変換	その他関数	○
22	grp_fs_lseek	・ファイルの読書き位置の設定・変更	POSIX 対応	○
23	grp_fs_lseek4G	・ファイルの読書き位置の設定・変更 4G 対応版	POSIX 拡張	○
24	grp_fs_mount	・メディアの mount 処理 (メディア挿入時)	挿抜処理対応	○

25	grp_fs_open	・ファイルのオープン	POSIX 対応	○
26	grp_fs_opendir *2	・ grp_fs_readdir でディレクトリエントリ読み出しを行うためのディレクトリのオープン	POSIX 対応	×
27	grp_fs_read	・ファイルの読込み	POSIX 対応	○
28	grp_fs_readdir *2	・ディレクトリエントリの読み出し(POSIX 互換用)	POSIX 対応	×
29	grp_fs_read_part *4	・メディアからのパーティション情報の読み込み	フォーマット 関連	×
30	grp_fs_rename	・ファイル/ディレクトリ名称の変更	POSIX 対応	○
31	grp_fs_set_attr	・ファイル属性情報の設定・変更	chmod、utimes 対応	○
32	grp_fs_stat *1	・ファイル属性情報の取得	POSIX 対応	×
33	grp_fs_sync	・未反映キャッシュデータのメディアへの書戻し	POSIX 対応	○
34	grp_fs_task_ free_all_env	・全タスクのオープンファイル・カレントディレクトリの無効化	タスク管理	×
35	grp_fs_task_ free_env	・自タスクのオープンファイル・カレントディレクトリの無効化	タスク管理	×
36	grp_fs_task_ free_env by_id	・指定タスクのオープンファイル・カレントディレクトリの無効化	タスク管理	×
37	grp_fs_truncate	・指定サイズ以降のファイル領域の解放	POSIX 対応	○
38	grp_fs_unlink	・ファイル/ディレクトリの削除	POSIX 対応	○
39	grp_fs_unmount	・メディアの unmount 処理	挿抜処理対応	○
40	grp_fs_utimes *1	・ファイル/ディレクトリのアクセス・更新時刻情報の変更	POSIX 対応	×
41	grp_fs_write	・ファイルの書込み	POSIX 対応	○
42	grp_fs_write_part *4	・メディアへのパーティション情報の書き込み	フォーマット 関連	×

\*1 これらの関数の型宣言は、"include" ディレクトリ下の "grp\_fs\_conv.h" に含まれています。

\*2 これらの関数の型宣言は、"include" ディレクトリ下の "grp\_fs\_readdir.h" に含まれています。

\*3 これらの関数の型宣言は、"include" ディレクトリ下の "grp\_fat\_format.h" に含まれています。

\*4 これらの関数の型宣言は、"include" ディレクトリ下の "grp\_fs\_disk\_part.h" に含まれています。

\*5 この関数の型宣言は、"include" ディレクトリ下の "grp\_fat\_format\_sd.h" に含まれています。

なお、本参考ライブラリは、SD Card Association の規格書で規定されている情報を基に作成されていますので、本ライブラリ、および、本ライブラリソースに記載された情報を製品に利用する場合は、SD Card のライセンスが必要となります。SD Card のライセンスにつきましては、SD Card Association にお問い合わせ下さい。

"grp\_fat\_format\_sd.h"は、ご要望のあった場合にのみ、ご提供させて頂いています。

## 4.5.1 grp\_fat\_find\_type

【機能概要】 メディア/パーティションのフォーマットパラメータ計算

【使用形式】

```
#include "grp_fat_format.h"

int grp_fat_find_type(grp_uint32_t uiTotalSec, grp_uint32_t uiOffset, int iSecShift,
                     grp_fat_format_param_t *ptReq, grp_fat_format_param_t *ptRes);
```

【パラメータ】

uiTotalSec	入力	メディア/パーティションのトータルセクタ数
uiOffset	入力	パーティションの開始位置セクタ数
iSecShift	入力	メディアのセクタサイズのシフトカウント
ptReq	入力	フォーマット情報パラメータ（要求）（出力フィールドも変更なし）
ptRes	出力	フォーマット情報パラメータ（結果）

【機能詳細】

uiTotal、iSecShift で指定したトータルセクタ数、セクタサイズを持つメディア/パーティションを FAT 形式でフォーマットした場合の FAT タイプやクラスタサイズ等のフォーマットパラメータを計算します。

ptReq は、フォーマット情報パラメータの指定で、ptReq の指す構造体のうち、0 以外の値を設定したフィールドのパラメータは、指定の値をパラメータとして使用し、0 に設定されたフィールドのパラメータを計算またはデフォルトの値により求めます。そして、求めた結果のパラメータ値を、予め指定されたパラメータ値と共に、ptRes 指定された領域に格納します。

なお、予め指定したパラメータでフォーマットできない場合は、GRP\_FS\_ERR\_BAD\_PARAM エラーとなります。

フォーマット情報パラメータの詳細につきましては、3.12.1 節を参照下さい。

本関数は、grp\_fs\_write\_part でパーティション情報を設定する際に、メディアサイズから FAT タイプを特定して、パーティションタイプ情報を決定する場合や、SD カード等でデータクラスタ領域の境界調整を行なうためのパーティション開始位置の調整セクタ数を求める場合に使用します。特に、SD カード等では、ptReq の uiOption フィールドに GRP\_FAT\_ADJ\_BY\_START オプションを指定して使用します。このオプションを指定しますと、uiOffset で指定したパーティション開始位置セクタ数込みで、データクラスタ領域を ptReq の uiAlign フィールドで指定した境界に位置づけます。そして、パーティション開始位置の調整セクタ数を ptRes の uiAdjust フィールドに返します。また、uiAdjust 値を含め、FAT ファイルシステムとして使用できなかったセクタ数を ptRes の uiNotUsed フィールドに返します。

GRP\_FAT\_ADJ\_BY\_START オプション指定がない場合は、uiOffset パラメータは無視し、ptReq の uiRsvSecCnt フィールドの値を調整することで、データクラスタ領域を ptReq の uiAlign フィールドで指定した境界に調整します。そして、ptRes の uiAdjust フィールドには、uiRsvSecCnt を調

整した差分のセクタ数を、`uiNotUsed` には、調整後も FAT ファイルシステムとして使用できなかったセクタ数を返します。

なお、本関数のパラメータとして必要な、メディア/パーティションのサイズ、セクタサイズ等は、`grp_fs_open_dev`、`grp_fs_ioctl_dev` 等を使って求めることが可能です。

【リターン値】

0	正常終了
GRP_FS_ERR_BAD_PARAM	指定のフォーマット情報パラメータでは、フォーマットできない

## 4.5.2 grp\_fat\_format

【機能概要】 メディア/パーティションの初期化

【使用形式】

```
#include "grp_fat_format.h"
int grp_fat_format(const char *pcDev,
                  grp_fat_format_param_t *ptParam, grp_fs_media_info_t *ptMedia);
```

【パラメータ】

pcDev	入力	フォーマット対象のデバイス名称
ptParam	入力/出力	フォーマット情報パラメータ
ptMedia	入力/出力	メディア情報パラメータ

【機能詳細】

pcDev で指定したメディア/パーティションを FAT ファイルシステムとしてフォーマットし、初期化します。なお、本バージョンでは、フォーマットに必要な必要最小限のデータのみの書き込みを行い、メディア/パーティション全体にわたるデータの上書き削除や、アクセス不能セクタの検出、不良クラスタ登録等を行いません。

ptParam は、FAT タイプやクラスタサイズなどのフォーマット情報パラメータの指定です。

ptParam の aucVolLab には、ボリュームラベルを指定します。ボリュームラベルを指定する場合は、11 文字で指定を行います。11 文字に満たない場合は、'¥0' 終端が可能です。実際に書き込まれる際に後ろがスペースで埋められます。11 文字で記述する場合は、'¥0' 終端は出来ません。また、書き込まれるボリュームラベルは、文字列末尾の '¥0' を含めることは出来ません。

ボリュームラベルで使用できる文字は、ショートファイル名で指定できる文字（英大文字、数字、'.' '\$%' '-' '@' '~' '!' '() {} ^' '#' '&') に加え、空白が使えます。英小文字を指定した場合は自動的に英大文字に変換されます。これ以外の文字を指定した場合は、GRP\_FS\_ERR\_BAD\_PARAM を返します。

ptParam で指された構造体のうち、値が 0 でないフィールドのパラメータは、同パラメータ値を使用し、値が 0 のフィールドのパラメータは、**GR-FILE** のデフォルト値や計算により求めた値を使用してフォーマットを実行します。

ptMedia は、フォーマット対象のメディア/パーティションのトータルサイズやセクタサイズ等のメディア情報のパラメータです。通常、フォーマットに必要なメディア情報は、デバイスドライバから得るため、明示的に指定する必要は殆どありませんが、ドライバから返される値を上書きして、小さなサイズでフォーマットしたり、必要なメディア情報を返さないドライバの場合に、本パラメータで指された領域に必要な値を設定してコールします。ptParam 同様、値が 0 のフィールドが、**GR-FILE** での自動設定を、0 以外の値を設定したフィールドが明示的に指定したことを意味します。

フォーマットが成功した場合、ptParam、ptMedia で指された領域には、**GR-FILE** での自動設定した値も含めて、使用されたパラメータ値が返されます。なお、ptParam の uiRsvSecCnt フィールドは、バウンダリの関係で、要求した値が補正されて返る場合があります。また、ptParam の uiClst、uiFatSec、aucVolSer、uiNotUsed、uiAdjust フィールドは、結果を返すためのフィールドで、要求時に値を設定しても無視されます。

ptParam、ptMedia に NULL を指定した場合は、すべてのフィールドに 0 を指定したのと同じ効果を持ちます。但し、使用された結果のパラメータ値は返りません。

ptParam、ptMedia で指された領域に設定する値の詳細につきましては、3.12.1 節をご参照下さい。

なお、本関数は、対象のメディア/パーティションが mount されていない状態で実行して下さい。mount した状態で実行しますと、GRP\_FS\_ERR\_BUSY エラーとなります。

また、パーティションの設定/変更が必要な場合は、本関数を実行する前に、grp\_fs\_write\_part を使用して予めパーティションの設定/変更を行なって下さい。既に、パーティションが設定されたメディアに対し、パーティションレスでフォーマットしたい場合は、pcDev で指定するデバイス名称を"USB0\*" のように、パーティション番号文字を '\*' で指定したパーティションレスデバイス名を使用して下さい。

特に、SD カード等では、まず、GRP\_FAT\_ADJ\_BY\_START オプション付で grp\_fat\_find\_type を実行し、パーティション開始位置の調整セクタ数を求め、その値を使って、grp\_fs\_write\_part によりパーティション設定を行ってから、本関数を実行します。その際、ptParam の uiOption には、grp\_fat\_find\_type 実行時と同様に GRP\_FAT\_ADJ\_BY\_START オプションを指定して実行して下さい。このオプションを指定して実行しますと、パーティション開始位置込みで、データクラスタ領域を ptParam の uiAlign フィールドで指定した境界に位置づけます。その結果、ptParam の uiAdjust フィールドおよび uiNotUsed フィールドには、それぞれ、パーティション開始位置の調整セクタ数と、FAT ファイルシステムとして使用できなかったセクタ数が返りますが、既にパーティション設定時にこれらの値を考慮して、正しくパーティション設定を行なっていれば、これらのフィールドは共に 0 になっています。そうでなければ、パーティション設定が正しくないことを意味しますので、再度、パーティション設定をやり直す必要があります。

また、SD カードの場合、各ファイルの作成/アクセス時刻情報は使用せず、ディレクトリ内の同情報を格納するフィールドは 0 とすべきと規定していますので、ptParam の uiOption に GRP\_FAT\_NO\_CRT\_ACCTIME を指定して使用して下さい。本オプションを指定しますと、フォーマット時にルートディレクトリ下にボリュームラベルを作成する際、作成/アクセス時刻情報を 0 として作成します。

なお、上記のルールに従い、SD カードに対し、SD 規格に従ったパーティション設定からフォーマットまで行なうライブラリ関数 grp\_fat\_format\_sd を別途参考として用意しています。(但し、利用に当たっては、SD Card のライセンスの取得が必要です。)

ptParam の uiOption に GRP\_FAT\_ADJ\_BY\_START オプション指定がない場合は、パーティションの開始位置に関係なく、ptParam の uiRsvSecCnt フィールドの値を調整することで、データクラスタ領域を ptParam の uiAlign フィールドで指定した境界に調整します。そして、ptParam の uiAdjust フィールドには、uiRsvSecCnt を調整した差分のセクタ数を、uiNotUsed には、調整後も FAT ファイルシステムとして使用できなかったセクタ数を返します。なお、uiAlign が 0 の場合のクラスタ境界とします。

## 【リターン値】

GRP_FAT_TYPE_12	FAT12 でフォーマット完了
GRP_FAT_TYPE_16	FAT16 でフォーマット完了
GRP_FAT_TYPE_32	FAT32 でフォーマット完了
GRP_FS_ERR_BAD_DEV	指定したデバイスが見つからない
GRP_FS_ERR_BAD_PARAM	フォーマット情報パラメータ、メディア情報パラメータ、 指定したタイプ/クラス値でフォーマットできない、 トータルサイズ情報がドライバから得られない、
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	パーティション情報が不正
GRP_FS_ERR_NOMEM	I/O バッファを確保できない
GRP_FS_ERR_BUSY	指定したデバイスが mount されている



## 4.5.3 grp\_fat\_format\_sd (参考ライブラリ)

【機能概要】 SD カード、SDHC カードのフォーマット

【使用形式】

```
#include "grp_fat_format_sd.h"
int grp_fat_format_sd(const char *pcDev,
                      grp_uchar_t *pucVolLab, grp_uint32_t uiTotalSec,
                      grp_fat_format_param_t *ptParam, grp_fs_media_info_t *ptMedia);
```

【パラメータ】

pcDev	入力	フォーマット対象の SD カードのデバイス名称
pucVolLab	入力	ボリュームラベル名称（不必要な場合 NULL を指定）
uiTotalSec	入力	メディアサイズのセクタ数（自動でよい場合 0 を指定）
ptParam	出力	フォーマット情報パラメータ（出力のみ）
ptMedia	出力	メディア情報パラメータ（出力のみ）

【機能詳細】

pcDev で指定した SD/SDHC カードを SD/SDHC カードの規格に従い、パーティション設定をし、SD/SDHC カードのサイズに対応して、同規格で推奨されたフォーマットパラメータにて FAT ファイルシステムとしてフォーマットして初期化します。なお、本バージョンでは、フォーマットに必要な必要最小限のデータのみの書き込みを行い、メディア/パーティション全体にわたるデータの上書き削除や、アクセス不能セクタの検出、不良クラスタ登録等はいりません。

pcDev に指定する SD/SDHC カードのデバイス名称は、“sd0” のようにパーティション番号文字は指定せず、SD/SDHC カードに対応したデバイス種別名称とサブデバイス番号までを指定します。

pucVolLab は、フォーマットしたメディアに付けるボリュームラベル名称で、英大文字 11 文字以内で指定します。NULL を指定しますと、ボリュームラベルなしとします。

uiTotalSec は、メディアサイズのセクタ数の指定です。デバイスドライバで、メディアのサイズは自動的に検出しますので、通常、本パラメータは 0 を指定します。0 以外を指定しますと、実際のメディアサイズに関係なく、指定した領域分だけを FAT ファイルシステムとしてフォーマットします。但し、実際のサイズ（保護領域等を除いたサイズ）以上を指定しますと、GRP\_FS\_ERR\_BAD\_PARAM エラーとなります。

ptParam、ptMedia には、フォーマット結果のフォーマットパラメータ情報やパーティション情報を返す領域を指定します。汎用の grp\_fat\_format 関数では、これらのパラメータで示された構造体の各フィールドは、入力/出力の両方がありますが、本 grp\_fat\_format\_sd 関数では、出力のみとなっています。また、grp\_fat\_format 関数では、結果情報が不要の場合、NULL を指定しても OK ですが、本関数では、必ず、結果を格納する領域を確保して、コールする必要があります。

さらに、本関数では、フォーマットパラメータを決定するために、SD/SDHC カードのメディア全体のサイズ情報が必要なため、SD/SDHC カードのデバイスドライバがフォーマット対象のユーザ領域だけでなく、もう 1 つの領域である保護領域の情報を返す機能を持っていることを前提としています。具体的には、SD/SDHC カードのデバイスドライバが、デバイス番号内のサブデバイス

番号に GRP\_FAT\_SD\_PROT\_SUBDEV (0x08) のビットがたっているか否かで、SD/SDHC カードメモリのユーザデータ領域と、保護領域を区別し、それぞれのサブデバイス番号に対するオープンで、対応する領域のサイズを返すことを前提としています。例えば、サブデバイス番号が 1 の場合、SD/SDHC カードのデバイスドライバのオープン関数では、2 つ目の SD/SDHC デバイスのユーザデータ領域のサイズを返し、サブデバイス番号が 9 の場合、同じ 2 つめの SD/SDHC デバイスの保護領域のサイズを返すといった具合です。なお、保護領域のサブデバイス番号によるオープン処理でデバイスドライバからエラーが返ってきた場合は、エラーとせず、保護領域のサイズを 0 と仮定して、メディアサイズを計算します。

ptParam、ptMedia で指された領域の詳細につきましては、3.1 2.1 節をご参照下さい。

なお、本関数は、SD Card Association の規格書で規定された情報を基に作成されていますので、本関数、および、本関数のソースの一部を製品に利用する場合は、SD Card のライセンスが必要です。SD/SDHC カードのカードサイズに対応した、SD 規格推奨のフォーマットパラメータにつきましては、同規格の「SD Sepecification Part 2 File System Specification Ver. 2.00」をご参照下さい。

SD Card のライセンスにつきましては、SD Card Association にお問い合わせ下さい。

本参考ライブラリは、ご要望のある場合のみご提供しています。

#### 【リターン値】

GRP_FAT_TYPE_12	FAT12 でフォーマット完了
GRP_FAT_TYPE_16	FAT16 でフォーマット完了
GRP_FAT_TYPE_32	FAT32 でフォーマット完了
GRP_FS_ERR_BAD_DEV	指定したデバイスが見つからない、サイズ等が正しくない
GRP_FS_ERR_BAD_PARAM	uiTotalSec 値が正しくない トータルサイズ情報がドライバから得られない、
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	I/O バッファを確保できない
GRP_FS_ERR_BUSY	指定したデバイスが mount されている

## 4.5.4 grp\_fs\_chdir

【機能概要】 カレントディレクトリの設定

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_chdir(const grp_uchar_t *pucDir);
```

【パラメータ】

pucDir            入力                      カレントディレクトリのパス名   or   NULL

【機能詳細】

コールしたタスクのカレントディレクトリを pucDir で指定したパス名に設定します。pucDir に NULL または、長さ 0 の文字列 (“”) を指定した場合は、コールしたタスクのカレントディレクトリを無効化します。

なお、grp\_fs\_chdir を用いてカレントディレクトリを設定した場合は、ファイルシステムを unmount する前に、すべてのタスクのカレントディレクトリを同ファイルシステム外に設定するか、無効化する必要があります。unmount しようとするファイルシステム上にカレントディレクトリを持つタスクが存在しますと、GRP\_FS\_ERR\_BUSY のエラーで、unmount 処理がエラーとなります。無効化する前に強制 unmount を行った場合でも、カレントディレクトリを無効化する必要があります。無効化を行わない場合、mount 後の処理が正常に行えない場合があります。

また、カレントディレクトリを設定しない状態、あるいは、無効化した状態でも、**GR-FILE** の各処理関数は使用可能です。その場合、ファイル名の指定はフルパスで行なう必要があります。

【リターン値】

0	正常終了
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したディレクトリが存在しない
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスクの数がシステムの上限值を超えた
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.5 grp\_fs\_check\_fs\_dev

【機能概要】再挿入メディアのチェック

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_check_fs_dev(const char *pcDev,
                        grp_uchar_t *pucVolName, int *piVolNameLen,
                        grp_uint32_t *puiVolSerNo);
```

【パラメータ】

pcDev	入力	チェックすべきメディアのデバイス名
pucVolName	出力	読出したボリューム名を格納する領域のアドレス NULL の場合、ボリューム名は格納しない
piVolNameLen	入力	*piVolNameLen : 上記格納領域のサイズ
	出力	*piVolNameLen : 読出したボリューム名の長さ piVolNameLen が NULL の場合、ボリューム名は格納しない
PuiVolSerNo	出力	読出したボリュームシリアル番号を格納する領域のアドレス NULL の場合、シリアル番号は格納しない

【機能詳細】

unmount 処理されずに取出されたメディアが再挿入された際に、挿抜処理のシステムアプリケーションで本関数を使用します。

本関数は、pcDev で指定したメディアのボリューム名とシリアル番号をチェックし、unmount 処理されずに取出されたメディアかどうかを判定します。pcDev で指定するデバイス名は、メディアが挿入されたデバイスのデバイスドライバテーブルに登録されたデバイスタイプ名にサブデバイス番号、パーティション番号文字を付加した形で指定します。

もし、ボリューム名とシリアル番号が一致した場合は、unmount 処理されずにメディアが取出された際に、挿抜処理のシステムアプリケーションが実行した grp\_fs\_invalidate\_fs\_dev による同メディアへの I/O 抑止設定を解除し、さらに、grp\_fs\_sync 相当の処理を実行して、未反映となっていたキャッシュを同メディアに書戻します。ボリューム名とシリアル番号が一致しない場合は、同メディアへの I/O 抑止設定を継続します。

pucVolName、piVolNameLen、puiVolSerNo に NULL 以外を指定した場合は、メディアから読込んだボリューム名とシリアル番号の情報を指定した領域に格納します。この情報は、リターン値が 0 の場合、あるいは、GRP\_FS\_ERR\_NEED\_CHECK の場合にのみ有効です。

なお、メディアから読込んだボリューム名が、\*piVolNameLen で指定した領域サイズより大きい場合は、エラーとなります。

## 【リターン値】

0	取出されたメディアと同一メディア
GRP_FS_ERR_NEED_CHECK	取出されたメディアと異なる
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	pcDev で指定したデバイスが正しくない
GRP_FS_ERR_BAD_PARAM	pcDev で指定した領域が正しくない (NULL 等)。 ボリューム名の格納領域サイズが不足。

## 4.5.6 grp\_fs\_check\_volume

【機能概要】 未 mount メディアのボリューム名の取得

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_check_volume(const char *pcDev, const char *pcFsType,
                        grp_uchar_t *pucVolName, int *piVolNameLen,
                        grp_uint32_t *puiVolSerNo);
```

【パラメータ】

pcDev	入力	チェックすべきメディアのデバイス名
pcFsType	入力	チェックすべきデバイスのファイルシステムタイプ名 (FAT ファイルシステムの場合、“fat”を指定)
pucVolName	出力	読出したのボリューム名を格納する領域のアドレス。 NULL の場合、ボリューム名は格納しない
piVolNameLen	入力	*piVolNameLen : 上記格納領域のサイズ
	出力	*piVolNameLen : 読出したボリューム名の長さ piVolNameLen が NULL の場合、ボリューム名は格納しない
puiVolSerNo	出力	読出したのボリュームシリアル番号を格納する領域のアドレス。 NULL の場合、シリアル番号は格納しない

【機能詳細】

pcDev で指定したメディアを pcFsType で指定したファイルシステムとして解釈し、同メディアのボリューム名とシリアル番号を取得します。pcDev で指定するデバイス名は、メディアが挿入されたデバイスのデバイスドライバテーブルに登録されたデバイスタイプ名にサブデバイス番号とパーティション番号文字を付加した形で指定します。PcFsType で指定するファイルシステム名は、ファイルシステムテーブルに登録されたファイルシステムのタイプ名称を指定します。FAT ファイルシステムの場合は、“fat”を指定します。

pucVolName、piVolNameLen、puiVolSerNo に NULL 以外を指定した場合は、メディアから読込んだボリューム名とシリアル番号の情報を指定した領域に格納します。この情報は、リターン値が 0 の場合にのみ有効です。なお、メディアから読込んだボリューム名が、\*piVolNameLen で指定した領域サイズより大きい場合は、エラーとなります。

本関数は、挿抜処理のシステムアプリケーションで使用し、grp\_fs\_get\_error でセーブしたキャッシュデータがある場合、メディアが挿入された際に、grp\_fs\_mount を実行する前に本関数を実行することで、セーブしたキャッシュと同じメディアであるかどうかのチェックに使用します。同一メディアであることが判明した場合、挿抜処理のシステムアプリケーションは、利用者に確認を行い、デバイスドライバの I/O 関数を使用して、セーブしたキャッシュデータを同メディアに書戻してから grp\_fs\_mount を実行します。

なお、本関数は、まだ mount されていないメディアに対するボリューム名の取得関数です。すでに mount されているメディアのボリューム情報の取得は、grp\_fs\_get\_mnt、

grp\_fs\_get\_mnt\_by\_dev、grp\_fs\_get\_mnt\_by\_name を用いて行って下さい。

【リターン値】

0	取出されたメディアと同一メディア
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	pcDev で指定したデバイスが正しくない
GRP_FS_ERR_BAD_FSNAME	pcFsType で指定したファイルシステムタイプ名が正しくない
GRP_FS_ERR_BAD_PARAM	pcDev、pcFsType で指定した領域が正しくない (NULL 等)。 ボリューム名の格納領域サイズが不足。

## 4.5.7 grp\_fs\_chmod

【機能概要】 ファイルの保護モードの設定・変更

【使用形式】

```
#include "grp_fs_conv.h"
int grp_fs_chmod(const grp_uchar_t * pucPath, grp_uint32_t uiProt);
```

【パラメータ】

pucPath	入力	ファイル保護モードを設定・変更するファイル/ディレクトリ名
uiProt	入力	設定するファイルの保護モード（以下のモードの bit or で指定）
GRP_FS_PROT_RUSR	0400	オーナー read 可能
GRP_FS_PROT_WUSR	0200	オーナー write 可能
GRP_FS_PROT_XUSR	0100	オーナー実行可能
GRP_FS_PROT_RGRP	0040	オーナーグループ内 read 可能
GRP_FS_PROT_WGRP	0020	オーナーグループ内 write 可能
GRP_FS_PROT_XGRP	0010	オーナーグループ内実行可能
GRP_FS_PROT_ROTH	0004	他のユーザ read 可能
GRP_FS_PROT_WOTH	0002	他のユーザ write 可能
GRP_FS_PROT_XOTH	0001	他のユーザ実行可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが EXE”、“COM”、“DLL”、“BAT” の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、GRP\_FS\_PROT\_RUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pucPath で指定したファイル/ディレクトリの保護モードを uiProt に従い設定・変更します。

なお、FAT ファイルシステムの隠しファイル属性等、ファイルシステム固有の属性を設定する場合は、grp\_fs\_set\_attr を用いて行って下さい。

【リターン値】

0	正常終了
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_PERMIT	親ディレクトリに対する write が許可されていない
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名が不正である（NULL 等）
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）



#### 4.5.8 grp\_fs\_close

【機能概要】 ファイルのクローズ

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_close(int iFhdl);
```

【パラメータ】

iFhdl	入力	open 関数等で返ってきたファイルハンドル
-------	----	------------------------

【機能詳細】

grp\_fs\_open 関数等でオープンしたファイルをクローズします。

なお、タスク毎にオープンしたファイルが管理されていますので、ファイルをオープンした場合は、オープンしたタスクが本 grp\_fs\_close 関数を実行して必ずクローズ処理をするか、あるいは、タスクの消滅時に、OS または、別のタスクが、消滅したタスクのオープンファイルの無効化処理を grp\_fs\_task\_free\_env\_by\_id 等を使用して行って下さい。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

#### 4.5.9 grp\_fs\_closedir

【機能概要】 grp\_fs\_opendir でオープンしたディレクトリのクローズ

【使用形式】

```
#include "grp_fs_readdir.h"
int grp_fs_closedir(DIR *ptDirHdl);
```

【パラメータ】

ptDirHdl	入力	grp_fs_opendir 関数等で返ってきたファイルハンドル
----------	----	----------------------------------

【機能詳細】

grp\_fs\_opendir 関数でオープンしたディレクトリファイルをクローズします。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.10 grp\_fs\_create

【機能概要】 ファイル/ディレクトリの作成

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_create(const grp_uchar_t *pucPath, grp_uint32_t uiType,
                  grp_uint32_t uiProt, grp_uint32_t uiAttr);
```

【パラメータ】

pucPath	入力	作成するファイル/ディレクトリのパス名
uiType	入力	作成するファイルのタイプ
GRP_FS_FILE_FILE	1	通常ファイル
GRP_FS_FILE_DIR	2	ディレクトリ
GRP_FS_FILE_LINK	3	リンクファイル (FAT では指定不可)
GRP_FS_FILE_OTHER	4	その他のファイルシステム依存ファイル (FAT では指定不可)
uiProt	入力	ファイル/ディレクトリの保護モード
GRP_FS_PROT_RUSR	0400	オーナー read 可能
GRP_FS_PROT_WUSR	0200	オーナー write 可能
GRP_FS_PROT_XUSR	0100	オーナー実行可能
GRP_FS_PROT_RGRP	0040	オーナーグループ内 read 可能
GRP_FS_PROT_WGRP	0020	オーナーグループ内 write 可能
GRP_FS_PROT_XGRP	0010	オーナーグループ内実行可能
GRP_FS_PROT_OTH	0004	他のユーザ read 可能
GRP_FS_PROT_WOTH	0002	他のユーザ write 可能
GRP_FS_PROT_XOTH	0001	他のユーザ実行可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが **EXE**、**COM**、**DLL**、**BAT** の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、GRP\_FS\_PROT\_RUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

uiAttr	入力	ファイルシステム依存の属性情報
FAT ファイルシステムでは以下が指定可能です。これらは、"fat.h" に定義されています。		
FAT_ATTR_HIDDEN	0x02	隠しファイル
FAT_ATTR_SYSTEM	0x04	システムファイル
FAT_ATTR_ARCHIVE	0x20	バックアップ要

【機能詳細】

pucPath で指定したファイル/ディレクトリを uiProt で指定した保護モードで作成します。作成するファイルのタイプは、uiType で指定します。また、uiAttr により、ファイルシステム依存の属性を指定します。特別な属性設定が必要ない場合は、0 を指定します。

コンパイルスイッチ「GRP\_FS\_UPDATE\_ARCHIVE」を有効にしていた場合は、作成されるファイル/ディレクトリには ARCHIVE 属性がセットされます。

## 【リターン値】

0	作成成功
GRP_FS_ERR_PERMIT	親ディレクトリに対して <b>write</b> 権限がない
GRP_FS_ERR_NOT_FOUND	指定したパス中のディレクトリが存在しない
GRP_FS_ERR_EXIST	作成しようとしたディレクトリが既に存在する
GRP_FS_ERR_TOO_MANY	オープン中のファイル/タスク数がシステムの規定値を超えた
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pcDir で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.1 1 grp\_fs\_err

【機能概要】 エラー番号/エラーメッセージ変換

【使用形式】

```
#include "grp_fs_if.h"
char *grp_fs_err(int iErrNo, char *pcMsgBuf);
```

【パラメータ】

iErrNo	入力	<b>GR-FILE</b> 関数から返ってきたエラー番号
pcMsgBuf	出力	<b>GR-FILE</b> で規定されていないエラー番号の場合に、エラーメッセージを格納する 32 バイト以上の領域のアドレス。NULL の場合は、本関数がスタティックに保持した領域を使用する。

【機能詳細】

**GR-FILE** 関数から返ってきたエラー番号をシンボリックなエラーメッセージに変換し、変換結果のメッセージを格納した領域のアドレスをリターン値として返します。

iErrNo で指定したエラー番号が、**GR-FILE** で規定されたエラー番号の場合は、エラーメッセージテーブルに登録されたメッセージのアドレスを返します。規定されていないエラー番号の場合は、下記の変換を行い、pcMsgBuf で指定した領域、あるいは、本関数の持つスタティックな領域に変換結果の文字列を書込み、同領域のアドレスを返します。pcMsg に NULL を指定しますと、規定されていないエラー番号の場合、スタティックな領域を使って、エラーメッセージを返しますので、複数のタスクで同時に使用する可能性がある場合は、pcMsg に個別の領域のアドレスを指定して下さい。

正值の場合： 10 進数文字列に変換します

規定されていない負値の場合： -0XXXXXXXX の形で負値の 16 進文字列に変換します

【リターン値】

変換結果のエラーメッセージへのポインタ

## 4.5.1 2 grp\_fs\_get\_attr

【機能概要】 ファイル属性情報の取得

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_get_attr(const grp_uchar_t *pucPath, grp_fs_dir_ent_t *ptAttr);
```

【パラメータ】

pucPath	入力	ファイル属性情報を取得したいファイル/ディレクトリのパス名
ptAttr	出力	ファイル属性情報を格納する領域のアドレス

本パラメータの構造体の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照

【機能詳細】

pucPath で指定したファイル/ディレクトリの属性情報を取得し、ptAttr で指定した領域に格納します。本関数では、ptAttr の各フィールドをコール前に初期設定する必要はありません。本関数からのリターン時には、ptAttr で指定した構造体データのうち、iDev、uiFid、ucType、uiProtect、iSize、iCTime、iMtime、iAtime、uiAttr、uiMisc フィールドのみ有効な値を設定して返します。その他のフィールドの値は、有効性が保証されません。なお、FAT ファイルシステムの場合、uiMisc フィールドは、意味のある情報が入っていません。

得られる属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、grp\_fs\_dir\_ent 構造体の iSize メンバ変数は型が符号なし 32 ビットに変更され、名称も uiSize に変更されます。

【リターン値】

0	属性情報取得成功
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名、ptAttr が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.13 grp\_fs\_get\_cwd

【機能概要】 カレントディレクトリ名の取得

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_get_cwd(grp_uchar_t *pucPath, int iPathBufLen, int iSepChar);
```

【パラメータ】

pucPath	出力	取得したカレントディレクトリ名を格納する領域のアドレス
iPathBufLen	入力	上記格納領域のサイズ
iSepChar	入力	パス中の各ディレクトリ名間を区切るための文字

【機能詳細】

本関数をコールしたタスクのカレントディレクトリ名を pcBuf で指定した領域に返します。

カレントディレクトリが設定されていない場合は、既にマウントされているファイルシステムがあれば、最初にマウントされたファイルシステムのルートへのパスを返します。マウントされているファイルシステムがない場合は、GRP\_FS\_ERR\_NOT\_FOUND を返します。また、各コンポーネント長が、**GR-FILE** の config パラメータ GRP\_FS\_MAX\_COMP-1 よりも長い場合は、GRP\_FS\_MAX\_COMP-1 の長さで切り捨てられ、pucPath に返されます。

iPathBufLen で指定したサイズ内にカレントディレクトリ情報が収まらない場合や、その他のエラーが発生した場合も、エラーをリターン値として返します。

【リターン値】

0	カレントディレクトリ取得成功
GRP_FS_ERR_NOT_FOUND	カレントディレクトリが設定されておらず、かつ、マウントされているファイルシステムがない
GRP_FS_ERR_TOO_LONG	iPathBufLen で指定したサイズ内に収まらない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.14 grp\_fs\_get\_dirent

【機能概要】 ディレクトリエントリ情報の取得

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_get_dirent(int iFhdl, grp_fs_dir_ent_t *ptDirent);
```

【パラメータ】

iFhdl	入力	open 関数でオープンしたサーチ対象のディレクトリファイルのファイルハンドル、または、-1 (カレントディレクトリ)
ptDirent	入力	下記フィールドを設定した grp_fs_dir_ent_t 構造体のアドレス
		pucName: ファイル名を格納する領域のアドレス
		sNameSize: ファイル名の格納領域のサイズ
		uiStart: ディレクトリ内のサーチ開始位置
	出力	サーチ開始位置以降で見つかったディレクトリエントリの情報を本パラメータに指定した領域に格納する。なお、pucName で指定した領域には、見つかったファイル名を、sNameSize フィールドは、同ファイル名の長さを、uiStart には、同エントリのディレクトリ内の開始位置を設定する。本パラメータの構造体の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照。

【機能詳細】

iFhdl で指定したディレクトリファイルを検索し、ptDirent で指定したサーチ開始位置以降で見つかったフリーでないディレクトリエントリ情報を ptDirent で指定した領域に返します。

サーチ対象のディレクトリファイルは、本関数をコールする前に grp\_fs\_open 関数を使って read only でオープンし、得られたファイルハンドルを本関数に指定します。カレントディレクトリ内のディレクトリエントリを検索する場合は、オープンを行わず、ファイルハンドルとして -1 を指定します。

