

Backup Library for Maple Storage

VMSFS(仮称)

Version 1.06

Reference and Tutorial

1998/9/22 Rev.1.04

***(C) SEGA Enterprises,LTD.
SYSTEM R&D Dept. S.Uchida***

目次

第1章 バックアップライブラリ概要	
1.1 特徴	4
1.2 仕様	4
1.3 複数バックアップRAM の識別	4
1.4 バックアップRAM の容量	4
1.5 関数種別	5
第2章 一般アプリケーションへの組み込み	
2.1 バックアップライブラリの組み込み	6
2.2 バックアップライブラリの組み込みに関する関数の説明	8
第3章 一般アプリケーションでのファイルアクセス	
3.1 バックアップRAM の接続確認	11
3.2 バックアップRAM 情報の取得	11
3.2.1 フォーマット状態	11
3.2.2 空き容量	11
3.2.3 最大容量その他の情報	12
3.2.4 ファイルの存在確認	12
3.2.5 バックアップRAM のアクセス状態	12
3.3 ファイルのセーブ	13
3.3.1 セーブのリクエスト	13
3.3.2 セーブの終了判定	13
3.3.3 セーブに要する時間	13
3.4 ファイルのロード	14
3.4.1 ロードのリクエスト	14
3.4.2 ロードの終了判定	14
3.4.3 ロードに要する時間	14
3.5 ファイルの削除	15
3.5.1 削除のリクエスト	15
3.5.2 削除の終了判定	15
3.5.3 削除に要する時間	15
3.6 コールバック関数	16
3.6.1 完了コールバック	16
3.6.2 バックアップRAM の接続と切断	17
3.6.3 経過コールバック	17
3.6.4 コールバックに関する注意事項	17
3.6 エラー処理	18
3.7 情報取得に関する関数の説明	18
3.8 ファイルアクセスに関する関数の説明	22
第4章 特殊アプリケーションでのファイルアクセス	
4.1 ディレクトリ情報の取得	25
4.2 ファイルのコピー	26
4.3 ファイルの一部読み込み	27
4.4 バックアップRAM のフォーマット	27
4.5 ファイルアクセスに関する関数の説明	28

第5章 実行ファイル

5.1 実行ファイルの特徴	31
5.2 実行ファイルの存在確認	31
5.3 実行ファイル用空き容量の取得	31
5.4 実行ファイル用空き容量の確保	32
5.5 実行ファイルのセーブ	33
5.6 実行ファイルの部分更新	33
5.7 実行ファイルに関する関数の説明	34

第6章 バックアップファイルフォーマット

6.1 ファイルの外部インタフェース	37
6.2 ファイルの内部フォーマット	39
6.3 バックアップファイルユーティリティ関数	46
6.4 バックアップファイルユーティリティ関数の説明	48
6.5 VMS ボリュームアイコン	50

第7章 新機能「動的ワーク確保」

7.1 動的ワーク確保の使用方法	53
7.1.1 初期化	53
7.1.2 バックアップRAM の接続	53
7.1.3 ワークの確保とドライブのマウント	53
7.1.4 ドライブのアンマウントとワークの開放	54
7.1.5 バックアップRAM の切断	54
7.2 動的ワーク確保に関する関数の説明	55

第1章 バックアップライブラリ概要

1.1 特徴

本ライブラリは、コントロールパッド等に接続される VMS をはじめとする Maple ストレージメディアへのファイルアクセスを可能にする忍拡張ライブラリです。

1.2 仕様

本ライブラリの仕様は以下のとおりです。また、この仕様は将来変更の可能性があります。

項目	内容
対応ハードウェア	Maple Bus 仕様ストレージファンクションデバイス
対応デバイス数	8

表 1-1.仕様

1.3 複数のバックアップRAM の識別

Dreamcast では、ポート A～ポート D のコントローラにそれぞれ最大2個までのバックアップ RAM が接続可能です。合計で最大8個までのバックアップ RAM が接続できます。本ライブラリでは、初期化時に指定した任意のドライブにアクセス可能です。

ドライブ番号	ドライブ指定定数	接続場所
0	BUD_DRIVE_A1	ポート A のコントローラの拡張コネクタ 1 に接続されたバックアップ RAM
1	BUD_DRIVE_A2	ポート A のコントローラの拡張コネクタ 2 に接続されたバックアップ RAM
2	BUD_DRIVE_B1	ポート B のコントローラの拡張コネクタ 1 に接続されたバックアップ RAM
3	BUD_DRIVE_B2	ポート B のコントローラの拡張コネクタ 2 に接続されたバックアップ RAM
4	BUD_DRIVE_C1	ポート C のコントローラの拡張コネクタ 1 に接続されたバックアップ RAM
5	BUD_DRIVE_C2	ポート C のコントローラの拡張コネクタ 2 に接続されたバックアップ RAM
6	BUD_DRIVE_D1	ポート D のコントローラの拡張コネクタ 1 に接続されたバックアップ RAM
7	BUD_DRIVE_D2	ポート D のコントローラの拡張コネクタ 2 に接続されたバックアップ RAM

表 1-2.ドライブ番号

1.4 バックアップRAM の容量

本ライブラリでは、初期化時に指定した最大容量までのバックアップ RAM に対応します。どのような容量のバックアップ RAM が存在するかは現在のところ未定となっています。

現在次のようなバックアップ RAM があります。

名称	容量	最大ファイル数
VMS(ビジュアルメモリ)	128KB	200

表 1-3.バックアップ RAM の容量

1.6 関数種別

本ライブラリには、TYPE AとTYPE Bの2種類の関数があり、以下のような特徴があります。

タイプ	説明
TYPE A	<ul style="list-style-type: none">・即時完了復帰するタイプ。buStat()によってステータスを取得する必要はなく、コールバックもありません。・このタイプの関数の終了直後には、他のファイルシステム関数を呼び出すことができます。・このタイプの関数は主に、バックアップ RAM 接続時にキャッシュされる情報を参照するだけのものが多いため、即時完了復帰が可能です。
TYPE B	<ul style="list-style-type: none">・即時復帰しますが、その時点で処理は終了しておらず、buStat()によってステータスを毎フレーム取得して終了を待つ必要があります。・ステータスが終了を示さない限り、同じドライブに対してファイルシステム関数を呼び出すことはできません。呼び出した場合はエラーを返します。・処理終了時に指定の完了コールバック関数をコールバックします。・処理中に経過コールバック関数をコールバックします。・このタイプの関数の返値は、BUD_ERR_OK または BUD_ERR_BUSY です。・引数で示されるアドレスにデータを取得する関数(buLoadFile()など)の場合、ステータスが完了を示すまでは有効データが格納されないので注意してください。・引数で示されるアドレスのデータをバックアップ RAM に書き込む関数(buSaveFile()等)の場合、ステータスが完了を示すまでデータの内容を保証してください。

第2章

一般アプリケーションへの組み込み

本ライブラリには多くの関数が用意されていますが、それらのほとんどは BOOT ROM や一部のファイルユーティリティ的な機能を含むアプリケーションのみが使用することを想定されています。一般的なゲームアプリケーションの場合であれば、データのロード、セーブ、状態取得など一部の関数を使用するだけでバックアップ RAM へのファイルアクセスが容易に行えます。

2.1 バックアップライブラリの組み込み

1. バックアップライブラリの初期化

バックアップライブラリの初期化には、ライブラリのワークエリアの先頭アドレス、対応するドライブ数、最大容量、そして初期化完了コールバック関数が必要です。

```
#include <shinobi.h>
#include <sg_bup.h>

#define MAX_DRIVES 8

Uint32 bupWork[BUM_WORK_SIZE(BUD_CAPACITY_128KB, MAX_DRIVES) / sizeof(Uint32)];

static void init_callback(void)
{
}

void BackupInit(void)
{
    buInit(BUD_USE_DRIVE_ALL, BUD_CAPACITY_128KB, bupWork, init_callback);
}

void BackupExit(void)
{
    buExit();
}
```

この例では 8 個すべてのドライブに対応し、それぞれ 128KB までのバックアップ RAM に対応するよう初期化しています。

コントローラ A に接続されたバックアップ RAM(2 個)のみに対応する場合は、以下のようになります。

```
#define MAX_DRIVES 2

void BackupInit(void)
{
    buInit(BUD_USE_DRIVE_A1 | BUD_USE_DRIVE_A2, BUD_CAPACITY_128KB,
          bupWork, init_callback);
}
```

初期化時に指定されなかったドライブにバックアップ RAM が接続されても認識しません。しかし、アプリケーションで対応する必要のないドライブはなるべく指定しないでください。それによりワーク容量とライブラリの負荷が軽減されます。

2.コールバック関数の登録

必要であれば、ファイルアクセス等の処理完了時、経過時のコールバック関数を登録することが可能です。それには以下のようにします。

```
Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    return BUD_CBRET_OK;
}

Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Sint32 max)
{
    return BUD_CBRET_OK;
}

static void init_callback(void)
{
    buSetCompleteCallback(complete_callback);
    buSetProgressCallback(progress_callback);
}
```

例のように、処理完了時に呼び出したい関数を buSetCompleteCallback()で、一定処理経過時に呼び出したい関数を buSetProgressCallback()で、それぞれ登録します。登録は先ほどの初期化コールバック関数中で行ってください。この例では、コールバック関数では何もせずにただ BUD_CBRET_OK を返しています。

コールバックを利用することにより、ロード、セーブの終了やバックアップ RAM の接続、取り外しを容易に検出することができます。また、長時間かかる処理の進行具合も知ることができます。

コールバック関数の詳細については次章で解説します。

2.2 バックアップライブラリの組み込みに関する関数の説明

関数	機能	タイプ
bulnit	ライブラリを初期化する	A
buExit	ライブラリを終了する	A
buSetCompleteCallback	処理完了時のコールバック関数を登録する	A
buSetProgressCallback	一定処理経過時のコールバック関数を登録する	A
BU_INITCALLBACK	初期化終了時のコールバック関数型	-
BU_COMPLETECALLBACK	処理完了時のコールバック関数型	-
BU_PROGRESSCALLBACK	一定処理経過時のコールバック関数型	-

表 2-1.関数一覧

・API

bulnit

ライブラリ初期化

TYPE A

書式

Sint32 bulnit(Sint32 capacity, Sint32 driveflag, void* work, BU_INITCALLBACK func)

