USFiles*Plus*™

Filesystem Utility

User's Manual

Revision 1.02
July 2009

Embedded Solution Partner
日新システムズ

はじめに

この度は、USFilesPlus Filesystem Utility をお買い上げ頂き有り難うございます。

このマニュアルは、USFilesPlus Filesystem Utility の導入をスムーズに行って頂く為のマニュアルです。

USFilesPlus の詳細につきましては、User’s Manual、USFilesPlus Quick Start Guide をご覧ください。

改訂履歴

日付	Revision	改訂内容
2004 年 7 月 21 日	1.00	初版
2006 年 3 月 27 日	1.01	Ver.3.10 に対応
2009 年 7 月 21 日	1.02	Ver.3.21 に対応 SD フォーマット機能に関して追記

目　次

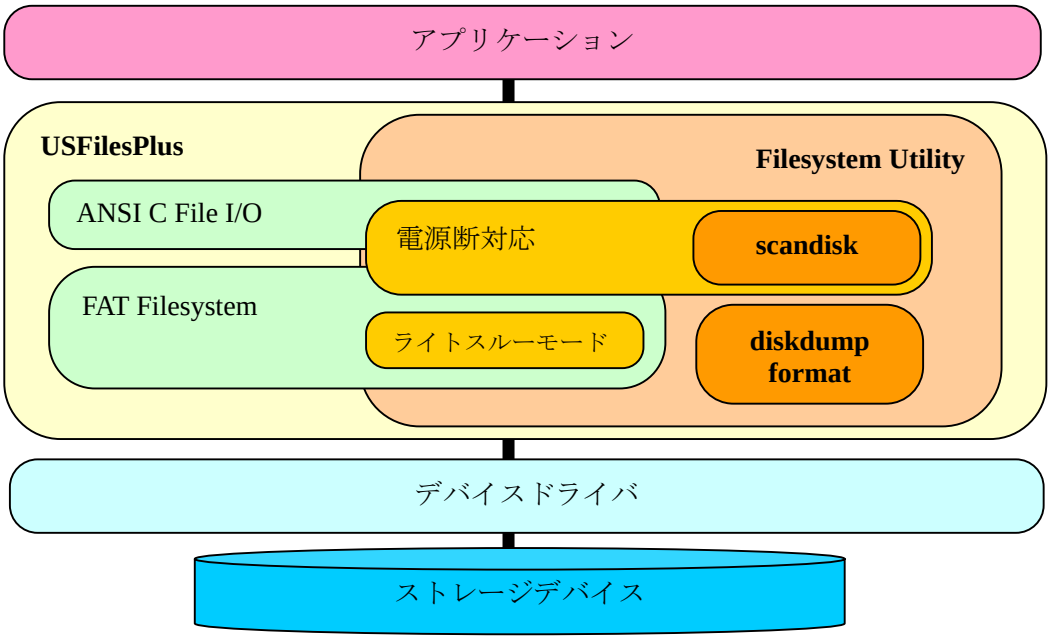
1. USFilesPlus Filesystem Utility について

1.1. USFilesPlus Filesystem Utilityとは

USFilesPlus Filesystem Utility は、下表に示す機能を実装するための
USFilesPlus FAT ファイルシステム専用のユーティリティです。

機能	概要
diskdump 機能	ディスク内のダンプデータの取得／書込を行います。
format 機能	デバイスの初期化（フォーマット）を行います。
scandisk 機能	デバイスの不整合の修正を行います。
電源断対応	FAT のクリーンフラグを使用して、電源断後の再起動時に scandisk の実行を行います。 ファイルリネーム時の電源断によるリネーム途中のファイルの消滅防止を行います。
ライトスルーモード	バッファ（キャッシュ）を使用せず、直接デバイスに書き込みを行います。

本ユーティリティは、C 言語のソースコード提供となっており、
各機能は、diskdump、format、scandisk は C 言語の関数として、また電源断対応および
ライトスルーモードは、メイクファイル等の設定により FAT ファイルシステム内に
実装する構成となっております。下図にモジュール構成イメージ図を記載します。



1.2. USFilesPlus Filesystem Utilityの制限事項

・本ユーティリティは、USFilesPlus Ver.3.10 以降でのみ動作します。

・本ユーティリティの保証範囲は、USFilesPlus 上での動作のみとします。

・他のシステムにて電源断および不正書き込みが行われたストレージデバイスでは、
電源断対応処理が正常に動作しない場合があります。

・電源断時に書き込みを行っていたファイルのデータは消滅する場合があります。

2. インストール

USFilesPlus Filesystem Utility をインストールする前に、下記の 2 つの必要事項をチェックしてください。

※以降に示す手順は、開発環境を Windows2000/XP 上とした場合の説明です。
また、作業用ドライブを C ドライブ、CD-ROM ドライブを E ドライブとしています。

- 開発環境に Opus make がインストールされているかどうか。
  USFilesPlus の makefile は Opus make を使用して動作するように設計されています。