サーチ開始位置の指定は、ptDirent で指された構造体の uiStart フィールドに設定して本関数をコールすることで行います。ディレクトリエントリを検索する場合、カレントディレクトリファイル以外では、まず、grp\_fs\_open でオープンし、その後、uiStart フィールドを 0 に設定して本関数をコールします。ディレクトリエントリが見つかった場合、本関数は、見つかった情報を ptDirent で指定した構造体に格納し、メディア上のディレクトリエントリのサイズをリターン値として返します。以降、次のディレクトリエントリを見つけるには、前回のコールで返ってきた uiEnd フィールドの値を uiStart フィールドに設定してコールすることで行います。フリーでないエントリが見つからなかった場合は、リターン値として 0 を返します。リターン値が 0 の場合は、カレントディレクトリ以外では、grp\_fs\_close でクローズ処理を行い、サーチ処理を終了します。(下記サンプルコード参照)



また、見つかったディレクトリエントリのファイル名を格納するため、本関数をコールする際には、同ファイル名を格納するための領域を確保し、`ptDirent` で指定した構造体の `pucName` フィールドと `sNameSize` フィールドに、同領域のアドレスと、同領域のサイズを設定する必要があります。`pucName` 値は、一回設定すると、本関数をコールしても変更されませんが、`sNameSize` フィールドは、本関数からのリターン時に、取得したディレクトリエントリのファイル名の長さが設定されます。従いまして、`sNameSize` フィールドは、本関数を毎回コールする前に再設定することが必要です。

実際のファイル名の長さが、`sNameSize` で指定した長さを超えている場合は、ファイル名は `sNameSize` で指定した長さが返り NULL 文字が付加されません。また、`sNameSize` には `sNameSize` で指定した値-1 が返されます。

実際のファイル名の長さが、`sNameSize` 未満の場合は、NULL 文字付で、`pucName` で指定した領域に格納されます。

本関数により得られる属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

なお、FAT ファイルシステムの場合、ロングファイル名のディレクトリエントリは、物理的には、複数のエントリからなっているため、本関数では、これらのエントリを読み、論理的な 1 つのディレクトリエントリとして返します。この場合、`pucName` で指定した領域には、これらの複数エントリから得られるロングファイル名を格納します。default で提供している UNICODE-シフト JIS 変換を使用している場合、日本語のロングファイル名は、シフト JIS に変換して返します。但し、ロングファイル用のディレクトリエントリは、ファイル名以外の情報がないため、`iDev`、`ucType`、`pucName`、`sNameSize`、`uiAttr`、`uiStart`、`uiEnd` 以外のフィールドは、0 に設定して返します。同ロングファイル名に対応したファイルのサイズや時刻情報は、同ロングファイルのディレクトリエントリに続く、ショートネームのディレクトリエントリに格納されていますので、本関数を再度コールすることで得ることができます。

#### 【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、`grp_fs_dir_ent` 構造体の `iSize` メンバ変数は型が符号なし 32 ビットに変更され、名称も `uiSize` に変更されます。

#### 【リターン値】

正值	メディア上のディレクトリエントリのサイズ (ディレクトリエントリ情報取得成功)
0	フリーエントリ以外のディレクトリエントリ情報なし
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DIR	ファイルハンドルで指定したファイルがディレクトリでない
GRP_FS_ERR_BAD_PARAM	<code>ptDirent</code> および <code>pucName</code> フィールドで指定した領域が正しくない
	または、 <code>ptDirent</code> の <code>uiStart</code> フィールドの値が正しくない
GRP_FS_ERR_FS	ファイルシステムが正しくない

GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の参照）
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

## 【サンプルコード】

```

#include "grp_fs_if.h"

int dir_search (void) {
    int          iFhdl;          /* ファイルハンドル */
    int          iRet;          /* リターン値 */
    grp_fs_dir_ent_t tDirent;    /* ディレクトリエントリ情報 */
    grp_uchar_t   aucNameBuf[GRP_FS_MAX_COMP]; /* ファイル名格納領域 */
    grp_uchar_t   aucLongName[GRP_FS_MAX_COMP]; /* ロングファイル名格納領域 */

    iFhdl = grp_fs_open("ディレクトリ名", GRP_FS_O_RDONLY, 0); /* ディレクトリオープン */
    if (iFhdl < 0) /* オープン失敗 */
        return(-1); /* エラーリターン */
    tDirent.pucName = aucNameBuf; /* ファイル名格納領域設定 */
    tDirent.sNameSize = sizeof(aucNameBuf); /* 格納領域サイズ設定 */
    tDirent.uiStart = tDirent.uiEnd = 0; /* サーチ開始位置初期化 */
    while ((iRet = grp_fs_get_dirent(iFhdl, &tDirent)) > 0) { /* 次エン트리検索 */
        switch(tDirent.ucType) { /* ファイルタイプ判定 */
            case GRP_FS_FILE_FILE: /* 通常ファイル(ショートエン트리) */
                /* 通常ファイルの処理 */
                . . .
                break;
            case GRP_FS_FILE_DIR: /* ディレクトリ(ショートエン트리) */
                /* ディレクトリファイルの処理 */
                . . .
                break;
            case GRP_FS_FILE_LINK: /* ロングファイル名 */
                strcpy((char *)aucLongName, (char *)aucNameBuf); /* ロングファイル名セーブ */
                tDirent.sNameSize = sizeof(aucNameBuf); /* 格納領域サイズ再設定 */
                tDirent.uiStart = tDirent.uiEnd; /* サーチ開始位置前進 */
                iRet = grp_fs_get_dirent(iFhdl, &tDirent); /* 次エン트리取得 */
                if (iRet <= 0) /* 次エン트리なし/エラー */
                    || (tDirent.ucType != GRP_FS_FILE_FILE /* 通常ファイル以外 */
                        && tDirent.ucType != GRP_FS_FILE_DIR) { /* ディレクトリ以外 */
                        iRet = XXXX; /* リターン値設定 */
                        goto out; /* 処理終了 */
                    }
                /* ロングファイル名の処理 */
                . . .
                break;
            default: /* その他 */
                /* エラー処理 */
                . . .
                break;
        }
        tDirent.sNameSize = sizeof(aucNameBuf); /* 格納領域サイズ再設定 */
        tDirent.uiStart = tDirent.uiEnd; /* サーチ開始位置前進 */
    }
out:
    grp_fs_close(iFhdl); /* ディレクトリファイルクローズ */
    . . .
    return(iRet);
}

```

## 4.5.15 grp\_fs\_get\_error

【機能概要】 未反映キャッシュバッファの読出し・解放

【使用形式】

```
#include "grp_fs_if.h"

grp_int32_t grp_fs_get_error(int iMode, int iDev, grp_uint32_t uiBlk,
                             grp_uchar_t *pucBuf, grp_uint32_t uiSize,
                             grp_uint32_t *puiNeed);
```

【パラメータ】

iMode	入力	キャッシュバッファの読出しモードの指定 (bit or で指定)
GRP_FS_GE_CONTENT		キャッシュバッファ情報+キャッシュデータの読出し
GRP_FS_GE_RELEASE		キャッシュデータ読出し後、キャッシュバッファを解放
GRP_FS_GE_DIRTY		メディアに反映できていないデータを持つキャッシュバッファの読出し
		(本ビット指定がない場合は、write を失敗したバッファのみ)
GRP_FS_GE_FBUF		ファイル管理ブロックキャッシュの読出し
GRP_FS_GE_DBUF		ファイルデータキャッシュの読出し
iDev	入力	読出し対象のデバイス番号
GRP_FS_GE_DEV_ANY		全てのデバイス
uiBlk	入力	読出し対象のブロック番号
GRP_FS_GE_BLK_ANY		全てのブロック
pucBuf	出力	読出したキャッシュデータを格納する領域のアドレス
uiSize	入力	上記格納領域のサイズ
puiNeed	出力	必要な格納領域のサイズ情報を格納する変数のアドレス

【機能詳細】

本関数は、挿抜処理のシステムアプリケーション等で使用し、write エラーやメディアの不当取出しで、反映不能となったキャッシュデータをキャッシュバッファから読出し、同キャッシュバッファの解放を行います。

読出し対象のキャッシュデータは、iMode、iDev、uiBlk で指定します。

まず、iMode で、GRP\_FS\_GE\_FBUF、GRP\_FS\_GE\_DBUF を指定し、ファイル管理ブロックキャッシュか、ファイルデータキャッシュか、両方かを指定します。

さらに、実際にキャッシュデータを読出す場合、GRP\_FS\_GE\_CONTENT も指定します。

GRP\_FS\_GE\_CONTENT 指定がある場合は、反映不能となったキャッシュバッファの情報を grp\_fs\_err\_binfo\_t の構造体（詳細は、前述の”アプリケーションインタフェース関連の構造体”の節参照）の形で指定の領域に格納し、さらに、同キャッシュバッファのデータも同構造体に引続き格納します。GRP\_FS\_GE\_CONTENT を指定がない場合は、キャッシュバッファ情報のみを指定した領域に格納します。

本関数は、default では、write を失敗したキャッシュバッファのみを取得の対象とします。しかし、メディアを取外した場合等では、同メディアに対するキャッシュデータは同メディアが再挿入されない限り反映不能という状態となります。このようなケースでは、データ更新後書戻しが行われて

いないキャッシュバッファも取得対象とする必要がありますので、このような場合には、さらに、iMode に GRP\_FS\_GE\_DIRTY を or で指定します。

また、反映不能となったデータを読み出し後、同キャッシュバッファを解放する場合は、GRP\_FS\_GE\_RELEASE を or で指定します。なお、GRP\_FS\_GE\_RELEASE を指定した場合は、GRP\_FS\_GE\_CONTENT の指定が無くても、GRP\_FS\_GE\_CONTENT が指定されたと仮定します。

特定のデバイス、特定のブロックのキャッシュバッファのみを対象とする場合は、iDev、uiBlk で対象のデバイスやブロック番号を指定します。デバイスやブロックを特定しない場合は、iDev、uiBlk に GRP\_FS\_GE\_DEV\_ANY、GRP\_FS\_GE\_BLK\_ANY を指定します。

一方、読み出したデータの格納場所、および、同領域のサイズは、pucBuf、uiSize パラメータで指定します。格納すべきデータが uiSize より大きい場合は、uiSize で指定したサイズに収まる分だけ格納し、実際に格納したサイズを本関数のリターン値として返します。

なお、指定した対象の全キャッシュデータを格納できた場合も、uiSize が小さく、部分的にしか格納できなかった場合でも、puiNeed で指定した領域には、全対象データを格納するのに必要なサイズを返します。この機能を利用すれば、最初 uiSize に 0 を指定して、処理に必要な領域のサイズを確認し、必要なサイズの領域を確保して、再度、本関数を実行するということが可能です。また、最初、GRP\_FS\_GE\_CONTENT を指定せず、反映不能となったキャッシュバッファの情報だけ取出し、取出した情報に従い、iDev、uiBlk を指定して、1 ブロックずつ読み出すという方法も可能です。

本関数により読み出したデータは、一旦セーブし、次回対応するメディアが挿入された際に、mount を行う前に、挿抜処理のシステムアプリケーション等が、利用者に確認して、デバイスドライバの I/O 関数を使って書戻します。セーブされたデータに対応したメディアが挿入されたか否かは、セーブ時に、grp\_fs\_get\_mnt\_by\_dev を用いて取得したメディアのボリューム名とシリアル番号をデータと一緒にセーブし、メディア挿入時に grp\_fs\_check\_volume を使って挿入されたメディアのボリューム名とシリアル番号を確認することで行います。

#### 【リターン値】

0 または 正值	指定した領域に格納したデータのサイズ
GRP_FS_ERR_BAD_PARAM	iMode に GRP_FS_GE_FBUF、GRP_FS_GE_DBUF のいずれの指定もない。 pucBuf で指定された領域が正しくない。

## 4.5.16 grp\_fs\_get\_mnt

【機能概要】 全 mount 情報の取得

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_get_mnt(int iMaxCnt, grp_fs_mnt_info_t *ptMntInfo);
```

【パラメータ】

iMaxCnt	入力	取得する mount 情報の最大数
ptMntInfo	出力	mount 情報を格納する領域のアドレス (iMaxCnt 個数分の領域が必要)

【機能詳細】

すでに mount されている全メディアの mount 情報を取得し、ptMntInfo で指定した領域に格納します。iMaxCnt が現在 mount されているメディアの数より小さい場合は、iMaxCnt 分だけ取得します。

mount 情報として返す構造体 grp\_fs\_mnt\_info\_t の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【リターン値】

0 または 正值	取得した mount 情報の数
GRP_FS_ERR_BAD_PARAM	ptMntInfo で指定した領域が正しくない (NULL 等)

## 4.5.17 grp\_fs\_get\_mnt\_by\_dev

【機能概要】 デバイス番号による特定 mount 情報の取得

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_get_mnt_by_dev(int iDev, grp_fs_mnt_info_t *ptMntInfo);
```

【パラメータ】

iDev	入力	mount 情報取得対象のデバイス番号
ptMntInfo	出力	mount 情報を格納する領域のアドレス

【機能詳細】

iDev で指定したデバイスメディアの mount 情報を取得し、ptMntInfo で指定した領域に格納します。 mount 情報として返す構造体 grp\_fs\_mnt\_info\_t の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【リターン値】

0	取得成功
GRP_FS_ERR_BAD_DEV	iDev で指定したデバイスメディアは、mount されていない
GRP_FS_ERR_BAD_PARAM	ptMntInfo で指定した領域が正しくない (NULL 等)

## 4.5.18 grp\_fs\_get\_mnt\_by\_name

【機能概要】 デバイス名による特定 mount 情報の取得

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_get_mnt_by_name(const char *pcDevName, grp_fs_mnt_info_t *ptMntInfo);
```

【パラメータ】

pcDevName	入力	mount 情報取得対象のデバイス名
ptMntInfo	出力	mount 情報を格納する領域のアドレス

【機能詳細】

pcDevName で指定したデバイスメディアの mount 情報を取得し、ptMntInfo で指定した領域に格納します。mount 情報として返す構造体 grp\_fs\_mnt\_info\_t の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【リターン値】

0	取得成功
GRP_FS_ERR_BAD_DEV	pcDevName で指定したデバイスメディアは mount されていない
GRP_FS_ERR_BAD_PARAM	ptMntInfo で指定した領域が正しくない (NULL 等)



## 4.5.19 grp\_fs\_init

【機能概要】 **GR-FILE** の初期化

【使用形式】

```
#include "grp_fs_if.h"
int  grp_fs_init(void);
```

【パラメータ】

なし

【機能詳細】

**GR-FILE** を初期化、あるいは、再初期化し、使用可能な状態とします。本関数は、グローバル構造体変数 `grp_fs_param` に設定された各種パラメータを参照し、キャッシュバッファ等、**GR-FILE** で必要な各種領域を確保します。従いまして、設定値を変更する場合は、本関数をコールする前に、`grp_fs_param` の値を変更する必要があります。

すでに、`grp_fs_init` を実行し、再度実行した場合は、前回確保した領域を解放後、再度確保しなおします。但し、セマフォ等のリソースを解放は行いません。

`grp_fs_param` に設定する各種パラメータの値については、前述の「各種パラメータの設定・変更」の節を参照下さい。

【リターン値】

0	初期化成功
GRP_FS_ERR_NOMEM	メモリが足りない
GRP_FS_ERR_BAD_PARAM	<code>grp_fs_param</code> で指定されたパラメータが正しくない

## 4.5.20 grp\_fs\_invalidate\_fs\_dev

【機能概要】 メディアに対する I/O 抑止設定（メディア不当取出し時）

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_invalidate_fs_dev(const char *pcDevName);
```

【パラメータ】

pcDevName	入力	I/O 抑止対象のデバイス名
-----------	----	----------------

【機能詳細】

pcDevName で指定したデバイスメディアに対する以降 **GRP-FILE** を通したすべての I/O 要求を抑止し、GRP\_FS\_ERR\_IO エラーで返します。但し、キャッシュ上にあるデータで処理が可能な要求は、エラーとはなりません。

本関数は、unmount 処理をされずにメディアが不当に取出された際に、挿抜処理を行うシステムアプリケーション等で使用します。

【リターン値】

0	I/O 抑止設定成功
GRP_FS_ERR_BAD_DEV	pcDevName で指定したデバイスは、mount されていない

## 4.5.2 1 grp\_fs\_lookup\_dev

【機能概要】 デバイス名称→デバイス番号変換

【関数形式】

```
#include "grp_fs_if.h"
int grp_fs_lookup_dev (const char *pcDev);
```

【パラメータ】

pcDev	入力	アプリケーションタスク空間に格納された検索対象のデバイス名 (デバイスタイプ名称+サブデバイス番号+パーティション番号 文字)
-------	----	---

【機能詳細】

**GR-FLE** のデバイステーブルを検索して、pcDev で指定したデバイス名に対応するデバイス番号を返します。

pcDev には、"USB1a"のように、デバイスタイプ名称にサブデバイス番号をとパーティション番号文字を付加した形のデバイス名を指定します。本関数は、指定されたデバイスタイプ名称と、デバイステーブル **grp\_fs\_dev\_tbl** の pcDevName フィールドを比較し、一致したエントリのエントリ番号に、パーティション番号とサブデバイス番号を 3.5.3 節で示した形でエンコードしたデバイス番号を返します。

【リターン値】

正值	デバイス番号
GRP_FS_ERR_BAD_DEV	指定のデバイスタイプ名を持つデバイステーブルエントリが存在しない
GRP_FS_ERR_BAD_PARAM	pcDev で指定したアドレスが正しくない

## 4.5.2 2 grp\_fs\_lseek

【機能概要】 ファイルの読書き位置の設定・変更

【使用形式】

```
#include "grp_fs_if.h"
grp_ioffset_t grp_fs_lseek(int iFhdl, grp_ioffset_t iOffset, int iMode);
```

【パラメータ】

iFhdl	入力	grp_fs_open 関数等で返ってきたファイルハンドル
iOffset	入力	ファイルの読書き位置情報
iMode	入力	ファイルの読書き位置情報の解釈方法
GRP_FS_SEEK_SET		uiOffset で指定した値を読書き位置に設定
GRP_FS_SEEK_CUR		現在の読書き位置+iOffset の値を読書き位置に設定 (GRP_FS_O_APPEND モードでオープンされている場合は、 GRP_FS_SEEK_END と同じ)
GRP_FS_SEEK_END		現在のファイルサイズ+iOffset の値を読書き位置に設定

【機能詳細】

iFhdl で指定したファイルの read/write 時に使用するファイルの読書き位置を iOffset、iMode に従い設定し、設定した読書き位置を返します。

なお、ファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、write 時にエラーとなります。また、読書き位置が負値になった場合は、0 に自動補正します。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と grp\_fs\_lseek4G などの 4G 対応 API を混在して使用しないでください。grp\_fs\_lseek4G などで 2GB 以上のオフセットに移動したのちに本 API を使用し、オフセットが 2GB 以上になった場合、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0 または 正値	設定した読書き位置
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_BAD_PARAM	iMode パラメータが正しくない

## 4.5.23 grp\_fs\_lseek4G

【機能概要】 ファイルの読書き位置の設定・変更 4G 対応版

【使用形式】

```
#include "grp_fs_conv.h"
int grp_fs_lseek4G(int iFhdl, grp_uioffset_t uiOffset, int iMode,
                  grp_uioffset_t *puiResultOffset);
```

【パラメータ】

iFhdl	入力	grp_fs_open 関数等で返ってきたファイルハンドル
uiOffset	入力	ファイルの読書き位置情報
iMode	入力	ファイルの読書き位置情報の解釈方法
GRP_FS_SEEK_SET		uiOffset で指定した値を読書き位置に設定
GRP_FS_SEEK_CUR		現在のファイル読書き位置+uiOffset の値を読書き位置に設定 (GRP_FS_O_APPEND モードでオープンされている場合は、 GRP_FS_SEEK_END と同じ)
GRP_FS_SEEK_END		現在のファイルサイズ+uiOffset の値を読書き位置に設定
GRP_FS_SEEK_MINUS		読書き位置の算出の際に uiOffset を負値として扱う。 (本モードのみでの指定は不可。GRP_FS_SEEK_CUR/ GRP_FS_SEEK_END と bit or で指定)
puiResultOffset	出力	設定後のファイルの読書き位置情報

【機能詳細】

本機能は、コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合に使用できます。

iFhdl で指定したファイルの read/write 時に使用するファイルの読書き位置を uiOffset, iMode に従い設定し、puiResultOffset に設定した読書き位置を返します。

なお、現在のファイルサイズを超えた位置に読書き位置を設定することは可能ですが、FAT ファイルシステムの場合は、read/write 時にエラーとなります。

また、読書き位置にオーバーフローもしくはアンダーフローが発生した場合には、エラーを返します。(この場合のオーバーフローは 4G-1 バイトを超えるオフセットを指定した場合を指し、アンダーフローは 0 バイトを下回るオフセットを指定した場合を指します)

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、本 API と grp\_fs\_lseek などの 4G 未対応 API を混在して使用しないでください。本 API で 2GB 以上のオフセットに移動したのちに grp\_fs\_lseek などを使用し、オフセットが 2GB 以上になった場合、ファイルの読書き位置が負値として認識されるため 0 に自動補正されます。

【リターン値】

0 正常終了

GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_BAD_PARAM	iMode パラメータが正しくない
GRP_FS_ERR_BAD_OFF	uiOffset パラメータが正しくない

## 4.5.2 4 grp\_fs\_mount

【機能概要】 メディアの mount 処理（メディア挿入時）

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_mount(const char *pcDevName, const grp_uchar_t *pucMp,
                 const char *pcFsType, int iMode);
```

【パラメータ】

pcDevName	入力	mount 対象のメディアのデバイス名
pucMp	入力	mount 先ディレクトリのパス名（ルートからの絶対パス）
pcFsType	入力	mount するファイルシステムタイプの名前 （FAT ファイルシステムの場合、“fat”を指定する）
iMode	入力	mount モードの指定（以下を bit or で指定）
GRP_FS_RDONLY		read-only で mount
GRP_FS_FORCE_MOUNT		強制 mount（前回 unmount 処理されていないメディアの mount 等）に使用
GRP_FS_NO_UPD_ACCTIME		アクセス時刻記録のメディアへの反映抑止
GRP_FS_NO_MNT_FLAG		メディアへのマウント中フラグの書き込み抑止
GRP_FS_NO_CRT_ACCTIME		メディア上の作成/アクセス時刻記録なし
GRP_FS_SYNC_ALL		write-through 方式のキャッシュ反映方式使用
GRP_FS_SYNC_FL_CLOSE		each-close 方式のキャッシュ反映方式使用
GRP_FS_SYNC_FS_CLOSE		last-close 方式のキャッシュ反映方式使用

なお、GRP\_FS\_SYNC\_ALL、GRP\_FS\_SYNC\_FL\_CLOSE、GRP\_FS\_SYNC\_FS\_CLOSE は、多くともいずれか 1 つだけを選択します。これらをいずれも指定しない場合は、unmount 方式のキャッシュ反映方式を使用します。

【機能詳細】

pcDevName で指定したデバイスメディアを、pcFsType で指定したファイルシステムとして、pucMp で指定したディレクトリに mount し、同メディアをアクセス可能とします。本関数は、メディアが挿入された際に、挿抜処理を行うシステムアプリケーション等で使用します。

pcDevName で指定するデバイス名は、メディアが挿入されたデバイスのデバイスドライバテーブルに登録されたデバイスタイプ名にサブデバイス番号とパーティション番号文字を付加した形で指定します。

PcFsType で指定するファイルシステム名は、ファイルシステムテーブルに登録されたファイルシステムのタイプ名称を指定します。FAT ファイルシステムの場合は、“fat”を指定します。

pucMp で指定する mount 先ディレクトリのパス名は、個別のファイルシステムとして扱う場合は、新たなドライブ名を指定し、既に mount されたファイルシステムの一部に接続する場合は、接続する同ファイルシステムのディレクトリをルートからの絶対パスで指定します。例えば、“D:”ドライブとして独立したファイルシステムの形で mount する場合は、“D:”と指定します。既に、’/’に mount されたファイルシステムの “/mnt” ディレクトリ下に mount する場合は、“/mnt” と指定します。詳細は、「ドライブ型+階層化 mount」の節を参照下さい。

iMode パラメータでは、read-only、強制 mount、キャッシュの反映方式などを指定します。

強制 mount オプション **GRP\_FS\_FORCE\_MOUNT** は、前回 unmount 処理されていないメディアを mount する際等に使用します。例えば、FAT16/32 ファイルシステムの場合、unmount 処理しないでメディアを取出しますと、unmount 処理がされていないという情報がメディアに残ったままになっています。このようなメディアを mount しようとしますと、default では、**GRP\_FS\_ERR\_NEED\_CHECK** というエラーを返します。そこで、**GRP\_FS\_ERR\_NEED\_CHECK** エラーが返ってきたメディアを再度強制的に mount する場合に、**GRP\_FS\_FORCE\_MOUNT** オプションを指定します。**GRP\_FS\_READONLY** を指定した場合も、書き込みを行わないため、エラーとせず、強制的な mount を行います。

iMode で **GRP\_FS\_NO\_UPD\_ACCTIME** を指定しますと、ファイルのアクセス時刻記録のメディアへの反映を抑止し、アクセスだけでは反映せず、更新等があった場合に反映します。

iMode で **GRP\_FS\_NO\_CRT\_ACCTIME** を指定しますと、メディア上は、ファイルの作成/アクセス時刻記録はなしとして扱います。このモードを FAT ファイルシステムに使用した場合、メディア上のディレクトリ内にあるファイルの作成/アクセス時刻情報フィールドを、作成/アクセス時刻情報として読み出しますが、作成/更新時には同フィールドを 0 に設定します。また、**GRP\_FS\_NO\_UPD\_ACCTIME** と同様に、アクセスだけの時刻情報の反映を抑止します。但し、**GR-FILE** 内では、作成時刻やアクセス時刻を管理していますので、**grp\_fs\_get\_attr**、**stat**、**get\_fs\_get\_dirent** 等で返ってくるファイルの作成/更新時刻情報は、メディア上の値とは異なり、0 以外の値となる場合があります。

また、iMode で **GRP\_FS\_NO\_MNT\_FLAG** を指定しますと、現在、同メディアはマウント中であるというフラグをメディアに書き込むことを抑止します。例えば、FAT16、FAT32 のファイルシステムでは、FAT 1 にキャッシュが CLEAN ビットという情報があり、**GR-FILE** では、通常、mount 時にこのビットを落とし、unmount 時に再度設定するという処理を行なっています。iMode で **GRP\_FS\_NO\_MNT\_FLAG** を設定しますと、この CLEAN ビットの操作を行なわないようにします。

iMode で指定するキャッシュの反映方式の詳細につきましては、「キャッシュの write 制御」の節を参照下さい。

mount 処理の中では、メディアのライトプロテクト状態は確認せず、処理を行おうとします。その結果、ライトプロテクト状態のメディアは **GRP\_FS\_ERR\_IO** が発生する事があります。予め mount 処理前にメディアのライトプロテクト状態を **grp\_fs\_ioctl\_dev0**などで確認し、mount オプションを決定して下さい。**grp\_fs\_ioctl\_dev0**でのライトプロテクト状態取得はデバイスドライバ依存ですので、適宜ポーティングを行ってからご使用下さい。デバイスドライバにライトプロテクト状態を取得する機能が無い場合、mount 処理と mount オプションを組合せ、数回実行する事でライトプロテクト状態を予測する事が出来ます。サンプルフローに手順を示します。ただし、サンプルフローの手順を行ったとしても、メディアのライトプロテクト状態は正確には判断できません。

複数のタスクで複数のメディアを同時にマウントを行うとエラーとなる場合がありますのでご注意ください。

#### 【リターン値】

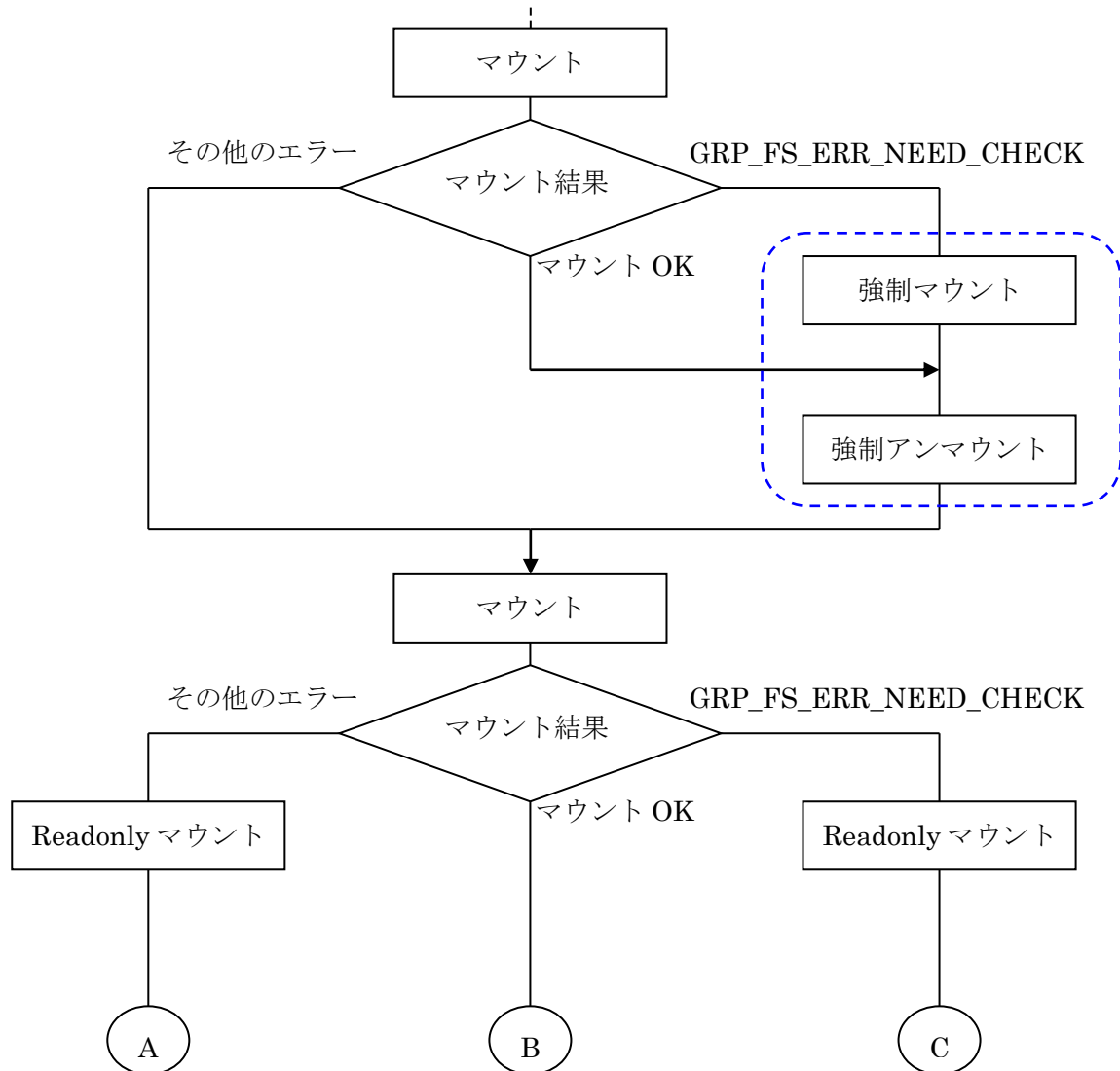
0	mount 成功
<b>GRP_FS_ERR_NEED_CHECK</b>	前回正常に unmount されておらず、ファイルシステムに不整合がある可能性がある



GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	pcDevName で指定したデバイス名が正しくない
GRP_FS_ERR_PERMIT	write 保護がかかっている write 可能モードでは mount できない
GRP_FS_ERR_TOO_MANY	マウント中のメディアの数やオープン中のファイル/タスク数がシステムの規定値を超えた
GRP_FS_ERR_EXIST	既に、mount されている
GRP_FS_ERR_BUSY	指定したデバイスまたはマウント先ディレクトリは、既に、mount されている、あるいは、アクセス中である
GRP_FS_ERR_TOO_LONG	マウント先ディレクトリとして指定したパス名が長すぎる
GRP_FS_ERR_NOMEM	mount 処理に必要なメモリが確保できない 使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 【サンプルフロー】

メディアのライトプロテクト状態を簡易的に確認するサンプルフローを示します。  
本サンプルフローを用いても、ライトプロテクト状態は確実にわかりません。



[-] 前回 **unmount** 処理されていないという情報をクリアする為に、一度強制マウント/アンマウントを行います。本処理には、書き込みが出来るか確認の意味も含まれます。

- A Readonly マウントの結果、マウントできた場合はライトプロテクト状態の可能性があります。
- B 読み書き可能なマウント状態。
- C Readonly マウントの結果、マウントできた場合は前回 **unmount** 処理されていないメディアで、かつ、ライトプロテクト状態の可能性があります。

A、C の Readonly マウントの結果、エラーの場合は、マウントは行えません。エラーコードを確認しエラー処理を行います。

## 4.5.25 grp\_fs\_open

【機能概要】 ファイルのオープン

【使用形式】

```
#include "grp_fs_if.h"

int grp_fs_open(const grp_uchar_t *pucFile, int iMode, grp_uint32_t uiProt);
```

【パラメータ】

pucFile	入力	オープン/生成するファイルのパス名
iMode	入力	オープンモード (GRP_FS_O_APPEND 以下のモードは、bit or で指定可能)
GRP_FS_O_RDONLY	0x0000	read のみのモードでオープン
GRP_FS_O_WRONLY	0x0001	write のみのモードでオープン
GRP_FS_O_RDWR	0x0002	read/write 両方が可能なモードでオープン
GRP_FS_O_APPEND	0x0008	ファイルへの追加 write モードでオープン (write 要求毎に、常にファイルの最後に位置づけて処理)
GRP_FS_O_CREAT	0x0200	ファイルが存在しない場合、ファイルを生成
GRP_FS_O_TRUNC	0x0400	ファイルをオープン後、ファイルを空の状態にする
GRP_FS_O_EXCL	0x0800	GRP_FS_O_CREAT が指定され、かつ、既に同ファイルが存在する場合エラーで返す。
GRP_FS_O_DIRECT_IO	0x8000	キャッシュ上に対応するデータがない場合は、可能な限りキャッシュバッファを使用せず、アプリケーションバッファとメディアとの間で直接、かつ、連続したブロックで I/O を行う。
uiProt	入力	iMode で GRP_FS_O_CREAT を指定した場合の作成される ファイルの保護モード
GRP_FS_PROT_RUSR	0400	オーナー read 可能
GRP_FS_PROT_WUSR	0200	オーナー write 可能
GRP_FS_PROT_XUSR	0100	オーナー実行可能
GRP_FS_PROT_RGRP	0040	グループ内 read 可能
GRP_FS_PROT_WGRP	0020	グループ内 write 可能
GRP_FS_PROT_XGRP	0010	グループ内実行可能
GRP_FS_PROT_ROTH	0004	他のユーザ read 可能
GRP_FS_PROT_WOTH	0002	他のユーザ write 可能
GRP_FS_PROT_XOTH	0001	他のユーザ実行可能

FAT ファイルシステムの場合は、ユーザの概念がありません。その為「オーナー」、「グループ」、「他のユーザ」の保護モードはオーナービット値の指定で統一されます。この時、「グループ」、「他のユーザ」の指定は無視されます。但し、隠しファイルの場合は、「グループ」、「他のユーザ」ビットを自動的に 0 に設定します。さらに、FAT ファイルシステムには実行可否の概念がなく、情報も保持できないため、「実行可能」ビットを指定しても無視されます。「実行可能」ビットは、ファイル名のサフィックスが EXE、COM、DLL、BAT の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、GRP\_FS\_PROT\_RUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

【機能詳細】

pucFile で指定したファイルをオープン/生成し、read/write 処理のためのファイルハンドルを返します。ファイル生成等のオープン時の処理モードや、read のみ、write のみ、read/write 両方等

のオープン後の処理モードを `iMode` で指定します。`iMode` では、`GRP_FS_O_RDONLY`、`GRP_FS_O_WRONLY`、`GRP_FS_O_RDWR` の何れか 1 つを指定して、`read/write` の区別を指定し、その他のモードは、オプションとして `or` の形で追加で指定します。`GRP_FS_O_RDONLY`、`GRP_FS_O_WRONLY`、`GRP_FS_O_RDWR` のいずれも指定しなかった場合は、`GRP_FS_O_RDONLY` が仮定されます。`GRP_FS_O_APPEND` を指定しても、`GRP_FS_O_WRONLY`、または、`GRP_FS_O_RDWR` を指定していなかった場合は、`write` できないことに注意して下さい。また、`iMode` で `GRP_FS_O_CREAT` を指定した場合の生成するファイルの保護モードを `uiProt` で指定します。なお、`iMode` に `GRP_FS_O_CREAT` を指定しない場合も、`uiProt` を指定する必要がありますが、その場合、`uiProt` の値は、無視されます。

`puFile` には、通常のファイル以外に、ディレクトリを指定することも可能です。但し、ディレクトリは、`O_RDONLY` モードでしかオープンできません。

`GRP_FS_O_DIRECT_IO` は、**GR-FILE** 固有のオプションで、キャッシュ上に対応するデータがない場合は、可能な限りキャッシュバッファを使用せず、アプリケーションの指定したバッファとメディアとの間で直接 I/O を行い、しかもメディア上で連続したブロックは一括して I/O を行なう指定です。本オプションは、一回の `read/write` 要求のサイズが大きく、頻繁には `read` しないファイルの I/O を高速に行うのに適したオプションです。なお、本オプションは、ファイルシステム依存部でサポートされている場合のみ有効です。**GR-FILE** の FAT ファイルシステムでは、シングル空間のシステムで、`GRP_FS_FAT_DIRECT_IO` オプション付で **GR-FILE** を構築した場合に有効となります。

`GRP_FS_O_DIRECT_IO` には使用上の注意点があります。

詳しくは、本マニュアルの「3.8 アプリケーションバッファとメディア間の直接/連続ブロック I/O (ダイレクト I/O 機能)」を参照下さい。

コンパイルスイッチ「`GRP_FS_UPDATE_ARCHIVE`」を有効にしていた場合は、作成されるファイル/ディレクトリには `ARCHIVE` 属性がセットされます。

#### 【リターン値】

0 または 正值	<code>read/write</code> 処理のためのファイルハンドル
<code>GRP_FS_ERR_PERMIT</code>	<code>GRP_FS_O_CREAT</code> を指定した場合で、親ディレクトリまたは同ファイルに対して <code>write</code> 権限がない
<code>GRP_FS_ERR_TOO_MANY</code>	オープン中のファイル/タスク数がシステムの規定値を超えた
<code>GRP_FS_ERR_NOT_FOUND</code>	指定したパス中のディレクトリが存在しない
<code>GRP_FS_ERR_EXIST</code>	<code>GRP_FS_O_CREAT</code> 指定がなく、指定したファイルが存在しない <code>GRP_FS_O_CREAT</code> かつ <code>GRP_FS_O_EXCL</code> 指定があり、指定したファイルが既に存在する
<code>GRP_FS_ERR_IO</code>	I/O エラー
<code>GRP_FS_ERR_TOO_LONG</code>	指定したパス名が長すぎる
<code>GRP_FS_ERR_BAD_PARAM</code>	<code>puFile</code> で指定したパス名が不正である (NULL 等)
<code>GRP_FS_ERR_FS</code>	ファイルシステムが正しくない
<code>GRP_FS_ERR_NOMEM</code>	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.26 grp\_fs\_opendir

【機能概要】 grp\_fs\_readdir でディレクトリエントリ読み出しを行うためのディレクトリのオープン

【使用形式】