パラメータ

capacity

対応ストレージメディア最大容量

BUD_CAPACITY_128KB

BUD_CAPACITY_256KB

BUD_CAPACITY_512KB

BUD_CAPACITY_1MB

driveflag

対応ドライブ指定

BUD_USE_DRIVE_ALL

すべてのドライブ

BUD_USE_DRIVE_A1

ポート A の1番ドライブ

BUD_USE_DRIVE_A2

ポート A の2番ドライブ

BUD_USE_DRIVE_B1

ポート B の1番ドライブ

BUD_USE_DRIVE_B2

ポート B の2番ドライブ

BUD_USE_DRIVE_C1

ポート C の1番ドライブ

BUD_USE_DRIVE_C2

ポート C の2番ドライブ

BUD_USE_DRIVE_D1

ポート D の1番ドライブ

BUD_USE_DRIVE_D2

ポート D の2番ドライブ

work

ワークバッファ(4バイト境界)

func

初期化終了コールバック

戻り値

BUD_ERR_OK

正常終了

BUD_ERR_INVALID_PARAM

パラメータ異常

機能

・ファイルシステムを初期化します。

・初期化に成功した場合、指定の関数をコールバックします。コールバック関数は bulnit() からリターンする前に呼び出されます。

参照備考

BUM_WORK_SIZE マクロ、BU_INITCALLBACK

・最大容量やドライブ数によって、必要なワーク容量が大幅に異なります。アプリケーションで対応する必要のない容量のバックアップ RAM やドライブはなるべく指定しないようにしてください。

・不要なドライブは指定しないことにより、ワーク容量の削減と、ライブラリの高速度化が可能です。

・以下に、ワーク容量の目安を記します。

最大容量	1ドライブあたりのワーク容量	8ドライブでのワーク容量
128KB	8KB	64KB
256KB	14KB	112KB
512KB	28KB	220KB
1MB	55KB	440KB

・使用ドライブ指定には BUD_USE_DRIVE_XXX 定数を必ず指定してください。BUD_DRIVE_XXX 定数を誤って指定することのないようにしてください。

例

```
/* 128KB までのバックアップ RAM 2 個分のバッファを確保
Uint32 work[BUM_WORK_SIZE(BUD_CAPACITY_128KB, 2) / sizeof(Uint32)];

void init_callback(void)
{
}

/* ポート A の1番、2番ドライブに対応する */
buInit(BUD_CAPACITY_128KB,
        BUD_USE_DRIVE_A1 | BUD_USE_DRIVE_A2,
        work, init_callback);
```

buExit

ライブラリ終了

TYPE A

書式	Sint32 buExit(void)
パラメータ	なし
戻り値	BUD_ERR_OK 正常終了 BUD_ERR_BUSY 何らかの TYPE B 関数が処理中である
機能	・ファイルシステムを終了します。
参照	
備考	・何らかのファイル処理が行われている場合、終了することができないため、BUD_ERR_BUSY を返します。
例	do { } while(buExit() != BUD_ERR_OK);

buSetCompleteCallback

TYPE A

書式	void buSetCompleteCallback(BU_COMPLETECALLBACK func)
パラメータ	func 完了コールバック関数アドレス
戻り値	なし
機能	・処理完了時のコールバック関数を指定します。
参照	
備考	・初期化終了コールバック関数中で呼び出してください。
例	<pre>Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 status, Uint32 param) { return BUD_CBRET_OK; } Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Uint32 max) { return BUD_CBRET_OK; } void init_callback(void) { buSetCompleteCallback(complete_callback); buSetProgressCallback(progress_callback); }</pre>

buSetProgressCallback

TYPE A

書式	void buSetProgressCallback(BU_PROGRESSCALLBACK func)
パラメータ	func 経過コールバック関数アドレス

戻り値	なし
機能	・処理中の経過コールバック関数を指定します。
参照	
備考	・初期化終了コールバック関数中で呼び出してください。
例	buSetCompleteCallback の例を参照してください。

BU_COMPLETECALLBACK

コールバック関数

書式	typedef Sint32 (*BU_COMPLETECALLBACK)(Sint32 drive, Sint32 op, Sint32 status, Uint32 param)
パラメータ	drive ドライブ番号 op オペレーションコード status ステータス param ユーザーパラメータ
戻り値	BUD_CBRET_OK を返してください。
機能	・
参照	
備考	
例	<pre>Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 status, Uint32 param) { switch (op) { case BUD_OP_FORMATDISK: printf("フォーマットが終了しました。¥n"); break; case BUD_OP_MOUNT: printf("ドライブ%d にバックアップ RAM が接続されました。¥n"); break; : } return BUD_CBRET_OK; }</pre>

BU_PROGRESSCALLBACK

コールバック関数

書式	typedef Sint32 (*BU_PROGRESSCALLBACK)(Sint32 drive, Sint32 op, Sint32 count, Sint32 max)
パラメータ	drive ドライブ番号 op オペレーションコード count 進行状況カウント max 進行状況最大値
戻り値	BUD_CBRET_OK を返してください。
機能	・
参照	
備考	・進行状況カウントはあくまでも目安です。カウントと最大値を比較して終了を判定するなどの処理は行わないでください。 ・進行状況カウントは最大値を超える場合があります。
例	<pre>Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Sint32 max) { switch (op) { case BUD_OP_FORMATDISK: printf("フォーマット中です ...%d/%d¥n", count, max); break; : } }</pre>

第3章

一般アプリケーションでのファイルアクセス

本ライブラリには多くの関数が用意されていますが、それらのほとんどは BOOT ROM や一部のファイルユーティリティ的な機能を含むアプリケーションのみが使用することを想定されています。一般的なゲームアプリケーションの場合であれば、データのロード、セーブ、状態取得など一部の関数を使用するだけでバックアップ RAM へのファイルアクセスが容易に行えます。

3.1 バックアップRAM の接続確認

バックアップ RAM が接続されているかどうかを調べるには、関数 `buIsReady()` を使用します。

```
if (buIsReady(BUD_DRIVE_A1)) {  
    /* バックアップ RAM が接続されている */  
}
```

3.2 バックアップRAM の情報取得

3.2.1 フォーマット状態

バックアップ RAM がフォーマットされているかどうかを調べるには、関数 `buIsFormatDisk()` を使用します。

```
switch (buIsFormatDisk(BUD_DRIVE_A1)) {  
    case BUD_ERR_OK:  
        /* フォーマットされている */  
        break;  
    case BUD_ERR_NO_DISK:  
        /* バックアップ RAM が接続されていない */  
        break;  
    case BUD_ERR_BUSY:  
        /* バックアップ RAM は BUSY 状態のため調べられない */  
        break;  
}
```

フォーマットされていない場合、フォーマット以外の処理をすることはできません。

3.2.2 空き容量

バックアップの空き容量を調べるには、関数 `buGetDiskFree()` を使用します。

```
Sint32 free;  
  
free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_NORMAL);  
  
if (free < 0) return NG;
```

空き容量の取得に成功すれば、free には 0 以上の値が返されます。これはそのバックアップ RAM の空きブロック数となります。(1 ブロックは 5 1 2 バイト)
エラーが発生した場合には、負の値が返されます。

3.2.3 最大容量その他の情報

バックアップの情報は、関数 buGetDiskInfo() を使用しても調べることができます。
BUS_DISKINFO 構造体の詳細は構造体リファレンスを参照してください。

```
BUS_DISKINFO info;
Sint32 ret;

ret = buGetDiskInfo(BUD_DRIVE_A1, &info);

if (ret < 0) return NG;
```

3.2.4 ファイルの存在確認

ファイルアクセスの前に、指定のファイルが存在するかどうかを調べることができます。

```
Sint32 ret;

ret = buIsExistFile(BUD_DRIVE_A1, "SAVEDATA_001");

switch (ret) {
    case BUD_ERR_OK:
        /* ファイルがある */
        break;
    case BUD_ERR_FILE_NOT_FOUND:
        /* ファイルなし */
        break;
    default:
        /* その他のエラー */
        break;
}
```

3.2.5 バックアップ RAM のアクセス状態

バックアップ RAM にアクセス中(何らかのファイルアクセスやフォーマット等の処理中)かどうかを調べるには次のようにします。

```
if (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

処理中である場合には、そのドライブに対して他の処理をすることはできません。他の処理を行う関数を呼び出した場合はエラーが返ります。

3.3 ファイルのセーブ

3.3.1 セーブのリクエスト

ファイルをセーブするには、関数 buSaveFile()を使用します。

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
extern UInt8 SaveData[];    /* セーブするデータ(512*10 バイト) */

blocks = 10;                /* ファイルサイズは10 ブロック */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00);
/* ベリファイを行う | コピーフラグを 00H に設定する */

time.year = 1998;          /* タイムスタンプの設定 */
time.month = 6;            /* 1998/6/25(木) 23:59:59 */
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks, &time, flag);

if (ret == BUD_ERR_OK) {
    /* セーブリクエストに成功した */
} else {
    /* セーブリクエストに失敗した(BUSY) */
}
```

3.3.2 セーブの終了判定とエラーチェック

ファイルセーブが完了したかどうかは、関数 buStat()を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

セーブ時にエラーが発生したかどうかは、関数 buGetLastError()を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {
    /* セーブ成功 */
} else {
    /* エラー発生 */
}
```

3.3.3 セーブに要する時間

ファイルセーブに要する時間は、VMS の場合ほぼ以下の式で計算できます。

$$\text{time(フレーム数)} = (\text{ブロック数} + 2) * 5$$

上記の例では、10 ブロックセーブしていますので、約 60 フレーム(1秒)かかること

になります。ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無、エラーの有無により変化することがあります。また、ベリファイを行う場合はデータの読み出し処理を行うため、セーブ時間は多少長くなります。

3.4 ファイルのロード

3.4.1 ロードのリクエスト

ファイルをロードするには、関数 `buLoadFile()` を使用します。

```
Sint32 ret;
Sint32 blocks;
extern Uint8 SaveData[];    /* ロードバッファ */

blocks = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA"); /* ファイルサイズを取得 */

if (blocks <= 0) return NG;
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks);

if (ret == BUD_ERR_OK) {
    /* ロードリクエストに成功した */
} else {
    /* ロードリクエストに失敗した(BUSY) */
}
```

3.4.2 ロードの終了判定とエラーチェック

ファイルロードが完了したかどうかは、関数 `buStat()` を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

ロード時にエラーが発生したかどうかは、関数 `buGetLastError()` を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {
    /* ロード成功 */
} else {
    /* エラー発生 */
}
```

3.4.3 ロードに要する時間

ファイルロードに要する時間は、VMS の場合ほぼ以下の式で計算できます。

time(フレーム数) = ブロック数

上記の例では、10ブロックロードしていますので、約 10 フレーム(167 ミリ秒)かかることになります。ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無、エラーの有無により変化することがあります。

3.5 ファイルの削除

3.5.1 削除のリクエスト

ファイルを削除するには、関数 `buDeleteFile()` を使用します。

```
Sint32 ret;

ret = buDeleteFile(BUD_DRIVE_A1, "SAVEDATA");

if (ret == BUD_ERR_OK) {
    /* 削除リクエストに成功した */
} else {
    /* 削除リクエストに失敗した(BUSY) */
}
```

3.5.2 削除の終了判定

ファイル削除が完了したかどうかは、関数 `buStat()` を使用して調べます。

```
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
    /* 処理中 */
}
```