- USFilesPlus 製品 CD-ROM があるかどうか。

上記の条件がそろっている場合、下記の手順でインストールを行います。

1. USFilesPlus の CD-ROM を CD-ROM ドライブ（**E:**）に挿入してください。

2. USFilesPlus FAT ファイルシステム（E:¥disk1¥usf ディレクトリ下のファイル）を全て C:¥ussw ディレクトリ以下にコピーします。また、VFAT（E:¥disk1¥usfvfat）、FAT32（E:¥disk1¥usf32）等が必要な場合は、各ディレクトリ下のファイルを全て C:¥ussw ディレクトリ以下にコピーします。

3. USFilesPlus Filesystem Utility（E:¥disk1¥usfut ディレクトリ下のファイル）を全て C:¥ussw ディレクトリ以下にコピーします。

4. BSP ソースコード（E:¥disk2¥i8086 ディレクトリ下のファイル）を全て C:¥ussw ディレクトリ下にコピーします。
   （ここでは、i8086 プラットフォームとして説明しています。）

5. C:¥ussw ディレクトリ下の属性を書き込み可能に変更します。

3. 設定

USFilesPlus にて、お客様の環境にて開発を可能にするには、３つのメイクファイル、「**Config.mak**」、「**Compiler.mak**」、「**sio.mak**」および「**devtab.c**」を編集する必要があります。

3.1. Config.makの設定

USFilesPlus をインストールしたディレクトリの直下にある「**Config.mak**」を下記のように編集します。

1. USFilesPlus をインストールしたディレクトリを指定するために、***USROOTDIR*** 宣言部分をインストールしたディレクトリに変更します。ここでは、インストールディレクトリを「C:¥ussw」としています。

   USROOTDIR = C:¥ussw

2. ***CPU*** 宣言部分にターゲットの CPU を指定します。先頭の「#」は、コメントアウトシンボルを示しており、先頭の「#」を消去することで宣言が有効となり、先頭に「#」をつけることで宣言が無効となります。

   ここでは、「i8086」をターゲットの CPU として指定します。
   CPU = i8086　　# Intel real mode,　　COMPILERS: borland, msoft, cadul

3. ***COMPILER*** 宣言部分にコンパイラを指定します。

   ここでは、コンパイラを Borland C/C++として指定します。
   COMPILER = borland　　# Borland C/C++

4. ***PRODLIST*** 宣言部分に使用する機能を指定します。
   最低限 USFilesPlus FAT ファイルシステムが必要ですので、「#」にてコメントアウトされている場合はこれを除去してください。
   FAT32 プロダクトが必要な場合は、同様にコメントアウトを除去してください。
   また、これら以外の必要のないプロダクトはコメントアウトしてください。

   ```
   PRODLIST += usf        # USFiles file system
   PRODLIST += usf32      # USFiles FAT32 support
   ```

5. ***PRODLIST*** 宣言部分に USFilesPlus Filesystem Utility 宣言を追記します。

   ```
   PRODLIST += usfut      # USFiles Filesystem Utility
   ```

6. ***RTOS*** 宣言部分に使用する RTOS を指定します。
   ここでは RTOS を使用しない「none」を指定します。

   ```
   RTOS = none            # No RTOS used
   ```

7. ***TRACE_DEBUG*** 宣言部分にデバッグトレース出力モードを指定します。デフォルトは、３が指定されています。ターゲットのデバッグトレース出力をなしにするには０を、冗長な出力が必要な場合は９を指定してください。

   ```
   TRACE_DEBUG = 3
   ```

ここで設定した内容にしたがって、U S Software のライブラリ「ussw.lib」がビルドされます。その他の詳細な宣言については、「USFiles User’s Manual」を参照してください。

3.2. Compiler.makの設定

USFilesPlus は、コマンドラインでのコンパイラ、アセンブラ、リンカ、ライブラリアン等を必要とします。

<USFilesPlus インストールディレクトリ>¥config¥<CPU>¥<コンパイラ>ディレクトリ下にある「**Compiler.mak**」を編集して、これらの開発ツールを適切に指定する必要があります。CPU およびコンパイラディレクトリは「**Config.mak**」にて指定した CPU およびコンパイラとなります。

その他のターゲットでのユーザオプションには、違ったオプションが存在します。
このオプションは、ボードレベルのサポートとなり、開始アドレス、ベースアドレス、クロックレート、エンディアン等々の設定があります。

ここで、最も重要な項目は、ツールチェーンパスの宣言部分です。
PTH にツールチェーンがインストールされているパスを指定します。

ここでは､Borland C/C++が C ドライブの **c:¥*bc5*** にインストールされているものとして、***PTH*** を下記のように指定します。

```
PTH = C:¥bc5        # Where the compiler is
```

コンパイラ、アセンブラ、リンカ、ライブラリアンについても、***PTH*** に基づいて指定され、下記の項目に宣言しています。必要であれば、これらの部分も編集してください。