```
#include "grp_fs_readdir.h"
DIR *grp_fs_opendir(grp_uchar_t *pucPath);
```

【パラメータ】

pucPath	入力	オープンするディレクトリファイル名称
---------	----	--------------------

【機能詳細】

POSIX 互換インタフェースである grp\_fs\_readdir を使ってディレクトリエントリの読み出しを行うためのディレクトリファイルのオープンを行います。pucPath パラメータでオープンするディレクトリファイル名を指定します。

【リターン値】

NULL	オープン失敗
NULL 以外	grp_fs_readdir、grp_fs_closedir で使用するファイルハンドル 情報

## 4.5.27 grp\_fs\_read

【機能概要】 ファイルの読み込み

【使用形式】

```
#include "grp_fs_if.h"
grp_isize_t grp_fs_read(int iFhdl, grp_uchar_t *pucBuf, grp_isize_t iSize);
```

【パラメータ】

iFhdl	入力	open 関数等で返ってきたファイルハンドル
pucBuf	出力	読込んだデータを格納する領域のアドレス
iSize	入力	読み込むデータのサイズ

【機能詳細】

iFhdl で指定したファイルから、iSize バイト分データを読み込み、pucBuf で指定した領域に格納します。指定したファイルの現在の読書き位置から同ファイルの最後まで残りバイト数が iSize 分より少ない場合は、残りバイト数分だけを指定した領域に格納します。

リターン値としては、実際に読めたバイト数を返します。

なお、オープン時に GRP\_FS\_O\_DIRECT\_IO の指定がなかった場合は、キャッシュバッファを介して read 処理を行い、キャッシュ上にあれば、実際にメディアからは read せず、キャッシュからデータを返します。GRP\_FS\_O\_DIRECT\_IO 指定があり、ファイルシステム依存部で同オプションをサポートしている場合も、キャッシュ上にある場合の動作は同じですが、対象のデータがキャッシュ上にない場合は、可能な限りキャッシュバッファを使用せず、メディアからアプリケーションの指定したバッファに対し、直接連続ブロックで I/O を行います。

**GR-FILE** の FAT ファイルシステムでは、シングল空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファへの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続していれば、連続した単位で、一括してメディアから read します。

【リターン値】

0 または 正值	実際に読めたバイト数
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	read 可能なモードでオープンされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

## 4.5.28 grp\_fs\_readdir

【機能概要】 ディレクトリエントリの読み出し（POSIX 互換用）

【使用形式】

```
#include "grp_fs_readdir.h"
struct dirent *grp_fs_readdir(DIR *ptDirHdl);
```

【パラメータ】

ptDirHdl          入力                  grp\_fs\_opendir 関数で返ってきたファイルハンドル

【機能詳細】

ptDirHdl で指定したディレクトリファイルから、次の有効なディレクトリエントリを読み出し、読み出した情報へのポインタをリターン値として返します。返すディレクトリ情報の構造は、以下のとおりです。

```
struct dirent {
    int          d_dev;          /* デバイス番号（3.5.3 節 参照）          */
    int          d_type;         /* ファイルタイプ                          */
                                /* DT_UNKNOWN (0)   不明                    */
                                /* DT_REG (1)   ファイル(ショート名ファイル) */
                                /* DT_DIR (2)   ディレクトリ                */
                                /* DT_LINK (3)  リンクファイル(ロング名ファイル) */
    grp_uint32_t d_ino;          /* ファイル番号                            */
    grp_int32_t  d_size          /* ファイルサイズ                          */
    コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を無効にした場合の定義です。符
    号付 32 ビット変数型で定義されます。
    grp_uint32_t d_size          /* ファイルサイズ                          */
    コンパイルオプション「GRP_FS_ENABLE_OVER_2G」を有効にした場合の定義です。符
    号なし 32 ビット変数型で定義されます。
    int          d_reclen;       /* ファイル名の長さ                        */
    char         d_name[NAME_MAX]; /* ファイル名称バッファ                    */
};
```

FAT の場合、ロング名称エントリの d\_ino、d\_size フィールドは、0 が設定されています。同エントリに対応したファイルのファイル番号（クラスタ番号）、ファイルサイズ情報は、次の readdir で得られる同ロング名称に対応したショート名称のエントリの d\_ino、d\_size フィールドの値を参照して下さい。

d\_name にはファイル名称が返されますが、ファイル名称が **GR-FILE** の config パラメータ GRP\_FS\_MAX\_COMP-1 よりも長い場合は、GRP\_FS\_MAX\_COMP-1 の長さで切り捨てられ、d\_name に返されます。

## 【リターン値】

NULL

最終まで読んで次の空でないエントリは存在しなかった、あるいは、エラーが発生した

NULL 以外

読み出したディレクトリ情報へのポインタ



## 4.5.29 grp\_fs\_read\_part

【機能概要】 メディアからのパーティション情報の読み込み

【使用形式】

```
#include "grp_fs_disk_part.h"
int grp_fs_read_part(const char *pcDev, grp_fs_dk_part_t *ptPart);
```

【パラメータ】

pcDev	入力	メディアのデバイス名称
ptPart	出力	読み込んだパーティション情報を格納する領域へのポインタ (grp_fs_dk_part_t 4 エントリ分の領域を用意)

【機能詳細】

pcDev で指定したメディアからパーティション情報を読み出し、ptPart で指定された領域に格納します。

pcDev は、対象メディアの指定で、パーティション情報は、メディアの先頭から読み出す必要があるため、デバイス名としては、"USB0\*" のように、パーティション番号文字として '\*' を指定したパーティションレスデバイス名を指定する必要があります。通常のパーティション分割に対応したデバイス名称を指定した場合は、GRP\_FS\_ERR\_BAD\_DEV エラーとなります。

ptPart は、読み込んだパーティションデータを格納する領域のアドレスで、grp\_fs\_dk\_part\_t 4 エントリ分の領域を用意しておく必要があります。本関数は、メディアから読み込んだパーティションデータをアクセスしやすい形に変換し、ptPart で指定された領域に格納します。

ptPart で指された領域に格納されるデータの詳細につきましては、3.1 2.2 節を参照下さい。

【リターン値】

0	パーティション情報の読み込み成功
GRP_FS_ERR_BAD_DEV	指定したデバイスが存在しない、あるいは、パーティションレスタイプのデバイス名指定でない
GRP_FS_ERR_NOMEM	I/O バッファを確保できない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	パーティションレス (BPB が先頭に存在)
GRP_FS_ERR_NOT_FOUND	パーティション情報が見つからない

## 4.5.3 0 grp\_fs\_rename

【機能概要】 ファイル/ディレクトリ名称の変更

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_reaname(const grp_uchar_t *pucOld, const grp_uchar_t *pucNew);
```

【パラメータ】

pucOld	入力	名称を変更するファイル/ディレクトリのパス名
pucNew	入力	新しいファイル/ディレクトリ名称

【機能詳細】

pucOld で指定したファイル/ディレクトリの名称を pucNew で指定した名称に変更します。pucOld と pucNew で指定したディレクトリが異なる場合は、ファイルの移動になります。但し、pucOld と pucNew が異なるファイルシステム上にある場合は、エラーとなります。

コンパイルスイッチ「GRP\_FS\_UPDATE\_ARCHIVE」を有効にしていた場合は、名称変更されたファイル/ディレクトリには ARCHIVE 属性がセットされます。

【リターン値】

0	名称変更成功
GRP_FS_ERR_PERMIT	pucNew の親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	pucOld で指定したファイル/ディレクトリが存在しない pucNew 指定したファイル/ディレクトリのパス中のディレクトリが存在しない
GRP_FS_ERR_EXIST	pucNew で指定したファイル/ディレクトリが既に存在する
GRP_FS_ERR_XFS	pucOld と pucNew が異なるファイルシステム上にある
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pucOld、pucNew で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.3 1 grp\_fs\_set\_attr

【機能概要】 ファイル属性情報の設定・変更

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_set_attr(const grp_uchar_t *pucPath, grp_fs_dir_ent_t *ptAttr);
```

【パラメータ】

pucPath	入力	ファイル属性情報を設定・変更したいファイル/ディレクトリのパス名
ptAttr	入力	ファイル属性情報を格納した領域のアドレス

本パラメータの構造体の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照

【機能詳細】

pucPath で指定したファイル/ディレクトリの属性情報を ptAttr で指定した情報に従い、設定・変更します。この設定・変更は、ptAttr のフィールドのうち、uiProtect、iCTime、iMtime、iAtime、uiAttr、uiMisc フィールドの値を用いて行います。その他のフィールドは、設定しても無視します。

なお、FAT ファイルシステムの場合、uiMisc の情報は使用しません。また、uiAttr フィールドは、FAT\_ATTR\_HIDDEN、FAT\_ATTR\_SYSTEM、FAT\_ATTR\_ARCHIVE ビットのみ有効です。

ptAttr で指定する属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【リターン値】

0	属性情報設定・変更成功
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名、ptAttr が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.3 2 grp\_fs\_stat

【機能概要】 ファイル属性情報の取得

【使用形式】

```
#include "grp_fs_conv.h"
```

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」が無効な場合

```
int grp_fs_stat(const grp_uchar_t *pucPath, struct stat *ptStat);
```

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」が有効な場合

```
int grp_fs_stat(const grp_uchar_t *pucPath, grp_fs_stat_t *ptStat);
```

【パラメータ】

pucPath	入力	ファイル属性情報を取得したいファイル/ディレクトリのパス名
ptStat	出力	ファイル属性情報を格納する領域のアドレス

ptStat はコンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」の設定により使用する型が異なります。

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」が無効な場合

```
struct stat {
    int          st_dev;      デバイス番号 (3.5.3 節 参照)
    grp_uint32_t st_ino;      ファイル ID
    grp_uint32_t st_mode;     保護モード/ファイルタイプ
                             S_IRUSR   0400   ファイルオーナー read 可能
                             S_IWUSR   0200   ファイルオーナー write 可能
                             S_IXUSR   0100   ファイルオーナー実行可能
                             S_IRGRP   0040   ファイルオーナーグループ内 read 可能
                             S_IWGRP   0020   ファイルオーナーグループ内 write 可能
                             S_IXGRP   0010   ファイルオーナーグループ内実行可能
                             S_IROTH   0004   他のユーザ read 可能
                             S_IWOTH   0002   他のユーザ write 可能
                             S_IXOTH   0001   他のユーザ実行可能
                             S_IFMT    0170000 ファイルタイプビットマスク
                             S_IFDIR   0040000 ディレクトリファイル
                             S_IFREG   0100000 通常ファイル
                             S_IFLNK   0120000 リンクファイル(ロングファイル名)
    int          st_nlink;    リンクカウント (FAT の場合、常に 1)
    int          st_uid;      ファイルオーナーのユーザ ID (FAT の場合、常に 0)
```

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは、オーナービット値と同じです。隠しファイルの場合は、グループ、他のユーザのビットが 0 となります。さらに、FAT には実行可否の概念がなく、情報も保持できないため、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが EXE”、“COM”、“DLL”、“BAT” の場合、1 で、その他の場合は、0 となります。また、read 不可の概念がないため、S\_IRUSR のビットは常に 1 となります。

```

int          st_gid;          ファイルオーナーグループのグループ ID (FAT の場合、常に 0)
grp_int32_t  st_atime;        最終アクセス時刻 (1970/1/1 からのトータル秒)
                                   但し FAT の場合、精度は日単位

grp_int32_t  st_mtime;        最終更新時刻 (1970/1/1 からのトータル秒)
grp_int32_t  st_ctime         ファイルの生成時刻 (1970/1/1 からのトータル秒)
grp_int32_t  st_size          ファイルサイズ (符号付 32 ビット)
};

```

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」が有効な場合

```

grp_fs_stat_t {
    int          st_dev;
    grp_uint32_t st_ino;
    grp_uint32_t st_mode;
    int          st_nlink;
    int          st_uid;
    int          st_gid;
    grp_int32_t  st_atime;
    grp_int32_t  st_mtime;
    grp_int32_t  st_ctime
    grp_uint32_t st_size          ファイルサイズ (符号なし 32 ビット)
};

```

st\_size 以外は上記 struct stat の説明を参照してください

#### 【機能詳細】

pucPath で指定したファイル/ディレクトリの属性情報を取得し、ptStat で指定された領域に格納します。

なお、st\_mode フィールドに格納されたファイルのタイプ情報判定のために、以下のマクロも定義されています。

```

#define S_ISDIR(iMode) (((iMode) & S_IFMT) == S_IFDIR)    /* ディレクトリ */
#define S_ISREG(iMode) (((iMode) & S_IFMT) == S_IFREG)    /* 通常ファイル */
#define S_ISLNK(iMode) (((iMode) & S_IFMT) == S_IFLNK)    /* リンクファイル */

```

#### 【リターン値】

0	属性情報取得成功
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名、ptStat が不正である (NULL 等)
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.33 grp\_fs\_sync

【機能概要】 未反映キャッシュデータのメディアへの書戻し

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_sync(int iMode);
```

【パラメータ】

iMode	入力	書戻しモードの指定
GRP_FS_SYNC_FAILED		一度 write を失敗したキャッシュデータも再トライする
GRP_FS_SYNC_HINT		FAT32 の残りクラスタのヒント情報など、ファイルシステム依存のヒント情報のメディアへの書き戻しを行なう

【機能詳細】

未反映のキャッシュデータをメディアに書込み、メモリ上のキャッシュバッファにキャッシュされたデータとメディアとの整合性を確保します。

なお、unmount 処理せずにメディアを取外したり、何らかの理由により一度 write エラーとなったキャッシュデータについては、iMode に GRP\_FS\_SYNC\_FAILED を指定した場合のみ、メディアへの書戻しをトライします。指定がない場合、同データの書戻しは行いません。

また、GRP\_FS\_SYNC\_HINT を指定すると、FAT32 の残りクラスタのヒント情報など、ファイルシステム依存のヒント情報のメディアへの書き戻しも行います。FAT ファイルシステムの場合は、バージョン 1.11f から FAT ファイルシステム依存の処理で、GRP\_FS\_SYNC\_HINT 指定なしでも、デフォルト FAT32 の残りクラスタのヒント情報の書き戻しを行なうようになっていますが、依存コードに定義されているグローバル変数 grp\_fat\_sync\_hint の値を 0 に変更しますと、バージョン 1.11e 以前と同様に、新たに追加された GRP\_FS\_SYNC\_HINT 指定なしでは、FAT32 の残りクラスタのヒント情報の書き戻しを行いません。そこで、そのようなケースで同情報を書き戻す際には、本オプションを指定して行います。

【リターン値】

0	書戻し成功
GRP_FS_ERR_IO	I/O エラー

#### 4.5.3 4 grp\_fs\_task\_free\_all\_env

【機能概要】 全タスクのオープンファイル・カレントディレクトリの無効化

【使用形式】

```
#include "grp_fs_if.h"
int  grp_fs_task_free_all_env(void);
```

【パラメータ】

なし

【機能詳細】

各タスクでオープンされた全てのファイルをクローズし、カレントディレクトリも無効化します。  
本関数は、強制的な `unmount` 処理等、システムをクリーンな状態にする場合に使用します。

【リターン値】

0	無効化成功
GRP_FS_ERR_BUSY	ファイル I/O 中で無効化できない

#### 4.5.35 grp\_fs\_task\_free\_env

【機能概要】 自タスクのオープンファイル・カレントディレクトリの無効化

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_task_free_env(void);
```

【パラメータ】

なし

【機能詳細】

自タスクでオープンしたファイルをすべてクローズし、自タスクのカレントディレクトリも無効化します。本関数は、自タスクの終了処理等で使用します。

【リターン値】

0	無効化成功
GRP_FS_ERR_BUSY	ファイル I/O 中で無効化できない



## 4.5.36 grp\_fs\_task\_free\_env\_by\_id

【機能概要】 指定タスクのオープンファイル・カレントディレクトリの無効化

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_task_free_env_by_id(grp_fs_task_t tTaskId);
```

【パラメータ】

tTaskId	入力	無効化したいタスクの ID
---------	----	---------------

【機能詳細】

tTaskId で指定したタスクがオープンしたファイルをすべてクローズし、同タスクのカレントディレクトリも無効化します。本関数は、対象のタスクが終了した際、OS や同タスクの監視タスク等による後処理等で使用します。

【リターン値】

0	無効化成功
GRP_FS_ERR_NOT_FOUND	指定したタスクの管理情報が見つからない
	オープン中のファイル、カレントディレクトリ設定がない
GRP_FS_ERR_BUSY	ファイル I/O 中で無効化できない

## 4.5.37 grp\_fs\_truncate

【機能概要】 指定サイズ以降のファイル領域の解放

【使用形式】

```
#include "grp_fs_conv.h"
int grp_fs_truncate(int iFhdl, grp_uint32_t uiOffset);
```

【パラメータ】

iFhdl	入力	grp_fs_open 関数等で返ってきたファイルハンドル
uiOffset	入力	ファイル領域を解放する開始オフセット

【機能詳細】

iFhdl で指定されたファイルのファイルサイズを、uiOffset で指定されたサイズに切り詰めます。切り詰められたファイルのファイルサイズは uiOffset となります。uiOffset 以降の領域は解放されます。

uiOffset にファイルサイズ以上の値を指定した場合はエラーとなります。

【リターン値】

0	正常終了
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	同ファイルに対して write 権限がない
GRP_FS_ERR_BAD_OFF	uiOffset の値が正しくない（ファイルサイズを超えている）
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）
GRP_FS_ERR_SHOULD_CLOSE	指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある

## 4.5.38 grp\_fs\_unlink

【機能概要】 ファイル/ディレクトリの削除

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_unlink(const grp_uchar_t *pucPath);
```

【パラメータ】

pucPath	入力	削除するファイル/ディレクトリのパス名
---------	----	---------------------

【機能詳細】

pucPath で指定したファイル/ディレクトリを削除します。pucPath で指定したファイルがディレクトリの場合、削除するディレクトリにファイルやディレクトリが存在する場合はエラーとなり、削除することができません。また、“.”、“..”、アクセス中のファイルを削除しようとした場合も、エラーとなり削除することができません。

【リターン値】

0	削除成功
GRP_FS_ERR_PERMIT	親ディレクトリに対して write 権限がない
GRP_FS_ERR_NOT_FOUND	指定したファイルが存在しない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_BUSY	指定したファイルがディレクトリで、同ディレクトリが空でない “.”、“..”、アクセス中のファイルを削除しようとした
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.39 grp\_fs\_unmount

【機能概要】 メディアの unmount 処理

【使用形式】

```
#include "grp_fs_if.h"
int grp_fs_unmount(const char *pcDevName, int iMode);
```

【パラメータ】

pcDevName	入力	unmount 対象のメディアのデバイス名
iMode	入力	unmount モードの指定
GRP_FS_FORCE_UMOUNT		強制 unmount

【機能詳細】

pcDevName で指定したデバイスメディアのキャッシュデータの書戻しを行い、さらに、unmount 処理を行ったかどうかを示す情報をファイルシステムが持つ場合、同情報を設定し、メディアを取外し可能な状態にします。

但し、同メディア上のファイルがオープンされている場合、あるいは、同メディア上にカレントディレクトリが設定されている場合、本 unmount 処理は、GRP\_FS\_ERR\_BUSY エラーでエラーリターンします。同メディア上のファイルのオープンや、カレントディレクトリ設定を強制的にクローズ/無効化し、強制的に、unmount 処理を行う場合は、iMode に GRP\_FS\_FORCE\_UMOUNT を指定します。GRP\_FS\_FORCE\_UMOUNT を指定した場合でも、内部的には「強制クローズ」状態としてファイルハンドルやカレントディレクトリを保持します。「強制クローズ」状態をクリアし、ファイルハンドルの解放やカレントディレクトリの無効化を行うには、開いていたファイルを grp\_fs\_close 関数等を使用して、クローズし、カレントディレクトリを grp\_fs\_chdir 関数を使用して無効化して下さい。詳しくは 2 章 表 2.1 「GR-FILE の前提条件・制限」 # 1、および、3.3.1 節 (2) 「タスク毎のファイル管理機能」をご参照下さい。

コンパイルスイッチ「GRP\_FS\_ASYNC\_UNMOUNT」を有効にしていた場合は、アクセスしていないデバイスへのアンマウントを行う事が出来ますが、アンマウントを行おうとするデバイスへタスクがアクセスしている場合は、「GRP\_FS\_ASYNC\_UNMOUNT」を無効にしていた場合と同様に強制アンマウントも GRP\_FS\_ERR\_BUSY エラーでエラーリターンします。

コンパイルスイッチ「GRP\_FS\_ASYNC\_UNMOUNT」を無効にしていた場合は、デバイスへのアクセスに限らず、1 つでもタスクの処理が GR-FILE 内部で「待ち」の状態では、強制 unmount も GRP\_FS\_ERR\_BUSY エラーでエラーリターンします。この場合は、全てのタスクの処理が GR-FILE よりリターンした時点で、再び強制 unmount を行ってください。

【リターン値】

0	unmount 成功
GRP_FS_ERR_BAD_DEV	pcDevName で指定したデバイスメディアが mount されていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BUSY	同メディアのファイルがオープン中、あるいは、同メディア上に

	カレントディレクトリが設定されている
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.4 0 grp\_fs\_utimes

【機能概要】 ファイル/ディレクトリのアクセス・更新時刻情報の変更

【使用形式】

```
#include "grp_fs_conv.h"
int grp_fs_utimes(const grp_uchar_t *pucPath, struct timeval *ptTimes);
```

【パラメータ】

pucPath	入力	アクセス・更新時刻を変更するファイル/ディレクトリ名
ptTimes	入力	アクセス時刻 (ptTimes[0]) と更新時刻(ptTimes[1])情報が入った配列へのポインタ。それぞれの時刻情報の構造は以下のとおり

```
struct timeval {
    long    tv_sec;        1970/1/1 からのトータル秒
    long    tv_usec;       秒以下の値 (単位マイクロ秒)。但し、秒以下の情報は
                           ファイルシステム上の保持していないので、本値は
                           無視される。
};
```

なお、FAT のファイルシステムの場合は、日単位の精度でしかアクセス時刻情報をファイルシステム上に保持していないので、アクセス時刻情報は、日単位の精度で指定します。

【機能詳細】

pucPath で指定したファイル/ディレクトリのアクセス・更新時刻情報を ptTimes に従い変更します。

【リターン値】

0	設定成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOT_FOUND	指定したファイル/ディレクトリが存在しない
GRP_FS_ERR_PERMIT	親ディレクトリに対する write が許可されていない
GRP_FS_ERR_BAD_PARAM	pucPath で指定したパス名が不正である (NULL 等)
GRP_FS_ERR_TOO_LONG	指定したパス名が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.5.4 1 grp\_fs\_write

【機能概要】 ファイルの書込み

【使用形式】

```
#include "grp_fs_if.h"
grp_isize_t grp_fs_write(int iFhdl, grp_uchar_t *pucBuf, grp_isize_t iSize);
```

【パラメータ】

iFhdl	入力	grp_fs_open 関数等で返ってきたファイルハンドル
pucBuf	入力	書込むデータの格納された領域のアドレス
iSize	入力	書込むデータのサイズ

【機能詳細】

pucBuf で指定した領域に格納されたデータを、iFhdl で指定したファイルに iSize バイト分書込みます。ファイルシステムの残り領域が iSize 分より少ない場合は、残り領域分だけ書込みます。

リターン値としては、実際に書けたバイト数を返します。

なお、オープン時に GRP\_FS\_O\_DIRECT\_IO の指定がなかった場合は、キャッシュバッファを介して write 処理を行ない、write 要求処理時にはキャッシュに書込むだけで、メディアへの反映は、キャッシュの write 制御方式に従ったタイミングで行います。

一方、GRP\_FS\_O\_DIRECT\_IO 指定があり、ファイルシステム依存部で同オプションをサポートしている場合は、可能な限りキャッシュバッファを使用せず、アプリケーションの指定したバッファからメディアに対し、直接連続ブロックで I/O を行います。この場合も、データ境界等の関係で、直接 I/O できない部分については、キャッシュバッファを介して I/O する形になります。

**GR-FILE** の FAT ファイルシステムでは、シングল空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファからの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続して確保できれば、連続した単位で、一括してメディアに対し write します。

GRP\_FS\_O\_DIRECT\_IO オプションに関しては「3.8 アプリケーションバッファとメディア間の直接/連続ブロック I/O (ダイレクト I/O 機能)」を参照下さい。

コンパイルスイッチ「GRP\_FS\_UPDATE\_ARCHIVE」を有効にしていた場合は、書き込まれるファイルには ARCHIVE 属性がセットされます。

【リターン値】

0 または 正值	実際に書けたバイト数
GRP_FS_ERR_FHDL	指定したファイルハンドルが正しくない
GRP_FS_ERR_PERMIT	write 可能なモードでオープンされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注

参照)

GRP\_FS\_ERR\_SHOULD\_CLOSE 指定したファイルハンドルは、強制アンマウント処理により無効化されているため、クローズする必要がある



## 4.5.4 2 grp\_fs\_write\_part

【機能概要】 メディアへのパーティション情報の書き込み

【使用形式】

```
#include "grp_fs_disk_part.h"
int grp_fs_write_part(const char *pcDev, int iAuto, grp_fs_dk_part_t *ptPart);
```

【パラメータ】

pcDev	入力	メディアのデバイス名称
iAuto	入力	1 パーティション設定の自動設定指示
ptPart	入力/出力	書き込むパーティション情報を格納した領域へのポインタ (grp_fs_dk_part_t 4 エントリ分のデータ)

iAuto が 0 以外の場合、設定したパーティション情報が書き戻される

【機能詳細】

ptPart で指定された領域に格納された 4 エントリ分のパーティション情報を pcDev で指定したメディアに対し書き込みます。

pcDev は、対象メディアの指定で、パーティション情報は、メディアの先頭に書く必要があるため、デバイス名としては、"USB0\*" のように、パーティション番号文字として '\*' を指定したパーティションレスデバイス名を指定する必要があります。通常のパーティション分割に対応したデバイス名称を指定した場合は、GRP\_FS\_ERR\_BAD\_DEV エラーとなります。

ptPart は、メディアに書き込むパーティション情報が格納された領域のアドレスを指定します。本関数をコールする前に、同領域に、4 エントリ分のパーティションデータを格納しておくことが必要です。このパーティションデータを作成するために必要なメディアのサイズ情報や、FAT タイプ情報は、デバイス直接制御インタフェースの grp\_fs\_open\_dev、grp\_fs\_ioctl\_dev や、GR-FILE 固有インタフェースの grp\_fat\_find\_type などを利用して求めることが可能です。

ptPart で指された領域に格納すべきデータの詳細につきましては、3.1 2.2 節を参照下さい。

iAuto は、指定したメディア全体を 1 パーティションとして簡単にパーティション設定するためのパラメータです。本パラメータを 0 以外に設定してコールしますと、ptPart で指定した領域には、予めメディアに書き込むデータを用意する必要がなく、GR-FILE がデバイスドライバと連携して、自動的に必要なパーティション情報を作成し、メディアに書き込みを行ないます。そして、設定したパーティション情報を、ptPart で指された領域に書き戻します。なお、デバイスドライバが自動設定に必要なメディアサイズ情報等を返さない場合は、GRP\_FS\_ERR\_PARAM エラーとなります。

## 【リターン値】

0	パーティション情報の読み込み成功
GRP_FS_ERR_BAD_DEV	指定したデバイスが存在しない、あるいは、パーティションレス タイプのデバイス名指定でない
GRP_FS_ERR_BUSY	指定したデバイスはマウント中である
GRP_FS_ERR_NOMEM	I/O バッファを確保できない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_PARAM	パーティション情報が正しくない <ul style="list-style-type: none"><li>・アクティブパーティションが複数ある</li><li>・パーティション開始位置情報、終了位置情報に範囲外の値 が設定されている</li><li>・パーティション開始位置情報が終了位置情報より大きい</li></ul>

#### 4.6 デバイス直接制御アプリケーションインタフェース

3. 2 節で説明しましたとおり、**GR-FILE** では、メディアの取出し/ロック/アンロックや物理フォーマット等、デバイス固有機能をアプリケーションから利用するために、デバイス直接制御インタフェースを提供しています。また、本インタフェースにて、デバイス固有機能だけでなく、メディアへの直接の I/O 機能を提供しています。

なお、各関数でパラメータとして使用するデバイス番号は、`grp_fs_lookup_dev` を使って、デバイス名称から得ることができます。

表 4-8 に、**GR-FILE** でサポートしているデバイス直接制御インタフェースの一覧を示します。

表 4-8 デバイス直接制御アプリケーションインタフェース一覧

#	インタフェース名	内容・機能
1	<code>grp_fs_open_dev</code>	デバイスのオープン
2	<code>grp_fs_close_dev</code>	デバイスのクローズ
3	<code>grp_fs_read_dev</code>	デバイスからのデータ読み込み
4	<code>grp_fs_write_dev</code>	デバイスへのデータの書き込み
5	<code>grp_fs_ioctl_dev</code>	デバイス固有の I/O 制御

これらのインタフェースは、“include” 下の “`grp_fs_dev_io_if.h`” に定義されています。

以下本説では、表 4-8 に示したデバイス直接制御アプリケーションインタフェースの詳細を説明します。

## 4.6.1 grp\_fs\_open\_dev

【機能概要】 デバイスのオープン

【関数形式】

```
#include "grp_fs_dev_io_if.h"

int grp_fs_open_dev(int iDev, int iRWOOpen, grp_fs_dev_io_info_t *ptDevIoInfo);
```

【パラメータ】

iDev	入力	オープンするデバイスメディアのデバイス番号
iRWOOpen	入力	オープンモード (0 : read-only、0 以外: read/write)
ptDevIoInfo	出力	デバイス I/O 制御情報

【機能詳細】

iDev で指定したデバイスメディアをオープンし、その属性情報や、同デバイスメディアに対する I/O 処理時に指定するデバイスハンドルを ptDevInfo で示す領域に返します。

iRWOOpen は、オープンモードの指定で、0 以外の場合は、対象のメディアが write 可能かをチェックし、write 可能でなければ、エラーを返します。

ptDevIoInfo で示す領域の構造体の形式は以下のとおりです。

```
typedef struct grp_fs_dev_io_info {
    grp_int32_t      iHandle;          /* デバイス I/O ハンドル */
    grp_uint32_t     uiOff;            /* 開始位置オフセット */
    grp_uint32_t     uiSize;           /* トータルブロック数 */
    int              iSzShift;         /* ブロックのシフト値 */
} grp_fs_dev_io_info_t;
```

ptDevIoInfo で示す構造体の iHandle フィールドには、以降の I/O 要求で使用するデバイスハンドルを設定して返します。**GR-FILE** では、デバイスドライバがハンドルを用いて I/O する方式と、デバイス名称に対応したデバイス番号を用いて I/O を方式の両方をサポートしています。従いまして、デバイス番号により I/O を行うタイプのドライバの場合は、本 iHandle には、特に意味のある値を設定しないケースもあります。

また、ptDevIoInfo で示す構造体の uiOff フィールドには、I/O 要求を出す際のメディア内の開始オフセットを物理ブロック（セクタ）単位で設定して返します。パーティションレスメディアの場合は、通常 0 を設定して返しますが、パーティション分割されているメディアでは、iDev で指定したパーティションのメディア内のオフセットを設定して返します。

ptDevIoInfo で示す構造体の uiSize フィールドには、指定されたデバイスメディア（またはパーティション）のトータル物理ブロック（セクタ）数を設定して返します。デバイスメディアによっては、トータル物理ブロック数を返す機能をサポートしておらず、0 を返してくるものもあります。

最後に、ptDevIoInfo で示す構造体の iSzShift フィールドには、物理ブロック（セクタ）サイズのシフト値を設定して返します。デバイス直接制御インタフェースの grp\_fs\_read\_dev、

grp\_fs\_write\_dev を使って、デバイスメディアに対し I/O 要求を出す際には、I/O オフセット値、I/O バイト数は、この物理ブロック（セクタ）サイズ単位で指定して行います。また、I/O オフセット値としては、パーティション内のオフセット値に、上記 uiOff フィールドで得られた値を加えて指定します。

**【リターン値】**

0	オープン成功
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない
GRP_FS_ERR_NOT_SUPPORT	オープン処理がサポートされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_PARAM	ptDevIoInfo で指定したアドレスが正しくない
その他	エラー。GRP_FS_ERR_IO 等、 <b>GR-FILE</b> で規定したエラー番号

## 4.6.2 grp\_fs\_close\_dev

【機能概要】 デバイスのクローズ

【関数形式】

```
#include "grp_fs_dev_io_if.h"
int grp_fs_close_dev (grp_int32_t iHandle, int iDev);
```

【パラメータ】

iHandle	入力	grp_fs_open_dev で得られたデバイス I/O ハンドル
iDev	入力	クローズするデバイスメディアのデバイス番号

【機能詳細】

iHandle、iDev で指定したデバイスメディアをクローズします。

【リターン値】

0	クローズ成功
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない
GRP_FS_ERR_NOT_SUPPORT	クローズ処理がサポートされていない
GRP_FS_ERR_IO	I/O エラー
その他	エラー。GRP_FS_ERR_IO 等、 <b>GR-FILE</b> で規定したエラー番号

## 4.6.3 grp\_fs\_read\_dev

【機能概要】 デバイスからのデータの読み出し

【関数形式】