削除時にエラーが発生したかどうかは、関数 `buGetLastError()` を使用して調べます。

```
if (buGetLastError(BUD_DRIVE_A1) == BUD_ERR_OK) {
    /* 削除成功 */
} else {
    /* エラー発生 */
}
```

3.5.3 削除に要する時間

ファイル削除に要する時間は、VMS の場合はほぼ5フレームとなります。ただし、この時間はあくまでも予想時間であり、実際には他のドライブのアクセスの有無、エラーの有無により変化することがあります。

3.6 コールバック関数

ロードやセーブをはじめとして、時間を要する処理のバックグラウンド処理を容易にするために、コールバックをサポートしています。

3.6.1 完了コールバック

完了コールバックはロード、セーブなどの処理が完了した時点で、ユーザーがあらかじめ登録しておいた関数が呼び出されるものです。完了コールバック関数は、例えば以下のように記述します。

```
Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    switch (op) {
        case BUD_OP_MOUNT:
            printf("バックアップ RAM%d が接続されました。¥n", drive);
            break;
        case BUD_OP_UNMOUNT:
            printf("バックアップ RAM%d が取り外されました。¥n", drive);
            break;
        case BUD_OP_LOADFILE:
            printf("ロードが終了しました。¥n");
            break;
        case BUD_OP_SAVEFILE:
            printf("セーブが終了しました。¥n");
            break;
        case BUD_OP_DELETEFILE:
            printf("削除が終了しました。¥n");
            break;
    }

    return BUD_CBRET_OK;
}
```

パラメータ drive にはドライブ番号が、op には下表のオペレーションコードが渡されます。オペレーションコードにはこれ以外にも、各関数に対応したものがあります。

オペレーションコード	処理	対応関数
BUD_OP_MOUNT	バックアップ RAM の接続	バックアップ RAM の接続、buRemount()
BUD_OP_UNMOUNT	バックアップ RAM の切断	バックアップ RAM の取り外し
BUD_OP_DELETEFILE	ファイルの削除	buDeleteFile()
BUD_OP_LOADFILE	ファイルのロード	buLoadFile()
BUD_OP_SAVEFILE	ファイルのセーブ	buSaveFile()

また、その処理が成功したかどうか、何らかのエラーが発生したかどうかなどの完了ステータスが stat に渡されます。どの処理がどういう完了ステータスを取りうるのかなど、詳細は関数リファレンスを参照してください。

param にはユーザーが指定したパラメータが渡されます。

コールバック関数中で、ファイルシステム関数を呼び出すことができます。これにより、例えばバックアップ RAM の接続コールバック関数中で、buGetDiskInfo()等の関数を使用して情報を取得したり、ファイルの存在を確認したり、buLoadFile()を使用して自動的にファイルを読み込む等の処理が可能です。

3.6.2 バックアップRAM の接続と切断

完了コールバックを利用して、バックアップ RAM の接続と取り外しを容易に検出することができます。

```
Uint32 gBackupRAMReady[8];

Sint32 complete_callback(Sint32 drive, Sint32 op, Sint32 stat, Uint32 param)
{
    switch (op) {
        case BUD_OP_MOUNT:
            if (buIsFormatDisk(drive) == BUD_ERR_OK) {
                gBackupRAMReady[drive] = TRUE;
            }
            break;
        case BUD_OP_UNMOUNT:
            gBackupRAMReady[drive] = FALSE;
            break;
    }

    return BUD_CBRET_OK;
}
```

3.6.3 経過コールバック

経過コールバックを利用すると、どの程度処理が経過したかを容易に知ることができます。

```
Uint32 gProgress[8];

Sint32 progress_callback(Sint32 drive, Sint32 op, Sint32 count, Sint32 max)
{
    gProgress[drive] = count * 100 / max;
    printf("%d %% completed.\n", gProgress[drive]);

    return BUD_CBRET_OK;
}
```

3.6.4 コールバックに関する注意事項

- ・コールバック関数は、Maple 割り込みをトリガとして(割り込みハンドラとして)呼び出されます。複数プロセス(スレッド)からの呼び出しに対応していないような他のライブラリ関数、アプリケーション関数を呼び出さないようにしてください。
- ・ファイルアクセス中にバックアップ RAM が取り外された場合、まず現在処理中の処理に対して、完了コールバックが発生します。その際の完了ステータスは BUD_ERR_NO_DISK となります。引き続いて同一割り込み中にバックアップ RAM 切断コールバック(BUD_OP_UNMOUNT)が発生します。

3.7 エラー処理

ファイルアクセスには、予期せぬエラーが発生することがあります。今回バックアップ RAM がリムーバブルメディアであるため、ロードやセーブ中にバックアップ RAM が抜かれたりすることが予想されます。

3.8 情報取得に関する関数の説明

関数	機能	タイプ
bulsReady	バックアップ RAM が接続されているかどうかを調べる	A
bulsFormat	バックアップ RAM がフォーマット済みかどうかを調べる	A
buGetDiskFree	空き容量を調べる	A
buGetDiskInfo	バックアップ RAM の各種情報を取得する	A
bulsExistFile	指定ファイルが存在するかどうかを調べる	A
buGetFileSize	指定ファイルのサイズを調べる	A
buGetFileInfo	指定ファイルの各種情報を取得する	A
buStat	処理が完了しているかどうかを調べる	A
buGetLastError	最後に発生したエラーを取得する	A

表 3-1.関数一覧

構造体	機能
BUS_DISKINFO	バックアップ RAM の情報を格納する
BUS_FILEINFO	ファイルの情報を格納する
BUS_TIME	タイムスタンプを格納する

表 3-2.構造体一覧

•API

bulsReady

TYPE **A**

書式 Sint32 bulsReady(Sint32 drive)
 パラメータ drive ドライブ番号
 戻り値 TRUE ドライブが接続されていて、アクセス可能である
 FALSE ドライブが接続されていない
 機能
 参照
 備考
 例
 ・ドライブの接続状況を返します。

bulsFormat

TYPE **A**

書式 Sint32 bulsFormat(Sint32 drive)
 パラメータ drive ドライブ番号
 戻り値 BUD_ERR_OK フォーマットされている
 BUD_ERR_UNFORMAT フォーマットされていない
 BUD_ERR_NO_DISK ディスクがない
 BUD_ERR_BUSY 何らかの TYPE B 関数が処理中である
 機能
 参照
 備考
 例
 ・指定ドライブのディスクがフォーマット済みかどうかを調べます。

```
buIsFormat ( BUD_DRIVE_A1 );
```

buGetDiskFree

TYPE **A**

書式	Sint32 buGetDiskFree(Sint32 drive, Sint32 type)
パラメータ	drive ドライブ番号 type 調べる空きブロック数のタイプ BUD_FILETYPE_NORMAL 通常ファイル空き容量 BUD_FILETYPE_EXECUTABLE 実行ファイル空き容量
戻り値	0 あるいは正 空きブロック数 BUD_ERR_UNFORMAT フォーマットされていない BUD_ERR_NO_DISK ディスクがない BUD_ERR_BUSY 何らかの TYPE B 関数が処理中である
機能	・指定ドライブの空き容量をブロック数で取得します。 ・通常ファイル空き容量には、実行ファイル空き容量も含まれます。
参照 備考 例	 Sint32 free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_NORMAL);

buGetDiskInfoTYPE **A**

書式	Sint32 buGetDiskInfo(Sint32 drive, BUS_DISKINFO* info)
パラメータ	drive ドライブ番号 info ディスク情報構造体のアドレス
戻り値	BUD_ERR_OK 正常終了 BUD_ERR_UNFORMAT フォーマットされていない BUD_ERR_NO_DISK ディスクがない BUD_ERR_BUSY 何らかの TYPE B 関数が処理中である
機能	・指定ドライブのディスク情報を取得します。
参照 備考 例	BUS_DISKINFO 構造体 BUS_DISKINFO info; buGetDiskInfo(BUD_DRIVE_A1, &info);

buIsExistFile

ファイル関数

TYPE **A**

書式	Sint32 buIsExistFile(Sint32 drive, const char* fname)
パラメータ	drive ドライブ番号 fname ファイル名
戻り値	BUD_ERR_OK ファイルあり BUD_ERR_FILE_NOT_FOUND ファイルなし BUD_ERR_UNFORMAT フォーマットされていない BUD_ERR_NO_DISK ディスクがない BUD_ERR_BUSY 何らかの TYPE B 関数が処理中である
機能	・指定ファイルが存在するかどうかを調べます
参照 備考 例	 switch (buIsExistFile(BUD_DRIVE_A1, "test.bin")) { case BUD_ERR_OK: /* ファイルあり */ break; case BUD_ERR_FILE_NOT_FOUND: /* ファイルなし */ break; default: /* その他のエラー */ break; }

buGetFileSize

ファイル関数

TYPE **A**

書式	Sint32 buGetFileSize(Sint32 drive, const char* fname)
----	---

パラメータ	drive	ドライブ番号
	fname	ファイル名
戻り値	0 あるいは正	空きブロック数
	BUD_ERR_FILE_NOT_FOUND	ファイルがない
	BUD_ERR_UNFORMAT	フォーマットされていない
	BUD_ERR_NO_DISK	ディスクがない
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である
機能	・ファイルのサイズをブロック数で取得します。	
参照		
備考		
例	<pre>Sint32 size = buGetFileSize(BUD_DRIVE_A1, "SAVEDATA_001");</pre>	

buGetFileInfo		ファイル関数	TYPE A
書式	Sint32 buGetFileInfo(Sint32 drive, const char* fname, BUS_FILEINFO* info)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
	info	ファイル情報構造体のアドレス	
戻り値	BUD_ERR_OK	正常終了	
	BUD_ERR_FILE_NOT_FOUND	ファイルがない	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である	
機能	・ファイルの情報を取得します。		
参照			
備考			
例	<pre>BUS_FILEINFO info; buGetFileInfo(BUD_DRIVE_A1, "SAVEDATA_001", &info);</pre>		

buStat		TYPE A
書式	Sint32 buStat(Sint32 drive)	
パラメータ	drive	ドライブ番号
戻り値	BUD_STAT_READY	処理は終了している
	BUD_STAT_BUSY	何らかの TYPE B 関数が処理中である
機能	・処理が終了しているかどうかを調べます。	
参照		
備考		
例	<pre>buLoadFile(BUD_DRIVE_A1, file, buf, 1); while (buStat(BUD_DRIVE_A1) != BUD_STAT_BUSY); if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;</pre>	

buGetLastError		TYPE A
書式	Sint32 buGetLastError(Sint32 drive)	
パラメータ	drive	ドライブ番号
戻り値	エラーコード	
機能	・指定ドライブで最後に発生したエラーを調べます。	
参照		
備考		
例	buStat() の例を参照してください。	