CC	: コンパイラ
AS	: アセンブラ
LNK	: リンカ
LIBR	: ライブラリアン

また、各ツールチェーンのオプションも下記の項目にて指定してください。

CFLAGS	: コンパイラオプション
AFLAGS	: アセンブラオプション
LFLAGS	: リンカオプション

3.3. Sio.makの設定

USFilesPlus は、詳細機能選択を、

<USFilesPlus インストールディレクトリ>¥config¥sio.mak

を編集することで行います。ここでは、C:¥ussw¥config¥sio.mak となります。

1. USFilesPlus VFAT を使用する場合、***VFAT*** 宣言を 1 に変更します。
   デフォルトは 0 となっています。

```
#
# Set VFAT = 1 if long file names and/or Joliet extensions are desired.
#
VFAT = 0        # 0    No long file name support
                # 1    Include long file names
```

2. USFilesPlus にて、漢字コード（Shift-JIS）を使用する場合、***FAKEUNICODE*** 宣言を 0 に変更します。デフォルトは 1 となっています。

```
#
# Set FAKEUNICODE = 0 if you want Kanji support with long file names
#
FAKEUNICODE = 1         # 0    Kanji supported in Unicode long file names
                        # 1    Only ASCII supported in Unicode long file names
```

3. ライトスルーモードを使用する場合、***WRITETHROUGH*** 宣言を 1 に変更します。
   デフォルトは 0 となっています。

```
#
# Set WRITETHROUGH = 1 if you want writing through buffers mode
#
WRITETHROUGH = 0        # 0    Write through mode OFF
                        # 1    Write through mode ON
```

3.4. devtab.c の設定

ストレージデバイス情報の宣言を

＜USFilesPlusインストールディレクトリ＞¥siosrc¥devtab.c

ファイルに記載する必要があります。

設定方法および詳細な宣言については、「USFilesPlus Quick Start Guide」および「USFiles User's Manual」を参照してください。

4. コンパイル

前述の通り、USFilesPlus はライブラリおよびテストプログラムを作成するために Opus make ユーティリティを使用します。

その他の make ユーティリティを使用する場合は、Opus make の実行名部分を宣言しなおす必要があります。違ったアプローチとしては、簡単なバッチファイルを作成する方法があります。この場合、make ユーティリティのコマンドラインオプションに留意する必要があります。

通常、makefile はライブラリをコンパイルするようになっています。

特定のプログラムをコンパイルしたい場合は、make コマンドの引数にプログラム名を指定してください。また、全てのライブラリを削除したい場合は、make コマンドの引数として「clean」と指定してください。この引数を指定することで、ライブラリファイル、オブジェクトファイル、実行ファイル、マップファイルを全て削除します。

なお、詳細については、「USFilesPlus Quick Start Guide」および「USFiles User's Manual」を参照してください。

5. ライブラリのビルド

config.mak、**compiler.mak** および **sio.mak** が正しく指定されていれば、USFilesPlus ライブラリをビルドすることができます。ビルドは、下記の手順にて行います。

1. USFilesPlus がインストールされているディレクトリに移動します。
   ここでは、「c:¥ussw」に USFilesPlus がインストールされているものとします。

   c:¥>cd　ussw

2. make コマンドを実行してください。
   ここでは、make コマンドを「make」とします。

   c:¥ ussw >make

6. 詳細内容

以降に USFilesPlus Filesystem Utility の詳細について記載します。

6.1. diskdump機能

USFilesPlus Filesystem Utility での diskdump 機能に関する詳細を以下に記載します。

6.1.1.概要

diskdump 機能とは、ストレージデバイスの特定論理セクタ位置からのダンプデータ取得／書き込みを行う機能です。この操作を行うために、USFilesPlus Filesystem Utility では、以降に示す２つの関数があります。

6.1.2.getd_total_sects

関数名

getd_total_sects

宣言

uint32 getd_total_sects(const char *_drive_)

引数

drive　　デバイステーブルに指定されているドライブ文字

戻り値

０以外の数値	正常終了（全セクタ数を返します。）
０	異常終了（errno にエラー情報が設定されます。）

説明

*drive*にて指定したストレージデバイス上の全論理セクタ数の取得を行います。

ファイル

＜USFilesPlus インストールディレクトリ＞¥usfut¥diskdump.c

6.1.3.mt_diskdump

関数名

mt_diskdump

宣言

int mt_diskdump(const char *_drive_, uint16 _options_,
uint32 _lsect_, byte *_buf_, int _len_)

引数

drive　デバイステーブルに指定されているドライブ文字
options　オプションフラグ

オプションフラグ	詳細
DMP_OPT_READ	ダンプデータの取得を行います。
DMP_OPT_WRITE	ダンプデータの書き込みを行います。
DMP_OPT_MBR	MBR（Master Boot Record）領域を操作可能にします。

lsect　開始論理セクタ
buf　ダンプデータバッファへのポインタ
len　ダンプデータバッファのサイズ（bytes）

戻り値

0 以上の数値　正常終了（取得したダンプデータのバイト数を返します。）
－1　異常終了（errno にエラー情報が設定されます。）

説明

drive にて指定したストレージデバイスにおいて、*lsect* にて指定した論理セクタを起点として、ダンプデータの取得および書き込みを行います。

ファイル

＜USFilesPlus インストールディレクトリ＞¥usfut¥diskdump.c

6.1.4.diskdump サンプルプログラム

diskdump にはサンプルプログラムがあり、サンプルプログラムのソースファイルは**＜USFilesPlus インストールディレクトリ＞¥appsrc** ディレクトリ下にある「**dmptest.c**」となっています。