```
#include "grp_fs_dev_io_if.h"
int grp_fs_read_dev(grp_int32_t iHandle, int iDev,
                    grp_uint32_t uiDevBlk, grp_uchar_t *pucBuf, grp_isize_t iCnt);
```

【パラメータ】

iHandle	入力	grp_fs_open_dev で得られたデバイス I/O ハンドル
iDev	入力	read 対象のデバイスメディアのデバイス番号
uiDevBlk	入力	read 開始物理ブロックオフセット
pucBuf	出力	read したデータを格納する領域のアドレス
iCnt	入力	read する物理ブロック数

【機能詳細】

iHandle、iDev で指定したデバイスメディアからデータを読み込み、pucBuf で指定した領域に読込んだデータを格納します。

ハンドルを用いて I/O 処理を行うデバイスドライバの場合は、iHandle を用いて I/O 処理を行い、デバイス番号を用いて I/O 処理を行うデバイスドライバは、iDev を用いて I/O 処理を行います。

uiDevBlk には、read を開始する物理ブロック（セクタ）番号を、iCnt には read する物理ブロック（セクタ）数を指定します。uiDevBlk には、対応するデバイスメディアのパーティション内でのブロックオフセット値に、オープン時に返ってきた同パーティションのメディア内での開始物理ブロックオフセットを加えて、指定します。

【リターン値】

0 または 正值	実際に読込んだ物理ブロック（セクタ）数
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない
GRP_FS_ERR_NOT_SUPPORT	read 処理がサポートされていない
GRP_FS_ERR_IO	I/O エラー
その他	エラー。GRP_FS_ERR_IO 等、GR-FILE で規定したエラー番号

## 4.6.4 grp\_fs\_write\_dev

【機能概要】 デバイスへのデータの書き込み

【関数形式】

```
#include "grp_fs_dev_io_if.h"
int grp_fs_write_dev (grp_int32_t iHandle, int iDev,
                     grp_uint32_t uiDevBlk, grp_uchar_t *pucBuf, grp_isize_t iCnt);
```

【パラメータ】

iHandle	入力	grp_fs_open_dev で得られたデバイスハンドル
iDev	入力	write 対象のデバイスメディアのデバイス番号
uiDevBlk	入力	write 開始物理ブロックオフセット
pucBuf	入力	write するデータを格納した領域のアドレス
iCnt	入力	write する物理ブロック数

【機能詳細】

pucBuf で指定した領域に格納された iCnt ブロック分のデータを、iHandle、iDev で指定したデバイスメディアに書込みます。

ハンドルを用いて I/O 処理を行うデバイスドライバの場合は、iHandle を用いて I/O 処理を行い、デバイス番号を用いて I/O 処理を行うデバイスドライバは、iDev を用いて I/O 処理を行います。

uiDevBlk には、対応するデバイスメディアのパーティション内でのブロックオフセット値に、オープン時に返ってきた同パーティションのメディア内での開始物理ブロックオフセットを加えて、指定します。

【リターン値】

0 または 正值	実際に書込んだ物理ブロック（セクタ）数
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない
GRP_FS_ERR_NOT_SUPPORT	write 処理がサポートされていない
GRP_FS_ERR_IO	I/O エラー
その他	エラー。GRP_FS_ERR_IO 等、GR-FILE で規定したエラー番号



## 4.6.5 grp\_fs\_ioctl\_dev

【機能概要】 デバイス固有の I/O 制御

【関数形式】

```
#include "grp_fs_dev_io_if.h"
int grp_fs_ioctl_dev(int iDev, grp_uint32_t uiCmd void *pvParam);
```

【パラメータ】

iDev	入力	read 対象のデバイスメディアのデバイス番号
uiCmd	入力	実行する I/O 制御の機能番号
pvParam	入力/出力	実行する I/O 制御に渡すパラメータ

【機能詳細】

iDev で指定したデバイスに対し、uiCmd で指定したデバイス固有の I/O 制御機能を、pvParam で指定したパラメータを渡して、実行します。

uiCmd、pvParam の内容は、実行するデバイス固有機能に依存しますが、**GR-FILE** では、以下の機能に関して、具体的な機能番号や、パラメータを規定しています。

機能番号	機能、および、パラメータ
GRP_FS_DEV_CTL_GET_MEDIA	メディア情報の取得。grp_fs_media_info_t の型を持つ領域のアドレスを pvParam に指定し、取得したメディア情報を格納します。(grp_fs_media_info_t の詳細については、4.2.2 項のアプリケーションインタフェース関連の構造体を参照)
GRP_FS_DEV_CTL_EJECT	メディアの取り出し制御。メディアの取り出し制御指示番号を設定した int 型の領域のアドレスを pvParam に設定し、設定した取り出し制御指示番号に従い、メディアの取り出し制御を行います。取り出し制御指示番号としては、以下を指定します。
	GRP_FS_DEV_LOCK_MEDIA 取り出しロックを設定
	GRP_FS_DEV_UNLOCK_MEDIA 取り出しロックを解除
	GRP_FS_DEV_EJECT_MEDIA メディアを排出
GRP_FS_DEV_CTL_FORMAT	メディアの物理フォーマット。メディアに対し、物理的なフォーマットを実行します。pvParam には NULL を指定します。
GRP_FS_DEV_CTL_GET_WRITE_PROTECT	メディアのライトプロテクト状態を取得します。 pvParam には int 型の領域が設定されています。

なお、規定されている上記機能も含め、各機能をサポートするかどうかは、各デバイス依存です。対象のデバイスが、ioctl 自体をサポートしていない場合、あるいは、uiCmd で指定した機能をサポートしていない場合は、GRP\_FS\_ERR\_NOT\_SUPPORT エラーを返します。

## 【リターン値】

0	指定したデバイス固有機能実行成功
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない
GRP_FS_ERR_NOT_SUPPORT	指定したデバイス固有機能がサポートされていない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_PARAM	pvParam で指定したアドレスが正しくない
その他	エラー。GRP_FS_ERR_IO 等、 <b>GR-FILE</b> で規定したエラー番号

#### 4.7 ファイルシステム抽象化インタフェース

本節で規定するファイルシステム抽象化インタフェースは、ファイルシステム依存処理部と **GR-FILE** のファイルシステム非依存の共通処理部とのインタフェースです。

3.9 節で示しましたとおり、**GR-FILE** のファイルシステム共通処理部とファイルシステム依存処理部のインタフェースは、ファイルシステムタイプ名称と、同ファイルシステム依存の処理関数テーブルへのポインタを **GR-FILE** のファイルシステムテーブル `grp_fs_type_tbl[]` に登録することで行います。表 4-9 に、ファイルシステムテーブル `grp_fs_type_tbl` の各エントリの構造体 `grp_fs_type_tbl_t` の内容を示します。また、表 4-10 に、各エントリからポイントされるファイルシステム依存の処理関数テーブルの構造体 `grp_fs_op_t` の内容を示します。

表 4-9 ファイルシステムテーブルエントリの構造体 `grp_fs_type_tbl_t` の内容

#	構造体メンバ名	タイプ	意味
1	<code>pcFsName</code>	<code>const char *</code>	ファイルシステムタイプ名称
2	<code>ptFsOp</code>	<code>grp_fs_op_t *</code>	ファイルシステム依存処理関数テーブルへのポインタ

表 4-10 ファイルシステム依存処理関数テーブルの構造体 `grp_fs_op_t` の内容

#	構造体メンバ名	タイプ	意味
1	<code>pfnOpenRoot</code>	<code>grp_fs_open_root_t *</code>	ルートディレクトリの取得関数
2	<code>pfnMount</code>	<code>grp_fs_mount_t *</code>	<code>mount</code> 処理関数
3	<code>pfnUmount</code>	<code>grp_fs_umount_t *</code>	<code>unmount</code> 処理関数
4	<code>pfnOpen</code>	<code>grp_fs_open_t *</code>	ファイルのオープン処理関数
5	<code>pfnClose</code>	<code>grp_fs_close_t *</code>	ファイルのクローズ/属性反映処理関数
6	<code>pfnRead</code>	<code>grp_fs_read_t *</code>	ファイルの <code>read</code> 処理関数
7	<code>pfnWrite</code>	<code>grp_fs_write_t *</code>	ファイルの <code>write</code> 処理関数
8	<code>pfnCreate</code>	<code>grp_fs_create_t *</code>	ファイル/ディレクトリの作成処理関数
9	<code>pfnUnlink</code>	<code>grp_fs_unlink_t *</code>	ファイル/ディレクトリの削除処理関数
10	<code>pfnRename</code>	<code>grp_fs_rename_t *</code>	ファイル/ディレクトリの名称変更処理関数
11	<code>pfnGetAttr</code>	<code>grp_fs_get_attr_t *</code>	ファイル/ディレクトリの属性取得関数
12	<code>pfnSetAttr</code>	<code>grp_fs_set_attr_t *</code>	ファイル/ディレクトリの属性設定・変更関数
13	<code>pfnTruncate</code>	<code>grp_fs_truncate_t *</code>	ファイルデータの削除関数
14	<code>pfnGetDirEnt</code>	<code>grp_fs_get_dirent_t *</code>	ディレクトリエントリ情報の取得関数
15	<code>pfnMatchComp</code>	<code>grp_fs_match_comp_t *</code>	ディレクトリエントリの検索関数
16	<code>pfnCheckVolume</code>	<code>grp_fs_check_volume_t *</code>	ボリューム名の取得関数
17	<code>pfnSync</code>	<code>grp_fs_sync_t *</code>	ファイルシステム依存キャッシュの書き戻し処理

以下本節では、上記ファイルシステム依存処理関数テーブルの各メンバの詳細インタフェースを説明します。なお、これらの構造体の型宣言は、“`grp_fs_cfg.h`” に定義されています。上記インタフェースに従ったファイルシステム依存関数の実現をサポートするための **GR-FILE** 関数については、「ファイルシステム依存関数向け **GR-FILE** 関数」の節を参照下さい。

## 4.7.1 grp\_fs\_open\_root\_t \*pfnOpenRoot

【機能概要】 ルートディレクトリの取得関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_open_root_t xxx_open_root;

int xxx_open_root(grp_fs_info_t *ptFs, grp_fs_file_t **pptFile);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
pptFile	出力	ルートディレクトリのファイル管理情報へのポインタを格納する領域のアドレス

【機能詳細】

ptFs で指定されたファイルシステムのルートディレクトリをオープンし、そのファイル管理情報へのポインタを pptFile で指定した領域に格納します。

【リターン値】

0	取得成功
GRP_FS_TOO_MANY	オープン中のファイルの数がシステムの上限值を超えた
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.2 grp\_fs\_mount\_t \*pfnMount

【機能概要】 mount 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_mount_t xxx_mount;

int xxx_mount(grp_fs_info_t *ptFs, int iMode);
```

【パラメータ】

ptFs	入出力	ファイルシステム情報へのポインタ
	入力	grp_fs_info_t 構造体の ucDevBlkShift までのフィールド
	出力	grp_fs_info_t 構造体の ucFsFBlkShift 以降のフィールド
iMode	入力	mount モードの指定（以下を bit or で指定）
	GRP_FS_RDONLY	read-only で mount
	GRP_FS_FORCE_MOUNT	強制 mount (前回 unmount 処理されていないメディアの mount 等)に使用)
	GRP_FS_NO_UPD_ACCTIME	アクセス時刻記録のメディアへの反映抑止
	GRP_FS_NO_MNT_FLAG	メディアへのマウント中フラグの書き込み抑止
	GRP_FS_NO_CRT_ACCTIME	メディア上の作成/アクセス時刻記録なし
	GRP_FS_SYNC_ALL	write-through 方式のキャッシュ反映方式使用
	GRP_FS_SYNC_FL_CLOSE	each-close 方式のキャッシュ反映方式使用
	GRP_FS_SYNC_FS_CLOSE	last-close 方式のキャッシュ反映方式使用

なお、GRP\_FS\_SYNC\_ALL> GRP\_FS\_SYNC\_FL\_CLOSE>GRP\_FS\_SYNC\_FS\_CLOSE の順でキャッシュ反映方式が 2 つ以上指定された場合は 1 つを選択します。これらのいずれの指定もない場合は、unmount 方式のキャッシュ反映方式を使用します。

【機能詳細】

ptFs で指定された grp\_fs\_info\_t 構造体の ucDevBlkShift フィールドまでの情報を使い、対象のファイルシステムの管理情報をメディアから読出し、ucFsFBlkShift フィールド以降のフィールドを設定して、同ファイルシステムをアクセス可能な状態にします。

なお、unmount 処理が行われたかどうかを示す情報をメディアに持つ場合は、iMode に GRP\_FS\_NO\_MNT\_FLAG が指定されていないければ、同情報を mount 中であるという情報に変更します。また、同ファイルシステムに対応してファイルシステム依存の情報を保持する場合は、必要な領域を確保して依存情報を格納後、ptFs の pvFsInfo に設定します。

【リターン値】

0	mount 処理成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	必要なメモリが確保できない、あるいは、使用可能なキャッシュバッファがない（エラー番号の節の注参照）

## 4.7.3 grp\_fs\_umount\_t \*pfnUmount

【機能概要】 unmount 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_umount_t xxx_umount;

int xxx_umount(grp_fs_info_t *ptFs, int iMode);
```

【パラメータ】

ptFs	入出力	ファイルシステム情報へのポインタ
	入力	grp_fs_info_t 構造体全フィールド
	出力	grp_fs_info_t 構造体の pvFsInfo をリセット
iMode	入力	mount モードの指定（以下を bit or で指定）
	GRP_FS_FORCE_UMOUNT	強制 unmount
	GRP_FS_REVOKE_MOUNT	ファイルシステム依存情報の解放のみ （ <b>GR-FILE</b> 再初期化時）

【機能詳細】

ptFs で指定されたファイルシステムの unmount 処理を行います。

共通部の処理で既に、キャッシュバッファの書戻し処理等が済んでいますので、本ファイルシステム依存の unmount 処理では、例えば、unmount 処理が行われたかどうかを示す情報をメディアに持つ場合に、メディアへの unmount 情報の設定を行い、ptFs の pvFsInfo フィールドに保持したファイルシステム依存情報をクリアして、確保した領域の解放を行います。

なお、iMode に GRP\_FS\_REVOKDE\_MOUNT が指定されている場合や、read-only のファイルシステムの場合は、メディアへの unmount 情報の設定処理等の write 処理は行いません。read-only のファイルシステムかどうかは、ptFS の usStatus フィールドに GRP\_FS\_STAT\_RDONLY のビットが設定されているかどうかで判定します。usStatus フィールドに GRP\_FS\_STAT\_NO\_MNT\_FLAG が設定されている場合も、unmount 情報のメディアへの設定を抑止します。

また、iMode に GRP\_FS\_FORCE\_UMOUNT が指定されている場合は、unmount 情報の設定処理等の write 処理でエラーが発生しても、正常終了としてリターン値を返します。

【リターン値】

0	unmount 処理成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	必要なメモリが確保できない、あるいは、 使用可能なキャッシュバッファがない（エラー番号の節の注参照）

## 4.7.4 grp\_fs\_open\_t \*pfnOpen

【機能概要】 ファイルのオープン処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_open_t  xxx_open;
grp_fs_open_t  grp_fs_file_open_common;

int  xxx_open (
    grp_fs_info_t      *ptFs,
    grp_fs_file_t      *ptDir,
    const grp_uchar_t  **ppucPath,
    int                iMode,
    grp_uchar_t        *pucComp,
    grp_fs_file_t      **pptOpened);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
ptDir	入力	検索開始ディレクトリ (NULL の場合ルートからサーチ開始)
ppucPath	入力	*ppucPath : オープン対象のファイルのパス名
	出力	*ppucPath : 残りのパス名の先頭アドレス
iMode	入力	オープンモードの指定 (以下を bit or で指定)
GRP_FS_OPEN_EXEC		実行可能モードでオープン
GRP_FS_OPEN_READ		read 可能モードでオープン
GRP_FS_OPEN_WRITE		write 可能モードでオープン
GRP_FS_OPEN_APPEND		追加 write モードでオープン (常にファイルの終わりに seek して write 時)
GRP_FS_OPEN_PARENT		対象ファイルの親ディレクトリのオープン
pucComp	出力	GRP_FS_OPEN_PARENT の場合の最終ファイルコンポーネント名を格納する領域のアドレス。なお、本パラメータで指定した領域は、GRP_FS_OPEN_PARENT 以外でも、本関数内でのファイルサーチ処理に使用し、GRP_FS_MAX_COMP バイトの領域が確保されていることを前提とする。
pptOpened	出力	オープンした対象ファイル、または、親ディレクトリ、または、検索中に検出したファイルシステム跨りとなるディレクトリのファイル管理情報へのポインタを格納する領域のアドレス

【機能詳細】

ptFs で指定されたファイルシステムの ptDir で指定されたディレクトリから検索を開始し、\*ppucPath で得られるパス名のファイルまたはその親ディレクトリをオープンして、pptOpened で指定された領域に、オープン結果のファイル管理情報へのポインタを返します。

iMode パラメータに GRP\_FS\_OPEN\_PARENT が指定されている場合は、指定の対象ファイル

の親ディレクトリまでをサーチしてオープンし、指定がない場合は、対象のファイルをオープンします。また、オープン対象のファイル、または、親ディレクトリに対し、iMode で指定された read/write/実行等が許可可能かどうかをチェックします。指定された属性が許可できない場合は、リターン値として、GRP\_FS\_ERR\_PERMIT を返します。

pucComp パラメータには、検索の際、パス名から各コンポーネント名を切出し保持するための作業領域として使用可能な GRP\_FS\_MAX\_COMP バイトの領域のアドレスが指定されています。この pucComp で指定された領域には、GRP\_FS\_OPEN\_PARENT が iMode パラメータに指定されている場合、親ディレクトリ以下の未検索の最終コンポーネント名を格納して返します。

通常は、対象のファイル、または、その親ディレクトリをオープンして返しますが、指定されたパス名をたどる中で、他のファイルシステムを mount したディレクトリに達した場合、あるいは、ルートディレクトリまで上っていた状態で、さらに、”..” が指定されている場合は、そこで、検索を打ち切ります。この場合、\*pptFile には、そのディレクトリ、または、ルートディレクトリのファイル管理情報へにポインタを返し、\*ppucPath には、残ったパス名のアドレスを設定して、リターン値として、GRP\_FS\_COMP\_MIDDLE(正值)を返します。

なお、本オープン処理関数は、ファイルシステムに依存した固有の関数を用意してもよいですが、共通処理関数として、grp\_fs\_file\_open\_common を用意しており、通常は、この関数をオープン処理関数として使用します。この grp\_fs\_file\_open\_common は、さらに、ファイルシステム依存処理関数テーブルの pfnMatchComp や pfnOpenRoot を使用して実現されています。

#### 【リターン値】

0	オープン処理成功
GRP_FS_COMP_MIDDLE	パス名を辿る中でファイルシステムを跨り、処理を中断した
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_PERMIT	指定されたオープンモードを許可できない
GRP_FS_ERR_BAD_NAME	パス名に不正な文字コードが含まれている
GRP_FS_ERR_NOT_FOUND	指定されたファイル、または、親ディレクトリが見つからない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)



## 4.7.5 grp\_fs\_close\_t \*pfnClose

【機能概要】 ファイルのクローズ／属性反映処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_close_t xxx_close;

int xxx_close (grp_fs_file_t *ptFile, int iMode);
```

【パラメータ】

ptFile	入力	クローズ/属性反映対象のファイル管理情報へのポインタ
iMode	入力	クローズモードの指定（以下のいずれかを 1 つを指定）
GRP_FS_CLOSE_RELEASE		完全にクローズし、リソースを解放
GRP_FS_CLOSE_CACHE		キャッシュと使用可能な状態でクローズ
GRP_FS_CLOSE_UPDATE		ファイル属性情報の更新のみ実施

【機能詳細】

ptFile で指定されたファイルのクローズ処理または属性反映処理を行います。iMode により、処理の内容が異なります。

GRP\_FS\_CLOSE\_RELEASE は、unmount 処理等で指定されるモードで、対象のファイルを完全にクローズし、同ファイルの領域マップのキャッシュ情報等も解放します。

GRP\_FS\_CLOSE\_CACHE は、通常のファイルクローズ処理等で指定されるモードで、最新の属性情報の反映等を行い、対象のファイルのクローズ処理を実施しますが、クローズ後も、次回オープン要求時にキャッシュとして使用可能なように領域マップのキャッシュ情報等は保持したままとします。

GRP\_FS\_CLOSE\_UPDATE は、sync 処理等で指定されるモードで、同ファイルを引続きオープン状態に保ったまま、最新の属性情報のみをメディアに反映します。

【リターン値】

0	オープン処理成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）

## 4.7.6 grp\_fs\_read\_t \*pfnRead

【機能概要】 ファイルの read 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_read_t xxx_read;

grp_int32_t xxx_read (grp_fs_file_t *ptFile, grp_uint32_t uiFsBlk,
                      grp_uint32_t uiBlkOff, grp_uchar_t *pucBuf,
                      grp_isize_t iSize, int iMode);
```

【パラメータ】

ptFile	入力	read 対象ファイルのファイル管理情報へのポインタ
uiFsBlk	入力	ファイル内でのブロック番号 (キャッシュブロック単位)
uiBlkOff	入力	uiFsBlk で指定したブロック内でのオフセット
pucBuf	出力	read したデータを格納する領域のアドレス (アプリケーション空間のアドレス)
iSize	入力	read するバイト数
iMode	入力	I/O モード。GRP_FS_OPEN_DIRECT_IO ビットが設定されている場合、キャッシュ上になければ、可能な限りキャッシュバッファを使用せず、メディアからアプリケーションバッファに対し、直接連続ブロックで I/O を行います。

【機能詳細】

ptFile で指定されたファイルから iSize バイト分データを読み出し、pucBuf に指定された領域に格納します。ファイル内でのデータの読み出し開始位置は、uiFsBlk、uiBlkOff で指定されます。uiFsBlk で、キャッシュブロック単位でのファイル内オフセットが指定され、さらに、uiBlkOff で同ブロック内でのオフセットが指定されます。

指定位置からファイルの終わりまでの残りバイト数が、iSize より小さい場合、その残りバイト数分を読み込んで、pucBuf に指定された領域に格納し、そのバイト数をリターン値として返します。

なお、iMode で GRP\_FS\_OPEN\_DIRECT\_IO の指定がない場合は、本関数は、直接デバイスドライバの I/O 関数を使って実現するのではなく、grp\_fs\_read\_buf / grp\_fs\_write\_buf 関数を使用し、キャッシュを使いながら読書きを行います。また、pucBuf で指定されたアドレスは、アプリケーションプログラムのアドレスですので、アプリケーションプログラムのアドレス空間が異なるシステムでは、単純なメモリコピーは使用できません。従いまして、データを pucBuf に返す場合は、単純なコピールーチンを使用するのではなく、grp\_fs\_copyout を使用し、アドレス空間が異なる場合でも対応できるようにする必要があります。

iMode に GRP\_FS\_OPEN\_DIRECT\_IO の指定がある場合は、キャッシュ上に対象のデータがなければ、可能な限りキャッシュバッファを使用せず、メディアからアプリケーションバッファに対し、直接連続ブロックで I/O を行います。本オプションのサポートは、任意で、デフォルトでサポ

ートしている FAT ファイルシステムでは、シングル空間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファからの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続していれば、連続した単位で、一括してメディアから read します。

**【リターン値】**

0 または 正值	実際に read できたバイト数
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_BAD_PARAM	pucBuf で指定された領域が正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.7 grp\_fs\_write\_t \*pfnWrite

【機能概要】 ファイルの write 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_write_t xxx_write;

grp_int32_t xxx_write (grp_fs_file_t *ptFile, grp_uint32_t uiFsBlk,
                        grp_uint32_t uiBlkOff, grp_uchar_t *pucBuf,
                        grp_isize_t iSize, int iMode);
```

【パラメータ】

ptFile	入力	write 対象ファイルのファイル管理情報へのポインタ
uiFsBlk	入力	ファイル内でのブロック番号 (キャッシュブロック単位)
uiBlkOff	入力	uiFsBlk で指定したブロック内でのオフセット
pucBuf	入力	write するデータを格納した領域のアドレス (アプリケーション空間のアドレス)
iSize	入力	write するバイト数
iMode	入力	I/O モード。GRP_FS_OPEN_DIRECT_IO ビットが設定されている場合、可能な限りキャッシュバッファを使用せず、アプリケーションバッファからメディアに対し、直接連続ブロックで I/O を行います。

【機能詳細】

pucBuf で指定された領域に格納された iSize バイト分データを、ptFile で指定されたファイルに書込みます。ファイル内でのデータの書込み開始位置は、uiFsBlk、uiBlkOff で指定されます。UiFsBlk で、キャッシュブロック単位でのファイル内オフセットが指定され、さらに、uiBlkOff で同ブロック内でのオフセットが指定されます。

ファイルシステムのフリーな残りバイト数が iSize より小さい場合、その残りバイト数分を書込み、そのバイト数をリターン値として返します。

なお、iMode で GRP\_FS\_OPEN\_DIRECT\_IO の指定がない場合は、本関数は、直接デバイスドライバの I/O 関数を使って実現するのではなく、grp\_fs\_read\_buf / grp\_fs\_write\_buf 関数を使用し、キャッシュを使いながら読書きを行います。また、pucBuf で指定されたアドレスは、アプリケーションプログラムのアドレスですので、アプリケーションプログラムのアドレス空間が異なるシステムでは、単純なメモリコピーは使用できません。従いまして、pucBuf からデータを読み出す場合は、単純なコピールーチンを使用するのではなく、grp\_fs\_copyin を使用し、アドレス空間が異なる場合でも対応できるようにする必要があります。

iMode に GRP\_FS\_OPEN\_DIRECT\_IO の指定がある場合は、可能な限りキャッシュを使用せず、アプリケーションバッファからメディアに対し、直接連続ブロックで I/O を行います。本オプションのサポートは、任意で、デフォルトでサポートしている FAT ファイルシステムでは、シングル空

間のシステムで、GRP\_FS\_FAT\_DIRECT\_IO オプション付で **GR-FILE** を構築した場合にのみ、GRP\_FS\_O\_DIRECT\_IO オプションの指定が有効です。本オプションによるアプリケーションバッファからの直接 I/O の対象は、キャッシュブロック単位部分のデータで、同ブロックがメディア上で連続して確保できれば、連続した単位で、一括してメディアに対し **write** します。

**【リターン値】**

0 または 正值	実際に <b>write</b> できたバイト数
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_BAD_PARAM	pucBuf で指定された領域が正しくない
GRP_FS_ERR_BAD_OFF	オフセットが正しくない（ファイルの終わりを越えている等）
GRP_FS_ERR_PERMIT	ファイルシステムに対して <b>write</b> 権限がない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）

## 4.7.8 grp\_fs\_create\_t \*pfnCreate

【機能概要】 ファイル/ディレクトリの作成処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_create_t xxx_create;

int xxx_create (
    grp_fs_info_t *ptFs, grp_fs_file_t *ptDir,
    const grp_uchar_t **ppucPath,
    grp_uint32_t uiType, grp_uint32_t uiProtect, grp_uint32_t uiAttr,
    grp_uchar_t *pucComp, grp_fs_file_t **pptFile);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
ptDir	入力	検索開始ディレクトリ (NULL の場合ルートからサーチ開始)
ppucPath	入力	*ppucPath : 作成対象ファイルのパス名
	出力	*ppucPath : 残りのパス名の先頭アドレス
uiType	入力	ファイルタイプ情報
GRP_FS_FILE_FILE		通常ファイル
GRP_FS_FILE_DIR		ディレクトリ
GRP_FS_FILE_LINK		リンクファイル
		なお、FAT ではロングファイル名を意味するが、ロングファイルも create 時は GRP_FS_FILE_FILE を指定する。
GRP_FS_FILE_OTHER		その他のタイプのファイル (FAT では本 create 関数では指定不可)
uiProtect	入力	ファイル/ディレクトリの保護モード
GRP_FS_PROT_RUSR	0400	オーナー read 可能
GRP_FS_PROT_WUSR	0200	オーナー write 可能
GRP_FS_PROT_XUSR	0100	オーナー実行可能
GRP_FS_PROT_RGRP	0040	グループ内 read 可能
GRP_FS_PROT_WGRP	0020	グループ内 write 可能
GRP_FS_PROT_XGRP	0010	グループ内実行可能
GRP_FS_PROT_ROTH	0004	他のユーザ read 可能
GRP_FS_PROT_WOTH	0002	他のユーザ write 可能
GRP_FS_PROT_XOTH	0001	他のユーザ実行可能

但し、FAT ファイルシステムの場合は、ユーザの概念がないため、グループ、他のユーザのビットは指定しても、オーナービット値の指定を使用します。但し、隠しファイルの場合は、グループ、他のユーザビットを自動的に 0 に設定します。さらに、FAT には実行可否の概念がなく、情報も保持できないため、指定に係らず、実行可否ビットは、ディレクトリ、または、ファイル名のサフィックスが "EXE"、"COM"、"DLL"、"BAT" の場合、自動的に 1 に設定し、その他の場合は、0 とします。また、read 不可の概念がないため、GRP\_FS\_PROT\_RUSR の指定がなかった場合でも、同ビットが指定されたものとして扱います。

uiAttr            入力            ファイルシステム依存の属性情報

FAT ファイルシステムの場合、以下を指定可能です。これらは、"fat.h" に定義されています。

FAT_ATTR_HIDDEN	0x02	隠しファイル
FAT_ATTR_SYSTEM	0x04	システムファイル

FAT_ATTR_ARCHIVE	0x20	バックアップ要
pucComp	入力	ファイル名検索時のパスコンポーネントを格納する作業領域のアドレス GRP_FS_MAX_COMP バイトの領域が確保されていることを前提とする。
pptFile	出力	作成/発見した対象ファイル、または、検索中に検出したファイルシステム跨りとなるディレクトリのファイル管理情報へのポインタを格納する領域のアドレス、

### 【機能詳細】

ptFs で指定されたファイルシステムの ptDir で指定されたディレクトリから検索を開始し、\*ppucPath で得られるパス名を持つファイルまたはディレクトリを作成します。作成するファイルのタイプ、保護モード、ファイルシステム依存の属性は、それぞれ、ucType、uiProtect、uiAttr で指定されています。

pucComp パラメータには、検索の際、パス名から各コンポーネント名を切出し保持するための作業領域として使用可能な GRP\_FS\_MAX\_COMP バイトの領域のアドレスが指定されています。

通常は、対象のファイル、または、ディレクトリを作成し、作成したファイル/ディレクトリのファイル管理情報へのポインタを\*pptFile に格納して返しますが、指定されたパス名をたどる中で、他のファイルシステムを mount したディレクトリに達した場合、あるいは、ルートディレクトリまで上っていた状態で、さらに、“.” が指定されている場合は、そこで、検索処理を打ち切ります。この場合、\*pptFile には、そのディレクトリ、または、ルートディレクトリのファイル管理情報へのポインタを返し、\*ppucPath には、残ったパス名のアドレスを設定して、リターン値として、GRP\_FS\_COMP\_MIDDLE(正值)を返します。

対象のファイルが既に存在した場合は、\*pptFile に発見したファイルのファイル管理情報へのポインタを格納し、リターン値として GRP\_FS\_ERR\_EXIST を返します。

なお、本関数の実現に当たっては、通常、grp\_fs\_file\_open\_common を用い、作成対象のファイル/ディレクトリの親ディレクトリをオープンし、親ディレクトリ下にファイルを作成する部分をファイルシステム依存の形で実現するという形で行います。

### 【リターン値】

0	作成成功
GRP_FS_COMP_MIDDLE	パス名を辿る中でファイルシステムを跨り、処理を中断した
GRP_FS_ERR_EXIST	対象のファイルが既に存在する
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_PERMIT	親ディレクトリに write 権限がない
GRP_FS_ERR_BAD_NAME	パス名に不正な文字コードが含まれている
GRP_FS_ERR_NOT_FOUND	親ディレクトリが見つからない
GRP_FS_ERR_TOO_LONG	ファイル名称が長すぎる
GRP_FS_ERR_BAD_TYPE	ファイルタイプが正しくない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)



## 4.7.9 grp\_fs\_unlink\_t \*pfnUnlink

【機能概要】 ファイル/ディレクトリの削除処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_unlink_t xxx_unlink;

int xxx_unlink (
    grp_fs_info_t *ptFs, grp_fs_file_t *ptDir,
    const grp_uchar_t **ppucPath,
    grp_uchar_t *pucComp, grp_fs_file_t **pptFile);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
ptDir	入力	検索開始ディレクトリ (NULL の場合ルートからサーチ開始)
ppucPath	入力	*ppucPath : 削除対象ファイル/ディレクトリのパス名
	出力	*ppucPath : 残りのパス名の先頭アドレス
pucComp	入力	ファイル名検索時のパスコンポーネントを格納する作業領域の アドレス GRP_FS_MAX_COMP バイトの領域が確保されて いることを前提とする。
pptFile	出力	検索中に検出したファイルシステム跨り場合、ファイルシステム 跨りとなるディレクトリのファイル管理情報へのポインタを格納 する領域のアドレス

【機能詳細】

ptFs で指定されたファイルシステムの ptDir で指定されたディレクトリから検索を開始し、\*ppucPath で得られるパス名を持つファイルまたはディレクトリを削除します。

pucComp パラメータには、検索の際、パス名から各コンポーネント名を切出し保持するための作業領域として使用可能な GRP\_FS\_MAX\_COMP バイトの領域のアドレスが指定されています。

通常は、対象のファイル、または、ディレクトリを削除し、\*pptFile には、特に情報を設定しませんが、指定されたパス名をたどる中で、他のファイルシステムを mount したディレクトリに達した場合、あるいは、ルートディレクトリまで上っていた状態で、さらに、".." が指定されている場合は、そこで、検索処理を打ち切ります。この場合、\*pptFile には、そのディレクトリ、または、ルートディレクトリのファイル管理情報へにポインタを返し、\*ppucPath には、残ったパス名のアドレスを設定して、リターン値として、GRP\_FS\_COMP\_MIDDLE(正値)を返します。

なお、本関数の実現に当たっては、通常、grp\_fs\_file\_open\_common を用い、削除対象のファイル/ディレクトリの親ディレクトリをオープンし、親ディレクトリ下にファイルを削除する部分をファイルシステム依存の形で実現するという形で行います。



## 【リターン値】

0	削除成功
GRP_FS_COMP_MIDDLE	パス名を辿る中でファイルシステムを跨り、処理を中断した
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_PERMIT	親ディレクトリに <b>write</b> 権限がない
GRP_FS_ERR_BAD_NAME	パス名に不正な文字コードが含まれている
GRP_FS_ERR_TOO_LONG	ファイル名称が長すぎる
GRP_FS_ERR_NOT_FOUND	親ディレクトリが見つからない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.10 grp\_fs\_rename\_t \*pfnRename

【機能概要】 ファイル/ディレクトリ名称変更関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_rename_t xxx_rename;

int xxx_rename (
    grp_fs_info_t *ptFs,
    grp_fs_file_t *ptOldDir, const grp_uchar_t *pucOld,
    grp_fs_file_t *ptNewDir, const grp_uchar_t *pucNew);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
ptOldDir	入力	名称変更対象のファイル/ディレクトリの親ディレクトリの ファイル管理情報へのポインタ
pucOld	入力	名称変更対象のファイル/ディレクトリ名
ptNewDir	入力	新しいファイル/ディレクトリの親ディレクトリのファイル 管理情報へのポインタ
pucNew	入力	新しいファイル/ディレクトリ名

【機能詳細】

ptFs で指定されたファイルシステムの ptOldDir ディレクトリ下にある、pucOld で指定されたファイル名を持つファイル/ディレクトリを、ptNewDir ディレクトリ下にある、pucNew で指定されたファイル/ディレクトリ名に変更します。

【リターン値】

0	名称変更成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_PERMIT	親ディレクトリに write 権限がない
GRP_FS_ERR_BAD_NAME	パス名に不正な文字コードが含まれている
GRP_FS_ERR_TOO_LONG	ファイル名称が長すぎる
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.1 1 grp\_fs\_get\_attr\_t \*pfnGetAttr

【機能概要】 ファイル/ディレクトリの属性情報の取得関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_get_attr_t xxx_get_attr;

int xxx_get_attr (grp_fs_file_t *ptFile, grp_fs_dir_ent_t *ptAttr);
```

【パラメータ】

ptFile	入力	属性取得対象ファイル/ディレクトリのファイル管理情報へのポインタ
ptAttr	出力	ファイル属性情報を格納する領域のアドレス 本パラメータの構造体の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照

【機能詳細】

ptFile で指定されたファイル/ディレクトリの属性情報を取得し、ptAttr で指定した領域に格納します。本関数は、ptAttr で指された構造体データのうち、iDev、uiFid、ucType、uiProtect、iSize、iCTime、iMtime、iAtime、uiAttr、uiMisc フィールドに有効な値を設定します。その他のフィールドは、必ずしも有効な値を設定する必要はありません。なお、FAT ファイルシステムの場合、uiMisc フィールドも、特別有効な値を規定していません。

属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、grp\_fs\_dir\_ent 構造体の iSize メンバ変数は型が符号なし 32 ビットに変更され、名称も uiSize に変更されます。

【リターン値】

0	属性取得成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.1.2 grp\_fs\_set\_attr\_t \*pfnSetAttr

【機能概要】 ファイル/ディレクトリの属性情報の設定・変更関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_set_attr_t xxx_set_attr;

int xxx_set_attr (grp_fs_file_t *ptFile, grp_fs_dir_ent_t *ptAttr);
```