・構造体

BUS_DISKINFO

構造体

定義	typedef struct { Sint8 volume[32]; Uint16 total_blocks; Uint16 total_user_blocks; Uint16 free_blocks; Uint16 free_user_blocks; Uint16 total_exe_blocks; Uint16 free_exe_blocks; Uint16 block_size; Uint16 icon_no; BUS_TIME time; } BUS_DISKINFO;																				
説明	バックアップ RAM の情報を格納する構造体です。																				
メンバ	<table> <tr><td>volume</td><td>ボリュームデータ</td></tr> <tr><td>total_blocks</td><td>全ブロック数</td></tr> <tr><td>total_user_blocks</td><td>全ユーザーブロック数(最大ファイル数)</td></tr> <tr><td>free_blocks</td><td>空きブロック数</td></tr> <tr><td>free_user_blocks</td><td>空きユーザーブロック数</td></tr> <tr><td>total_exe_blocks</td><td>全実行ファイル用ブロック数</td></tr> <tr><td>free_exe_blocks</td><td>空き実行ファイル用ブロック数</td></tr> <tr><td>block_size</td><td>ブロックサイズ(512 固定)</td></tr> <tr><td>icon_no</td><td>アイコン番号(0 ~ 255)</td></tr> <tr><td>time</td><td>フォーマットされた時刻</td></tr> </table>	volume	ボリュームデータ	total_blocks	全ブロック数	total_user_blocks	全ユーザーブロック数(最大ファイル数)	free_blocks	空きブロック数	free_user_blocks	空きユーザーブロック数	total_exe_blocks	全実行ファイル用ブロック数	free_exe_blocks	空き実行ファイル用ブロック数	block_size	ブロックサイズ(512 固定)	icon_no	アイコン番号(0 ~ 255)	time	フォーマットされた時刻
volume	ボリュームデータ																				
total_blocks	全ブロック数																				
total_user_blocks	全ユーザーブロック数(最大ファイル数)																				
free_blocks	空きブロック数																				
free_user_blocks	空きユーザーブロック数																				
total_exe_blocks	全実行ファイル用ブロック数																				
free_exe_blocks	空き実行ファイル用ブロック数																				
block_size	ブロックサイズ(512 固定)																				
icon_no	アイコン番号(0 ~ 255)																				
time	フォーマットされた時刻																				
参照	buGetDiskinfo(), BUS_TIME																				

BUS_FILEINFO

構造体

定義	typedef struct { Uint32 filesize; Uint16 blocks; Uint8 type; Uint8 copyflag; Uint16 headerofs; BUS_TIME time; } BUS_FILEINFO;																		
説明	ファイルの情報を格納する構造体です。																		
メンバ	<table> <tr><td>filesize</td><td>ファイルサイズ(バイト数)</td></tr> <tr><td>blocks</td><td>使用ブロック数</td></tr> <tr><td>type</td><td>ファイルタイプ</td></tr> <tr><td></td><td> <table> <tr><td>BUD_FILETYPE_NORMAL</td><td>通常ファイル</td></tr> <tr><td>BUD_FILETYPE_EXECUTABLE</td><td>実行ファイル</td></tr> </table> </td></tr> <tr><td>copyflag</td><td>コピーフラグ(FFH=コピー不可)</td></tr> <tr><td>headerofs</td><td>ヘッダオフセット(0 固定)</td></tr> <tr><td>time</td><td>タイムスタンプ</td></tr> </table>	filesize	ファイルサイズ(バイト数)	blocks	使用ブロック数	type	ファイルタイプ		<table> <tr><td>BUD_FILETYPE_NORMAL</td><td>通常ファイル</td></tr> <tr><td>BUD_FILETYPE_EXECUTABLE</td><td>実行ファイル</td></tr> </table>	BUD_FILETYPE_NORMAL	通常ファイル	BUD_FILETYPE_EXECUTABLE	実行ファイル	copyflag	コピーフラグ(FFH=コピー不可)	headerofs	ヘッダオフセット(0 固定)	time	タイムスタンプ
filesize	ファイルサイズ(バイト数)																		
blocks	使用ブロック数																		
type	ファイルタイプ																		
	<table> <tr><td>BUD_FILETYPE_NORMAL</td><td>通常ファイル</td></tr> <tr><td>BUD_FILETYPE_EXECUTABLE</td><td>実行ファイル</td></tr> </table>	BUD_FILETYPE_NORMAL	通常ファイル	BUD_FILETYPE_EXECUTABLE	実行ファイル														
BUD_FILETYPE_NORMAL	通常ファイル																		
BUD_FILETYPE_EXECUTABLE	実行ファイル																		
copyflag	コピーフラグ(FFH=コピー不可)																		
headerofs	ヘッダオフセット(0 固定)																		
time	タイムスタンプ																		
参照	buGetFileInfo(), BUS_TIME																		

BUS_TIME

構造体

定義	typedef struct { Uint16 year; Uint8 month;
----	--

	Uint8 day; Uint8 hour; Uint8 minute; Uint8 second; Uint8 dayofweek; } BUS_TIME;
説明	タイムスタンプを格納する構造体です。
メンバ	year 年(1998 ~) month 月(1 ~ 12) day 日(1 ~ 31) hour 時(0 ~ 23) minute 分(0 ~ 59) second 秒(0 ~ 59) dayofweek 曜日(日 = 0, 月 = 1, ... 土 = 6)
参照	BUS_DISKINFO, BUS_FILEINFO, buSaveFile()

3.9 ファイルアクセスに関する関数の説明

関数	機能	タイプ
buSaveFile	ファイルをセーブする	B
buLoadFile	ファイルをロードする	B
buDeleteFile	ファイルを削除する	B
buSetFileAttr	ファイル属性を変更する	B

表 3-3.関数一覧

buLoadFile		ファイル関数	TYPE B
書式	Sint32 buLoadFile(Sint32 drive, const char* fname, void* buf, Uint32 nblock)		
パラメータ	drive ドライブ番号 fname ファイル名 buf 読み込みアドレス(4バイト境界) nblock 読み込むブロック数		
戻り値	BUD_ERR_OK 処理要求を受け付けた BUD_ERR_BUSY 処理中のため要求を受け付けられなかった		
完了ステータス	BUD_ERR_OK 正常終了 BUD_ERR_NO_DISK ディスクがない BUD_ERR_UNFORMAT フォーマットされていない BUD_ERR_FILE_NOT_FOUND ファイルがない BUD_ERR_CANNOT_OPEN ファイルが開けない BUD_ERR_FILE_BROKEN ファイルが壊れている		
機能	・ファイルをロードします。 ・通常のアプリケーションであればこの関数でファイル全体をロードします。 ・読み込むブロック数に0を指定するとファイル全体をロードします。		
参照	・読み込みアドレスには必ず4バイト境界を指定してください。		
備考			
例	<pre> Sint32 ret; ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, 16); if (ret != BUD_ERR_OK) return NG; while (1) { if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break; } if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG; return OK; </pre>		

buSaveFile		ファイル関数	TYPE B
書式	Sint32 buSaveFile(Sint32 drive, const char* fname, const void* buf, Uint32 nblock, const BUS_TIME* time, Sint32 flag)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
	buf	セーブするデータアドレス(4バイト境界)	
	nblock	セーブするブロック数	
	time	タイムスタンプ	
	flag	フラグ	BUD_FLAG_VERIFY ベリファイあり BUD_FLAG_COPY(n) コピーフラグ
戻り値	BUD_ERR_OK	処理要求を受け付けた	
完了ステータス	BUD_ERR_BUSY	処理中のため要求を受け付けられなかった	
	BUD_ERR_OK	正常終了	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_DISK_FULL	ディスクフル	
	BUD_ERR_EXECFILE_EXIST	同名の実行ファイルが存在する	
	BUD_ERR_CANNOT_CREATE	ファイルが作成できない	
	BUD_ERR_VERIFY	ベリファイエラー	
機能	・ファイルをセーブします。 ・通常のアプリケーションであればこの関数でファイルを丸ごとセーブします。 ・同名ファイルがある場合、古いファイルは消去されます。ただし同名ファイルが実行ファイルであった場合はエラーとなります。		
参照備考	・ベリファイエラーが発生した場合でも、ファイルはセーブされています。 ・データアドレスには必ず4バイト境界を指定してください。		
例	<pre> Sint32 ret; BUS_TIME time; time.year = 1998; : /* ベリファイあり、コピーフラグ FFH(コピー禁止)として保存する */ ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, 16, &time, BUD_FLAG_VERIFY BUD_FLAG_COPY(0xff)); if (ret != BUD_ERR_OK) return NG; while (1) { if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break; } if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG; return OK; </pre>		

buDeleteFile		ファイル関数	TYPE B
書式	Sint32 buDeleteFile(Sint32 drive, const char* fname)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
戻り値	BUD_ERR_OK	処理要求を受け付けた	
完了ステータス	BUD_ERR_BUSY	処理中のため要求を受け付けられなかった	
	BUD_ERR_OK	正常終了	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_FILE_NOT_FOUND	ファイルがない	
機能	・ファイルを削除します。		
参照備考			

例

```
buDeleteFile(BUD_DRIVE_A1, "SAVEFILE_001");
if (ret != BUD_ERR_OK) return NG;
while (1) {
    if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;
return OK;
```

buSetFileAttr		ファイル関数	TYPE B
書式	Sint32 buSetFileAttr(Sint32 drive, const char* fname, Uint16 header, Uint8 copyflag)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
	header	ヘッダオフセット(無視されます)	
	copyflag	コピーフラグ(00H ~ FFH)	
戻り値	BUD_ERR_OK	処理要求を受け付けた	
	BUD_ERR_BUSY	処理中のため要求を受け付けられなかった	
完了ステータス	BUD_ERR_OK	正常終了	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_FILE_NOT_FOUND	ファイルがない	
機能	・ファイルの属性をを変更します。		
	BUS_FILEINFO, buGetFileInfo()		
参照	・すでにあるファイルのコピーフラグを変更する場合に使用してください。		
	・ヘッダオフセットは無視されます。0を指定してください。		
備考	・コピー不可フラグを付ける(コピーフラグを FFH にする) */		
	/* コピー不可フラグを付ける(コピーフラグを FFH にする) */		
例	<pre>buSetFileAttr(BUD_DRIVE_A1, "SAVEFILE_001", 0, 0xff); if (ret != BUD_ERR_OK) return NG; while (1) { if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break; } if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG; return OK;</pre>		

第4章

特殊アプリケーションでのファイルアクセス

ファイル管理ユーティリティのようなアプリケーションの場合、バックアップ RAM にあるすべてのファイルのファイル名を取得したり、ファイルのコピー、情報の表示等を行う必要があります。ここでは、そのようなアプリケーションのために用意されている関数を解説します。