6.2. format機能

USFilesPlus Filesystem Utility での format 機能に関する詳細を以下に記載します。

6.2.1.概要

format 機能とは、ストレージデバイスの初期化を行う機能です。
この操作を行うために、USFilesPlus Filesystem Utility では、**mt_format** 関数を使用します。

6.2.2.フォーマット形式

FAT のフォーマット形式には、大きく分けて「標準フォーマット」、「物理フォーマット」の 2 種類存在し、標準フォーマットにもクラスタチェックを行う／行わない等の種別があります。よって、USFilesPlus Filesystem Utility としては下表に示す種別を設けることで、フォーマット形式の分類を行います。

	通常フォーマット	クイックフォーマット	フルフォーマット
BPB 初期化	○	△※1	○
FAT 初期化	○	○	○
ルートディレクトリ領域初期化	○	○	○
クラスタチェック	×	×	○
物理フォーマット	×	×	○

○：処理を行う、×：処理を行わない

※1 FAT12/16 の場合は、BPB のボリュームシリアル ID およびボリュームラベルのみを変更します。また、FAT32 の場合は、これに加えてディスク空きクラスタ数を変更します。

また、ハードディスク等の ATA デバイスにおいては、メディアメーカー毎に物理フォーマット形式が異なるため、USFilesPlus のサンプルデバイスドライバ（BIOS／LBA デバイスドライバ）では **format** はインプリメントされていません。
これらのことから、USFilesPlus Filesystem Utility での物理フォーマット処理は、デバイスドライバの **format** を呼び出す構成としております。
したがいまして、実際には、デバイスドライバの **format** がインプリメントされていない場合、物理フォーマットは行われません。

6.2.3.クラスタサイズについて

ハードディスク（ATA デバイス）に対してフォーマットを行った場合は、MBR（Master Boot Record）または EBR（Extended Boot Record）からハードディスクのパーティションに割り当てられたセクタ数（論理セクタ数）を取得することで、クラスタサイズを決定します。また、FAT タイプは、システム ID をもとに決定します。

FAT タイプ	システム ID
FAT12	0x01
FAT16	0x04、0x06、0x0e
FAT32	0x0b、0x0c

システムIDに対応したFATタイプの最大セクタ数よりも取得したセクタ数が大きい場合は、最大セクタ数からクラスタサイズを計算します。

最大セクタ数は、FAT タイプから求められ、各 FAT タイプの最大クラスタサイズを超えないようにクラスタあたりのセクタ数を計算します。

最大クラスタサイズは、FAT12 は 4085（2^{12}-11）、FAT16 は 65525（2^{16}-11）と規定されています。FAT32 の最大クラスタ数は 262144（2^{18}）～67108853（2^{26} - 11）の間でディスクサイズ毎に決定します。ディスクサイズ毎のクラスタサイズ・クラスタあたりのセクタ数、およびクラスタサイズ・最大クラスタ数をそれぞれ下表に記載します。

<table>
<tr><th></th><th colspan="3">クラスタサイズ /クラスタあたりのセクタ数</th></tr>
<tr><th>ディスクサイズ</th><th>FAT12</th><th>FAT16</th><th>FAT32</th></tr>
<tr><td>16MB 未満</td><td>4KB(4096bytes) /8</td><td>-</td><td rowspan="3">-</td></tr>
<tr><td>16～127MB</td><td rowspan="9">-</td><td>2KB(2048bytes) /4</td></tr>
<tr><td>128～255MB</td><td>4KB(4096bytes) /8</td></tr>
<tr><td>256～511MB</td><td>8KB(8192bytes) /16</td></tr>
<tr><td>512～1023MB</td><td>16KB(16384bytes) /32</td><td rowspan="3">4KB(4096bytes) /8</td></tr>
<tr><td>1024～2047MB</td><td>32KB(32768bytes) /64</td></tr>
<tr><td>2～4GB 未満</td><td rowspan="4">-</td></tr>
<tr><td>4～8GB 未満</td></tr>
<tr><td>8～16GB 未満</td><td>8KB(8192bytes) /16</td></tr>
<tr><td>16～32GB 未満</td><td>16KB(16384bytes) /32</td></tr>
<tr><td>32GB 以上</td><td>32KB(32768bytes) /64</td></tr>
</table>

ディスクサイズ	クラスタサイズ	最大クラスタ数
512～1023MB	4KB(4096bytes) /8	262144（2^{18}）
1024～2047MB		524288（2^{19}）
2～4GB 未満		1048576（2^{20}）
4～8GB 未満		2097152（2^{21}）
8～16GB 未満	8KB(8192bytes) /16	
16～32GB 未満	16KB(16384bytes) /32	
32～64GB 未満	32KB(32768bytes) /64	
：	：	：
512GB～1TB 未満	32KB(32768bytes) /64	33554432（2^{25}）
1～2TB 未満	32KB(32768bytes) /64	67108864（2^{26}）