【パラメータ】

ptFile	入力	属性設定・変更対象ファイル/ディレクトリのファイル管理情報へのポインタ
ptAttr	入力	ファイル属性情報を格納した領域のアドレス 本パラメータの構造体の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照

【機能詳細】

ptFile で指定されたファイル/ディレクトリの属性情報を ptAttr で指定した情報に従い、設定・変更します。この設定・変更は、ptAttr のフィールドのうち、uiProtect、iCTime、iMtime、iAtime、uiAttr、uiMisc フィールドの値を用いて行います。その他のフィールドは、無視します。

なお、FAT ファイルシステムの場合、uiMisc の情報も使用しません。また、uiAttr フィールドは、FAT\_ATTR\_HIDDEN、FAT\_ATTR\_SYSTEM、FAT\_ATTR\_ARCHIVE ビットのみ有効です。

ptAttr で指定する属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

【リターン値】

0	属性情報設定・変更成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.13 grp\_fs\_truncate\_t \*pfnTruncate

【機能概要】 ファイルデータの削除関数

【関数形式】

```
#include "grp_fs_cfg.h"
```

```
grp_fs_truncate_t xxx_truncate;
```

```
int xxx_truncate (grp_fs_file_t *ptFile, grp_uint32_t uiFsBlk, grp_uint32_t uiBlkOff);
```

【パラメータ】

ptFile	入力	データ削除対象ファイルのファイル管理情報へのポインタ
uiFsBlk	入力	削除開始位置のファイル内でのブロック番号 (ファイルシステムブロック単位)
uiBlkOff	入力	削除開始位置の uiFsBlk で指定したブロック内でのオフセット

【機能詳細】

ptFile で指定されたファイルについて、指定された位置以降のデータを削除し、ファイルサイズを変更します。データの削除開始位置は、uiFsBlk、uiBlkOff で指定されます。uiFsBlk で、ファイルシステムのブロックサイズ単位でのファイル内オフセットが指定され、さらに、uiBlkOff で同ブロック内でのオフセットが指定されます。

【リターン値】

0	正常終了
GRP_FS_ERR_PERMIT	同ファイルに対して write 権限がない
GRP_FS_ERR_BAD_OFF	削除開始位置が正しくない (ファイルサイズを超えている)
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.14 grp\_fs\_get\_dirent\_t \*ptnGetDirEnt

【機能概要】 ディレクトリエントリ情報の取得関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_get_dirent_t xxx_get_dirent;

int xxx_get_dirent (grp_fs_file_t *ptDir, grp_fs_dir_ent_t *ptDirent);
```

【パラメータ】

ptDir	入力	取得対象ディレクトリのファイル管理情報へのポインタポインタ
ptDirent	入力	下記フィールドを設定した grp_fs_dir_ent_t 構造体のアドレス
		pucName: ファイル名を格納する領域のアドレス
		sNameSize: ファイル名の格納領域のサイズ
		uiStart: ディレクトリ内のサーチ開始位置
	出力	サーチ開始位置以降で見つかったディレクトリエントリの情報を 本パラメータに指定した領域に格納する。なお、pucName で指定 した領域には、見つかったファイル名を、sNameSize フィールド は、同ファイル名の長さを、uiStart には、同エントリのディレク トリ内の開始位置を設定する。 本パラメータの構造体の詳細については、前述の「アプリケー ションインタフェース関連の構造体」の節を参照

【機能詳細】

ptDir で指定したディレクトリファイルを検索し、ptDirent で指定したサーチ開始位置以降で見つかったフリーでないディレクトリエントリ情報を ptDirent で指定した領域に返します。属性情報の詳細については、前述の「アプリケーションインタフェース関連の構造体」の節を参照下さい。

リターン値は、得られたメディア上のディレクトリエントリのサイズで、フリーでないエントリが見つからなかった場合は、0 を返します。

なお、FAT ファイルシステムの場合、ロングファイル名のディレクトリエントリは、物理的には、複数のエントリからなっているため、本関数では、これらのエントリを読み込み、論理的な 1 つのディレクトリエントリとして返します。この場合、pucName で指定した領域には、これらの複数エントリから得られるロングファイル名を格納します。default で提供している UNICODE-シフト JIS 変換を使用している場合、日本語のロングファイル名は、シフト JIS に変換して返します。但し、ロングファイル用のディレクトリエントリは、ファイル名以外の情報がないため、iDev、ucType、pucName、sNameSize、uiAttr、uiStart、uiEnd 以外のフィールドは、0 で返します。

【注意事項】

コンパイルオプション「GRP\_FS\_ENABLE\_OVER\_2G」を有効にした場合、grp\_fs\_dir\_ent 構造体の iSize メンバ変数は型が符号なし 32 ビットに変更され、名称も uiSize に変更されます。

## 【リターン値】

正值	メディア上のディレクトリエントリのサイズ (ディレクトリエントリ情報取得成功)
0	フリーでないディレクトリエントリ情報なし
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DIR	ptDir がディレクトリでない
GRP_FS_ERR_BAD_PARAM	ptDirent で指定した uiStart が正しい値でない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.7.15 grp\_fs\_match\_comp\_t \*pfnMatchComp

【機能概要】 ディレクトリエントリの検索関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_match_comp_t xxx_match_comp;

int xxx_match_comp (
    grp_fs_file_t *ptDir,    grp_uchar_t *pucComp, int iPurge,
    grp_fs_file_t **pptFile, int iNeed, grp_uint32_t *puiOff);
```

【パラメータ】

ptDir	入力	検索対象ディレクトリのファイル管理情報へのポインタ
pucComp	入力	検索するファイル名
iPurge	入力	検索対象のファイル名が見つかった場合、ファイル名称キャッシュを削除するか否か（0：削除しない、0 以外：削除する）
pptFile	出力	見つかったファイル/ディレクトリのファイル管理情報へのポインタを格納する領域のアドレス
iNeed	入力	新規作成時に必要とするフリーなディレクトリエントリのサイズ 既存ファイル/ディレクトリの検索時は、0 が設定されている
puiOff	出力	iNeed で指定されたサイズのフリーエントリの開始オフセットを格納する領域のアドレス

【機能詳細】

ptDir で指定されたディレクトリ内を検索し、pucComp で示されたファイル名を持つファイル/ディレクトリのファイル管理情報へのポインタを\*pptFile に設定して返します。

検索対象のファイル/ディレクトリが見つからず、かつ、iNeed パラメータに 0 以外が設定されていた場合、iNeed パラメータで指定されたサイズ以上の連続したフリーエントリの開始オフセットを puiOff で指定された領域に設定して返します。

また、検索対象のファイル/ディレクトリが見つかり、iPurge に 0 以外が指定されている場合は、対応するファイル名称キャッシュを削除します。

【リターン値】

0	検索成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_TOO_LONG	ファイル名称が長すぎる
GRP_FS_ERR_NOT_FOUND	指定されたファイル/ディレクトリが見つからない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない（エラー番号の節の注参照）



## 4.7.16 grp\_fs\_check\_volume\_t \*pfnCheckVolume

【機能概要】 ボリューム名の取得関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_check_volume_t xxx_check_volume;

int xxx_check_volume (
    int iDev, grp_fs_info_t *ptFs,
    grp_uchar_t *pucVolName, grp_uint32_t *puiVolSerNo);
```

【パラメータ】

iDev	入力	チェックするメディアのデバイス番号
ptFs	入力	同メディアがファイルシステムとして mount されている場合のファイルシステム情報へのポインタ (grp_fs_check_fs_dev の場合)。grp_fs_check_volume の場合は、NULL が設定されている。
pucVolName	出力	得られたボリューム名を格納する領域のアドレス 本領域は、GRP_FS_VOL_NAME_LEN のサイズを持つ
puiVolSerNo	出力	得られたボリュームシリアル番号を格納する領域のアドレス

【機能詳細】

iDev で指定されたメディアをチェックし、同メディアのボリューム名とシリアル番号を pucVolName、puiVolSerNo で指定された領域に返します。本関数は、grp\_fs\_check\_fs\_dev と grp\_fs\_check\_volume の 2 つの関数からコールされる場合があります。

grp\_fs\_check\_fs\_dev からコールされる場合は、同メディアが既に mount されているという状況のため、ptFs パラメータに同メディアのファイルシステム情報へのポインタが設定されて渡ってきます。一方、grp\_fs\_check\_volume からコールされる場合は、ファイルシステムとして mount されていないという状況のため、ptFs パラメータは NULL で渡ってきます。本関数では、ptFs が NULL でない場合は、同ファイル管理情報に格納されたデバイスアクセスのためのハンドルを使用して I/O を行い、ボリューム名情報を取得します。ptFs が NULL の場合は、本関数内で対象デバイスのオープン/クローズを行い、得られたハンドルを用いて、ボリューム名情報を取得します。

【リターン値】

正值	得られたボリューム名の長さ
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	デバイス番号が正しくない

## 4.7.17 grp\_fs\_sync\_t \*pfnSync

【機能概要】 キャッシュ情報の書き戻し関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_sync_t xxx_sync;

int xxx_sync (grp_fs_info_t *ptFs, int iMode);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
iMode	出力	書き戻しモード
		GRP_FS_SYNC_HINT : ヒント情報の書き戻し

【機能詳細】

ptFs で指定されたファイルシステムメディアに対して、ファイルシステム依存でキャッシュしている情報を書き戻します。

iMode に GRP\_FS\_SYNC\_HINT の指定がある場合は、FAT32 の残りクラスタのヒント情報など、ファイルシステム依存のヒント情報のメディアへの書き戻しも行います。FAT ファイルシステムの場合は、バージョン 1.11f から FAT ファイルシステム依存の処理で、GRP\_FS\_SYNC\_HINT 指定なしでも、デフォルト FAT32 の残りクラスタのヒント情報の書き戻しを行なうようになっていますが、依存コードに定義されているグローバル変数 grp\_fat\_sync\_hint の値を 0 に変更しますと、バージョン 1.11e 以前と同様に、新たに追加された GRP\_FS\_SYNC\_HINT 指定なしでは、FAT32 の残りクラスタのヒント情報の書き戻しを行いません。そこで、そのようなケースで同情報を書き戻す際には、アプリケーションから本オプションを指定することで、FAT32 の残りクラスタのヒント情報の書き戻しを行います。

【リターン値】

0	書き戻し成功
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

#### 4.8 OS 抽象化インタフェース

本節で規定する OS 抽象化インタフェースは、プラットフォーム/OS 依存処理部と、**GR-FILE** とのインタフェースです。表 4-1 1 に、OS 抽象化インタフェースの一覧を示します。また、表 4-1 2 に、対象プラットフォームで提供されていることを前提としている標準ライブラリ関数の一覧を示します。

表 4-1 1 OS 抽象化インタフェース一覧

#	インタフェース名	内容・機能	備考
1	grp_fs_create_sem	排他制御のためのセマフォアの生成	GR-VOS、 $\mu$ ITORN 対応版
2	grp_fs_get_sem	排他制御のためのセマフォアロックの取得	GR-VOS、 $\mu$ ITORN 対応版
3	grp_fs_release_sem	排他制御のためのセマフォアロックの解放	GR-VOS、 $\mu$ ITORN 対応版
4	grp_fs_get_taskid	タスク ID の取得	GR-VOS、 $\mu$ ITORN 対応版
5	grp_fs_copyin	アプリケーション空間からのデータの取得	同一空間版を提供
6	grp_fs_copyout	アプリケーション空間へのデータの格納	同一空間版を提供
7	grp_fs_get_str	アプリケーション空間からの文字列の取得	同一空間版を提供
8	grp_fs_get_current_time	現在時刻情報の取得	POSIX、T-Kernel 版を提供
9	grp_fs_printf	エラーメッセージの書式付出力	printf 版を提供
10	grp_fs_char_cnt	1 文字のバイトカウント取得	シフト JIS 版を提供
11	grp_fs_char_to_unicode	多国語→UNICODE 変換	シフト JIS 版を提供
12	grp_fs_unicode_to_char	UNICODE→多国語変換	シフト JIS 版を提供
13	grp_fs_cmp_fname	ファイル名の比較	シフト JIS 版を提供
14	grp_fs_to_upper	文字列の英小文字大文字変換	シフト JIS 版を提供
15	grp_mem_alloc	メモリの確保	GR-VOS、 $\mu$ ITORN 対応版を提供
16	grp_mem_free	メモリの解放	同上
17	grp_fs_inform_io_err	I/O エラー時のプラットフォーム依存フック関数へのポインタ	特別な処理が必要な場合、処理関数を設定
18	grp_stdio_io_stdin	標準入力(stdin)からの入力処理関数へのポインタ	C 言語標準 I/O で標準入力を利用する場合に定義して設定
19	grp_stdio_io_stdout	標準出力(stdout/stderr)への出力処理関数へのポインタ	C 言語標準 I/O で標準出力を利用する場合に定義して設定

表 4-1 2 GR-FILE で前提とする標準ライブラリ関数

#	インタフェース名	内容・機能
1	strcpy/strncpy	文字列のコピー
2	strcmp/strncmp	文字列比較
3	memcpy/memmove	メモリのコピー関数
4	memset	メモリのクリア
5	sprintf/snprintf/ vsprintf/vsnprintf	メモリ領域への書式付データの書込み

表 4-1 1 の備考に示しますように、**GR-FILE** では、幾つかのプラットフォーム用の OS 抽象化インタフェースの実現例を提供しています。このうち、GR-VOS 対応とは、グレースシステムスのミドルウェアで利用している OS 抽象化インタフェースである GR-VOS を利用したものです。GR-VOS では、ITRON をサポートしています。GR-VOS の詳細については、GR-VOS の説明書をご参考下さい。

OS 抽象化インタフェースの型宣言は、#15、#16 以外は、“grp\_fs\_mdep\_if.h” に定義されています。#15、#16 は、“include” ディレクトリ下の “grp\_stdio.h” に定義されています。また、実現例は、“mdep\_vos/base/grp\_fs\_mdep\_if.c”、および、“mdep\_xxx”、“gr\_vos” ディレクトリ下のファイルに記述されています。OS/プラットフォーム依存部の実現をサポートする関数については、「デバイスドライ

バ、OS/プラットフォーム依存関数向けサポートライブラリ関数」の節を参照下さい。

以下本節では、表 4-11 で示した OS 抽象化インタフェースの詳細を説明します。

## 4.8.1 grp\_fs\_create\_sem

【機能概要】 排他制御のためのセマフォアの生成

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_create_sem (
    grp_fs_sem_t *ptSem,
    const char *pcName, int iInstance, int iInitCnt);
```

【パラメータ】

ptSem	出力	生成したセマフォアの ID を格納する領域のアドレス
pcName	入力	生成するセマフォアの名称。
iInstance	入力	同一名称で生成するセマフォアのインスタンス番号
iInitCnt	入力	生成するセマフォアのセマフォアカウンタの初期値

【機能詳細】

pcName、iInstance で識別されるセマフォアを生成し、生成したセマフォアの ID を ptSem で指定された領域に返します。また、生成したセマフォアのセマフォアカウンタの初期値は、iInitCnt で設定します。

なお、セマフォア ID のタイプ grp\_fs\_sem\_t は、プラットフォーム依存の型で、"mdep\_xxx/include" ディレクトリ下にある "grp\_fs\_mdep\_types.h" に定義します。

生成するセマフォアを識別する pcName、iInstance をどのように使用するかは、プラットフォーム依存で、無視して、常にユニークなセマフォアを生成しても構いません。

また、OS レス環境等で、複数のタスクで同時に **GR-FILE** の関数をコールしないことを保障できる場合は、例えば、grp\_fs\_sem\_t を int とし、本関数では、特別な生成処理は行わず、\*ptSem に常に 0 を設定し、リターン値も常に 0 を返すという形でも構いません。

【リターン値】

0	生成成功
その他	生成失敗

#### 4.8.2 grp\_fs\_get\_sem

【機能概要】 排他制御のためのセマフォアロックの取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
int  grp_fs_get_sem (grp_fs_sem_t  tSem);
```

【パラメータ】

tSem	入力	ロックを取得するセマフォアの ID
------	----	-------------------

【機能詳細】

tSem で指定されたセマフォアのロックを取得します。本関数は、tSem で指定されたセマフォアのセマフォアカウント値が正であれば、その値を 1 つ減らし、すぐにコール元に制御を返します。tSem で指定されたセマフォアのセマフォアカウント値が 0 以下であれば、正の値になるまで待ってから、その値を 1 つ減らして、コール元に制御を返します。リターン値は、セマフォア ID 不正等、特別なエラーが検出されない限り、0 を返します。

なお、OS レス環境等で、複数のタスクで同時に **GR-FILE** の関数をコールしないことを保障できる場合は、本関数では特別な処理を行わず、常に 0 を返すという形でも構いません。

【リターン値】

0	セマフォア取得成功
その他	セマフォア取得失敗

### 4.8.3 grp\_fs\_release\_sem

【機能概要】 排他制御のためのセマフォアロックの解放

【関数形式】

```
#include "grp_fs_mdep_if.h"
void grp_fs_release_sem (grp_fs_sem_t tSem);
```

【パラメータ】

tSem	入力	ロックを解放するセマフォアの ID
------	----	-------------------

【機能詳細】

tSem で指定されたセマフォアのロックを解放します。本関数は、tSem で指定されたセマフォアのセマフォアカウンタ値を 1 つ増やし、同セマフォアカウンタが正の値になるのを待っているタスクを起します。

なお、OS レス環境等で、複数のタスクで同時に **GR-FILE** の関数をコールしないことを保障できる場合は、本関数では特別な処理を一切行わないという形でも構いません。

【リターン値】

なし

#### 4.8.4 grp\_fs\_get\_taskid

【機能概要】 タスク ID の取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
grp_fs_task_t  grp_fs_get_taskid(void);
```

【パラメータ】

なし

【機能詳細】

本関数をコールしたタスクのタスク ID を返します。タスク ID のタイプ `grp_fs_task_t` は、プラットフォーム依存の型で、“`mdep_xxx/include`” ディレクトリ下にある “`grp_fs_mdep_types.h`” に定義します。

なお、OS レス環境等で、複数のタスクで同時に **GR-FILE** の関数をコールしないことを保障できる場合は、本関数では特別な処理を行わず、`grp_fs_task_t` を `int` として、常に 0 を返すという形でも構いません。

【リターン値】

本関数をコールしたタスクのタスク ID



## 4.8.5 grp\_fs\_copyin

【機能概要】 アプリケーション空間からのデータの取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
grp_int32_t grp_fs_copyin(void *pvDst, void *pvSrc, grp_isize_t iSize);
```

【パラメータ】

pvDst	出力	データのコピー先の <b>GR-FILE</b> 空間のアドレス
pvSrc	入力	データのコピー元のアプリケーションタスク空間のアドレス
iSize	入力	コピーバイト数

【機能詳細】

pvSrc で指定された **GR-FILE** のアプリケーションタスク空間のアドレスから iSize 分データを読み込み、pvDst で指定された **GR-FILE** 空間の領域にコピーします。

なお、**GR-FILE** を利用するアプリケーションタスクのアドレス空間と **GR-FILE** のアドレス空間が同じ場合は、本関数は memcpy と等しい機能を持ちます。

【リターン値】

0 または 正值	コピーできたバイト数
負値	エラー

## 4.8.6 grp\_fs\_copyout

【機能概要】 アプリケーション空間へのデータの格納

【関数形式】

```
#include "grp_fs_mdep_if.h"
grp_int32_t grp_fs_copyout(void *pvDst, void *pvSrc, grp_isize_t iSize);
```

【パラメータ】

pvDst	出力	データのコピー先のアプリケーションタスク空間のアドレス
pvSrc	入力	データのコピー元の <b>GR-FILE</b> 空間のアドレス
iSize	入力	コピーバイト数

【機能詳細】

pvSrc で指定された **GR-FILE** 空間のアドレスから iSize 分データを読み込み、pvDst で指定されたアプリケーションタスク空間の領域にコピーします。

なお、**GR-FILE** を利用するアプリケーションタスクのアドレス空間と **GR-FILE** のアドレス空間が同じ場合は、本関数は memcpy と等しい機能を持ちます。

【リターン値】

0 または 正值	コピーできたバイト数
負値	エラー

## 4.8.7 grp\_fs\_get\_str

【機能概要】 アプリケーション空間からの文字列取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_get_str(grp_uchar_t *pucDst, grp_uchar_t *pucSrc, int iSize);
```

【パラメータ】

pucDst	出力	文字列のコピー先の <b>GR-FILE</b> 空間のアドレス
pucSrc	入力	文字列のコピー元のアプリケーションタスク空間のアドレス
iSize	入力	コピー先領域のサイズ

【機能詳細】

pucSrc で指定された **GR-FILE** のアプリケーションタスク空間のアドレスから、pucDst で指定された **GR-FILE** 空間の領域に NULL で終端された文字列をコピーします。文字列の長さが、iSize 以上となる場合は、iSize-1 バイトだけコピーし、NULL で終端します。

なお、**GR-FILE** を利用するアプリケーションタスクのアドレス空間と **GR-FILE** のアドレス空間が同じ場合は、本関数は `strncpy` を使って実現することができます。

【リターン値】

0	文字列コピー成功
負値	エラー

#### 4.8.8 grp\_fs\_get\_current\_time

【機能概要】 現在時刻情報の取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_get_current_time (grp_int32_t *piTime);
```

【パラメータ】

piTime	出力	取得した現在時刻情報を格納する領域のアドレス
--------	----	------------------------

【機能詳細】

現在時刻を取得し、1970/1/1 からのトータル秒に変換した値を piTime で指定された領域に格納します。また、エラー発生時は piTime に 0 を返して下さい。なお、**GR-FILE** では、日付、時刻から 1970/1/1 からのトータル秒に変換するための関数 `grp_time_mktime` を提供しています。詳細については、「デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数」の節を参照下さい。

【リターン値】

0	取得成功
- 1	エラー

#### 4.8.9 grp\_fs\_printf

【機能概要】 エラーメッセージの書式付出力

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_printf(const char *pcMsg, ...);
```

【パラメータ】

pcMsg	入力	出力するメッセージの書式
-------	----	--------------

【機能詳細】

**GR-FILE** で発生したエラーを書式付で何らかの装置またはメディアに出力します。サポートする書式は、一般的な `printf` 関数相当とします。

【リターン値】

0 または 正值	出力したメッセージの長さ
-1	エラー

## 4.8.10 grp\_fs\_char\_cnt

【機能概要】 一文字のバイトカウント取得

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_char_cnt (const grp_uchar_t *pucStr);
```

【パラメータ】

pucStr	入力	バイトカウント取得対象の文字列のアドレス
--------	----	----------------------

【機能詳細】

プラットフォーム依存の文字コードで記述された pucStr で指定された文字列の先頭文字のバイト数を返します。例えば、シフト JIS コード系の場合、漢字は 2、ASCII 文字は 1、半角カナは 1、NULL 文字は 0、不当な文字コードは -1 を返します。

**GR-FILE** では、シフト JIS コード向けの関数 grp\_char\_sjis\_cnt を提供しています。

なお、ファイル名に多国語対応が不要な場合は、NULL 文字は 0、その他の文字は 1 を返す関数とします。

【リターン値】

0	先頭文字が NULL 文字
正值	先頭文字のバイト数
-1	不正な文字コード

## 4.8.1 1 grp\_fs\_char\_to\_unicode

【機能概要】 多国語→UNICODE 変換

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_char_to_unicode (const grp_uchar_t *pucStr, grp_uint32_t *puiCode);
```

【パラメータ】

pucStr	入力	変換対象の文字列のアドレス
puiCode	出力	変換された UNICODE を格納する領域のアドレス

【機能詳細】

プラットフォーム依存の文字コードで記述された pucStr で指定された文字列の先頭文字を UNICODE に変換し、4 バイトの値として puiCode で指定された領域に格納します。

**GR-FILE** では、シフト JIS コード向けの変換関数 grp\_char\_sjis\_to\_unicode を提供しています。

なお、ファイル名に多国語対応が不要な場合は、先頭文字を変換せず、1 バイトの文字コードの値を puiCode で指定された領域に設定する関数とします。

【リターン値】

0	先頭文字が NULL 文字
正值	先頭文字のバイト数
-1	不正な文字コード

## 4.8.1 2 grp\_fs\_unicode\_to\_char

【機能概要】 UNICODE→多国語変換

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_unicode_to_char (grp_uchar_t *pucStr, grp_uint32_t uiCode);
```

【パラメータ】

pucStr	出力	変換後の文字列を格納する領域のアドレス
uiCode	入力	変換する UNICODE 値

【機能詳細】

uiCode で指定された UNICODE 値を、プラットフォーム依存の文字コードの形式に変換し、pucStr で指定された領域に格納します。

**GR-FILE** では、シフト JIS コード向けの変換関数 `grp_char_unicode_to_sjis` を提供しています。

なお、ファイル名に多国語対応が不要な場合は、UNICODE 値を変換せず pucStr で指定された領域に格納する関数とします。

【リターン値】

0	UNICODE が NULL 文字
正值	変換後のバイト数
- 1	不正な文字コード



## 4.8.13 grp\_fs\_cmp\_fname

【機能概要】 英大文字変換後のファイル名比較

【関数形式】

```
#include "grp_fs_mdep_if.h"
int grp_fs_cmp_fname(const grp_uchar_t *pucName1, const grp_uchar_t *pucName2);
```

【パラメータ】

pucName1	入力	ファイル名文字列 1
pucName2	出力	ファイル名文字列 2

【機能詳細】

pucName1 と pucName2 で指定された文字列に含まれる英小文字を英大文字に変換し両者を文字列比較します。変換対象となる英小文字の文字コードは 0x61～0x7A の範囲です。pucName1 と pucName2 にシフト JIS の 2 バイトコードが含まれる場合は 2 バイトで 1 文字として扱われます。2 バイトコードの英小文字は変換対象には含まれません。

変換・比較は内部で行われ、変換・比較結果は pucName1 と pucName2 には影響しません。

【リターン値】

0	文字列が一致
- 1	文字列は不一致

## 4.8.14 grp\_fs\_to\_upper

【機能概要】 文字列中の半角英小文字を大文字に変換

【関数形式】

```
#include "grp_fs_mdep_if.h"
void grp_fs_to_upper(grp_uchar_t *pucUpName, const grp_uchar_t *pucOrgName);
```

【パラメータ】

pucOrgName	入力	変換元文字列の先頭アドレス
pucUpName	出力	変換結果を格納する領域のアドレス

【機能詳細】

pucOrgName に含まれる英小文字を英大文字に変換しながら pucUpName にコピーします。

変換対象となる英小文字の文字コードは 0x61～0x7A の範囲です。pucOrgName にシフト JIS の 2 バイトコードが含まれる場合は 2 バイトで 1 文字として扱われます。2 バイトコードの英小文字は変換対象には含まれません。

【リターン値】

なし

## 4.8.15 grp\_mem\_alloc

【機能概要】 メモリの確保

【関数形式】

```
#include "grp_fs_mdep_if.h"
void *grp_mem_alloc (grp_isize_t iSize);
```

【パラメータ】

iSize	入力	確保する領域のサイズ
-------	----	------------

【機能詳細】

iSize で指定された領域を確保し、確保した領域の先頭アドレスを返します。返すアドレスは、対象のプラットフォームで使用される主な型の変数用の領域として使用可能なアドレスバウンダリを持つアドレスとします。

なお、**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないプラットフォームのために、大きく割当てられた領域から切出して可変長領域として管理するライブラリ関数群 `grp_mem_vl_init`、`grp_mem_vl_add`、`grp_mem_vl_alloc`、`grp_mem_vl_free` を提供しています。詳細については、「デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数」の節を参照下さい。

【リターン値】

NULL	指定されたサイズの領域が確保できない
その他	確保した領域の先頭アドレス

## 4.8.16 grp\_mem\_free

【機能概要】 メモリの解放

【関数形式】

```
#include "grp_fs_mdep_if.h"
void grp_mem_free(void *pvMem);
```

【パラメータ】

pvMem	入力	解放する領域のアドレス
-------	----	-------------

【機能詳細】

pvMem パラメータで指定された領域を解放します。pvMem パラメータで指定するアドレスは、grp\_mem\_alloc で返ってきたアドレスとします。

なお、**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないプラットフォームのために、大きく割当てられた領域から切出して可変長領域として管理するライブラリ関数群 grp\_mem\_vl\_init、grp\_mem\_vl\_add、grp\_mem\_vl\_alloc、grp\_mem\_vl\_free を提供しています。詳細については、「デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数」の節を参照下さい。

【リターン値】

なし

## 4.8.17 grp\_fs\_inform\_io\_err

【機能概要】 I/O エラー時のプラットフォーム依存フック関数へのポインタ

【関数形式】

```
#include "grp_fs_mdep_if.h"

grp_fs_inform_io_err_func_t  *grp_fs_inform_io_err;

int  (*grp_fs_inform_io_err)(
    int  iDev, grp_uint32_t  uiDevBlk, grp_isize_t  iCnt, int  iMode);
```

【パラメータ】

iDev	入力	対象のデバイス番号
uiDevBlk	入力	対象のデバイスブロック番号
iCnt	入力	I/O カウント（デバイスブロック単位）
iMode	入力	I/O オペレーションの種別
GRP_FS_IO_READ		デバイス read 処理
GRP_FS_IO_WRITE		デバイス write 処理
GRP_FS_IO_REQ		デバイス write を伴うファイル I/O リクエスト
GRP_FS_IO_OP_ERR		実際にデバイス read/wrte エラー発生

【機能詳細】

デバイスの I/O エラーに対し、プラットフォーム依存の処理を行う場合、本関数ポインタ変数に同エラー処理関数を設定します。

本関数ポインタ変数にエラー処理関数を設定していないデフォルトの状態では、デバイスの I/O エラーが発生した場合、あるいは、grp\_fs\_invalidate\_fs\_dev で I/O 抑止状態にあるデバイスに対し、デバイスへの write 処理を伴うファイル I/O 要求を行った場合、同ファイル I/O 要求は GRP\_FS\_ERR\_IO でエラーリターンします。

本関数ポインタ変数に対し、プラットフォーム依存のエラー処理関数を設定しますと、上記のエラー発生時に、設定したエラー処理関数がコールされ、プラットフォーム依存の処理が行えます。例えば、利用者へのエラー状態の通知や、利用者からの処理方法の指示入力、さらに、ロギング等が行えます。なお、要求の処理をエラーとして返す場合は、同エラー処理関数は、GRP\_FS\_ERR\_IO 等 0 以外の値を返します。

また、同エラー処理関数がリターン値として 0 を返すことで、要求の処理をエラーでリターンするのではなく、不当に抜取られたメディアが挿入されるのを待ってから再実行するなど、I/O の再実行を指示することも可能です。但し、リターン値として 0 を返す場合は、I/O 抑止状態が解除されていることが必要です。I/O 抑止状態が解除されていない場合は、再度、同エラー処理関数がコールされます。

設定したプラットフォーム依存のエラー処理関数には、エラーとなったデバイスの番号、ブロック番号、I/O ブロックカウントが、iDev、uiDevBlk、iCnt パラメータで渡されます。また、iMode パラメータで、エラーとなったオペレーションの種別が渡されます。

iMode に、GRP\_FS\_IO\_READ または GRP\_FS\_IO\_WRITE と共に、GRP\_FS\_IO\_OP\_ERR が

設定されている場合は、同エラーが、実際にデバイスに対し I/O を行った結果のエラーであったことを示します。

iMode に GRP\_FS\_IO\_READ または GRP\_FS\_IO\_WRITE のみが設定されている場合は、実際にデバイスに対し I/O を行おうとした時点で、同デバイスが I/O 抑止状態であることを検出し、I/O 処理を抑止したことを示します。

iMode に GRP\_FS\_IO\_REQ が設定されている場合は、アプリケーションプログラムから I/O 要求を受けた時点で、対応するデバイスが I/O 抑止状態であることを検出し、同 I/O 要求処理を抑止したことを示します。この場合、物理的な I/O 対象のブロック番号、ブロックカウントの計算がまだ行われていない状態のため、uiDevBlk、iCnt には、0 が設定されています。

なお、設定したエラー処理関数には、**GR-FILE** システム全体の排他制御セマフォア grp\_fs\_ctl->tFsSem にロックがかかった状態で制御が渡されます。この状態では、**GR-FILE** が提供するすべてのアプリケーションインタフェースを使った I/O 要求は、同セマフォアによってブロックされます。従いまして、再挿入されたメディアのチェックを行うための関数 grp\_fs\_check\_fs\_dev 等、同エラー処理関数からアプリケーションインタフェース関数をコールする場合は、一旦同セマフォアを解放し、同エラー処理関数からリターンする前に再度同セマフォアを取得してリターンする必要があります。同エラー処理関数が挿抜処理アプリケーションと連携してメディアの再挿入待ち等を行う場合も、挿抜処理アプリケーション等にアプリケーションインタフェース関数の実行を許可するため、同様の処理が必要です。

#### 【リターン値】

0	要求の処理をエラーとせず、同処理を再度実行する
GRP_FS_ERR_IO 等	要求の処理をエラーとして処理する

## 4.8.18 grp\_stdio\_io\_stdin

【機能概要】 標準入力 (stdin) からの入力処理関数へのポインタ

【関数形式】

```
#include "grp_stdio.h"

grp_stdio_io_func_t *grp_stdio_io_stdin;

grp_isize_t (*grp_stdio_io_stdin) (
    FILE *ptFile, grp_uchar_t *pucBuf, grp_isize_t iSize);
```

【パラメータ】

ptFile	入力	FILE 構造体へのポインタ (= stdin)
pucBuf	出力	読込んだデータを格納する領域のアドレス
iSize	入力	読むデータの最大バイト数

【機能詳細】

**GR-FILE** の C 言語標準 I/O インタフェースで、標準入力(stdin)からの入力を可能とする場合、その処理関数のアドレスを本関数ポインタ変数に設定します。

本関数ポインタ変数に設定する関数は、標準入力からデータを入力し、pucBuf で指定された領域に入力したデータを格納する処理を行います。入力されたデータが iSize を超える場合は、iSize 分だけを pucBuf に指定された領域に格納します。入力されたデータが iSize を未満の場合は、入力されたデータを pucBuf に指定された領域に格納し、そのバイト数をリターン値として返します。

なお、デフォルトでは、本関数ポインタ変数には、標準入力への入力要求に対し、常にエラーを返す関数が設定されており、標準入力からの入力を可能とする必要がなければ、本関数ポインタ変数の初期化処理は不要です。ただし、NULL を設定した場合は不正呼び出しが発生する可能性があります。

【リターン値】

0	標準入力 EOF
正值	pucBuf に格納された入力データのバイト数
- 1	エラー

## 4.8.19 grp\_stdio\_io\_stdout

【機能概要】 標準出力 (stdin/stderr) への出力処理関数へのポインタ

【関数形式】