4.1 ディレクトリ情報の取得

バックアップ RAM にあるファイルのファイル名をすべて取得するには、関数 `buFindFirstFile()` と、`buFindNextFile()` を使用します。

```
Sint32 ret, files, blocks, total;
char fname[16];
BUS_FILEINFO info;

files = blocks = total = 0;

/* 第1ファイルのファイル名取得 */
ret = buFindFirstFile(drive, fname);
if (ret < 0) {
    if (ret == BUD_ERR_FILE_NOT_FOUND) goto end;
    else goto err;
}
if (buGetFileInfo(drive, fname, &info) < 0) goto err;
blocks = info.blocks;
total += blocks;
files++;

do {
    printf("%12s %10d bytes(%3d blocks)¥n", files, blocks * 512, blocks);
    /* 第2ファイル以降のファイル名取得 */
    ret = buFindNextFile(drive, fname);
    if (ret < 0) {
        if (ret == BUD_ERR_FILE_NOT_FOUND) goto end;
        else goto err;
    }
    if (buGetFileInfo(drive, fname, &info) < 0) goto err;
    blocks = info.blocks;
    total += blocks;
    files++;
} while (files < FILE_MAX);

end:
return OK;
err:
return NG;
```

4.2 ファイルのコピー

ファイルをコピーする関数を用意されていません。buLoadFile()と buSaveFile()を使用してコピーする必要があります。

```
Sint32 ret;
Sint32 blocks;
Sint32 saveflag;
extern Uint8 CopyBuffer[]; /* コピーバッファ */
BUS_FILEINFO info;

/* ファイル情報の取得 */
ret = buGetFileInfo(BUD_DRIVE_A1, "SAVEDATA_001", &info);
if (ret < 0) return NG;
blocks = info.blocks;

/* ファイルロード */
ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA_001", CopyBuffer, blocks);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

/* セーブフラグの設定: ベリファイ ON, コピーフラグを 00H とする */
saveflag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0x00);

/* ファイルのセーブ */
ret = buSaveFile(BUD_DRIVE_A2, "SAVEDATA_001", CopyBuffer, blocks,
                 &info.time, saveflag);

if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

上記の例では、ドライブ A1 から B1 にファイルをコピーしています。
実際のアプリケーションでは、以下のような点に注意が必要です。

- 1) コピー先のバックアップ RAM の接続、フォーマットチェック
- 2) コピー先の空き容量チェック
- 3) コピー先の同名ファイルの存在チェック

また、BOOT ROM および VMS 単体のファイル管理画面でのファイルコピーとの互換性を保つため、以下の点に注意してください。

- 1) 元のファイルの日付をコピー先ファイルにも反映させてください。
- 2) コピー元ファイルのコピーフラグはコピー先ファイルに反映されません。
コピー先ファイルのコピーフラグは 00H としてください。
- 3) コピーフラグが FFH のファイル（コピー不可ファイル）をコピーできません。

4.3 ファイルの一部読み込み

全ファイルの特定ブロックだけを読み込み、ヘッダを解析する等の処理が必要になる場合があります。その場合には、ファイルの一部読み込みが可能な関数 `buLoadFileEx()` を使用します。

```
Sint32 ret;
Sint32 blocks;
Sint32 start_block;
extern Uint8 LoadBuffer[]; /* ロードバッファ */

/* ファイル一部ロード */

start_block = 0;
blocks = 1;
ret = buLoadFileEx(BUD_DRIVE_A1, "SAVEDATA_001", LoadBuffer,
                  start_block, blocks);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

この例ではファイルの先頭ブロックを1ブロックだけ読み込んでいます。

4.4 フォーマット

バックアップ RAM は出荷時にフォーマットされていますが、フォーマットする必要がある場合には、関数 `buFormatDisk()` を使用してフォーマットできます。

```
Sint32 ret;
BUS_TIME time;
Uint8 volume[32];
Sint32 icon_no;

time.year = 1998; /* タイムスタンプの設定 */
time.month = 6; /* 1998/6/25(木) 23:59:59 */
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

icon_no = 0; /* アイコン番号の設定 */

/* ボディカラー青を設定する */
memset(volume, 0, sizeof(volume));
volume[0] = 0x01; /* ボディカラー情報あり */
volume[1] = 0xbf; /* B */
volume[2] = 0x00; /* G */
volume[3] = 0x00; /* R */
volume[4] = 0xff; /* A */

ret = buFormatDisk(BD_DRIVE_A1, volume, icon_no, &time, TRUE);
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

4.4.1 アイコン番号

BOOT ROM のファイル管理画面で表示される「デフォルトアイコン」の番号を指定します。
 BOOT ROM は124種類(1998/8/3 時点)のデフォルトアイコンを持っていますので、
 0～123の値を指定してください。
 また、デフォルトアイコンではなく、アプリケーションで任意のアイコンを保存し、BOOT ROM の
 ファイル管理画面で表示させることができます。これについては「6.5 VMS アイコンファイル」
 を参照してください。

4.4.2 ボディカラー情報

BOOT ROM のファイル管理画面で表示されるバックアップ RAM のボディカラー情報
 を持たせることができます。これはボリュームデータに以下の形式で指定してくだ
 さい。

- ・ボディカラー情報を持たせる場合

+0	+1	+2	+3	+4	+5	...	+31
01H	BLUE	GREEN	RED	ALPHA	00H	...	00H

ボリュームデータの先頭に 01H を格納してください。それ以外は禁止です。
 BLUE、GREEN、RED にそれぞれ青、緑、赤の色成分を 0～255 で設定します。
 ALPHA(透明度)の値は 128～255 の間で設定してください。127 以下の値は BOOT ROM
 での表示が見つらなくなるため指定禁止です。

- ・ボディカラー情報を持たせない場合

+0	+1	+2	+3	+4	+5	...	+31
00H	00H	00H	00H	00H	00H	...	00H

すべて 00H としてください。
 ボディカラー情報を持たせない場合、BOOT ROM のファイル管理画面ではデフォルト
 カラー（白）が使用されます。

4.5 ファイルアクセスに関する関数の説明

関数	機能	タイプ
buFindFirstFile	第1ファイルのファイル名を取得する	A
buFindNextFile	次のファイルのファイル名を取得する	A
buLoadFileEx	読み込み開始ブロックとブロック数を指定してファイルをロードする	B
buFormatDisk	バックアップ RAM をフォーマットする	B

表 4-1.関数一覧

buFindFirstFile

ファイル関数

TYPE **A**

書式 Sint32 buFindFirstFile(Sint32 drive, char* fname)
 パラメータ drive ドライブ番号
 fname ファイル名を格納するアドレス

戻り値	BUD_ERR_OK	正常終了
	BUD_ERR_FILE_NOT_FOUND	ファイルがない
	BUD_ERR_UNFORMAT	フォーマットされていない
	BUD_ERR_NO_DISK	ディスクがない
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である
機能	<ul style="list-style-type: none"> ・第1ファイルのファイル名を取得します。 ・ファイルが存在しない場合、fname[0] = '\0'となります。 ・通常、buFindNextFile()と組み合わせて、ドライブにある全ファイル名を取得するのに使用します。 	
参照備考例	<ul style="list-style-type: none"> ・ファイル名を格納する領域には最低 13 バイトの領域が必要です <pre>Sint32 ret, files, drive; char buf[16]; drive = BUD_DRIVE_A1; files = 0; ret = buFindFirstFile(drive, buf); if (ret < 0) { if (ret == BUD_ERR_FILE_NOT_FOUND) goto end; else goto err; } files++; do { printf("%s¥n", buf); ret = buFindNextFile(drive, buf); if (ret < 0) { if (ret == BUD_ERR_FILE_NOT_FOUND) goto end; else goto err; } files++; } while (1);</pre>	

buFindNextFile

ファイル関数

TYPE **A**

書式	Sint32 buFindNextFile(Sint32 drive, char* fname)	
パラメータ	drive	ドライブ番号
	fname	ファイル名を格納するアドレス
戻り値	BUD_ERR_OK	正常終了
	BUD_ERR_FILE_NOT_FOUND	ファイルがない
	BUD_ERR_UNFORMAT	フォーマットされていない
	BUD_ERR_NO_DISK	ディスクがない
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である
機能	<ul style="list-style-type: none"> ・次のファイルのファイル名を取得します。 ・通常、buFindFistFile()と組み合わせて、ドライブにある全ファイル名を取得するのに使用します。 	
参照備考	<ul style="list-style-type: none"> ・ファイル名を格納する領域には最低 13 バイトの領域が必要です ・ファイルが存在しない場合、fname[0] = '\0'となります。 	
例	buFindFirstFile()の例を参照してください。	

buLoadFileEx

ファイル関数

TYPE **B**

書式	Sint32 buLoadFileEx(Sint32 drive, const char* fname, void* buf, Uint32 start, Uint32 nblock)	
パラメータ	drive	ドライブ番号
	fname	ファイル名
	buf	読み込みアドレス(4バイト境界)
	start	読み込み開始ブロック

	nblock	読み込むブロック数
戻り値	BUD_ERR_OK	処理要求を受け付けた
	BUD_ERR_BUSY	処理中のため要求を受け付けられなかった
完了ステータス	0 または正	読み込んだブロック数
	BUD_ERR_NO_DISK	ディスクがない
	BUD_ERR_UNFORMAT	フォーマットされていない
	BUD_ERR_FILE_NOT_FOUND	ファイルがない
	BUD_ERR_CANNOT_OPEN	ファイルが開けない
	BUD_ERR_FILE_BROKEN	ファイルが壊れている
機能	・ファイルをロードします。 ・読み込み開始ブロックに 0 を指定するとファイルの先頭からロードします。 ・読み込むブロック数に 0 を指定するとファイル全体をロードします。	
参照 備考 例	・読み込みアドレスには必ず 4 バイト境界を指定してください。 <pre>Sint32 ret;</pre> <pre>ret = buLoadFileEx(BUD_DRIVE_A1, "SAVEDATA", SaveData, 0, 1); if (ret != BUD_ERR_OK) return; while (1) { if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break; }</pre>	

buFormatDisk

ドライブ関数

TYPE **B**

書式	Sint32 buFormatDisk(Sint32 drive, const Uint8* volume, Sint32 icon, BUS_TIME* time, Sint32 flag)				
パラメータ	drive	ドライブ番号			
	volume	ボリュームデータ			
	icon	アイコン番号			
	time	タイムスタンプ			
	flag	フォーマットフラグ	TRUE	完全フォーマット	
			FALSE	クイックフォーマット	
戻り値	BUD_ERR_OK	処理要求を受け付けた			
	BUD_ERR_BUSY	処理中のため要求を受け付けられなかった			
完了ステータス	BUD_ERR_OK	正常終了			
	BUD_ERR_NO_DISK	ディスクがない			
機能	・ディスクをフォーマットします。				
参照 備考 例	<pre>BUS_TIME time; Uint8 volume[32]; time.year = 1998; :</pre> <pre>buFormatDisk(BUD_DRIVE_A1, volume, 0, &time, FALSE);</pre>				

第5章 実行ファイル

バックアップ RAM の中でも、VMS のような携帯ゲーム機を兼ねたバックアップ RAM には、実行ファイルを保存することができます。実行ファイルが保存された VMS は、それ単体で携帯ゲーム機として機能します。

5.1 実行ファイルの特徴

実行ファイルには、以下の特徴があります。

- 1) バックアップ RAM の先頭ブロックから順に、常に連続して格納されます。
- 2) 最大サイズは 128 ブロック(64KB)です。
- 3) 1 つのバックアップ RAM に 1 つしか存在することができません。

5.2 実行ファイルの存在確認

バックアップ RAM に実行ファイルがあるかどうかは、関数 `buFindExecFile()` を使用して調べます。