6.2.4.ボリュームラベル

ボリュームラベルの設定は、BPB（Bios Parameter Block）のボリュームラベルとルートディレクトリ領域の最初のエントリにボリュームラベルのエントリを追記することで行います。ボリュームラベルは、11 bytes まで記載可能ですが、Shift-JIS の全角文字は 5 文字までとし、11 bytes 目が全角文字の先頭であった場合はエラーとなります。

6.2.5.SD フォーマットについて

バージョン 3.21 より、SD アソシエーションに準拠した SD フォーマットにも対応しました。但し、使用する場合は SD ドライバおよび、SPI ドライバが必要になります。また、物理フォーマットに関しましては、ご使用の SD ドライバおよび、SPI ドライバが ERASE コマンドに対応している場合に限ります。

尚、本 SD フォーマットのクイックフォーマットは、既存のフォーマットとは異なり、無条件で MBR の各パラメータの算出を行い、正常な値に書き換えます。

弊社より提供しているサンプルの SD ドライバの対応状況は下記になります。

	通常 フォーマット	クイック フォーマット	フル フォーマット
SPI ドライバ	○	○	○
標準ホストコントローラ対応 SD ドライバ	○	○	○
TE4301/4300 ※	○	○	×

※ 東京エレクトロンデバイス株式会社製 TE4300 ファームウェア及び TE4301 ファームウェアに対応したドライバです。

6.2.6.mt_format

関数名

mt_format

宣言

int mt_format(const char *_drive_, uint16 _option_, char *_label_)

引数

drive　デバイステーブルに指定されているドライブ文字
option　オプションフラグ

オプションフラグ	詳細
FMT_OPT_NORMAL	通常フォーマットを行います。
FMT_OPT_QUICK	クイックフォーマットを行います。
FMT_OPT_UNCON	フルフォーマットを行います。
FMT_OPT_INTERACTIVE	Character I/O を使用して対話形式でフォーマットを実行します。
FMT_OPT_ACTIVE	アクティブドライブとしてフォーマットを実行します。
FMT_OPT_IS_FDD	1.44MB フロッピーディスクとしてフォーマットを実行します。
FMT_OPT_IS_SD	SD カードとしてフォーマットを実行します。

label　ボリュームラベル文字列（最大 11 bytes）
ボリュームラベルが必要ない場合は、**NULL** を指定します。

戻り値

0　正常終了
－1　異常終了（errno にエラー情報が設定されます。）

説明

drive にて指定した書き込み可能なストレージデバイスに対して *options* に指定した内容で初期化を行います。また、ボリュームラベル名は、*label* に指定します。

ファイル

＜USFilesPlus インストールディレクトリ＞¥usfut¥format.c

6.2.7.format サンプルプログラム

format にはサンプルプログラムがあり、サンプルプログラムのソースファイルは**＜USFilesPlus インストールディレクトリ＞¥appsrc** ディレクトリ下にある「**fmttest.c**」となっています。

6.3. scandisk機能

USFilesPlus Filesystem Utility での scandisk 機能の詳細を以下に記載します。

6.3.1.概要

scandisk 機能とは、FAT ファイルシステムとして不正な状態となっているストレージデバイスを修復する機能です。この操作を行うために、USFilesPlus Filesystem Utility では、**mt_scandisk** 関数を使用します。

6.3.2.エラーと修復内容

FAT ファイルシステムとして、下表に示すファイルシステム異常に関連する様々なエラーが存在します。USFilesPlus Filesystem Utility では、それぞれのエラーに対して修復が行えるように、エラー毎に **mt_scandisk** の修復オプションを設けています。

エラー内容	エラーコード（errno）	scandisk 修復オプション
FAT1 と FAT2 が異なる場合	EBADFAT	SCN_OPT_EBADFAT
メディアディスクリプタが不正	EMEDIADESC	SCN_OPT_EMEDIADESC
ディレクトリ構造が不正	EDIRTREE	SCN_OPT_EDIRTREE
short エントリ名が不正	ESNAME	SCN_OPT_ESNAME
long エントリ名が不正 long エントリが 256 文字以上ある場合	ELNAME	SCN_OPT_ELNAME
FAT ：パスが 80 文字以上ある場合 VFAT：パスが 256 文字以上ある場合	EBIGPATH	SCN_OPT_EBIGPATH
エントリ内のファイルサイズが不正および ディレクトリエントリのサイズが 0 でない場合	EENTRYSIZE	SCN_OPT_EENTRYSIZE
エントリ内のクラスタ番号が不正	ECLUSTNUM	SCN_OPT_ECLUSTNUM
FAT チェーンが不正	ECLUSTLINK	SCN_OPT_ECLUSTLINK
破損クラスタがあった場合	ELOSTCLUST	SCN_OPT_ELOSTCLUST
破損クラスタを保存する際、ルートディレクトリエントリに空きがない場合	ERDFULL	SCN_OPT_ERDFULL
クロスリンクがあった場合	ECROSSLINK	SCN_OPT_ECROSSLINK
クロスリンクの修復時にディスクに空きがない場合	EDSKFUL	SCN_OPT_EDSKFUL
空き領域が不正（FAT32 のみ）	EFREECLUST	SCN_OPT_EFREECLUST
long エントリと short エントリのチェックサムが異なる場合	EDIRCHKSUM	SCN_OPT_EDIRCHKSUM
ディレクトリエントリ内のファイルサイズと FAT チェーン数が異なる場合	ELINKEND	SCN_OPT_ELINKEND