```
#include "grp_stdio.h"

grp_stdio_io_func_t *grp_stdio_io_stdout;

grp_isize_t (*grp_stdio_io_stdout) (
    FILE *ptFile, grp_uchar_t *pucBuf, grp_isize_t iSize);
```

【パラメータ】

ptFile	入力	FILE 構造体へのポインタ (= stdout/stderr)
pucBuf	入力	標準出力に出力するデータを格納した領域のアドレス
iSize	入力	出力するデータのバイト数

【機能詳細】

**GR-FILE** の C 言語標準 I/O インタフェースで、標準出力(stdout/stderr)に出力を可能とする場合、その処理関数のアドレスを本関数ポインタ変数に設定します。

本関数ポインタ変数に設定する関数は、pucBuf で指定された領域に格納された iSize 分のデータを、標準出力に出力する処理を行います。

なお、デフォルトでは、本関数ポインタ変数には、標準出力への出力要求に対し、常にエラーを返す関数が設定されており、標準出力への出力を可能とする必要がなければ、本関数ポインタ変数の初期化処理は不要です。ただし、NULL を設定した場合は不正呼び出しが発生する可能性があります。

【リターン値】

0 または 正值	標準出力に出力したバイト数
-1	エラー



#### 4.9 デバイスドライバインタフェース

本節で規定するデバイスドライバインタフェースは、メディアに対する I/O 機能を提供するデバイスドライバと、**GR-FILE** とのインタフェースです。

3.5.3 節で示しましたとおり、**GR-FILE** とデバイスドライバのインタフェースは、デバイスタイプ名称と、同デバイスの I/O 関数テーブルへのポインタを **GR-FILE** のデバイスドライバテーブル `grp_fs_dev_tbl[]` に登録することで行います。表 4-13 に、デバイスドライバテーブル `grp_fs_dev_tbl` の各エントリの構造体 `grp_fs_dev_tbl_t` の内容を示します。また、表 4-14 に、各エントリからポイントされるデバイスドライバの I/O 関数テーブルの構造体 `grp_fs_dev_op_t` の内容を示します。

表 4-13 デバイスドライバテーブルエントリの構造体 `grp_fs_dev_tbl_t` の内容

#	構造体メンバ名	タイプ	意味
1	<code>pcDevName</code>	<code>const char *</code>	デバイスタイプ名称
2	<code>ptOp</code>	<code>grp_fs_dev_op_t *</code>	デバイスの I/O 関数テーブルへのポインタ

表 4-14 デバイスドライバの I/O 関数テーブル構造体 `grp_fs_dev_op_t` 内容

#	構造体メンバ名	タイプ	意味
1	<code>pfnOpen</code>	<code>grp_fs_dev_open_t</code>	デバイスのオープン処理関数
2	<code>pfnClose</code>	<code>grp_fs_dev_close_t</code>	デバイスのクローズ処理関数
3	<code>pfnRead</code>	<code>grp_fs_dev_read_t</code>	デバイスの read 処理関数
4	<code>pfnWrite</code>	<code>grp_fs_dev_write_t</code>	デバイスの write 処理関数
5	<code>pfnIoctl</code>	<code>grp_fs_dev_ioctl_t</code>	デバイスの ioctl 処理関数 (オプション関数)

以下本節では、上記 I/O 関数テーブルの各メンバの詳細インタフェースを説明します。なお、これらの構造体の型宣言は、“`grp_fs_cfg.h`” に定義されています。Ioctl 処理のパラメータについては、“`include`” 下の “`grp_fs_dev_io_if.h`” に定義されています。

。

## 4.9.1 grp\_fs\_dev\_open\_t \*pfnOpen

【機能概要】 デバイスのオープン処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_dev_open_t xxx_open;

int xxx_open(int iDev, int iRWOOpen,
              grp_int32_t *piHandle,
              grp_uint32_t *puiOff, grp_uint32_t *puiSize, int *piSzShift);
```

【パラメータ】

iDev	入力	デバイス番号（3.5.3 節 参照）
iRWOOpen	入力	オープンモード（0：read-only、0 以外：read/write）
piHandle	出力	I/O 要求で使用するデバイスハンドルを格納する領域のアドレス
puiOff	出力	メディア内での開始オフセット（物理ブロック（セクタ）単位）を返す領域のアドレス
puiSize	出力	指定のデバイスメディアのトータルブロック（セクタ）数を返す領域のアドレス。トータルサイズが不明の場合は、0 を設定して返す。
piSzShift	出力	指定のデバイスメディアの物理ブロック（セクタ）サイズの 2 のべき乗数値を返す領域のアドレス

【機能詳細】

iDev で指定されたデバイスメディアをオープンし、その属性情報や、同デバイスメディアに対する I/O 処理時に指定するデバイスハンドルを指定された領域に返します。

iDev には、デバイス種別番号、パーティション番号、サブデバイス番号を 3.5.3 節で示した形でエンコードした値が格納されています。デバイスドライバの I/O 関数は、デバイス種別番号に対応してコールされますので、一般的には、デバイス種別番号をデバイスドライバ側で意識する必要はありませんが、複数のデバイスタイプでドライバの I/O 関数を共用している場合、デバイス種別番号を用いることでデバイスタイプを区別することが出来ます。また、サブデバイス番号は、同一種別のデバイス内で個々のデバイスを区別するのに使用します。パーティション番号は、対象のデバイスメディアが複数の論理的なパーティションから構成されている場合に、1 つのデバイスメディア内のパーティションを区別するのに使用します。

iRWOOpen は、オープンモードの指定で、0 以外の場合は、対象のメディアが write 可能かをチェックし、write 可能でなければ、エラーを返します。

その他のパラメータは、オープン結果の属性情報を返すためのパラメータです。

\*piHandle には、以降の I/O 要求で使用するデバイスハンドルを設定して返します。**GR-FILE** では、デバイスドライバがハンドルを用いて I/O する方式と、デバイス番号を用いて I/O を方式の両方をサポートしています。従いまして、デバイス番号により I/O を行うタイプのドライバの場合は、

本 `*piHandle` には、特に意味のある値を設定しなくてもよいですが、このようなケースでは、通常は `iDev` 値等を設定して返します。

また、`*puiOff` には、I/O 要求を出す際のメディア内の開始オフセットを物理ブロック（セクタ）単位で設定して返します。通常 `*puiOffset` には 0 を設定して返しますが、パーティション分割されているメディア等では、パーティション番号に合わせ、対応するパーティションのメディア内のオフセットを設定して返すという使い方をします。

`*puiSize` には、指定されたデバイスメディア（またはパーティション）のトータル物理ブロック（セクタ）数を設定して返します。この値を 0 以外に設定して返しますと、アプリケーションからのトータルブロック数を超える同デバイスメディアに対する I/O 要求は、**GR-FILE** でチェックし、エラーを返します。

最後に、`*puiSzShift` には、物理ブロック（セクタ）サイズの 2 のべき乗数値を設定して返します。**GR-FILE** では、デバイスに I/O 要求を出す際、I/O オフセット値、I/O バイト数をこの物理ブロック（セクタ）サイズ単位で指定して行います。具体的には、I/O オフセット値としては、要求の物理ブロックオフセット値＋上記 `*puiOff` の値を指定します。

なお、**GR-FILE** では、デバイスドライバの実装を簡単にするため、標準的なパーティションテーブルをアクセスするための関数を提供しています。詳細については、「デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数」の節を参照下さい。

#### 【リターン値】

0	オープン成功
その他	エラー。なお、 <code>GRP_FS_ERR_IO</code> 等、 <b>GR-FILE</b> で規定したエラー番号とします。

## 4.9.2 grp\_fs\_dev\_close\_t \*pfnClose

【機能概要】 デバイスのクローズ処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_dev_close_t xxx_close;

int xxx_close (grp_int32_t iHandle, int iDev);
```

【パラメータ】

iHandle	入力	pfnOpen で得られたデバイスハンドル
iDev	入力	デバイス番号（3.5.3 節 参照）

【機能詳細】

iHandle、iDev で指定されたデバイスメディアをクローズします。

ハンドルを用いて I/O 処理を行うデバイスドライバの場合は、iHandle を用いてクローズ処理を行い、デバイス番号を用いて I/O 処理を行うデバイスドライバは、iDev を用いてクローズ処理を行います。

なお、iDev パラメータは、ハンドルベースのデバイスドライバでも、デバイス番号ベースのエラーメッセージ出力等で使用します。

【リターン値】

0	クローズ成功
その他	エラー。なお、GRP_FS_ERR_IO 等、 <b>GR-FILE</b> で規定したエラー番号とします。

## 4.9.3 grp\_fs\_dev\_read\_t \*pfnRead

【機能概要】 デバイスの read 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_dev_read_t xxx_read;

int xxx_read (grp_int32_t iHandle, int iDev,
              grp_uint32_t uiDevBlk, grp_uchar_t *pucBuf, grp_isize_t iCnt);
```

【パラメータ】

iHandle	入力	pfnOpen で得られたデバイスハンドル
iDev	入力	デバイス番号 (3.5.3 節 参照)
uiDevBlk	入力	read 開始物理ブロックオフセット
pucBuf	出力	read したデータを格納する領域のアドレス
iCnt	入力	read する物理ブロック数

【機能詳細】

iHandle、iDev で指定されたデバイスメディアからデータを読み込み、pucBuf で指定された領域に読込んだデータを格納します。

ハンドルを用いて I/O 処理を行うデバイスドライバの場合は、iHandle を用いて I/O 処理を行い、デバイス番号を用いて I/O 処理を行うデバイスドライバは、iDev を用いて I/O 処理を行います。

uiDevBlk には、read を開始する物理ブロック（セクタ）番号が、iCnt に read する物理ブロック（セクタ）数が指定されています。uiDevBlk には、対応するデバイスメディアの論理的なパーティション内でのブロックオフセット値 + オープン時に返ってきた同パーティションのメディア内での開始物理ブロックオフセットが設定されてコールされます。

なお、iDev パラメータは、ハンドルベースのデバイスドライバでも、デバイス番号ベースのエラーメッセージ出力等で使用します。

【リターン値】

0 または 正值	実際に読込んだ物理ブロック（セクタ）数
その他	エラー。なお、GRP_FS_ERR_IO 等、GR-FILE で規定したエラー番号とします。

## 4.9.4 grp\_fs\_dev\_write\_t \*pfnWrite

【機能概要】 デバイスの write 処理関数

【関数形式】

```
#include "grp_fs_cfg.h"

grp_fs_dev_write_t xxx_write;

int xxx_write (grp_int32_t iHandle, int iDev,
               grp_uint32_t uiDevBlk, grp_uchar_t *pucBuf, grp_isize_t iCnt);
```

【パラメータ】

iHandle	入力	pfnOpen で得られたデバイスハンドル
iDev	入力	デバイス番号 (3.5.3 節 参照)
uiDevBlk	入力	write 開始物理ブロックオフセット
pucBuf	入力	write するデータを格納した領域のアドレス
iCnt	入力	write する物理ブロック数

【機能詳細】

pucBuf で指定された領域に格納された iCnt ブロック分のデータを、iHandle、iDev で指定されたデバイスメディアに書込みます。

ハンドルを用いて I/O 処理を行うデバイスドライバの場合は、iHandle を用いて I/O 処理を行い、デバイス番号を用いて I/O 処理を行うデバイスドライバは、iDev を用いて I/O 処理を行います。

uiDevBlk には、write を開始する物理ブロック（セクタ）番号が、iCnt に write する物理ブロック（セクタ）数が指定されています。UiDevBlk には、対応するデバイスメディアの論理的なパーティション内でのブロックオフセット値 + オープン時に返ってきた同パーティションのメディア内での開始物理ブロックオフセットが設定されてコールされます。

なお、iDev パラメータは、ハンドルベースのデバイスドライバでも、デバイス番号ベースのエラーメッセージ出力等で使用します。

【リターン値】

0 または 正值	実際に書込んだ物理ブロック（セクタ）数
その他	エラー。なお、GRP_FS_ERR_IO 等、GR-FILE で規定したエラー番号とします。

## 4.9.5 grp\_fs\_dev\_ioctl\_t \*pfnioctl

【機能概要】 デバイス固有の I/O 制御処理関数

【関数形式】

```
#include "grp_fs_dev_io_if.h"
#include "grp_fs_cfg.h"
grp_fs_dev_ioctl_t xxx_ioctl;

int xxx_ioctl(int iDev, grp_uint32_t uiCmd void *pvParam);
```

【パラメータ】

iDev	入力	デバイス番号 (3.5.3 節 参照)
uiCmd	入力	実行する I/O 制御の機能番号
pvParam	入力/出力	実行する I/O 制御に渡すパラメータ

【機能詳細】

iDev で指定されたデバイスに対し、uiCmd で指定されたデバイス固有の I/O 制御機能を、pvParam で指定されたパラメータで実行します。

uiCmd、pvParam の内容は、実行するデバイス固有機能に依存しますが、**GR-FILE** では、以下の機能に関して、具体的な機能番号や、パラメータを規定しています。

機能番号	機能、および、パラメータ
GRP_FS_DEV_CTL_GET_MEDIA	メディア情報の取得。pvParam に指定された grp_fs_media_info_t の型を持つ領域に、取得したメディア情報を格納します。(grp_fs_media_info_t の詳細については、4.2.2 項のアプリケーションインタフェース関連の構造体を参照)
GRP_FS_DEV_CTL_EJECT	メディアの取り出し制御。pvParam に指定された int 型の領域に格納されたメディアの取り出し制御指示番号に従い、メディアの取り出し制御を行います。取り出し制御指示番号としては、以下の指定があります。
	GRP_FS_DEV_LOCK_MEDIA 取り出しロックを設定
	GRP_FS_DEV_UNLOCK_MEDIA 取り出しロックを解除
	GRP_FS_DEV_EJECT_MEDIA メディアを排出
GRP_FS_DEV_CTL_FORMAT	メディアの物理フォーマット。メディアに対し、物理的なフォーマットを実行します。pvParam には NULL が指定されています。
GRP_FS_DEV_CTL_GET_WRITE_PROTECT	メディアのライトプロテクト状態を取得します。pvParam には int 型の領域が設定されています。

なお、規定されている上記機能も含め、各機能をサポートするかどうかは、各デバイス依存です。

対象のデバイスが、uiCmd で指定された機能をサポートしていない場合は、GRP\_FS\_ERR\_NOT\_SUPPORT エラーを返します。また、ioctl 機能自体を全くサポートしていない場合は、pfnIoctl 自体を NULL に設定しておいてもかまいません。

**【リターン値】**

0	指定されたデバイス固有機能実行成功
GRP_FS_ERR_NOT_SUPPORT	指定されたデバイス固有機能はサポートしていない
その他	エラー。なお、GRP_FS_ERR_IO 等、 <b>GR-FILE</b> で規定したエラー番号とします。



4.10 ファイルシステム依存関数向け GR-FILE フック関数

GR-FILE では、ファイルシステム依存関数向けに、フック関数を提供しています。表 4-15 に、ファイルシステム依存関数向けの GR-FILE フック関数の一覧を示します。

表 4-15 ファイルシステム依存関数向けの GR-FILE フック関数

#	関数名	機能
1	fat_interrupt_lookup	・ディレクトリ検索ループを中断するフック関数

以下、上記関数のインタフェースを説明します。

## 4.1 0.1 fat\_interrupt\_lookup

【機能概要】 ディレクトリ検索ループを中断するフック関数

【関数形式】

```
#include "fat.h"

int (*fat_interrupt_lookup)(int iDev, grp_uint32_t uiFid, grp_uint32_t uiOffset);
```

【パラメータ】

iDev	入力	I/O 対象ディレクトリのデバイス番号
uiFid	入力	I/O 対象ディレクトリの先頭クラスタ番号
iOffset	入力	上記ディレクトリの I/O オフセット値

【機能詳細】

ファイルシステム異常で、クラスタループがあった場合、`grp_fs_get_dirent/grp_fs_get_attr` 時のディレクトリサイズ計算や `grp_fs_open` などで行なわれるディレクトリ内のファイル検索の際に、**GR-FILE** 内でクラスタリンクを辿り、ループしてしまう現象が起きます。

**GR-FILE** には各 FAT タイプに許容された最大サイズでのディレクトリ内オフセットでループから抜ける処理がありますが、FAT で許容されるサイズは大きいため、実質的には、長時間 I/O ループとなり、無限ループに近い状況となります。

本フック関数は、アプリケーション側の判断で I/O ループを中断できる様に、ディレクトリのブロック単位での I/O のタイミングで、**GR-FILE** より呼ばれます。

本関数変数に、アプリケーションの中断判定用のフック関数を登録しておくことで、登録されたフック関数が呼ばれ、フック関数がエラーを返すことで、サイズ計算や検索処理を中断できます。

本関数変数の初期値は `NULL` となっています。登録を解除する場合は、本関数変数を使用していない状態（アンマウント状態等）で、本関数変数に `NULL` を設定します。

本関数変数への登録は、`grp_fs_init` 関数による **GR-FILE** の初期化後に行います。

【リターン値】

0	正常（処理継続）
負値	エラー。通常は、 <code>GRP_FS_ERR_FS</code> などを返す。 なお、 <b>GR-FILE</b> で使用していない、アプリ独自の中断やタイムアウトを表すようなエラー番号を返しても構いません。

## 【サンプルコード】

以下のサンプルコードは、オフセット値のみ判定し、ディレクトリ内オフセット値が 1MB を超えた場合、異常な状態として GRP\_FS\_ERR\_FS を返しています。

また、特定のデバイスのみ対象とする場合は、iDev を確認することで可能となります。

```
#include "fat.h"

#define    MAX_OFFSET    0x100000    /* 1MB */

int  gr_file_hook(int iDev,  grp_uint32_t  uiFid,  grp_uint32_t  uiOffset)
{
    int  iReturnCode;

    iReturnCode = 0;
    if(MAX_OFFSET < uiOffset) {          /* 1MB 以上か */
        iReturnCode = GRP_FS_ERR_FS; /* 1MB 以上のディレクトリオフセットは許さない */
    }

    return(iReturnCode);
}

int  main(・・・)
{
    /* 初期化处理 */
    ・・・

    grp_fs_init();
    fat_interrupt_lookup = gr_file_hook; /* フック関数を登録 */

    /* ファイル処理 */
    ・・・
}
```

## 4.1.1 ファイルシステム依存関数向けの GR-FILE 関数

**GR-FILE** では、ファイルシステム依存関数向けに、いくつかの関数を提供しています。表 4-16 に、ファイルシステム依存関数向けの **GR-FILE** 関数の一覧を示します。

表 4-16 ファイルシステム依存関数向けの GR-FILE 関数

#	関数名	機能
1	grp_fs_block_buf_mod grp_fs_unblock_buf_mod	・キャッシュバッファのロック取得/解除
2	grp_fs_block_file_op grp_fs_unblock_file_op	・ファイル操作のロック取得/解除
3	grp_fs_block_file_op_by_id grp_fs_unblock_file_op_by_id	・ファイル ID によるファイル操作のロック取得/解除
4	grp_fs_block_fs_mod grp_fs_unblock_file_op	・ファイルシステム操作のロック取得/解除
5	grp_fs_buf_fill_end	・キャッシュバッファのデータ読み込み終了通知
6	grp_fs_change_fid	・ファイル ID の変更
7	grp_fs_check_dev_busy	・デバイスのマウント中チェック
8	grp_fs_check_io_status	・I/O ready/再実行チェック
9	grp_fs_check_mnt_dev	・マウント済みデバイスの検索
10	grp_fs_close_file	・ファイルのクローズ
11	grp_fs_exec_dev_io	・デバイス I/O の実行
12	grp_fs_file_open_common	・ファイルのオープン
13	grp_fs_get_mount_root_attr	・マウントされたルートディレクトリの属性情報の取得
14	grp_fs_get_path_comp	・パスコンポーネントの取得
15	grp_fs_lookup_buf	・キャッシュバッファの取得
16	grp_fs_lookup_file_ctl	・ファイル管理情報の取得
17	grp_fs_lookup_fname_cache	・ファイル名称キャッシュの検索/削除
18	grp_fs_make_sname_another_method	・ショート名の生成で重複が続いた場合にベース名を変更する関数
19	grp_fs_purge_fname_cache_by_dev	・ファイル名称キャッシュのデバイス単位の削除
20	grp_fs_read_buf	・キャッシュバッファへの読み込み
21	grp_fs_set_access_time	・アクセスタイム情報の設定
22	grp_fs_set_fname_cache	・ファイル名称キャッシュの設定
23	grp_fs_unref_buf	・キャッシュバッファへの参照解放
24	grp_fs_wait_io	・キャッシュバッファの反映待ち
25	grp_fs_write_buf	・キャッシュバッファの書き込み

以下、上記関数のインタフェースを説明します。

## 4.1 1.1 grp\_fs\_block\_buf\_mod/grp\_fs\_unblock\_buf\_mod

【機能概要】 キャッシュバッファのロック取得/解除

【関数形式】

```
#include "grp_fs.h"

void  grp_fs_block_buf_mod (grp_fs_bio_t  *ptBio);
void  grp_fs_unblock_buf_mod (grp_fs_bio_t  *ptBio,  grp_ushort_t  usModStat);
```

【パラメータ】

ptBio	入力	ブロック I/O 情報へのポインタ
	出力	ptBio->ptBuf->usStatus フィールドのステータス情報を変更
usModStat	入力	更新状態/反映モード指示 (以下のビットを or で指定)
		0 データ更新なし
		GRP_FS_BSTAT_DIRTY データ更新
		GRP_FS_BSTAT_TSYNC メディアへの即反映要

【機能詳細】

キャッシュバッファのロックの取得/解放を行います。**GR-FILE** では、キャッシュバッファの更新を行う際には、必ず `grp_fs_block_buf_mod` をコールし、更新処理後、`grp_fs_unblock_buf_mod` をコールすることで、キャッシュバッファの更新処理の排他制御を行います。すでに、あるタスクが `grp_fs_block_buf_mod` でロックを確保している状態で、他のタスクで `grp_fs_block_buf_mod` をコールしますと、前のタスクが `grp_fs_unblock_buf_mod` を実行するまで、後のタスクは待ち状態になります。

ロック対象のキャッシュバッファは `ptBio->ptBuf` フィールドで指定します。また、ロック解放時には、`usStatus` パラメータで更新状態やメディアへの反映指示を行います。データを更新しなかった場合は、`usStatus` パラメータに 0 を指定します。キャッシュバッファのデータを更新した場合は、`GRP_FS_BSTAT_DIRTY` ビットを指定して実行します。さらに、更新情報のメディアへの反映を直ぐに行う場合は、`GRP_FS_BSTAT_TSYNC` ビットを指定して実行します。

【リターン値】

なし

## 4.1 1.2 grp\_fs\_block\_file\_op/grp\_fs\_unblock\_file\_op

【機能概要】 ファイル操作のロック取得/解除

【関数形式】

```
#include "grp_fs.h"
void  grp_fs_block_file_op (grp_fs_file_t  *ptFile);
void  grp_fs_unblock_file_op (grp_fs_file_t  *ptFile);
```

【パラメータ】

ptFile	入力	ファイル管理情報へのポインタ
	出力	ptFile-> usStatus フィールドのステータス情報を変更

【機能詳細】

各ファイル操作に対するロックの取得/解放を行います。**GR-FILE** では、各ファイルに対して read や write 等の操作を行う際には、必ず **grp\_fs\_block\_file\_op** をコールし、操作終了後、**grp\_fs\_unblock\_file\_op** をコールすることで、各ファイル操作の排他制御を行います。すでに、あるタスクが **grp\_fs\_block\_file\_op** でロックを確保している状況で、他のタスクで **grp\_fs\_block\_file\_op** をコールしますと、前のタスクが **grp\_fs\_unblock\_file\_op** を実行するまで、後のタスクは待ち状態になります。

【リターン値】

なし

## 4.1 1.3 grp\_fs\_block\_file\_op\_by\_id/grp\_fs\_unblock\_file\_op\_by\_id

【機能概要】 ファイル ID によるファイル操作のロック取得/解除

【関数形式】

```
#include "grp_fs.h"
grp_fs_file_t *grp_fs_block_file_op_by_id (grp_fs_info_t *ptFs, grp_uint32_t uiFid);
void grp_fs_unblock_file_op_by_id (grp_fs_info_t *ptFs);
```

【パラメータ】

ptFs	入力	ファイルシステム管理情報へのポインタ
uiFid	入力	ロック対象のファイル ID
	出力	<対応するファイル管理情報が存在する場合> 同管理情報の usStatus フィールドを変更 <対応するファイル管理情報が存在しない場合> ptFs->uiFsBusyFid を uiFid に設定 ptFs->usStatus フィールドを変更

【機能詳細】

grp\_fs\_block\_file\_op/grp\_fs\_unblock\_file\_op と同様に、各ファイル操作に対するロックの取得/解放を行います。但し、ファイル管理情報ではなく、ファイルシステム管理情報と、ファイル ID を指定して行ないます。本関数は、ファイルシステム管理情報が得られていない状態で、ファイル ID だけがわかっている場合に使用します。

grp\_fs\_block\_file\_op\_by\_id は、ptFs、uiFid で指定したファイルに対応するファイル管理情報が既に存在する場合、同管理情報に操作ロックを設定し、同管理情報のポインタを返します。対応するファイル管理情報が存在しない場合は、同管理情報を生成せず、ptFs で指定したファイルシステム管理情報に、操作ロック対象のファイル ID を記憶して、操作ロックを設定します。後者の場合、リターン値としては、NULL を返します。

grp\_fs\_block\_file\_op\_by\_id で設定した操作ロックの解除は、同関数から返ってきた値により、コールする関数を変えて行なう必要があります。NULL でないファイル管理情報ポインタが返ってきた場合は、grp\_fs\_close\_file を GRP\_FS\_FILE\_UNBLOCK オプション付きでコールすることで行います。grp\_fs\_block\_file\_op\_by\_id で NULL が返ってきた場合は、grp\_fs\_unblock\_file\_op\_by\_id をコールして行います。

【リターン値】

NULL	対応するファイル管理情報がなく、ファイルシステム管理情報にファイル ID を記憶して操作ロックを設定した
NULL 以外	操作ロックを設定したファイル管理情報へのポインタ

## 4.1 1.4 grp\_fs\_block\_fs\_mod/grp\_fs\_unblock\_fs\_mod

【機能概要】 ファイルシステム操作のロック取得/解除

【関数形式】

```
#include "grp_fs.h"
void grp_fs_block_fs_mod (grp_fs_info_t *ptFs);
void grp_fs_unblock_fs_mod (grp_fs_info_t *ptFs);
```

【パラメータ】

ptFs	入力	ファイルシステム管理情報へのポインタ
	出力	ptFs-> usStatus フィールドのステータス情報を変更

【機能詳細】

各ファイルシステム操作に対するロックの取得/解放を行います。**GR-FILE** では、ファイルシステムのフリーブロックキャッシュの変更等、ファイルシステム内の共通なリソースを変更する際には、必ず `grp_fs_block_fs_mod` をコールし、操作終了後、`grp_fs_unblock_fs_mod` をコールすることで、排他制御を行います。すでに、あるタスクが `grp_fs_block_fs_mod` でロックを確保している状態で、他のタスクで `grp_fs_block_fs_mod` をコールしますと、前のタスクが `grp_fs_unblock_fs_mod` を実行するまで、後のタスクは待ち状態になります。

【リターン値】

なし



## 4.1 1.5 grp\_fs\_buf\_fill\_end

【機能概要】 キャッシュバッファの読み込み終了通知

【関数形式】

```
#include "grp_fs.h"
void grp_fs_buf_fill_end(grp_fs_bio_t *ptBio, int iInvalidate);
```

【パラメータ】

ptBio	入力	ブロック I/O 情報へのポインタ
	出力	ptBio->ptBuf->usStatus フィールドのステータス情報を変更
iInvalidate	入力	バッファの無効化指示 (0 以外の場合)

【機能詳細】

キャッシュバッファへのデータの読み込みの終了を通知し、同バッファのデータ待ちをしているタスクを起こし、実行状態にします。

通知対象のキャッシュバッファは ptBio->ptBuf フィールドで指定します。また、キャッシュバッファへのデータの読み込みが失敗した場合は、iInvalidate パラメータに 0 以外を指定し、同バッファが無効であることを **GR-FILE** に伝えて、同バッファを解放します。データの読み込みが成功した場合は、iInvalidate パラメータに 0 を指定します。

【リターン値】

なし

## 4.1 1.6 grp\_fs\_change\_fid

【機能概要】 ファイル ID の変更

【関数形式】

```
#include "grp_fs.h"
void grp_fs_change_fid(grp_fs_file_t *ptFile, grp_uint32_t uiFid);
```

【パラメータ】

ptFile	入力	ファイル管理情報へのポインタ
uiFid	入力	新しいファイル ID
	出力	ptFile->uiFid : uiFid を設定 ptFile->ptHashFwd/Bwd : ハッシュリストを変更 ptFile->pptHashTop : ハッシュトップを変更

【機能詳細】

ptFile で指定したファイルのファイル ID を変更し、ハッシュリストを更新します。

【リターン値】

なし

## 4.1 1.7 grp\_fs\_check\_dev\_busy

【機能概要】 デバイスのマウント中チェック

【関数形式】

```
#include "grp_fs.h"
int grp_fs_check_dev_busy(int iDev);
```

【パラメータ】

iDev	入力	チェックするデバイス番号
------	----	--------------

【機能詳細】

iDev で指定したデバイス/メディアが既にマウント中であるかどうかをチェックします。パーティションレスのデバイス番号を指定した場合は、同一サブデバイスもチェック対象とします。

【リターン値】

0	マウントされていない
1	マウント中である

## 4.1 1.8 grp\_fs\_check\_io\_status

【機能概要】 I/O ready/再度実行チェック

【関数形式】

```
#include "grp_fs.h"
int *grp_fs_check_io_status(
    grp_fs_info_t *ptFs, grp_uint32_t uiDevBlk, grp_isize_t iCnt, int iMode);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
uiDevBlk	入力	対象のデバイスブロック番号
iCnt	入力	I/O カウント（デバイスブロック単位）
iMode	入力	I/O オペレーションの種別
GRP_FS_IO_READ		デバイス read 処理
GRP_FS_IO_WRITE		デバイス write 処理
GRP_FS_IO_REQ		デバイス write を伴うファイル I/O リクエスト
GRP_FS_IO_OP_ERR		実際にデバイス read/write エラー発生

【機能詳細】

本関数は、大きく 3 つのパターンで使用します。

1 つ目は、write が発生するアプリケーションインタフェース処理の開始時に、ptFs に対応したデバイスが I/O 抑止状態でないかどうかをチェックする際に使用します。この場合、I/O 対象部分の物理的なデバイスブロック番号や I/O カウント値をまだ計算していない状態のため、uiDevBlk、iCnt には 0 を設定し、iMode に GRP\_FS\_IO\_REQ を設定してコールします。本関数は、同要求に対し、ptFs に対応したデバイスが I/O 抑止状態かどうかチェックし、その結果をコール元に返します。I/O 抑止状態でない場合は 0 を、I/O 抑止状態の場合は GRP\_FS\_ERR\_IO を返します。なお、ファイルシステム依存処理部において、このタイプで本関数を利用する必要があるケースは、create、rename、unlink に対応した処理です。

2 つ目は、実際にデバイス I/O 処理を行う前に、同様に、ptFs に対応したデバイスが I/O 抑止状態でないかどうかをチェックする際に使用します。この場合、I/O 対象部分の物理的なデバイスブロック番号や I/O カウント値を uiDevBlk、iCnt に設定し、iMode に GRP\_FS\_IO\_READ または GRP\_FS\_IO\_WRITE を設定してコールします。本関数は、同要求に対し、ptFs に対応したデバイスが I/O 抑止状態かどうかチェックし、その結果をコール元に返します。I/O 抑止状態でない場合は 0 を、I/O 抑止状態の場合は GRP\_FS\_ERR\_IO を返します。なお、ファイルシステム非依存部でデバイス I/O 処理のラッパー関数を提供していますので、原則、ファイルシステム依存処理部において、このタイプで本関数を利用する必要があるケースありません。

3 つ目は、実際にデバイス I/O 処理を行った結果、I/O エラーであった場合に、プラットフォーム依存のエラー処理や再実行が必要かどうかをチェックするために使用します。この場合、エラーとなった I/O 対象部分の物理的なデバイスブロック番号や I/O カウント値を uiDevBlk、iCnt に設定し、iMode に GRP\_FS\_IO\_READ または GRP\_FS\_IO\_WRITE と共に、GRP\_FS\_IO\_OP\_ERR を設定してコールします。本関数は、同要求に対し、プラットフォーム依存の I/O エラー処理関数が

関数ポインタ変数 `grp_fs_inform_io_err` に設定されていれば、同エラー処理関数をコールし、プラットフォーム依存のエラー処理を実行して、同エラー処理関数からのリターン値を本関数のリターン値とします。I/O エラー処理関数が設定されていない場合は、`GRP_FS_ERR_IO` を返します。要求の I/O 処理の再実行が必要な場合は 0 を、要求の処理をエラーとして返す場合は、`GRP_FS_ERR_IO` 等 0 以外をリターン値として返します。

プラットフォーム依存のエラー処理関数は、例えば、利用者へのエラー状態の通知や、利用者からの処理方法の指示入力や、ロギング等を行います。また、I/O 処理を再実行するため、メディアの再挿入を待ったりする場合があります。

なお、ファイルシステム非依存部でデバイス I/O 処理のラッパー関数を提供していますので、原則、ファイルシステム依存処理部において、3 番目のタイプで本関数を利用する必要があるケースありません。

1 番目、および、2 番目のタイプにおいても、対応するデバイスが I/O 抑止状態の場合は、プラットフォーム依存の I/O エラー処理関数が関数ポインタ変数 `grp_fs_inform_io_err` に設定されていれば、同エラー処理関数をコールし、プラットフォーム依存のエラー処理を実行して、同エラー処理関数からのリターン値を本関数のリターン値とします。従いまして、例えば、同エラー処理関数を実行した時点では、I/O 抑止状態であっても、同エラー処理関数がメディアの再挿入等を待ち、I/O 抑止状態が解除されてから、同エラー処理関数が 0 でリターンするようなケースも考えられます。

#### 【リターン値】

0	要求の I/O 処理を実行可能、または、再実行要
0 以外 ( <code>GRP_FS_ERR_IO</code> 等)	要求の I/O 処理実行不可

## 4.1 1.9 grp\_fs\_check\_mnt\_dev

【機能概要】 マウント済みデバイスの検索

【関数形式】

```
#include "grp_fs.h"
grp_fs_info_t *grp_fs_check_mnt_dev(grp_fs_info_t *ptFs, int iDev);
```

【パラメータ】

ptFs	入力	マウント済みファイルシステム情報リストの先頭へのポインタ 通常、grp_fs_ctl->ptFsMnt を指定する。
iDev	入力	検索するデバイス番号

【機能詳細】

ptFs で指定したマウント済みのファイルシステム情報リストを検索し、iDev で指定したデバイス番号を持つファイルシステムのファイルシステム情報へのポインタを返します。

なお、ptFS には、通常、grp\_fs\_ctl->ptFsMnt を指定して検索します。

【リターン値】

NULL 以外	見つかったファイルシステム情報へのポインタ
NULL	指定のデバイス番号を持つマウント済みファイルシステムが見つからなかった

## 4.1 1.1 0 grp\_fs\_close\_file

【機能概要】 ファイルのクローズ

【関数形式】

```
#include "grp_fs.h"
void grp_fs_close_file(grp_fs_file_t *ptFile, int iMode);
```

【パラメータ】

ptFile	入力	クローズ対象のファイルのファイル管理情報へのポインタ
	出力	ptFile->iRefCnt を 1 減らし、0 になったら、クローズ処理実行。 また、0 になった場合、ptFile->ptFs->iFsOpen の 1 減らす。
iMode	入力	クローズモード（以下のビットを or で指定）
		GRP_FS_FILE_INVALID            管理情報を解放
		GRP_FS_FILE_UNBLOCK        grp_fs_unblock_file_op 実行

【機能詳細】

grp\_fs\_lookup\_file\_ctl 関数で取得したファイル管理情報へのポインタを ptFile に指定し、同ファイルのクローズ処理を行います。

本関数は、ptFile で指定したファイル管理情報の参照カウントフィールド iRefCnt を 1 つ減らし、0 になったらファイルシステム依存のクローズ関数を実行します。また、0 になった場合は、ファイルシステム内のオープン中のファイルカウントを表す ptFile->ptFs->iFsOpen を 1 減らし、キャッシュの反映方式に応じて、メディアへの反映処理を行います。

iMode パラメータは、クローズモードの指定で、GRP\_FS\_FILE\_INVALID ビットを設定して指定しますと、参照カウントが 0 になった場合、同管理情報をキャッシュとしては残さず解放します。また、GRP\_FS\_FILE\_UNBLOCK ビットを設定して指定しますと、grp\_fs\_unblock\_file\_op を実行し、ファイル操作のロックの解放も行います。

【リターン値】

なし

## 4.1 1.1 1 grp\_fs\_exec\_dev\_io

【機能概要】 デバイス I/O の実行

【関数形式】

```
#include "grp_fs.h"
grp_int32_t grp_fs_exec_dev_io (
    grp_fs_info_t    *ptFs,
    grp_uint32_t     uiBlk,
    grp_uchar_t      *pucBuf,
    grp_isize_t       iCnt,
    int               iOp
    int               iBlkKind);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
uiBlk	入力	キャッシュブロック番号
pucBuf	入力	書込むデータが格納されている領域のアドレス (iOp == 1 の場合)
	出力	読込んだデータを格納する領域のアドレス (iOp == 0 の場合)
iCnt	入力	読込むデータのデバイスブロック (セクタ) カウント
iOp	入力	I/O モード GRP_FS_IO_READ : read、GRP_FS_IO_WRITE : write
iBlkKind	入力	I/O ブロックの種別 GRP_FS_BUF_FILE: ファイル管理情報ブロック GRP_FS_BUF_DATA: データブロック

【機能詳細】

ptFs で指定したファイルシステムの iBlkKind で指定した種別のキャッシュブロック番号 uiBlk のデータを、対応するデバイスメディアから、あるいは、同デバイスメディアに I/O します。iOp に GRP\_FS\_IO\_WRITE ビットが立っている場合は、メディアへの書き込みを行い、その他の場合は、メディアからの読込みを行います。

読込んだデータを格納する領域のアドレス、あるいは、書込むデータの入った領域のアドレスは、pucBuf で指定します。I/O するサイズは、デバイスブロック (セクタ) 単位で、iCnt パラメータで指定します。