```
Sint32 ret;
char fname[16];

/* 実行ファイルのファイル名取得 */
ret = buFindExecFile(BUD_DRIVE_A1, fname);

switch (ret) {
    case BUD_ERR_OK:
        /* 実行ファイルあり。fname にはファイル名が格納されている。 */
        printf("実行ファイル:%s¥n", fname);
        break;
    case BUD_ERR_FILE_NOT_FOUND:
        printf("実行ファイルはありません。¥n");
        break;
    default:
        printf("エラー¥n");
        break;
}
```

5.3 実行ファイル用空き容量の取得

バックアップ RAM に実行ファイルを書き込める空き容量があるかどうかは、関数 `buGetDiskFree()` を使用して調べます。

```
Sint32 free;

free = buGetDiskFree(BUD_DRIVE_A1, BUD_FILETYPE_EXECUTABLE);

if (free < 0) return NG;
```

空き容量の取得に成功すれば、free には0以上の値が返されます。これはそのバックアップ RAM の実行ファイルが書き込み可能な空きブロック数となります。エラーが発生した場合には、負の値が返されます。

5.4 実行ファイル用空き容量の確保

実行ファイルを書き込むためには、先頭ブロックから連続した空きエリアが必要です。先ほどの buGetDiskFree(drive, BUD_FILETYPE_EXECUTABLE)では、この先頭ブロックからの連続空きブロック数を調べています。

連続空きブロック数が書き込みたい実行ファイルのサイズより少ない場合でも、バックアップ RAM 全体の空き容量が足りていれば、デフラグ処理を行うことによって連続したブロックを確保できる場合があります。デフラグ処理は以下の手順で行ってください。

- 1) buGetDiskFree(drive, BUD_FILETYPE_EXECUTABLE)を使用して、実行ファイルが書き込み可能な連続ブロック数を取得します。
- 2) 1)で取得したブロック数が十分であれば、実行ファイルをそのままセーブできます。デフラグ処理を行う必要はありません。
- 3) 連続ブロックが足りない場合、buGetDiskFree(drive, BUD_FILETYPE_NORMAL)を使用して、全体の空きブロック数を取得します。
- 4) 全体の空きブロック数が足りなければ、デフラグ処理をしても無意味ですので、実行ファイルの保存はできません。
- 5) 全体の空きブロック数が足りていれば、デフラグ処理によって連続ブロックを確保することができます。buDefragDisk()を使用してデフラグ処理を行います。

```
Sint32 ret;
Uint32 DefragWork[512 / sizeof(Uint32)];

ret = buDefragDisk(BUD_DRIVE_A1, DefragWork);
if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

注意

デフラグ処理はバックアップ RAM 全体を書き換える処理です。これにはそれなりの処理時間と、途中でエラーが発生するかもしれないというリスクが常に伴います。仮にデフラグ中にバックアップ RAM が抜かれたりすれば、そのバックアップ RAM の内容はほとんどの場合、全て破壊されます。特に必要のない限り、デフラグ処理を行うことは避けてください。

5.5 実行ファイルのセーブ

実行ファイルをセーブするには、関数 `buSaveExecFile()` を使用します。

```
Sint32 ret;
Sint32 blocks, flag;
BUS_TIME time;
extern Uint8 SaveData[];    /* セーブするデータ(512*10 バイト) */

blocks = 10;                /* ファイルサイズは10 ブロック */
flag = BUD_FLAG_VERIFY | BUD_FLAG_COPY(0xff);
/* ベリファイを行う | コピーフラグを FFH に設定する */
/* タイムスタンプの設定 */
time.year = 1998;           /* 1998/6/25(木) 23:59:59 */
time.month = 6;
time.day = 25;
time.hour = 23;
time.minute = 59;
time.second = 59;
time.dayofweek = 4;

ret = buSaveExecFile(BUD_DRIVE_A1, "SAVEDATA", SaveData, blocks, &time, flag);

if (ret != BUD_ERR_OK) return NG;
while (buStat(BUD_DRIVE_A1) == BUD_STAT_BUSY) {
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;

return OK;
```

5.6 実行ファイルの部分更新

実行ファイルには、VMS 等の携帯ゲーム機で動作し、データ保存のために自己一部書き換えをすることが考えられます。Dreamcast 本体アプリとの連動のために、本体からも実行ファイルの一部を書き換える必要がある場合があります。実行ファイルの場合は通常のデータファイルよりも大きいファイルである場合が多く、通常の `buSaveExecFile()` 関数でファイル全体を書き換えるには時間がかかります。またセーブ中に取り外された場合は実行ファイルが実行不可能になる場合がほとんどです。こういった用途のために、実行ファイルに限って一部書き換え可能な関数が用意されています。

```
Sint32 ret;

ret = buRewriteExecFile(BUD_DRIVE_A1, "GAMEFILE_VMS", buf, 15, 2);
```

この例ではファイルの 15 ブロック目から連続した 2 ブロックを、直接書き換えています。(ファイルの先頭ブロックを 0 ブロックとして)

5.7 実行ファイルに関する関数の説明

関数	機能	タイプ
buFindExecFile	実行ファイルのファイル名を取得する	A
buSaveExecFile	実行ファイルをセーブする	B
buDefragDisk	連続空きブロックを確保する	B
buRewriteExecFile	実行ファイルの一部を直接書き換える	B

表 5-1.関数一覧

• API

buFindExecFile		ファイル関数	TYPE A
書式	Sint32 buFindExecFile(Sint32 drive, char* fname)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
戻り値	BUD_ERR_OK	正常終了	
	BUD_ERR_FILE_NOT_FOUND	実行ファイルがない	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である	
機能	・実行ファイルのファイル名を取得します。		
参照			
備考	・ファイル名を格納する領域には最低 13 バイトの領域が必要です ・ファイルが存在しない場合、fname[0] = '\0' となります。 ・buFindFirstFile(), buFindNextFile() とはまったく独立に、互いに影響なく使用できます。		
例	<pre>char fname[16]; buFindExecFile(BUD_DRIVE_A1, fname);</pre>		

buSaveExecFile		ファイル関数	TYPE B
書式	Sint32 buSaveExecFile(Sint32 drive, const char* fname, const void* buf, Uint32 nblock, BUS_TIME* time, Sint32 flag)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
	buf	セーブするデータアドレス(4バイト境界)	
	nblock	セーブするブロック数	
	time	タイムスタンプ	
	flag	フラグ	BUD_FLAG_VERIFY ベリファイあり BUD_FLAG_COPY(n) コピーフラグ
戻り値	BUD_ERR_OK		
	BUD_ERR_BUSY		
完了ステータス	BUD_ERR_OK	正常終了	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_DISK_FULL	ディスクフル	
	BUD_ERR_EXEC_FILE_EXIST	実行ファイルはすでに存在する	
	BUD_ERR_FILE_EXIST	同名ファイルが存在する	
機能	・実行ファイルをセーブします。 ・通常のアプリケーションであればこの関数でファイルを丸ごとセーブします。 ・実行ファイルが存在する場合および同名ファイルが存在する場合はエラーとなります。		
参照			

備考 ・ベリファイエラーが発生した場合でも、ファイルはセーブされています。
 ・データアドレスには必ず4バイト境界を指定してください。

例

```
Sint32 ret;
BUS_TIME time;
time.year = 1998;
:
/* ベリファイあり、コピーフラグ FFH(コピー禁止)として保存する */
ret = buSaveExecFile(BUD_DRIVE_A1, "EXEFILE", SaveData, 16, &time,
    BUD_FLAG_VERIFY | BUD_FLAG_COPY(0xff));
if (ret != BUD_ERR_OK) return NG;
while (1) {
    if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;
return OK;
```

buDefragDisk		ドライブ関数	TYPE B
書式	Sint32 buDefragDisk(Sint32 drive, void* work)		
パラメータ	drive	ドライブ番号	
	work	ワークバッファ(4バイト境界 512バイト)	
戻り値	BUD_ERR_OK または BUD_ERR_BUSY		
完了ステータス	BUD_ERR_OK	正常終了	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_ACCESS_DENIED	実行ファイルが存在するため最適化処理不可能	
機能	・実行ファイル書き込みエリアを確保するために、ディスクの最適化を行います。		
参照			
備考	・最適化処理中は work で示されるワークバッファの内容を書き換えないでください。 ・ワークバッファは4バイト境界から始まる512バイトを確保してください。		
例	buDefragDisk(BUD_DRIVE_A1, work);		

buRewriteExecFile		ファイル関数	TYPE B
書式	Sint32 buRewriteExecFile(Sint32 drive, const char* fname, void* buf, Uint32 start, Uint32 nblock)		
パラメータ	drive	ドライブ番号	
	fname	ファイル名	
	buf	セーブするデータアドレス(4バイト境界)	
	start	書き換え開始ブロック	
	nblock	書き換えブロック数	
戻り値	BUD_ERR_OK BUD_ERR_BUSY		
完了ステータス	BUD_ERR_OK	正常終了	
	BUD_ERR_NO_DISK	ディスクがない	
	BUD_ERR_UNFORMAT	フォーマットされていない	
	BUD_ERR_FILE_NOT_FOUND	ファイルがない	
	BUD_ERR_INVALID_PARAM	ブロックの指定が間違っている	
機能	・実行ファイルの一部を直接書き換えます。 ・すでに存在する実行ファイルの、すでに存在するブロックを更新する場合にのみ使用できます。 ・実行ファイルが存在しない場合および実行ファイルに対して指定ブロックが大きすぎる場合エラーとなります。		
参照			
備考	・この関数でファイルのタイムスタンプ、属性等を変更することはできません。 ・データアドレスには必ず4バイト境界を指定してください。		

例

```
Sint32 ret;
ret = buRewriteExecFile(BUD_DRIVE_A1, "EXEFILE", SaveData, 15, 2);
if (ret != BUD_ERR_OK) return NG;
while (1) {
    if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
}
if (buGetLastError(BUD_DRIVE_A1) != BUD_ERR_OK) return NG;
return OK;
```