これらの修復オプションに加えて、以下のオプションを付加することで様々な設定にて scandisk を実行することが可能です。

オプション	詳細
SCN_OPT_CHECKONLY	エラーチェックのみを行い、修復は行いません。
SCN_OPT_INTERACTIVE	Character I/O を使用して対話形式で scandisk を実行します。
SCN_OPT_SAVE	破損クラスタをファイルに保存します。
SCN_OPT_SURFACE	クラスタスキャンを実行します。

また、全ての修復オプションを簡単に指定したり、一般的なオプションを指定した

オプション	詳細
SCN_OPT_FORCE	全てのエラーを自動的に修復します。
SCN_OPT_AUTOFIX	特定のエラーを自動的に修復します。

を設けることでより簡単に指定できるようにしております。
SCN_OPT_FORCE、SCN_OPT_AUTOFIX を指定した場合に修復される内容を下表に記載します。

scandisk 修復オプション	SCN_OPT_FORCE	SCN_OPT_AUTOFIX
SCN_OPT_EBADFAT	○	○
SCN_OPT_EMEDIADESC	○	○
SCN_OPT_EDIRTREE	○	○
SCN_OPT_ESNAME	○	○
SCN_OPT_ELNAME	○	○
SCN_OPT_EBIGPATH	○	×
SCN_OPT_EENTRYSIZE	○	○
SCN_OPT_ECLUSTNUM	○	○
SCN_OPT_ECLUSTLINK	○	○
SCN_OPT_ELOSTCLUST	○	○
SCN_OPT_ERDFULL	○	×
SCN_OPT_ECROSSLINK	○	○
SCN_OPT_EDSKFUL	○	×
SCN_OPT_EFREECLUST	○	○
SCN_OPT_EDIRCHKSUM	○	○
SCN_OPT_ELINKEND	○	○

6.3.3.mt_scandisk

関数名

mt_scandisk

宣言

int mt_scandisk(const char * *drive*, **uint32** *options*)

引数

drive　　デバイステーブルに指定されているドライブ文字
option　　オプションフラグ

オプションフラグ	詳細
SCN_OPT_FORCE	全てのエラーを自動的に修復します。
SCN_OPT_AUTOFIX	特定のエラーを自動的に修復します。
SCN_OPT_CHECKONLY	エラーチェックのみを行います。
SCN_OPT_INTERACTIVE	Character I/O を使用して対話形式でスキャンディスクを実行します。
SCN_OPT_SAVE	破損クラスタをファイルに保存します。
SCN_OPT_SURFACE	クラスタスキャンを実行します。

また、SCN_OPT_FORCE、SCN_OPT_AUTOFIX、SCN_OPT_CHECKONLY の代わりに以下のオプションを個別に指定することも可能です。

オプションフラグ	詳細
SCN_OPT_EBADFAT	FAT1 と FAT2 の差異を修復します。
SCN_OPT_EMEDIADESC	メディアディスクリプタのエラーを修復します。
SCN_OPT_EDIRTREE	ディレクトリ構造のエラーを修復します。
SCN_OPT_ESNAME	不正な short エントリ名のエントリを削除します。
SCN_OPT_ELNAME	不正な long エントリ名のエントリを削除します。
SCN_OPT_EBIGPATH	パス名の長さを修復します。
SCN_OPT_EENTRYSIZE	不正なファイルサイズのエントリを削除します。
SCN_OPT_ECLUSTNUM	不正なクラスタ番号のエントリを削除します。
SCN_OPT_ECLUSTLINK	不正な FAT チェーンの存在するエントリを削除します。
SCN_OPT_ELOSTCLUST	破損クラスタを修復します。
SCN_OPT_ERDFULL	破損クラスタを保存する際、ルートディレクトリエントリに空きがない場合、破損クラスタを削除します。
SCN_OPT_ECROSSLINK	クロスリンクを修復します。
SCN_OPT_EDSKFUL	クロスリンクの修復時にディスクに空きがない場合、クロスリンクしているエントリを削除します。
SCN_OPT_EFREECLUST	FAT32 の空き領域の不整合を修復します。
SCN_OPT_EDIRCHKSUM	long エントリと short エントリのチェックサムの不整合があった場合、そのエントリを削除します。
SCN_OPT_ELINKEND	FAT チェーンをファイルサイズと合致するように修正します。