【リターン値】

0 または 正值	I/O できたデバイスブロック数
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	ファイルシステムのデバイス番号が正しくない



## 4.1 1.1 2 grp\_fs\_file\_open\_common

【機能概要】 ファイルのオープン

【関数形式】

```
#include "grp_fs.h"

int grp_fs_file_open_common (
    grp_fs_info_t      *ptFs,
    grp_fs_file_t      *ptDir,
    const grp_uchar_t  **ppucPath,
    int                 iMode,
    grp_uchar_t        *pucComp,
    grp_fs_file_t      **pptOpened);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
ptDir	入力	検索開始ディレクトリ (NULL の場合ルートからサーチ開始)
ppucPath	入力	*ppucPath : オープン対象ファイルのパス名
	出力	*ppucPath : 残りのパス名の先頭アドレス
iMode	入力	オープンモードの指定 (以下を bit or で指定)
GRP_FS_OPEN_EXEC		実行可能モードでオープン
GRP_FS_OPEN_READ		read 可能モードでオープン
GRP_FS_OPEN_WRITE		write 可能モードでオープン
GRP_FS_OPEN_APPEND		追加 write モードでオープン
		(常にファイルの終わりに seek して write 時)
GRP_FS_OPEN_PARENT		対象ファイルの親ディレクトリのオープン
pucComp	出力	GRP_FS_OPEN_PARENT の場合の最終ファイルコンポーネント名を格納する領域のアドレス。なお、本パラメータで指定した領域は、GRP_FS_OPEN_PARENT 以外でも、本関数内でのファイルサーチ処理に使用し、GRP_FS_MAX_COMP バイトの領域が確保されていることを前提とする。
pptOpened	出力	オープンした対象ファイル、または、親ディレクトリ、または、検索中に検出したファイルシステム跨りとなるディレクトリのファイル管理情報へのポインタを格納する領域のアドレス

【機能詳細】

本関数は、**GR-FILE** のファイルシステム抽象化インタフェースのオープン関数に対応した関数です。機能の詳細については、ファイルシステム抽象化インタフェースのオープン関数 **pfnOpen** フィールドの説明を参照下さい。

## 【リターン値】

0	オープン処理成功
GRP_FS_COMP_MIDDLE	パス名を辿る中でファイルシステムを跨り、処理を中断した
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_PERMIT	指定されたオープンモードを許可できない
GRP_FS_ERR_BAD_NAME	パス名に不正な文字コードが含まれている
GRP_FS_ERR_NOT_FOUND	指定したファイル、または、親ディレクトリが見つからない
GRP_FS_ERR_FS	ファイルシステムが正しくない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.1 1.1 3 grp\_fs\_get\_mount\_root\_attr

【機能概要】 マウントされたルートディレクトリの属性情報の取得

【関数形式】

```
#include "grp_fs.h"
int grp_fs_get_mount_root_attr (grp_fs_file_t *ptFile, grp_fs_dir_ent_t *ptDirEnt);
```

【パラメータ】

ptFile	入力	マウント先ディレクトリのファイルシステム情報へのポインタ
ptDirEnt	出力	マウントされたファイルシステムのルートディレクトリの属性情報を格納する領域のアドレス

【機能詳細】

ptFile で指定したマウント先ディレクトリにマウントされたファイルシステムのルートディレクトリの属性情報を取得し、ptDirEnt で指定した領域に格納します。

本関数は、grp\_fs\_get\_dirent に対応したファイルシステム依存関数において、対応するディレクトリエントリが、ファイルシステムがマウントされたディレクトリファイルの場合、実際にマウントされたファイルシステムのルートディレクトリのディレクトリエントリ情報を返す際に使用します。

【リターン値】

正值	取得したディレクトリエントリのサイズ
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_FS	指定したディレクトリにマウントされたファイルシステムが見つからない
GRP_FS_ERR_NOMEM	使用可能なキャッシュバッファがない (エラー番号の節の注参照)

## 4.1 1.1 4 grp\_fs\_get\_path\_comp

【機能概要】 パスコンポーネントの取得

【関数形式】

```
#include "grp_fs.h"

int grp_fs_get_path_comp(const grp_uchar_t **ppucPath,
                        grp_uchar_t *pucComp, int iCompBufSize);
```

【パラメータ】

ppucPath	入力	*ppucPath : パス名
	出力	*ppucPath : 残りのパス名の先頭アドレス
pucComp	出力	切出した先頭コンポーネントを格納する領域のアドレス
iCompBufSize	入力	上記格納領域のサイズ

【機能詳細】

\*ppucPath で指定したパス名から、次の ‘/’ または ‘\’ または NULL 文字の手前までの文字列（コンポーネント）を切出し、pucComp で指定した領域に NULL 終端した文字列として格納します。さらに、見つかった ‘/’ または ‘\’ 文字をスキップし、次のコンポーネントの先頭アドレスを \*ppucPath に設定します。

リターン値としては、コンポーネントの切出し後、さらに次のコンポーネントが残っている場合は、GRP\_FS\_COMP\_MIDDLE を、最終コンポーネントの場合は、GRP\_FS\_COMP\_LAST を、パス名の終端に既に達していて、切出すコンポーネントがない場合は、0 を返します。また、切出したコンポーネントの長さが、iCompBufSize - 1 を超える場合は、GRP\_FS\_ERR\_TOO\_LONG を、パス名に不正な文字が含まれる場合は、GRP\_FS\_ERR\_BAD\_NAME を返します。

【リターン値】

0	パス名の終端
GRP_FS_COMP_LAST	最終コンポーネント
GRP_FS_COMP_MIDDLE	中間コンポーネント
GRP_FS_ERR_TOO_LONG	切出したコンポーネントの長さが iCompBufSize - 1 を超えた
GRP_FS_ERR_BAD_NAME	パス名に不正な文字が含まれている

## 4.1 1.1 5 grp\_fs\_lookup\_buf

【機能概要】 キャッシュバッファの取得

【関数形式】

```
#include "grp_fs.h"
int grp_fs_lookup_buf (
    grp_fs_info_t    *ptFs,
    grp_uint32_t     uiBlk,
    int              iBufKind,
    grp_int32_t       iSize,
    grp_fs_bio_t      ptBio);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
uiBlk	入力	キャッシュブロック番号
iBufKind	入力	キャッシュバッファの種別、および、割当て要求かどうかの区別 GRP_FS_BUF_FILE      ファイル管理ブロックキャッシュ GRP_FS_BUF_DATA      ファイルデータキャッシュ GRP_FS_BUF_ALLOC    割当て要求（種別情報と一緒に指定可）
iSize	入力	データサイズ
ptBio	出力	取得したキャッシュバッファの情報を設定するブロック I/O 構造体のアドレス

【機能詳細】

メディアからの読出したファイル管理ブロック、または、ファイルデータを、あるブロック単位でキャッシュしたキャッシュバッファから、ptFs で指定したファイルシステムの iBufKind で指定した種別のキャッシュで、キャッシュブロック番号が uiBlk で、データサイズが iSize のキャッシュを検索し、検索結果を ptBio で指定した構造体に返します。見つかったキャッシュバッファは、参照カウント（grp\_fs\_buf\_t 構造体の iRefCnt フィールド）を 1 増やして ptBio の構造体に返します。

検索対象のキャッシュバッファが見つからず、かつ、iBufKind に GRP\_FS\_BUF\_ALLOC ビットを一緒に指定した場合は、フリーなキャッシュバッファを割当て、参照カウントを 1 に設定して、同キャッシュバッファの情報を ptBio で指定した構造体に返します。フリーなバッファがない場合は、現在の参照カウントが 0 のキャッシュバッファで最も最近アクセスされていない同一種別のバッファを 1 つ取出し、同バッファを再利用して割当てます。但し、同キャッシュバッファにメディアに未反映のデータが含まれる場合は、同データをメディアに反映してから、割当てます。

本関数により取得したキャッシュバッファは、処理終了後、grp\_fs\_unref\_buf 関数を用い、参照カウントを元に戻します。

## 【リターン値】

0	指定のキャッシュバッファが見つかった
GRP_FS_ERR_NOT_FOUND	指定のキャッシュバッファが見つからなかった。 但し、iBufKind に GRP_FS_BUF_ALLOC ビットを指定した場合は、フリーなバッファを割当てて、ptBio で指定した構造体に同情報を返します。
GRP_FS_ERR_NOMEM	すべてのキャッシュバッファを（I/O エラー等で）使いきって、フリーなバッファを割当てることができない
GRP_FS_ERR_TOO_BIG	指定したデータサイズがキャッシュバッファサイズを超えている
GRP_FS_ERR_FS	ファイルシステムが不正である

## 4.1 1.1 6 grp\_fs\_lookup\_file\_ctl

【機能概要】 ファイル管理情報の検索

【関数形式】

```
#include "grp_fs.h"

int grp_fs_lookup_file_ctl(grp_fs_info_t *ptFs, grp_uint32_t uiFid, int iAlloc,
                           grp_fs_file_t **pptFile);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
uiFid	入力	検索するファイル ID
iAlloc	入力	0 以外の場合、検索対象のファイル情報が見つからなかった場合、フリーなエントリを割当てて
pptFile	出力	見つかったファイル管理情報へのポインタを格納する領域のアドレス

【機能詳細】

オープン中のファイル管理情報キャッシュの中から、ptFs で指定したファイルシステムでファイル ID が uiFid のファイル管理情報を検索し、見つかったファイル管理情報へのポインタを pptFile で指定した領域に返します。その際、見つかったファイル管理情報の参照カウント (grp\_fs\_file\_t 構造体の iRefCnt フィールド) は 1 増やして返します。

指定したファイル管理情報が見つからなかった場合、iAlloc が 0 でなければ、フリーなファイル管理情報を割当て、必要な情報を設定し、参照カウントを 1 に設定して返します。フリーなファイル管理情報がない場合は、GRP\_FS\_ERR\_NOMEM をリターン値として返します。また、iAlloc が 0 の場合は、\*pptFile に NULL を設定して返します。

\*pptFile に NULL 以外を設定して返した場合は、ptFs で指定したファイルシステムのオープン中ファイルカウントである ptFs->iFsOpen フィールドも 1 増やします。

なお、本関数で取得したファイル管理情報は、同ファイルに対する処理終了後、grp\_fs\_close\_file 関数を使って、参照カウントを減らします。

【リターン値】

0	指定のファイル管理情報が見つかった
-1	指定のファイル管理情報が見つからなかった。 但し、iAlloc が 0 でなければ、フリーなエントリを割当てて、同情報へのポインタを*pptFile に設定して返します。

## 4.1 1.1 7 grp\_fs\_lookup\_fname\_cache

【機能概要】 ファイル名称キャッシュの検索／削除

【関数形式】

```
#include "grp_fs.h"
int grp_fs_lookup_fname_cache(grp_fs_file_t *ptDir, const grp_uchar_t *pucName,
                             int iPurge, grp_fs_file_t **pptFile);
```

【パラメータ】

ptDir	入力	親ディレクトリのファイル管理情報へのポインタ
pucName	入力	検索するファイル名称（パスコンポネント名称）
iPurge	入力	キャッシュが見つかった場合、本パラメータが 0 以外であれば 見つかったキャッシュエントリを削除する
pptFile	出力	見つかったキャッシュに格納された対象ファイルのファイル管理 情報へのポインタを格納する領域のアドレス

【機能詳細】

ファイル名称キャッシュを検索し、親ディレクトリ情報とファイル名称が、それぞれ、ptDir、pucName にマッチする見つけます。指定のキャッシュが見つかった場合は、同キャッシュエントリに格納されている対象ファイルのファイル管理情報へのポインタを pptFile で示された領域に設定して返します。さらに、iPurge パラメータが 0 以外の場合、見つかったキャッシュエントリを削除します。

【リターン値】

0	指定のキャッシュが見つかった
GRP_FS_ERR_NOT_FOUND	指定のキャッシュが見つからなかった



## 4.1 1.1 8 grp\_fs\_make\_sname\_another\_method

【機能概要】 ショートファイル名のベース名生成

【関数形式】

```
#include "grp_fs_mdep_if.h"

int grp_fs_make_sname_another_method( const  grp_uchar_t  *pucLName,
                                       grp_uchar_t  *pucSName);
```

【パラメータ】

pucLName	入力	生成対象のロングファイル名が渡されます。
ptBio	出力	生成したショートファイル名のベース名を書き込みます。 ベース名なので 8 バイト文字列(NULL を入れると 9 バイト)で返します。

【機能詳細】

pucLName で指定されるロングファイル名から、ショートファイル名を生成します。

通常、ロングファイル名付きショートファイル名の生成は、ロングファイル名の先頭から使用できる文字を使い、必要があれば「~数字」を付加します。

似たようなロングファイル名が多数存在する場合、同名のショートファイル名が生成され、「~数字」の部分がインクリメントされます。ショートファイル名は重複して存在する事が許されていない為、「~数字」を付加した後のショートファイル名が重複していないか確認する必要があります。この確認処理はファイルが多数存在するほど多くの時間とアクセスを必要とします。

本関数は、通常のショートファイル名生成で重複回数が「GRP\_FS\_MAKE\_SNAME\_THRESHOLD」で定義される回数を超えた場合に一度だけ呼び出されます。呼び出された後、「~数字」の番号は 1 に再初期化され、本関数で生成したショートファイル名のベース名と「~数字」の組合せで再度重複確認が行われます。

本関数で生成されるショートファイル名のベース名を調整する事で、重複が少ないショートファイル名を生成し、重複確認に要する時間とアクセスを減らす事が出来ます。本関数の最適なインプリメントは使用されるロングファイル名のパターンにより異なる為、本関数はサンプル動作を行うソースが各依存ソースファイルに存在します。使用されるロングファイル名より最適な生成方法を実装して下さい。

本関数は、コンパイルスイッチ「GRP\_FS\_FAST\_MAKE\_SNAME」が有効な場合に呼び出されます。

【リターン値】

0	ショートファイル名のベース名の生成が成功
-1	ショートファイル名のベース名の生成が失敗

## 4.1 1.1 9 grp\_fs\_purge\_fname\_cache\_by\_dev

【機能概要】 ファイル名称キャッシュのデバイス単位の削除

【関数形式】

```
#include "grp_fs.h"
void grp_fs_purge_fname_cache_by_dev(int iDev);
```

【パラメータ】

iDev	入力	削除対象デバイスのデバイス番号
------	----	-----------------

【機能詳細】

ファイル名称キャッシュを検索し、iDev で指定したデバイス上のキャッシュであれば、同エントリを削除します。

【リターン値】

なし

## 4.1 1.2 0 grp\_fs\_read\_buf

【機能概要】 キャッシュバッファへの読み込み

【関数形式】

```
#include "grp_fs.h"

grp_int32_t grp_fs_read_buf(
    grp_fs_info_t    *ptFs,
    grp_uint32_t     uiBlk,
    int               iBufKind,
    grp_int32_t       iSize,
    grp_fs_bio_t      *ptBio);
```

【パラメータ】

ptFs	入力	ファイルシステム情報へのポインタ
uiBlk	入力	キャッシュブロック番号
iBufKind	入力	キャッシュバッファの種別 GRP_FS_BUF_FILE      ファイル管理ブロックキャッシュ GRP_FS_BUF_DATA      ファイルデータキャッシュ
iSize	入力	読み込むデータのサイズ
ptBio	出力	読み込んだデータを保持したキャッシュバッファの情報を格納する ブロック I/O 構造体のアドレス

【機能詳細】

キャッシュバッファを検索し、ptFs で指定したファイルシステムの iBufKind で指定した種別のキャッシュで、キャッシュブロック番号が uiBlk で、データサイズが iSize のキャッシュがあれば、同キャッシュの情報を ptBio で指定した構造体に設定して返します。

見つからなかった場合は、フリーなキャッシュバッファを割当て、同バッファにメディアから対象のデータを読み込んだ後、同キャッシュの情報を ptBio で指定した構造体に設定して返します。

キャッシュの検索には、grp\_fs\_lookup\_buf 関数を使用しています。従いまして、見つかったキャッシュバッファの参照カウント (grp\_fs\_buf\_t 構造体の iRefCnt フィールド) は 1 増えて返ってきます。また、フリーなバッファを割当てるとき、割当てたバッファがメディアへの未反映のデータが含まれていた場合は、同データをメディアに反映してから、割当てます。

本関数により取得したキャッシュバッファは、処理終了後、grp\_fs\_unref\_buf 関数を用い、参照カウントを元に戻します。

【リターン値】

0 または 正值	キャッシュバッファに読み込まれているデータのバイト数
GRP_FS_ERR_NOMEM	フリーなバッファを割当てることができない
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	ファイルシステムのデバイス番号が正しくない

## 4.1 1.2 1 grp\_fs\_set\_access\_time

【機能概要】 アクセスタイム情報の設定

【関数形式】

```
#include "grp_fs.h"
void grp_fs_set_access_time(grp_fs_file_t *ptFile);
```

【パラメータ】

ptFile	入力	ファイル管理情報へのポインタ ptFile->iATime フィールド
	出力	ptFile->iATime フィールド、 ptFile->usStatus フィールド (GRP_FS_FSTAT_UPD_ETIME)

【機能詳細】

ptFile で指定したファイル管理情報のアクセスタイム情報 (iATime フィールド) に現在時刻を設定します。さらに、同アクセスタイム情報のメディアへの反映が必要であることを示す GRP\_FS\_FSTAT\_UPD\_ETIME ビットを usStatus フィールドに設定します。

なお、ptFile->usStatus に GRP\_FS\_STAT\_DAY\_ACCTIME ビットが設定されている場合は、設定前のアクセスタイムと、設定した現在のアクセスタイムが同一日付でない場合のみ、GRP\_FS\_STAT\_UPD\_ETIME ビットを usStatus フィールドに設定します。この GRP\_FS\_STAT\_DAY\_ACCTIME ビットは、FAT ファイルシステム等、アクセス時刻情報が日付単位の情報しか持たないファイルシステムで設定します。

【リターン値】

なし

## 4.1 1.2 2 grp\_fs\_set\_fname\_cache

【機能概要】 ファイル名称キャッシュの設定

【関数形式】

```
#include "grp_fs.h"
grp_fs_fname_cache_t * grp_fs_set_fname_cache(grp_fs_file_t *ptDir,
                                              const grp_uchar_t *pucName, grp_fs_file_t *ptFile,
                                              grp_fs_fname_cache_t *ptAlias);
```

【パラメータ】

ptDir	入力	親ディレクトリのファイル管理情報へのポインタ
pucName	入力	ファイル名称 (パスコンポネント名称)
ptFile	入力	キャッシュ対象ファイルのファイル管理情報へのポインタ
ptAlias	入力	別名のキャッシュを持つ場合の、同キャッシュエントリへのポインタ

【機能詳細】

ptDir、pucName で指定された親ディレクトリ情報とファイル名称を持つキャッシュエントリを生成し、同エントリのファイル管理情報として ptFile で指定した値を設定します。既に、存在する場合は、同エントリのファイル管理情報を更新します。

また、ptAlias で指定した別名キャッシュ情報を同エントリに記憶します。

【リターン値】

NULL	メモリ不足等でキャッシュエントリの生成に失敗した
NULL 以外	生成/更新したキャッシュエントリのアドレス

## 4.1 1.2 3 grp\_fs\_unref\_buf

【機能概要】 キャッシュバッファへの参照解放

【関数形式】

```
#include "grp_fs.h"
void grp_fs_unref_buf (grp_fs_bio_t *ptBio);
```

【パラメータ】

ptBio	入力	grp_fs_lookup_buf または grp_fs_read_buf で得られたブロック I/O 構造体のアドレス
-------	----	--

【機能詳細】

grp\_fs\_lookup\_buf または grp\_fs\_read\_buf で得られたキャッシュバッファの参照カウントを 1 つ減らし、同キャッシュバッファを解放します。

なお、参照カウントが 0 になっても、同バッファの iDev フィールドが正値であれば、キャッシュとして保持し続けます。また、参照カウントが 0 になった場合、write-through 方式のキャッシュの場合、未反映のデータは、書戻しを行います。さらに、grp\_fs\_block\_buf\_mod でロックがかかっていた場合、grp\_fs\_unblock\_buf\_mod をコールし、ロックの解放も行います。

【リターン値】

なし

## 4.1 1.2 4 grp\_fs\_wait\_io

【機能概要】 キャッシュバッファの反映待ち

【関数形式】

```
#include "grp_fs.h"
int grp_fs_wait_io (int iDev, int iMode);
```

【パラメータ】

iDev	入力	キャッシュバッファの反映待ちを行うデバイスのデバイス番号
iMode	入力	反映待ちモードの指定
		GRP_FS_BUF_WAIT_INV 反映待ち後、バッファを解放
		GRP_FS_BUF_FORCE_INV 反映失敗の場合も無視して解放

【機能詳細】

iDev で指定したデバイスメディアのすべての未反映キャッシュデータを書戻し、その終了を待ちます。iMode に GRP\_FS\_BUF\_WAIT\_INV を指定した場合は、反映後、同バッファをキャッシュからはずし、フリーな状態として解放します。また、GRP\_FS\_BUF\_FORCE\_INV を指定した場合は、反映を失敗したバッファについても、エラーを無視してフリーな状態として解放します。iMode に 0 を指定した場合は、反映後も有効なキャッシュとして保持し続けます。

【リターン値】

0	反映成功
GRP_FS_ERR_IO	I/O エラー

## 4.1 1.2 5 grp\_fs\_write\_buf

【機能概要】 キャッシュバッファへの書戻し

【関数形式】

```
#include "grp_fs.h"
grp_int32_t  grp_fs_write_buf (grp_fs_bio_t  *ptBio);
```

【パラメータ】

ptBio	入力	書戻すキャッシュバッファ情報を保持したブロック I/O 構造体のアドレス (grp_fs_lookup_buf または grp_fs_read_buf で得たもの)
-------	----	---

【機能詳細】

ptBio で指定したキャッシュバッファにキャッシュされたデータを対応するデバイスメディアに書戻します。

ptBio で指定する構造体は、grp\_fs\_lookup\_buf または grp\_fs\_read\_buf で得られたものを指定します。実際には、grp\_fs\_lookup\_buf または grp\_fs\_read\_buf で得たキャッシュバッファに対し、grp\_fs\_block\_buf\_mod、grp\_fs\_unblock\_buf\_mod を使って、ロックをかけて同キャッシュバッファのデータを更新した後、強制的に同更新情報をメディアに反映したい場合に、本関数をコールします。

なお本関数は、対象のキャッシュバッファのロックが解除されていることを前提とします。

【リターン値】

0 または 正值	メディアに書込んだバイト数
GRP_FS_ERR_IO	I/O エラー
GRP_FS_ERR_BAD_DEV	ファイルシステムのデバイス番号が正しくない



## 4.1.2 デバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数

**GR-FILE** では、**GR-FILE** 向けのデバイスドライバや、OS/プラットフォーム依存関数の実装をサポートするため、いくつかの関数を提供しています。表 4-17 に、**GR-FILE** で提供するデバイスドライバ、OS/プラットフォーム依存関数向けサポートライブラリ関数の一覧を示します。

表 4-17 OS/プラットフォーム依存関数向けサポートライブラリ関数一覧

#	関数名	機能
1	grp_char_sjis_cnt	・ シフト JIS 文字のバイトカウント取得
2	grp_char_sjis_to_unicode	・ シフト JIS→UNICODE 変換
3	grp_char_unicode_to_sjis	・ UNICODE→シフト JIS 変換
4	grp_fs_get_part	・ パーティション情報の取得
5	grp_mem_vl_init / grp_mem_init_vl_pool	・ 可変長メモリプールの初期化
6	grp_mem_vl_add / grp_mem_add_vl_pool	・ 可変長メモリプールへのメモリの追加
7	grp_mem_vl_alloc / grp_mem_alloc_from_vl_pool	・ 可変長プールからのメモリの確保
8	grp_mem_vl_free / grp_mem_free_to_vl_pool	・ 可変長プールへのメモリの解放
9	grp_time_localtime	・ トータル秒→日付・時刻変換
10	grp_time_mktime	・ 日付・時刻→トータル秒変換
11	grp_time_set_base_year	・ トータル時刻秒のベース年の設定
12	grp_time_set_time_diff	・ 時差の設定・変更
13	grp_time_get_config	・ ベース年・時差情報の取得

以下、上記関数のインタフェースを説明します。

## 4.1 2.1 grp\_char\_sjis\_cnt

【機能概要】 シフト JIS 文字のバイトカウント取得

【関数形式】

```
#include "grp_char_conv.h"
int grp_char_sjis_cnt (const grp_uchar_t *pucStr);
```

【パラメータ】

pucStr	入力	バイトカウント取得対象の文字列のアドレス
--------	----	----------------------

【機能詳細】

pucStr で指定された文字列の先頭文字をシフト JIS コード系の文字と解釈し、バイト数を返します。例えば、シフト JIS の漢字文字は 2、ASCII 文字は 1、半角カナは 1、NULL 文字は 0、不正な文字コードは -1 を返します。

本関数は、**GR-FILE** の OS 抽象化インタフェース grp\_fs\_char\_cnt としてそのまま使用可能です。

【リターン値】

0	先頭文字が NULL 文字
正值	先頭文字のバイト数
- 1	不正な文字コード

## 4.12.2 grp\_char\_sjis\_to\_unicode

【機能概要】 シフト JIS→UNICODE 変換

【関数形式】

```
#include "grp_char_conv.h"
int grp_char_sjis_to_unicode (const grp_uchar_t *pucStr, grp_uint32_t *puiCode);
```

【パラメータ】

pucStr	入力	変換対象の文字列のアドレス
puiCode	出力	変換された UNICODE を格納する領域のアドレス

【機能詳細】

pucStr で指定された文字列の先頭文字をシフト JIS コード系の文字と解釈し、UNICODE に変換して、4 バイトの値として puiCode で指定された領域に格納します。

本関数は、**GR-FILE** の OS 抽象化インタフェース grp\_fs\_char\_to\_unicode としてそのまま使用可能です。

【リターン値】

0	先頭文字が NULL 文字
正值	先頭文字のバイト数
- 1	不正な文字コード

## 4.12.3 grp\_char\_unicode\_to\_sjis

【機能概要】 UNICODE→シフト JIS 変換

【関数形式】

```
#include "grp_char_conv.h"
```

```
int grp_char_unicode_to_sjis (grp_uchar_t *pucStr, grp_uint32_t uiCode);
```

【パラメータ】

pucStr	出力	変換後の文字列を格納する領域のアドレス
uiCode	入力	変換する UNICODE 値

【機能詳細】

uiCode で指定された UNICODE 値を、シフト JIS コードの形式に変換し、pucStr で指定された領域に格納します。

本関数は、**GR-FILE** の OS 抽象化インタフェース grp\_fs\_unicode\_to\_char としてそのまま使用可能です。

【リターン値】

0	UNICODE が NULL 文字
正值	変換後のバイト数
-1	不正な文字コード

## 4.1 2.4 grp\_fs\_get\_part

【機能概要】パーティション情報の取得

【関数形式】

```
#include "grp_fs_disk_part.h"
int grp_fs_get_part(grp_uchar_t *pucSecData, grp_fs_dk_part_t *ptPartInfo);
```

【パラメータ】

pucSecData	入力	先頭セクタのデータが格納されている領域のアドレス
ptPartInfo	出力	取得したパーティション情報を格納する領域のアドレス
		grp_fs_dk_part_t 構造体で GRP_FS_PART_CNT(4)個分の配列領域を指定する。

【機能詳細】

pucSecData で指定したメモリ領域に格納された先頭セクタのデータからパーティション情報を取出し、同パーティション情報を grp\_fs\_dk\_part\_t 構造体形式に変換して、ptPartInfo で指定した領域に格納します。ptPartInfo で指定した領域には、GRP\_FS\_PART\_CNT(4)個分のパーティション情報を、それぞれ grp\_fs\_dk\_part\_t 構造体の形で格納します。パーティション情報が見つかった場合は、GRP\_FS\_PART\_VALID を、パーティション情報が見つからず、pucSecData で指定された領域に FAT のブートパラメータブロックが直接入っている場合は、GRP\_FS\_PART\_LESS を、その他の場合は、GRP\_FS\_PART\_INVALID をリターン値として返します。

なお、pucSecData で指定した領域には、本関数を実行する前に、対象デバイスメディアからパーティション情報を含んだ先頭セクタのデータを読み込み、設定しておく必要があります。

ptPartInfo で指定した領域に格納される各パーティション情報の構造体 grp\_fs\_dk\_part\_t の形式は以下のとおりです。

```
typedef struct grp_fs_dk_part {
    grp_uchar_t    ucActive;        アクティブパーティションかどうかの区別
                                GRP_FS_PART_ACT    0x80 : アクティブ
                                GRP_FS_PART_NACT    0x00 : アクティブでない
    grp_uchar_t    ucPartType;      パーティションタイプ番号
    grp_fs_dk_chs_t tStartCHS;      開始アドレス (シリンダアドレス方式)
    grp_fs_dk_chs_t tEndCHS;        終了アドレス (シリンダアドレス方式)
    grp_uint32_t    uiStartSec;      開始アドレス (セクタアドレス方式)
    grp_uint32_t    uiEndSec;        終了アドレス (セクタアドレス方式)
} grp_fs_dk_part_t;
```

```
typedef struct grp_fs_dk_chs {          シリンダアドレス方式のアドレス情報
    grp_uchar_t      ucHead;           ヘッド番号
    grp_uchar_t      ucSec;            セクタ番号
    grp_ushort_t     usCyl;            シリンダー番号
} grp_s_dk_chs_t;
```

なお、`pcPartType` フィールドに設定されるパーティションタイプ番号用に、以下の `define` 定義を用意しています。

<code>GRP_FS_PART_NULL</code>	<code>0x00</code>	NULL パーティション
<code>GRP_FS_PART_FAT12</code>	<code>0x01</code>	FAT12
<code>GRP_FS_PART_FAT16_L32</code>	<code>0x04</code>	FAT16 < 32MB
<code>GRP_FS_PART_EXT</code>	<code>0x05</code>	拡張パーティション
<code>GRP_FS_PART_FAT16_H32</code>	<code>0x06</code>	FAT16 > 32MB
<code>GRP_FS_PART_NTFS</code>	<code>0x07</code>	NT ファイルシステム
<code>GRP_FS_PART_FAT32_CHS</code>	<code>0x0b</code>	FAT32 シリンダアドレス方式
<code>GRP_FS_PART_FAT32_LBA</code>	<code>0x0c</code>	FAT32 セクタアドレス方式
<code>GRP_FS_PART_FAT16_LBA</code>	<code>0x0e</code>	FAT16 セクタアドレス方式
<code>GRP_FS_PART_EXT_LBA</code>	<code>0x0f</code>	拡張パーティション (セクタアドレス方式)
<code>GRP_FS_PART_FAT32_HCHS</code>	<code>0x1b</code>	隠しパーティション FAT32 シリンダアドレス方式
<code>GRP_FS_PART_FAT32_HLBA</code>	<code>0x1c</code>	隠しパーティション FAT32 セクタアドレス方式
<code>GRP_FS_PART_FAT16_HLBA</code>	<code>0x1e</code>	隠しパーティション FAT16 セクタアドレス方式
<code>GRP_FS_PART_LINUX_SW</code>	<code>0x82</code>	LINUX スワップ/Solaris
<code>GRP_FS_PART_LINUX</code>	<code>0x83</code>	LINUX パーティション
<code>GRP_FS_PART_LINUX_EXT</code>	<code>0x85</code>	LINUX 拡張パーティション
<code>GRP_FS_PART_FREE_BSD</code>	<code>0xa5</code>	FreeBSD パーティション

本関数は、**GR-FILE** 用のデバイスドライバ等で使用します。

#### 【リターン値】

<code>GRP_FS_PART_VALID</code>	パーティション情報取得成功
<code>GRP_FS_PART_LESS</code>	パーティションがなく、直接 FAT のブートパラメータブロックで始まっている
<code>GRP_FS_PART_INVALID</code>	パーティション情報が見つからない

## 4.12.5 grp\_mem\_vl\_init / grp\_mem\_init\_vl\_pool

【機能概要】 可変長メモリプール初期化

【関数形式】

```
#include "grp_mem_vl_pool.h"
int  grp_mem_vl_init(char *pcPoolMem, grp_int32_t iPoolSize);
grp_mem_vl_ctl_t *grp_mem_init_vl_pool(char *pcPoolMem, grp_int32_t iPoolSize);
```

【パラメータ】

pcPoolMem	入力	分割管理するプール用メモリのアドレス
iPoolSize	入力	分割管理するプール用メモリのサイズ

【機能詳細】

**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないシステムで、**GR-FILE** の `grp_mem_alloc/grp_mem_free` 等の可変長メモリの取得/解放機能を実現するための関数群を提供しています。本関数は、その初期化処理関数で、分割管理するプール用のメモリのアドレスとサイズを `pcPoolMem`、`iPoolSize` パラメータで指定し、同領域の分割管理を可能とします。

なお、指定した領域上から、可変長領域を管理するための管理情報領域も確保するため、実際に可変長メモリとして利用可能なサイズは、指定したサイズ以下となります。

`grp_mem_init_vl_pool` は、可変長プールを区別して個別に初期化するための関数で、指定した領域を可変長プールとして管理するための管理情報のアドレスを返します。本関数で返ってきた管理情報を、`grp_mem_add_vl_pool`、`grp_mem_alloc_from_vl_pool`、`grp_mem_free_to_vl_pool` にパラメータとして指定することで、指定した可変長プールに対する処理が行えます。

`grp_mem_vl_init` は、デフォルトの可変長プールを初期化するための関数で、本初期化により、デフォルトの可変長プールに対する関数 `grp_mem_vl_add`、`grp_mem_vl_alloc`、`grp_mem_vl_free` が使用可能となります。これらの `grp_mem_vl_XXX` 関数は、実際には、上記、`grp_mem_XXX_vl_pool` を使ったマクロで定義されており、`grp_mem_vl_init` で、`grp_mem_init_vl_pool` から返ってきたアドレスを、デフォルトの可変長プールを管理する変数 `grp_mem_vl_ctl` に設定し、`grp_mem_vl_add`、`grp_mem_vl_alloc`、`grp_mem_vl_free` は、`grp_mem_vl_ctl` を使って処理する形となっています。

本関数に与えるメモリ領域は、環境/使用方法によっては、CPU の非キャッシュ領域に確保する必要があります。

【`grp_mem_init_vl_pool` のリターン値】

NULL	初期化失敗
NULL 以外	可変長プールの管理情報のアドレス

【`grp_mem_vl_init` のリターン値】

0	初期化成功
-1	初期化失敗

## 4.1 2.6 grp\_mem\_vl\_add / grp\_mem\_add\_vl\_pool

【機能概要】 可変長メモリプールへのメモリの追加

【関数形式】

```
#include "grp_mem_vl_pool.h"
int  grp_mem_vl_add(char *pcPoolMem, grp_int32_t iPoolSize);
int  grp_mem_add_vl_pool(
    grp_mem_vl_ctl_t *ptCtl, char *pcPoolMem, grp_int32_t iPoolSize);
```

【パラメータ】

ptCtl	入力	可変長メモリプール管理情報のアドレス
pcPoolMem	入力	分割管理するプール用メモリのアドレス
iPoolSize	入力	分割管理するプール用メモリのサイズ

【機能詳細】

**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないシステムで、**GR-FILE** の `grp_mem_alloc/grp_mem_free` 等の可変長メモリの取得/解放機能を実現するための関数群を提供しています。本関数は、分割管理した可変長メモリプールにメモリを追加する関数で、`pcPoolMem`、`iPoolSize` パラメータで指定した領域を、`ptCtl` パラメータで指定した可変長メモリプール、あるいは、デフォルトの可変長メモリプールに可変長メモリ領域として追加します。

なお、指定した領域上から、可変長領域を管理するための管理情報領域も確保するため、実際に可変長メモリとして利用可能なサイズは、指定したサイズ以下となります。

`grp_mem_add_vl_pool` は、特定の可変長メモリプールにメモリを追加する場合に使用する関数で、`ptCtl` パラメータに指定するアドレスは、`grp_mem_init_vl_pool` で返ってきた値を指定します。

`grp_mem_vl_add` は、デフォルトの可変長メモリプールにメモリを追加する場合に使用する関数で、本関数を呼ぶ前に `grp_mem_vl_init` を用いて初期化されていることが必要です。なお、`grp_mem_vl_add` は、`grp_mem_add_vl_pool` を使ったマクロ関数として定義されています。

【リターン値】

0	追加成功
- 1	追加失敗（サイズが小さすぎる）



## 4.12.7 grp\_mem\_vl\_alloc / grp\_mem\_alloc\_from\_vl\_pool

【機能概要】 可変長メモリプールからのメモリの確保

【関数形式】