第6章

バックアップファイルフォーマット

バックアップ RAM に保存するファイルは、どのようなものでも良いわけではありません。Dreamcast ソフトウェア作成基準に従った、正しい形式のファイルを保存しなくてはなりません。ここでは、バックアップファイルの外部インタフェースと内部フォーマットについて記述します。Dreamcat ソフトウェア作成基準も合わせて参照してください。

6.1 ファイルの外部インタフェース

6.1.1 ファイル名

ファイル名は必ず12文字指定します。ファイル名には半角英大文字、一部記号等、下図の網掛け部のコードが使用可能です。これ以外の文字は指定禁止となります。ライブラリ関数に指定する場合は、NULL 文字を含めて13バイトとなります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	.	p				-	タ	ミ		
1			!	1	A	Q	a	q			.	ア	チ	ム		
2			"	2	B	R	b	r			「	イ	ツ	メ		
3			#	3	C	S	c	s			」	ウ	テ	モ		
4			\$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			.	オ	ナ	ユ		
6			&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7			'	7	G	W	g	w			ア	キ	ヌ	ラ		
8			(8	H	X	h	x			イ	ク	ネ	リ		
9)	9	I	Y	i	y			ウ	ケ	ノ	ル		
A			*	:	J	Z	j	z			エ	コ	ハ	レ		
B			+	;	K	[k	{			オ	サ	ヒ	ロ		
C			,	<	L	\	l	!			ト	シ	フ	ワ		
D			-	=	M]	m	}			ユ	ス	ヘ	ン		
E			.	>	N	^	n	~			ヨ	セ	ホ	"		
F			/	?	O	_	o				ッ	ソ	マ	°		

図 6-1. ファイル名に指定可能な文字

1つのアプリケーションで複数のファイルを作成する場合は、必ず頭の9文字を統一し、後ろの3文字を変更してください。
BOOT ROM のファイル管理画面には、同一ゲームのファイルをまとめてコピーする等の機能があります。同一ゲームであるかどうかの判断は、ファイル名の先頭9文字で行っているため、ファイル名が適切でないとこれらの機能が正常に動作しません。

ファイル名の例: 「SAVEDATA_SYS」
「SAVEDATA_001」
「SAVEDATA_002」 等

6.1.2 ファイル属性

ファイル属性には、以下のようなものがあります。

1) コピーフラグ

コピーフラグとは、00H ~ FFH までの値を取るフラグです。ファイル保存時に任意に設定できます。

この値が FFH であるファイルは、BOOT ROM のファイル管理画面および VMS 単体のファイル管理画面でコピーできません。コピー禁止フラグとして使用できます。

コピーフラグ 00H ~ FEH のファイルは、BOOT ROM のファイル管理画面および VMS 単体のファイル管理画面でコピーすると、コピー先ファイルのコピーフラグは 00H になります。

2) 実行可能

VMS 等の携帯ゲーム機を兼ねたバックアップ RAM の場合、実行可能ファイルを 1 つだけ保存することができます。実行ファイルのサイズは最大 1 2 8 ブロックとなっています。

6.2 ファイルの内部フォーマット

バックアップファイルには、BOOT ROM のファイル管理画面や VMS 単体のファイル管理画面で表示するためのコメント、アイコン等のデータを必ず格納しなくてはなりません。バックアップファイルは、下のようなフォーマットになっています。なお、ワードデータ、ロングワードデータはリトルエンディアン形式とします。

ファイルフォーマットは通常ファイルと実行可能ファイルでヘッダの位置が異なっているため注意が必要です。

+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F
+0000	VMS コメント(16 バイト)														
+0010	BOOT ROM コメント (32 バイト)														
+0030	ゲーム名(ソートアイテム)(16 バイト)														
+0040	アイコン 枚数	アイコン スピード	ビジュア ルタイプ	CRC			セーブデータサイズ			(予約)					
+0050	(予約)														
+0060	アイコンパレットデータ (32 バイト)														
+0080	アイコンピクセルデータ1 (512 バイト)														
+0280	アイコンピクセルデータ2 (512 バイト)														
+0480	アイコンピクセルデータ3 (512 バイト)														
+nnnn	ビジュアルデータ(パレット/ピクセル) (2048/4544/8064 バイト)														
+nnnn	アプリケーションセーブデータ														

固定長ヘキサ領域

可変長ヘキサ領域

コーナー領域

図 6-2.通常ファイルのファイルフォーマット



図 6-3.実行可能ファイルのファイルフォーマット

- ・固定長ヘッダ(網掛け部) は必須データです。必ず全ての項目を正しく指定する必要があります。(CRC、セーブデータサイズを除く)
- ・(予約)となっている領域は全て 00H を格納してください。
- ・固定長ヘッダのうち、VMS 単体のファイル管理画面で参照するのは VMS コメントのみとなっています。
- ・可変長ヘッダを格納しない場合、アプリケーションセーブデータは+0280H から格納することができます(通常ファイルの場合)
- ・ファイルのアクセスは常にブロック単位で行われます。ファイル全体のサイズが512バイトの倍数になるようにアプリケーションセーブデータを調整し、ファイルの最終ブロックに不定データがなるべく格納されることのないようにしてください。
- ・固定長ヘッダのサイズが 280H バイトであるため、バックアップファイルは最低2ブロックを消費することになります。

6.2.1 VMS コメント

BOOT ROMとVMS単体のファイル管理画面の両方で表示されるコメントデータです。
16バイト以内で指定してください。下図の網掛け部の文字が使用できます。これ以外の文字は指定禁止となります。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0			SP	0	@	P	.	p				-	タ	ミ		
1			!	1	A	Q	a	q			.	ア	チ	ム		
2			"	2	B	R	b	r			、	イ	ツ	メ		
3			#	3	C	S	c	s			、	ウ	テ	モ		
4			\$	4	D	T	d	t			、	エ	ト	ヤ		
5			%	5	E	U	e	u			、	オ	ナ	ユ		
6			&	6	F	V	f	v			、	カ	ニ	ヨ		
7			'	7	G	W	g	w			、	キ	ヌ	ラ		
8			(8	H	X	h	x			、	ク	ネ	リ		
9)	9	I	Y	i	y			、	ケ	ノ	ル		
A			*	:	J	Z	j	z			、	コ	ハ	レ		
B			+	;	K	[k	{			、	サ	ヒ	ロ		
C			,	<	L	\	l	!			、	シ	フ	ワ		
D			=	=	M]	m	}			、	ス	ヘ	ン		
E			.	>	N	^	n	~			、	セ	ホ	"		
F			/	?	O	-	o				、	ソ	マ	*		

図 6-4.VMS コメントに指定可能な文字

6.2.2 BOOT ROM コメント

BOOT ROMのファイル管理画面で表示されるコメントデータです。全角文字(2バイト)、半角文字(1バイト)混在で、32バイトまで指定できます。
全角文字の場合は英数・かな・カナ・第1、第2水準までのSHIFT-JISコードで指定します。
半角文字の場合は図 1-2 の文字が指定できます。

6.2.3 ゲーム名(ソートアイテム)

BOOT ROMのファイル管理画面でファイルを表示する際、ソートするためのキーとして使用されます。以下の表を参照して、16バイトで格納してください。ゲーム名が16バイトに満たない場合、残りの領域には00Hを格納してください。

コード(10 進)	コード(16 進)	文字
0	00	スペース
1	01	ア
2	02	ァ
3	03	イ
4	04	ィ
5	05	ウ
6	06	ヴ
7	07	ゥ
8	08	エ
9	09	ェ
10	0A	オ
11	0B	ォ
12	0C	カ
13	0D	ガ
14	0E	キ
15	0F	ギ
16	10	ク
17	11	グ
18	12	ケ
19	13	ゲ
20	14	コ
21	15	ゴ
22	16	サ
23	17	ザ
24	18	シ
25	19	ジ
26	1A	ス
27	1B	ズ
28	1C	セ
29	1D	ゼ
30	1E	ソ
31	1F	ゾ
32	20	タ
33	21	ダ
34	22	チ
35	23	ヂ
36	24	ツ
37	25	ヅ
38	26	ッ
39	27	テ
40	28	デ
41	29	ト
42	2A	ド
43	2B	ナ
44	2C	ニ
45	2D	ヌ
46	2E	ネ
47	2F	ノ

コード(10 進)	コード(16 進)	文字
48	30	ハ
49	31	バ
50	32	パ
51	33	ヒ
52	34	ビ
53	35	ピ
54	36	フ
55	37	ブ
56	38	プ
57	39	ヘ
58	3A	ベ
59	3B	ペ
60	3C	ホ
61	3D	ボ
62	3E	ポ
63	3F	マ
64	40	ミ
65	41	ム
66	42	メ
67	43	モ
68	44	ヤ
69	45	ャ
70	46	ユ
71	47	ュ
72	48	ヨ
73	49	ョ
74	4A	ラ
75	4B	リ
76	4C	ル
77	4D	レ
78	4E	ロ
79	4F	ワ
80	50	ヰ
81	51	ヱ
82	52	ヲ
83	53	ン
84	54	0
85	55	1
86	56	2
87	57	3
88	58	4
89	59	5
90	5A	6
91	5B	7
92	5C	8
93	5D	9

表 6-1. ソートアイテムコード表

6.2.4 アイコン枚数

格納するアイコンの枚数を1～3で指定します。
最低1枚必ず格納しなくてはなりません。

6.2.5 アイコンアニメーションスピード

2枚、3枚のアイコンを格納する場合のパターン切り替えのスピードをフレーム数(1/30 秒単位)で指定します。

6.2.6 ビジュアルタイプ

格納するビジュアルコメントのタイプを指定します。

ビジュアルタイプ	格納するデータ
ビジュアルデータなし	0000H
TYPE A	0001H
TYPE B	0002H
TYPE C	0003H

表 6-2. ビジュアルタイプ

6.2.7 CRC コード

ファイル先頭から終わりまでをすべて加味した CRC コードです。
ただし、CRC の利用はアプリケーションまかせとなっています。BOOT ROM のファイル管理画面では参照しません。
これは、実行ファイルの場合は VMS 単体で動作し、データ保存のため自己書き換えが発生する場合があります。
CRC を利用しない場合は 0000H を格納しておきます。
後述のユーティリティ関数を利用すると、CRC の生成、チェックが自動で行えます。

6.2.8 セーブデータサイズ/ファイルサイズ

アプリケーションのセーブデータ部分のサイズをバイト数で指定します。
実行可能ファイルの場合はファイル全体のバイト数を指定してください。
BOOT ROM のファイル管理画面では参照しません。

6.2.9 アイコンデータ(パレットデータ/ピクセルデータ1, 2, 3)

BOOT ROM のファイル管理画面で表示されるアイコンです。セーブデータには最低1枚のアイコンを必ず格納しなくてはなりません。
以下のフォーマットでデータを作成してください。