戻り値

0	正常終了
－１	異常終了（errno にエラー情報が設定されます。）

説明

drive にて指定したストレージデバイスに対して、*options* に指定した内容で修復を行います。

ファイル

＜USFilesPlus インストールディレクトリ＞¥usfut¥scandisk.c

6.3.4.scandisk サンプルプログラム

scandisk にはサンプルプログラムがあり、サンプルプログラムのソースファイルは**＜USFilesPlus インストールディレクトリ＞¥appsrc** ディレクトリ下にある「**scdtest.c**」となっています。

6.4. 電源断対策

6.4.1.概要

電源断対策として、USFilesPlus に以下の機能の追加、および修正を行っております。

- ***fopen( )*** 時の scandisk 実行
- ファイルリネーム時のデータ保証改善
- エントリ作成時の属性フラグ
- キャッシュデータ書き込み順の変更

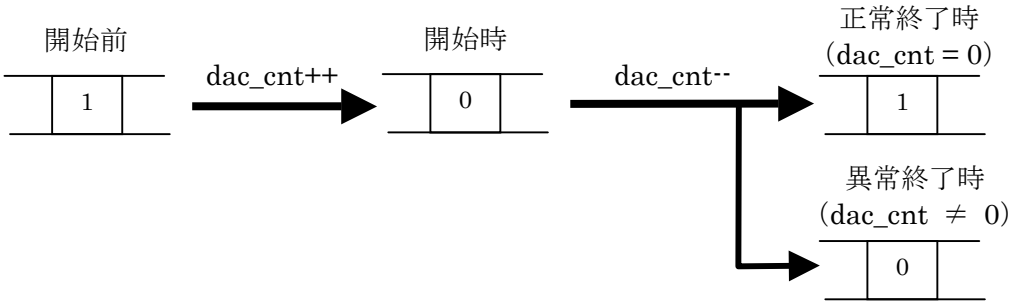
6.4.2. *fopen( )* 時の scandisk 実行

FAT の２つ目のエントリ（予約領域）のビットをクリーンフラグとして使用します。
各ファイルシステムにおいて、クリーンフラグとして使用されるビットは下表の通りです。

フォーマット形式	クリーンフラグのビット位置
FAT12	ビット 11（=MSB）
FAT16	ビット 15（=MSB）
FAT32	ビット 27

クリーンフラグは FAT ファイルシステムの利用開始時に“0”に設定し、終了時に“1”に設定するものとします。

ファイルオープン時に、デバイステーブルに設定されている書き込み可能なストレージデバイスのクリーンフラグが“0”として検出された場合、電源断が発生したものとみなし、自動的に scandisk を実行します。



また、複数のアプリケーションによるファイルシステム利用の開始／終了を判別するために、デバイステーブルに新たにディスクアクセスカウンタ（dac_cnt）を追加します。
ディスクアクセスカウンタにて、ファイルシステムの利用を開始（*fopen*）するときにカウントアップし、利用を終了（*fclose*）するときにカウントダウンすることで、クリーンフラグが現在デバイスに対してアクセスがない場合にのみ変更されるようにしています。

6.4.3.ファイルリネーム時のデータ保証改善

通常の FAT ファイルシステムでは、ファイルリネーム時に電源断が発生すると、旧ファイル、および新ファイルの両方のディレクトリエントリが消滅してしまう現象が発生していました。

```
a: (void) pcfm_delete (fp) ;          /* delete old dir entry */
b:nfp->mode &= ~MODE_ALTERED;     /* don't change file time */
c: (void) update_dir (nfp) ;
d:UNLOCK_FILESYSTEM () ;
```

図 6.1 リネーム処理（修正前）

この現象は、修正前のリネーム処理（*図 6.1*）は、旧ファイルのディレクトリエントリを削除（a行）した後、新ファイルのディレクトリエントリを追加（c行）しているために、発生する可能性がありました。

そこで、これを回避するために、現状のリネーム処理から図 3-5 で示す内容に修正することで、エントリが消滅しないようにしています。

```
a:nfp->mode &= ~MODE_ALTERED;     /* don't change file time */
b: (void) update_dir (nfp) ;
c: (void) pcfm_delete (fp) ;          /* delete old dir entry */
d:UNLOCK_FILESYSTEM () ;
```

図 6.2 リネーム処理（修正後）

図 6.2 の修正では、新ファイルのディレクトリエントリを追加（b行）した後、旧ファイルのディレクトリエントリを削除（c行）しているため、最悪の場合でも、旧ファイルのディレクトリエントリは残ることが保証され、下表のようにファイルの消滅を回避することができます。

電源断位置	旧ファイル	新ファイル
a	あり	なし
b	あり	なし
c	あり	あり
d	なし	あり

6.4.4.エントリ作成時の属性フラグ

エントリが作成される場合、まず空のエントリが作成されます。
この状態で電源断が発生し、ストレージデバイス上に旧ファイル・新ファイルの両エントリが存在する状態となった場合、どちらが正しいエントリかが判断できません。
これを回避するため、エントリの作成時に属性の上位から 2 ビット目を 1 に設定（0x40）し、属性・クラスタ番号・ファイルサイズ設定時にこのビットを 0 に戻すことで、正しいエントリを判断可能となるようにしています。