```
#include "grp_mem_vl_pool.h"
void *grp_mem_vl_alloc (grp_int32_t iSize);
void *grp_mem_alloc_from_vl_pool(grp_mem_vl_ctl_t *ptCtl, grp_int32_t iSize);
```

【パラメータ】

ptCtl	入力	可変長メモリプール管理情報のアドレス
iSize	入力	確保する領域のサイズ

【機能詳細】

**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないシステムで、**GR-FILE** の `grp_mem_alloc`/`grp_mem_free` 等の可変長メモリの取得/解放機能を実現するための関数群を提供しています。本関数は、その `grp_mem_alloc` に対応した機能を提供します。

本関数は、`ptCtl` パラメータで指定した可変長メモリプール、あるいは、デフォルトの可変長メモリプールから、`iSize` パラメータで指定したサイズの領域を確保し、確保した領域の先頭アドレスを返します。返すアドレスは、“`grp_mem_vl_pool.h`” に定義された `GRP_MEM_VL_ALIGN` の倍数のアドレス境界を持つ値を返します。

`grp_mem_alloc_from_vl_pool` は、特定の可変長メモリプールからメモリを確保する場合に使用する関数で、`ptCtl` パラメータに指定するアドレスは、`grp_mem_init_vl_pool` で返ってきた値を指定します。

`grp_mem_vl_alloc` は、デフォルトの可変長メモリプールからメモリを確保する場合に使用する関数で、本関数を呼ぶ前に `grp_mem_vl_init` を用いて初期化されていることが必要です。なお、`grp_mem_vl_alloc` は、`grp_mem_alloc_from_vl_pool` を使ったマクロ関数として定義されています。

【リターン値】

NULL	指定したサイズの領域が確保できない
その他	確保した領域の先頭アドレス

## 4.12.8 grp\_mem\_vl\_free / grp\_mem\_free\_to\_vl\_pool

【機能概要】 可変長メモリプールへのメモリの解放

【関数形式】

```
#include "grp_mem_vl_pool.h"
int  grp_mem_vl_free (void  *pvMem);
int  grp_mem_free_to_vl_pool(grp_mem_vl_ctl_t  *ptCtl, void  *pvMem);
```

【パラメータ】

ptCtl	入力	可変長メモリプール管理情報のアドレス
pvMem	入力	解放する領域のアドレス

【機能詳細】

**GR-FILE** では、ある一定サイズ以上の大きな領域割当て機能しか持たないシステムで、**GR-FILE** の `grp_mem_alloc`/`grp_mem_free` 等の可変長メモリの取得/解放機能を実現するための関数群を提供しています。本関数は、その `grp_mem_free` に対応した機能を提供します。

本関数は、`pvMem` パラメータで指定した領域を解放し、`ptCtl` パラメータで指定した可変長メモリプール、あるいは、デフォルトの可変長メモリプールに戻します。`pvMem` パラメータで指定するアドレスは、`grp_mem_free_to_vl_pool` の場合は、`grp_mem_alloc_from_vl_pool` で返ってきたアドレスを、`grp_mem_vl_free` の場合は、`grp_mem_vl_alloc` で返ってきたアドレスを指定します。`ptCtl` パラメータに指定する可変長メモリプールの管理情報のアドレスは、`grp_mem_init_vl_pool` で返ってきたアドレスを指定します。

なお、`grp_mem_vl_free` は、`grp_mem_free_to_vl_pool` を使ったマクロ関数として定義されています。

【リターン値】

0	解放成功
-1	解放失敗（指定のアドレスが正しくない）

## 4.12.9 grp\_time\_localtime

【機能概要】 トータル秒→日付・時刻変換

【関数形式】

```
#include "grp_time_lib.h"
int grp_time_localtime (grp_int32_t iTime, grp_time_tm_t *ptTM);
```

【パラメータ】

iTime	入力	変換するトータル秒値
ptTM	出力	変換後の日付・時刻情報を格納する領域のアドレス

【機能詳細】

iTime で指定したグリニッジ標準時刻の 1970/1/1 からのトータル秒値を、現地時刻の日付、時刻情報に変換し、ptTM で指定した領域に格納します。ptTM で指定する領域の構造は、以下のとおりです。

```
typedef struct grp_time_tm {
    grp_uchar_t    ucSec;           秒 (0 - 59)
    grp_uchar_t    ucMin;           分 (0 - 59)
    grp_uchar_t    ucHour;          時 (0 - 23)
    grp_uchar_t    ucDay;           日 (1 - 31)
    grp_uchar_t    ucMon;           月 (1 - 12)
    grp_uchar_t    ucWday;          曜日 (0 - 6)    (0 : 日曜)
    short          sYear;           年
} grp_time_tm_t;
```

なお、現地時刻の標準時刻からの時差は、grp\_time\_set\_time\_diff で予め設定しておきます。但し、日本時間の場合は、デフォルトで設定してありますので、設定の必要はありません。

また、grp\_time\_set\_base\_year を使って、トータル秒のベースを任意の年に変更することが可能です。

【リターン値】

0	変換成功
-1	変換失敗

## 4.1 2.1 0 grp\_time\_mktime

【機能概要】 日付・時刻→トータル秒変換

【関数形式】

```
#include "grp_time_lib.h"
grp_int32_t  grp_time_mktime (grp_time_tm_t  *ptTM);
```

【パラメータ】

ptTM	入力	変換する日付・時刻情報の入った領域のアドレス
------	----	------------------------

【機能詳細】

ptTM で指定した領域に格納されている現地時刻の日付、時刻情報を、グリニッジ標準時刻の 1970/1/1 からのトータル秒値に変換し、その値を返します。ptTM で指定する領域の構造は、以下のとおりです。このうち、ucWday フィールドは設定不要です。

```
typedef struct  grp_time_tm  {
    grp_uchar_t    ucSec;           秒 (0 - 5 9)
    grp_uchar_t    ucMin;           分 (0 - 5 9)
    grp_uchar_t    ucHour;          時 (0 - 2 3)
    grp_uchar_t    ucDay;           日 (1 - 3 1)
    grp_uchar_t    ucMon;           月 (1 - 1 2)
    grp_uchar_t    ucWday;          曜日 (0 - 6)   設定不要
    short          sYear;           年
}  grp_time_tm_t;
```

なお、現地時刻の標準時刻からの時差は、grp\_time\_set\_time\_diff で予め設定しておきます。但し、日本時間の場合は、デフォルトで設定してありますので、設定の必要はありません。

また、grp\_time\_set\_base\_year を使って、トータル秒のベースを任意の年に変更することが可能です。

【リターン値】

- 1 以外	トータル秒値
- 1	変換失敗

## 4.1 2.1 1 grp\_time\_set\_base\_year

【機能概要】 トータル時刻秒のベース年の設定

【関数形式】

```
#include "grp_time_lib.h"
void grp_time_set_base_year (int iBaseYear);
```

【パラメータ】

iBaseYear	入力	トータル時刻秒のベース年
-----------	----	--------------

【機能詳細】

grp\_time\_localtime、grp\_time\_mktime のトータル時刻秒のベース年を変更します。  
デフォルトでは、1970 が設定されています。

【リターン値】

なし

## 4.1 2.1 2 grp\_time\_set\_time\_diff

【機能概要】 時差の設定・変更

【関数形式】

```
#include "grp_time_lib.h"
void grp_time_set_time_diff (grp_int32_t iTimeDiff);
```

【パラメータ】

iTimeDiff	入力	グリニッジ標準時刻からの時差 (秒)
-----------	----	--------------------

【機能詳細】

grp\_time\_localtime、grp\_time\_mktime のグリニッジ標準時刻からの時差を設定します。  
デフォルトでは、日本時刻（+ 9 時間）の時差が設定されています。

【リターン値】

なし

## 4.1 2.1 3 grp\_time\_get\_config

【機能概要】 ベース年・時差情報の取得

【関数形式】

```
#include "grp_time_lib.h"
void grp_time_get_config (int *piBaseYear, grp_int32_t *piTimeDiff);
```

【パラメータ】

piBaseYear	出力	取得したトータル時刻秒のベース年を格納するアドレス
piTimeDiff	出力	取得したグリニッジ標準時刻からの時差（秒）を格納するアドレス

【機能詳細】

grp\_time\_localtime、grp\_time\_mktime で使用するトータル時刻秒のベース年情報と、グリニッジ標準時刻からの時差情報を取得し、それぞれ、piBaseYear、piTimeDiff で指定した領域に返します。

【リターン値】

なし

## 5. サンプルフロー

$\mu$ ITRON OS 上で **GR-FILE** を用いたシステムの概略フロー例を図 5-1 に示します。

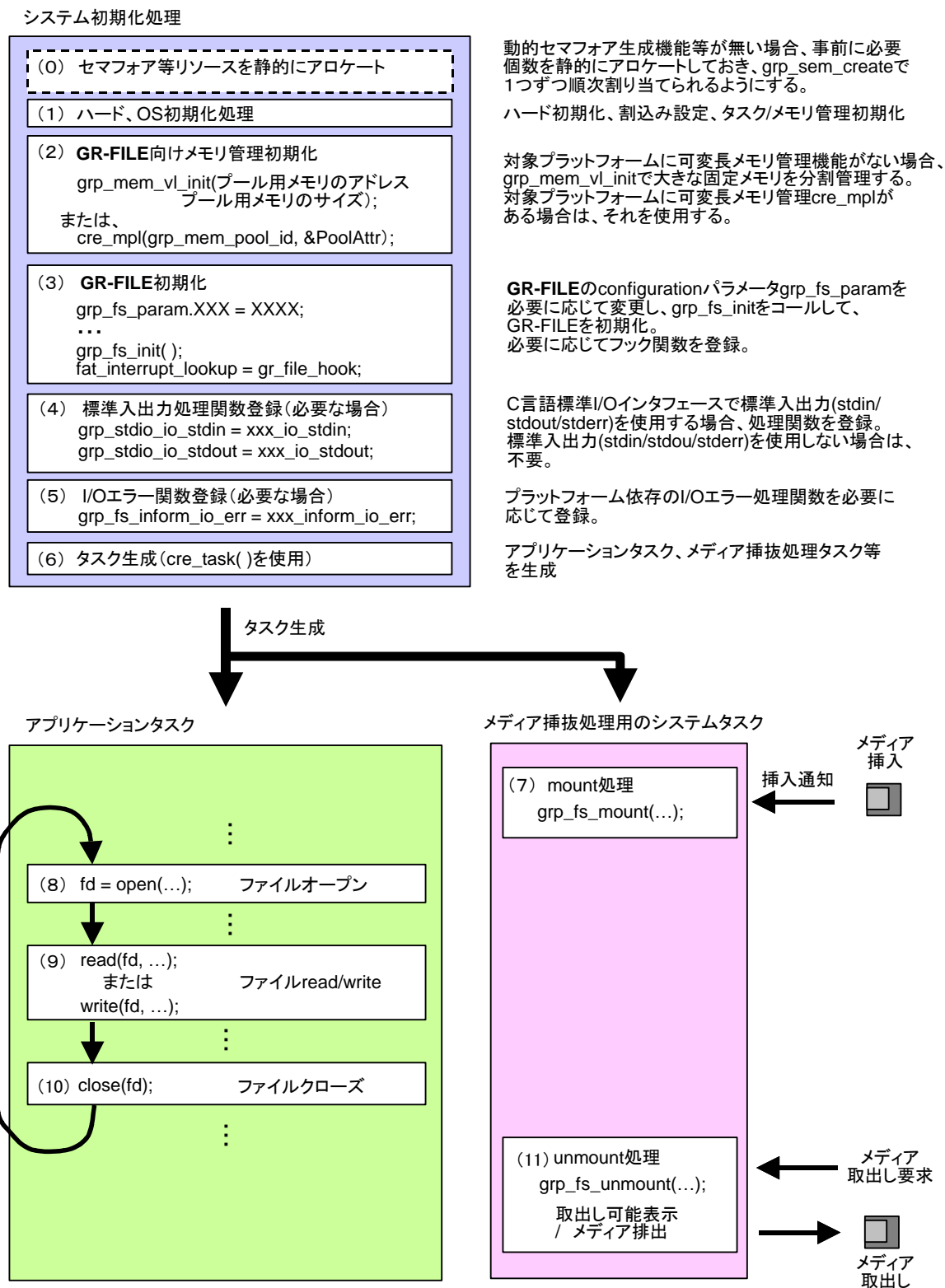


図 5-1 **GR-FILE** を用いたシステムの概略フロー例 ( $\mu$ ITRON OS の場合)



また、OS の違いを吸収する OS 抽象化インタフェース **GR-VOS** 上で **GR-FILE** を用いたシステムの概略フロー例を図 5-2 に示します。

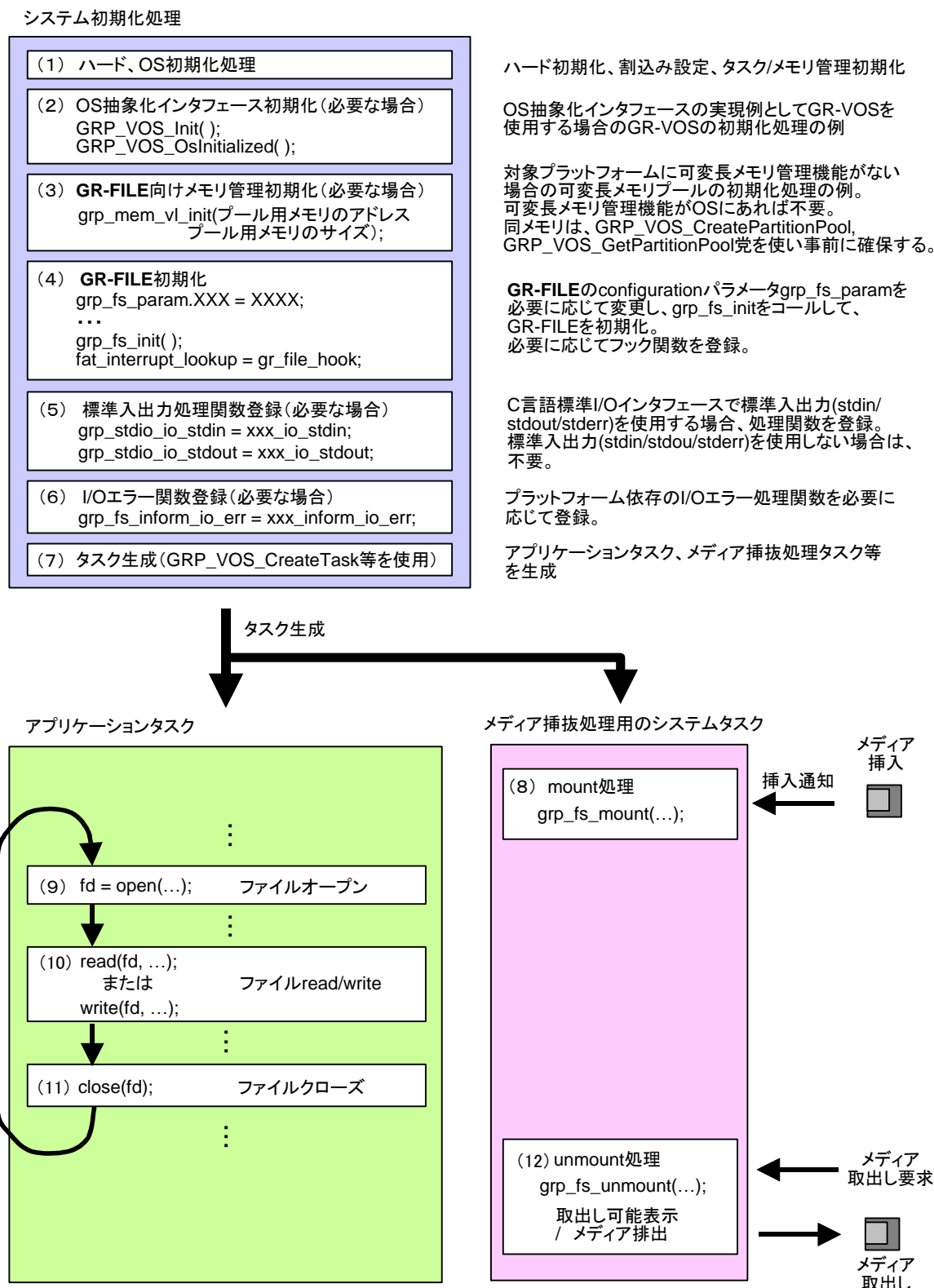


図 5-2 **GR-FILE** を用いたシステムの概略フロー例 (OS 抽象化インタフェース **GR-VOS** の場合)

さらに、上記メディア挿抜処理用のシステムタスクにおける、挿抜処理の概略処理フローの一例を図 5-3 に示します。

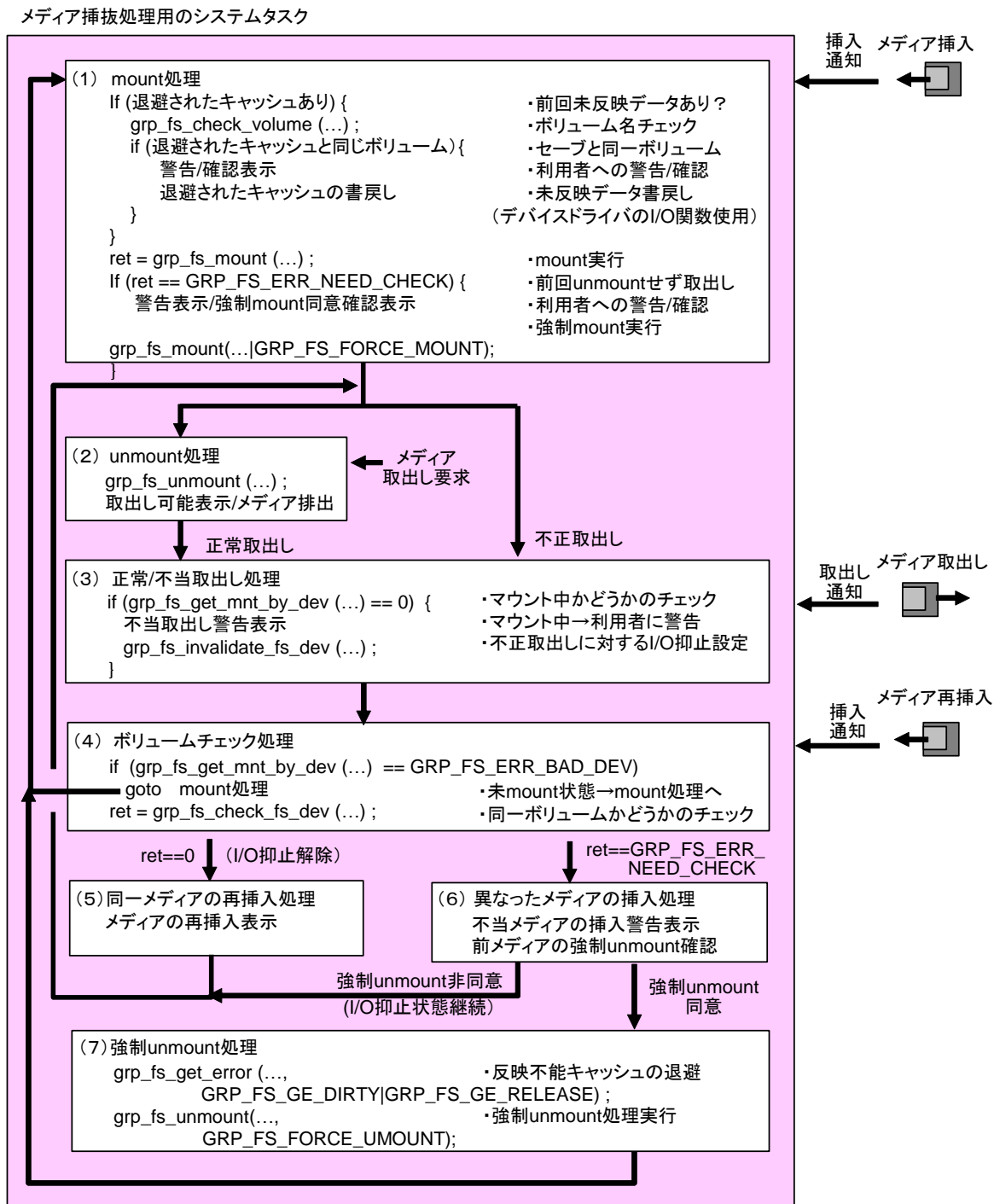


図 5-3 メディア挿抜処理の概略フローの一例

なお、わかり易さのために処理手順のようなイメージで処理間を結んだ形で示していますが、実際には、処理間の遷移は、挿入/取出し等のイベントに対応した状態遷移に対応しており、実際のプログラムは、イベント通知に対するアクションとして各処理を実行するようなイメージで実現する形となります。

また、わかり易さのために、(1) の `mount` 処理と、(4) ボリュームチェック処理を分離した形で示していますが、実際には、`mount` 処理の中で、最初に (4) の最初の `if` 文を実行し、挿入されたメディアが `mount` されていないならば、(1) の `mount` 処理を実行し、`mount` されていれば、(4) の同一ボリュームチェックを実行するという形になります。

ご参考として、上記の概略フローに対応したプラットフォーム依存の挿抜処理のサンプルコードを、“`sample/grp_fs_proc_event.c`” にて提供しています。

## 6. ソースファイルの構成と GR-FILE ライブラリの構築・使用方法

### 6.1 ソースファイルの構成

**GR-FILE** v1.24 のソースファイルの構成を表 6-1 に示します。なお、構成は、リリースバージョンにより変更になることがあります。

表 6-1 ソースファイルの構成

ディレクトリ/ファイル	内容
doc / lib /	<b>GR-FILE</b> ドキュメント <b>GR-FILE</b> の生成したライブラリを格納するディレクトリ 各ディレクトリ下の <b>Makefile</b> を使ってライブラリを作成しますと、以下の 2 つのライブラリファイルが生成されます。 grp_file.a : grp_file および grp_stdio ファイルの関数ライブラリ grp_lib.a : その他のファイルの関数ライブラリ
src / grp_fs / base / fat.c fat.h fat_format_def.h grp_fat_format.c grp_fs.c grp_fs.h grp_fs_cfg.c grp_fs_cfg.h grp_fs_conv_lib.c grp_fs_dev_io_if.c grp_fs_error.c grp_fs_get_cwd_lib.c grp_fs_get_disk_part.c grp_fs_io_disk_part.c grp_fs_mdep_if.h grp_fs_readdir.c grp_fs_set_disk_part.c grp_fs_trace.c grp_fs_trace.h grp_queue.h Makefile	<b>GR-FILE</b> ソースファイル ターゲット非依存 <b>GR-FILE</b> ソースファイル <b>GR-FILE</b> ベースファイル FAT ファイルシステム依存処理 FAT ファイルシステム依存処理定義ヘッダファイル FAT フォーマット関数の内部ヘッダファイル FAT フォーマット処理関数 ファイルシステム非依存処理 ファイルシステム非依存処理定義ヘッダファイル ファイルシステムタイプテーブル、デバイスドライバテーブル OS 抽象化/デバイスドライバインタフェース定義ヘッダファイル 変換処理関数 デバイス直接制御インタフェース関数 <b>GR-FILE</b> エラー番号→メッセージ変換ライブラリ カレントディレクトリの取得関数 パーティション情報の取得関数 パーティション情報の読書き関数 OS 抽象化インタフェース定義ヘッダファイル POSIX インタフェース関数の opendir / closedir / readdir 関数 パーティション情報の設定関数 I/O トレースオプションコード I/O トレースオプションコードヘッダファイル リスト処理マクロ定義ヘッダファイル <b>GR-FILE</b> 関数ライブラリ (lib / grp_file.a) 生成用 Makefile
include / grp_char_conv.h grp_fat_format.h grp_fat_param.h grp_fs_conv.h grp_fs_dev_io_if.h grp_fs_disk_part.h grp_fs_if.h grp_fs_param.h grp_fs_readdir.h grp_fs_sysdef.h grp_mem.h grp_sem.h grp_stdio.h grp_time.h grp_time_lib.h grp_types.h	アプリケーション向けターゲット非依存 <b>GR-FILE</b> ヘッダファイル 文字コード変換処理インタフェース定義 FAT フォーマットインタフェースの定義 <b>GR-FILE</b> のファイルシステム configuration パラメータ定義 POSIX 互換インタフェース定義 デバイス直接制御インタフェースの定義 パーティション設定/変更インタフェースの定義 <b>GR-FILE</b> 固有アプリケーションインタフェース定義 <b>GR-FILE</b> の configuration パラメータ定義 POSIX インタフェース opendir / closedir / readdir の定義 <b>GR-FILE</b> コンパイル定義 可変長メモリ管理インタフェース grp_mem_alloc / free の定義 セマフォ処理インタフェース定義 C 言語標準 I/O インタフェース定義 時刻設定/読出し関数インタフェース定義 時刻情報変換 (トータル秒⇔日付時刻) インタフェース定義 基本タイプ定義
lib / grp_char / grp_char_sjis_conv.c grp_char_sjis_tbl.h Makefile	<b>GR-FILE</b> ターゲット非依存ライブラリ 文字コード処理関数ライブラリソース シフト JIS ⇔ UNICODE 変換ライブラリ関数 シフト JIS ⇔ UNICODE 変換テーブル 文字コード処理関数生成、および、組込み向け関数ライブラリ (lib / grp_lib.a) への追加用 Makefile
grp_mem / grp_mem_vl_pool.c grp_mem_vl_pool.h Makefile	可変長メモリ管理関数ライブラリソース 可変長メモリ管理関数 可変長メモリ管理関数定義ヘッダファイル 可変長メモリ管理関数生成、および、組込み向け関数ライブラリ (lib / grp_lib.a) への追加用 Makefile
grp_stdio /	C 言語標準 I/O インタフェース関数ライブラリソース (オプション)

grp_stdio_default_io.c grp_stdio_fclose.c grp_stdio_fflush.c grp_stdio_fill.c grp_stdio_fopen.c grp_stdio_fprintf.c grp_stdio_fread.c grp_stdio_fseek.c grp_stdio_ftell.c grp_stdio_fwrite.c grp_stdio_getc.c grp_stdio_gets.c grp_stdio_put.c grp_stdio_vprintf.c Makefile	標準入出力(stdin / stdout / stderr)変数定義 fclose 処理 fflush 処理 I/O バッファ fill 処理 fopen 処理 fprintf 処理 fread 処理 fseek 処理 ftell 処理 fwrite 処理 getc 処理 gets 処理 putc 処理 vprintf 処理 C 言語標準 I/O インタフェース関数生成、および、 <b>GR-FILE</b> 関数ライブラリ (lib / grp_file.a) への追加用 Makefile
grp_time_lib / grp_time_lib.c Makefile	時刻情報変換 (トータル秒⇔日付時刻) 関数ライブラリソース 時刻情報変換 (トータル秒⇔日付時刻) 関数 時刻情報変換関数生成、および、組込み向け関数ライブラリ (lib / grp_lib.a) への追加用 Makefile
mdep_itron / base / grp_fs_dev_io.c grp_fs_dev_sw_tbl.c grp_fs_mdep_if.c include / grp_fs_mdep_types.h grp_mdep_sem.h lib / grp_itron_id.h grp_mem / grp_mem.c grp_sem / grp_sem.c grp_time / grp_time_get.c grp_time_set.c	μ ITRON 向けソースファイル  I/O 関数の空スタブコード デバイスドライバスイッチテーブル μ ITRON 向け OS 抽象化インタフェース  μ ITRON 向け <b>GR-FILE</b> タイプ定義 μ ITRON 向けセマフォアタイプ定義  μ ITRON 向けセマフォア、メモリプール ID 定義  μ ITRON 向け可変長メモリ管理関数 (grp_mem_alloc / free 関数)  μ ITRON 向けセマフォア関数  現在時刻取得関数の空スタブソース 時刻設定関数の空スタブソース
mdep_vos / base / grp_fs_dev_io.c grp_fs_dev_sw_tbl.c grp_fs_mdep_if.c include / grp_fs_mdep_types.h grp_mdep_sem.h grp_vos.h lib / grp_mem / grp_mem.c grp_sem / grp_sem.c grp_time / grp_time_get.c grp_time_set.c	VOS 向けソースファイル  I/O 関数の空スタブコード デバイスドライバスイッチテーブル VOS 向け OS 抽象化インタフェース  VOS 向け <b>GR-FILE</b> タイプ定義 VOS 向けセマフォアタイプ定義 VOS 向け基本タイプ定義  VOS 向け可変長メモリ管理関数 (grp_mem_alloc / free 関数)  VOS 向けセマフォア関数  現在時刻取得関数の空スタブソース 時刻設定関数の空スタブソース
mdep_vos2.xx / base / grp_fs_dev_io.c grp_fs_dev_sw_tbl.c grp_fs_mdep_if.c include / grp_fs_mdep_types.h grp_mdep_sem.h grp_vos.h lib / grp_mem / grp_mem.c grp_sem / grp_sem.c grp_time /	VOS2.xx 向けソースファイル  I/O 関数の空スタブコード デバイスドライバスイッチテーブル VOS2.xx 向け OS 抽象化インタフェース  VOS2.xx 向け <b>GR-FILE</b> タイプ定義 VOS2.xx 向けセマフォアタイプ定義 VOS2.xx 向け基本タイプ定義  VOS2.xx 向け可変長メモリ管理関数 (grp_mem_alloc / free 関数)  VOS2.xx 向けセマフォア関数

grp_time_get.c	現在時刻取得関数の空スタブソース
grp_time_set.c	時刻設定関数の空スタブソース
sample /	サンプルコード
app /	サンプルアプリケーション
readme.txt	サンプルアプリケーションの説明
vos	GR-VOS1.xx の環境用サンプル
gr_file_sample	シリアルを使ったコマンドライン型サンプルアプリケーション
cmd_line.c	コマンドライン関数
cmd_prog.c	プログラム I/F 関数のサンプル
cmd_tbl.c	コマンドテーブル
cmd_user.c	ユーザー I/F 関数のサンプル
usb_test.c	サンプルアプリケーションメイン関数
cmd.h	サンプルプログラムヘッダーファイル
usb_test.h	サンプルプログラムヘッダーファイル
gr_file_sample_no_console	デバッグ上で動作を確認する非コマンドライン型サンプルアプリケーション
test_main.c	サンプルアプリケーションメイン関数
test_main.h	サンプルプログラムヘッダーファイル
vos2.xx	GR-VOS2.xx の環境用サンプル
gr_file_sample	シリアルを使ったコマンドライン型サンプルアプリケーション
cmd_line.c	コマンドライン関数
cmd_prog.c	プログラム I/F 関数のサンプル
cmd_tbl.c	コマンドテーブル
cmd_user.c	ユーザー I/F 関数のサンプル
usb_test.c	サンプルアプリケーションメイン関数
cmd.h	サンプルプログラムヘッダーファイル
usb_test.h	サンプルプログラムヘッダーファイル
gr_file_sample_no_console	デバッグ上で動作を確認する非コマンドライン型サンプルアプリケーション
test_main.c	サンプルアプリケーションメイン関数
test_main.h	サンプルプログラムヘッダーファイル
base /	サンプル関数
grp_fat_format_sd.c	SD カードのフォーマット処理関数ライブラリ*1
grp_fat_format_sd.h	SD カードのフォーマット処理関数ライブラリヘッダファイル*1
grp_fs_dev_io_ram.c	RAM ディスク I/O 関数
grp_fs_dev_io_ram.h	RAM ディスク I/O 関数ヘッダファイル
grp_fs_proc_event.c	プラットフォーム依存の挿抜処理関数
grp_fs_proc_event.h	プラットフォーム依存の挿抜処理関数

\*1 本参考ライブラリは、ご要望された場合のみご提供しております。

本参考ライブラリは、SD Card Association の規格書で規定された情報を基に作成されていますので、本参考ライブラリの全体、または、一部を製品に利用する場合は、SD Card のライセンスが必要です。SD Card のライセンスにつきましては、SD Card Association にお問い合わせ下さい。

なお、OS 抽象化ライブラリ **GR-VOS** につきましては、同一 CD 内の **GR-FILE** のソースディレクトリと同一レベルにある "gr\_vos" ディレクトリ下をご参考下さい。(gr\_vos は弊社製の他の製品と同時購入時に添付されます)

## 6.2 GR-FILE の構築・使用方法

**GR-FILE** ライブラリの構築、アプリケーションの作成は、以下の手順で行います。

"mdep\_xxx"ディレクトリ名は、vos または itron の使用する環境に合わせて読み替えてください。

また、ポーティングについては、「ポーティングマニュアル」も参照ください。

### (1) OS/プラットフォーム依存タイプの定義

"mdep\_xxx/include" ディレクトリ下にある、"grp\_fs\_mdep\_types.h"、"grp\_mdep\_sem.h" を編集し、OS/プラットフォーム依存のタイプの定義を行います。

なお、32 ビットシステムでない場合は、include/grp\_types.h の基本タイプについても、変更が必要です。また、grp\_types.h の"int" を "long" に変更した場合は、fat.c、 grp\_fs.c 等のエラー出力用の grp\_fs\_printf フォーマットを変数タイプに応じて long 対応に変更する必要があります。

### (2) **GR-FILE** 用の OS/プラットフォーム依存関数の作成

"mdep\_xxx/base/grp\_fs\_mdep\_if.c" を編集し、**GR-FILE** 用の OS/プラットフォーム依存関数を作成します。提供時には、OS 仮想化ライブラリ **GR-VOS** 用に対応した、"mdep\_vos/base/grp\_fs\_mdep\_if.c"、"mdep\_vos2.xx/base/grp\_fs\_mdep\_if.c"、および、iTron 用の "mdep\_itron/base/grp\_fs\_mdep\_if.c"が入っています。

さらに、"mdep\_xxx/lib" ディレクトリ下にあるファイルを編集し、上記ファイル他から参照される OS/プラットフォーム依存ライブラリ関数を作成します。

### (3) デバイス I/O 関数の作成/登録

"mdep\_xxx/base/grp\_fs\_dev\_io.c" を編集し、**GR-FILE** 用のデバイス I/O 関数を作成します。提供時には、空のスタブ関数が入っています。

さらに、"mdep\_xxx/base/grp\_fs\_dev\_sw\_tbl.c" を編集し、作成したデバイス I/O 関数を grp\_fs\_dev\_tbl に登録します。

### (4) ファイルシステム依存関数の作成/登録（必要な場合）

提供している FAT ファイルシステム以外に独自でファイルシステムをサポートする場合は、"grp\_fs/base/fat.c" を参考に、同ファイルシステム用のファイルシステム依存関数を作成します。

さらに、"grp\_fs/base/grp\_fs\_cfg.c" を編集し、作成したファイルシステム関数を grp\_fs\_type\_tbl に登録します。

### (5) 各種パラメータの変更（必要な場合）

「各種パラメータの設定・変更」の節でしました、各種パラメータの define を必要に応じて変更します。



#### (6) ライブラリの作成

ソースファイルと一緒に提供しました **Makefile** を参考に、プラットフォームに応じたライブラリ構築スクリプトを作成し、**GR-FILE** のライブラリを作成します。なお、その際、「コンパイルオプション」の節で説明しました、コンパイルオプションを必要に応じて指定して構築します。

提供しました **Makefile** は、**LINUX** 上でライブラリを生成するためのサンプルです。前節で示しましたソースファイルの各ディレクトリの下にある **Makefile** を使ってそれぞれのソースファイルをコンパイルしますと、“grp\_fs/base” と “grp\_fs/lib/grp\_stdio” 下の関数群は、ライブラリファイル “lib/grp\_file.a” に生成・格納し、その他のディレクトリ下の関数群は、ライブラリファイル “lib/grp\_lib.a” に生成・格納する形となっています。

#### (7) アプリケーションの **GR-FILE** 依存部の作成

「サンプルフロー」の章で示しましたように、**GR-FILE** を用いたアプリケーションプログラムでは、ファイル I/O 処理だけでなく、**GR-FILE** 向けの初期化処理や、プラットフォームに依存したメディアの挿抜処理部分も **GR-FILE** で提供した関数を使用して作成します。なお、プラットフォーム依存のメディア挿抜処理のサンプルコードは、“sample/base/grp\_proc\_event.c” にて提供しています。

#### (8) アプリケーション実行モジュールの作成

(7) で作成したアプリケーションプログラムと、(6) で作成した **GR-FILE** のライブラリをリンクして、実行可能なロードモジュールを作成します。



## 組込み向けファイルシステム **GR-FILE**

発行年月：2020年 3 月 第 1.31版

発行：株式会社グレープシステム

E-Mail : [gr@support.grape.co.jp](mailto:gr@support.grape.co.jp)

URL : <http://www.grape.co.jp>

Copyright (C) 2003 - 2020 Grape Systems, Inc.

All rights reserved.