6.4.5.キャッシュデータ書き込み順の変更

現在キャッシュデータをストレージデバイスに書き込む際、キャッシュに格納された順序に関わらず、キャッシュエリアの先頭から書き込んでいます。このとき、本来のデータ書き込み順序と異なる場合が存在し、この時点で電源断が発生すると不正なエントリとして検出されない場合が存在します。

この現象を回避するため、キャッシュエリアの書き込み順を、キャッシュに格納された順にストレージデバイスに書き込むように変更しています。

6.4.6.設定

電源断対策の設定は、**3設定**における設定以外は、特に他の設定およびアプリケーションでの操作等の必要はありません。

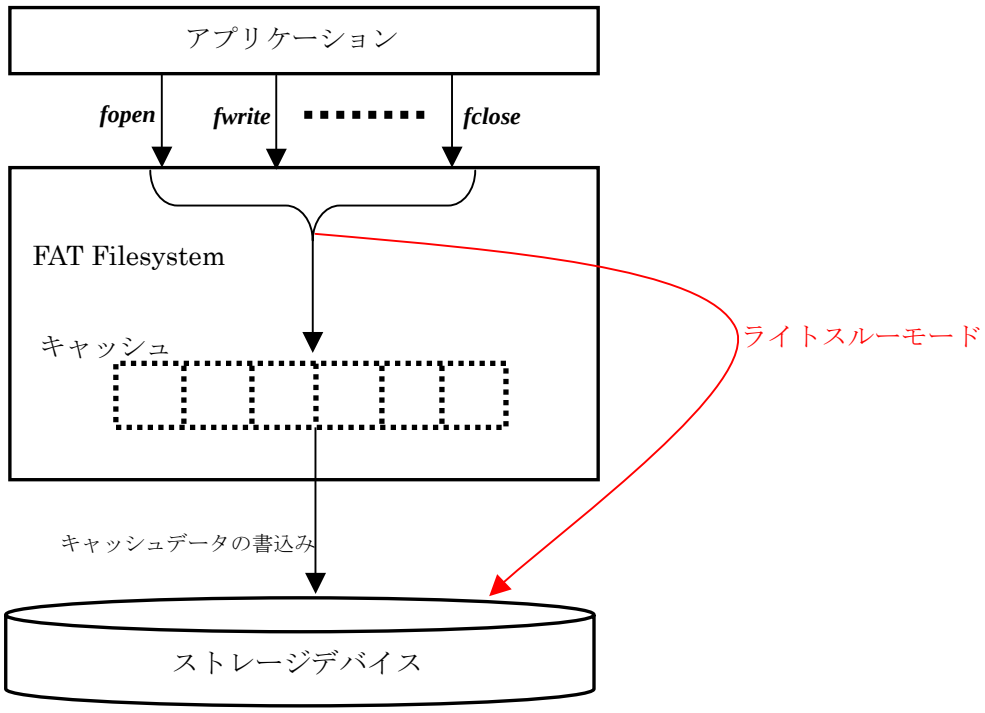
6.5. ライトスルーモード

6.5.1.概要

ファイルシステムには、アクセス速度を改善するためにキャッシュを使用しています。キャッシュデータは、以下タイミングでストレージデバイスに書き込まれます。

- ***fclose()***時
- ***fflush()***時

しかし、ストレージデバイスに書き込まれる前に電源断が発生すると、それまでにキャッシュに保持していたデータは破棄されてしまいます。
このようなデータ破棄を回避するために、キャッシュに格納するのではなく、直接ストレージデバイスに書き込むモードを追加しています。
ただし、アクセス速度が低下するため、本モードの処理を追加するかはコンパイル時に設定し、また、デバイスごとに使用するかを設定可能としています。



6.5.2.設定

ライトスルーモードを有効にするためにはまず、ライトスルーモードの処理をビルドする必要があります。ライトスルーモードの処理のビルドは **sio.mak** の **WRITETHROUGH** を設定してコンパイルすることで行います。

詳細につきましては**3.3Sio.makの設定**を参照ください。

ライトスルーモードを有効にするかどうかは、デバイスごとに設定することが可能で、デフォルトでは無効な設定となっています。

有効にするためには**<USFilesPlus インストールディレクトリ>¥siosrc** ディレクトリ下にある「**devtab.c**」のデバイステーブルで、有効にしたいデバイスの flag の値に **DVF_WTM** を設定します。

C ドライブでライトスルーモードを有効にしたい場合のデバイステーブルの例は以下の通りです。

```
    {
    &pcparmC,                   /* device dependent data */
    "C",                        /* name */
    FM_PCFM,                    /* device type = PC device */
    0xf,                        /* bits: text write read */
    0x84,                       /* unit# */
    0,                          /* partition */
    (DRIVER *)&lbadrv_s,        /* pointer to driver */
    &pcfm,                      /* pointer to file manager */
    NULL,                       /* pointer to FILE */
    DVF_WTM,                    /* flags */
    0                           /* # open paths (RAM) */
#ifdef USF_UTILITY
    ,
    0                           /* write counter for clean flag */
#endif
    }
```