項目	内容
サイズ	横 32x 縦 32 ピクセル
カラー形式	4ビット(16色パレット)
パレット形式	ARGB4444 形式 x1 6色 2バイト(16ビット)で1色を表現 リトルエンディアン形式
枚数	最低1枚必須、3枚まで格納可能 パレットは共通

格納方式	パレットデータを16色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。 1バイトで2ピクセルを表す。上位4ビットが左のドット、下位4ビットが右のドットとなる。
サイズ	1枚の場合 544バイト 2枚の場合 1056バイト 3枚の場合 1568バイト

表 6-3.アイコンデータフォーマット

2枚、3枚のアイコンを格納した場合、そのアイコンを指定したフレーム数(1/30 秒単位)で切り替えて、アニメーション表示されます。

6.2.10 ビジュアルコメント

BOOT ROM のファイル管理画面で表示されるビジュアルコメントです。ファイル中にこのビジュアルコメントを格納するかどうかは任意となります。格納する場合は、以下のフォーマットでデータを作成してください。

ビジュアルコメントにはピクセルデータ形式によって TYPE A、B、C の種類があり、以下のようなフォーマットになっています。

a) ビジュアルコメント TYPE A

項目	内容
サイズ	横 72x 縦 56 ピクセル
カラー形式	16ビットダイレクトカラー(ARGB4444)
パレット形式	なし
枚数	1枚または無し
格納方式	左上から右下へのピクセル情報を連続したデータとして格納する。 2バイト(16ビット)で1ピクセルを表す。 リトルエンディアン形式。
サイズ	8064バイト(16ブロック)

表 6-4.ビジュアルコメント TYPE A フォーマット

a) ビジュアルコメント TYPE B

項目	内容
サイズ	横 72x 縦 56 ピクセル
カラー形式	8ビット(256色パレット)
パレット形式	ARGB4444 形式 x16色または256色 2バイト(16ビット)で1色を表現 リトルエンディアン形式
枚数	1枚または無し
格納方式	パレットデータを256色分格納する。続いてピクセルデータを、左上から右下への情報を連続したデータとして格納する。 1ピクセルを1バイトで表す。
サイズ	4544バイト(9ブロック)

表 6-5.ビジュアルコメント TYPE B フォーマット

a) ビジュアルコメント TYPE C

項目	内容
サイズ	横 72x 縦 56 ピクセル
カラー形式	4ビット(16色パレット)
パレット形式	ARGB4444 形式 x16色 2バイト(16ビット)で1色を表現 リトルエンディアン形式
枚数	1枚または無し

格納方式	パレットデータを16色分格納する。続いてピクセルデータを左上から右下への情報を連続したデータとして格納する。 1バイトで2ピクセルを表す。上位4ビットが左のドット、下位4ビットが右のドットとなる。
サイズ	2048バイト(4ブロック)

表 6-6. ビジュアルコメント TYPE C フォーマット

ビジュアルタイプ A は使用ブロック数が多く、ビジュアルタイプ B、C に比べて特に高いクオリティを持つわけでもないため、なるべく使用しないでください。

6.2.11 アプリケーションセーブデータ

アプリケーションのデータを格納します。

ファイルのアクセスは常にブロック単位で行われます。ファイル全体のサイズが512バイトの倍数になるようにアプリケーションセーブデータを調整し、ファイルの最終ブロックに不定データがなるべく格納されることのないようにしてください。

6.3 バックアップファイルユーティリティ関数

バックアップファイルのイメージをメモリ上に作成したり、メモリ上に読み込んだファイルを解析するユーティリティ関数が用意されています。

なお、これらの関数は通常ファイルにのみ使用でき、実行可能ファイルに使用することはできませんので注意してください。

6.3.1 バックアップファイルイメージの作成

以下のように、BUS_BACKUPFILEHEADER 構造体を宣言し、パラメータを設定します。

```
extern Uint8 game_name[];          /* ゲーム名(ソートアイテム) */
extern Uint16 icon_palette[];     /* アイコンパレットデータ */
extern Uint8 icon_data[];        /* アイコンピクセルデータ */
extern Uint8 visual_data[];      /* ビジュアルデータ */
extern Uint8 save_data[];        /* アプリケーションセーブデータ */

BUS_BACKUPFILEHEADER hdr;        /* バックアップファイルヘッダ */

memset(&hdr, 0, sizeof(hdr));
strcpy(hdr.vms_comment, "VMS_COMMENT"); /* VMS コメントの設定 */
strcpy(hdr.btr_comment, "BOOT ROM 全角コメント"); /* BOOT ROM コメントの設定 */
memcpy(hdr.game_name, game_name, 16); /* ゲーム名(ソートアイテム)の設定 */
hdr.icon_palette = icon_palette; /* アイコンパレットデータの設定 */
hdr.icon_data = icon_data; /* アイコンピクセルデータの設定 */
hdr.icon_num = 1; /* アイコン枚数の設定 */
hdr.icon_speed = 1; /* アイコンスピードの設定 */
hdr.visual_data = visual_data; /* ビジュアルコメントの設定 */
hdr.visual_type = BUD_VISUALTYPE_C; /* ビジュアルタイプの設定 */
hdr.save_data = save_data; /* アプリケーションセーブデータ */
hdr.save_size = 0x400; /* アプリケーションセーブデータサイズ */
```

この例では、アイコン1枚、TYPE C のビジュアルデータ、400H のアプリケーションデータを格納すると仮定して、構造体を設定しています。

構造体を設定した後、関数 `buCalcBackupFileSize()` を使用して、作成したいバックアップファイルが何ブロック消費するかを計算します。この関数では、ファイル全体のバイト数が512バイトの倍数でない場合、次の512バイト境界までがファイル全体であると仮定してブロック数を計算します。

```
Sint32 nblock;

nblock = buCalcBackupFileSize(hdr.icon_num, hdr.visual_type, hdr.save_size);
```

ファイルイメージの作成には、当然作成先のワークメモリが必要です。ここでは `syMalloc()` 関数を利用してメモリを確保する例を示します。

```
void* buf;

buf = syMalloc(nblock * 512);    /* 使用ブロック数×512バイトを確保する */
if (buf == NULL) {
    /* メモリが確保できない */
    return NG;
}
```

関数 `buMakeBackupFileImage()` 使用して、メモリ上にファイルイメージを作成します。この関数では、ファイル全体のバイト数が512バイトの倍数でない場合、次の512バイト境界までの残りの領域は 00H で埋められます。この領域は CRC 計算には関係ありません。

```
nblock = buMakeBackupFileImage(buf, &hdr);

if (nblock < 0) {
    /* 構造体に不正なパラメータがある */
    return NG;
}
```

最後に、関数 `buSaveFile()` 使用して、メモリの内容を実際にバックアップ RAM にセーブします。必ず、セーブが終了してから確保したメモリを開放してください。

6.3.2 バックアップファイルイメージの取得

メモリ上に読み込まれたファイルを解析し、`BUS_BACKUPFILEHEADER` 構造体を作成する関数が用意されています。

```
extern UInt8 save_data[];    /* アプリケーションセーブデータ */
Sint32 ret;
BUS_BACKUPFILEHEADER hdr;    /* バックアップファイルヘッダ */

ret = buAnalyzeBackupFileHeader(&hdr, save_data);
switch (ret) {
    case BUD_ERR_OK:
        /* 正しく解析できた */
        break;
    case BUD_ERR_BUPFILE_CRC:
        /* 正しく解析できたが、CRC が異なっている */
        break;
    case BUD_ERR_BUPFILE_ILLEGAL:
        /* 正しい形式のファイルではない */
        break;
}
```

戻り値が BUD_ERR_OK であれば、構造体の各メンバが正しく設定されています。

戻り値が BUD_ERR_BUPTFILE_ILLEGAL の場合、正しい形式のバックアップファイルが読み込まれていない可能性があります。

戻り値が BUD_ERR_BUPTFILE_CRC の場合、固定長ファイルヘッダの CRC に書かれている値と、実際に計算した CRC の値が異なっています。この場合ファイルの一部が正常に読み込めていない可能性があります。

6.6 バックアップファイルユーティリティ関数の説明

関数	機能	タイプ
buMakeBackupFileImage	バックアップファイル形式メモリイメージを作成する	A
buCalcBackupFileSize	バックアップファイルイメージのブロックサイズを計算する	A
buAnalyzeBackupFileImage	メモリ上のファイルイメージを解析する	A

表 6-7.関数一覧

buMakeBackupFileImage		ユーティリティ関数	TYPE A
書式	Sint32 buMakeBackupFileImage(void* buf, BUS_BACKUPFILEHEADER* hdr)		
パラメータ	buf	バックアップファイル形式イメージ作成アドレス(4バイト境界)	
	hdr	バックアップファイルヘッダアドレス	
戻り値	正	バックアップファイルの使用ブロック数	
	BUD_ERR_INVALID_PARAM	ヘッダ不正	
機能	・メモリ上に Dreamcast バックアップファイル形式イメージを作成します。		
参照	buAnalyzeBackupFileImage(),buCalcBackupFileSize()		
備考	・イメージ作成アドレスは必ず4バイト境界を指定してください。		

```

例
    Sint32 ret, nblock;
    BUS_BACKUPFILEHEADER hdr;
    BUS_TIME time;
    extern UInt8 game_name[];
    extern UInt16 icon_palette[];
    extern UInt8 icon_data[];
    extern UInt8 visual_data[];
    extern UInt8 save_data[];

    time.year = 1998;
    :
    memset(&hdr, 0, sizeof(hdr));
    strcpy(hdr.vms_comment, "VMS_COMMENT");
    strcpy(hdr.btr_comment, "BOOT ROM 全角コメント");
    memcpy(hdr.game_name, game_name, 16);
    hdr.icon_palette = icon_palette;
    hdr.icon_data = icon_data;
    hdr.icon_num = 1;
    hdr.icon_speed = 1;
    hdr.visual_data = visual_data;
    hdr.visual_type = BUD_VISUALTYPE_A;
    hdr.save_data = save_data;
    hdr.save_size = 0x200 * 5;
    nblock = buCalcBackupFileSize(hdr.icon_num, hdr.visual_type,
                                   hdr.save_size);

    buf = syMalloc(nblock * 512);
    if (!buf) return NG;
    nblock = buMakeBackupFileImage(buf, &hdr);
    if (nblock < 0) goto err;
    ret = buSaveFile(BUD_DRIVE_A1, "SAVEDATA_001", buf, nblock,
                     &time, BUD_FLAG_VERIFY | BUD_FLAG_COPY(0));
    if (ret != BUD_ERR_OK) goto err;
    while (1) {
        if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break;
    }

    syFree(buf);
    return OK;

err:
    syFree(buf);
    return NG;

```

buCalcBackupFileSize

ユーティリティ関数

TYPE **A**

書式	Sint32 buCalcBackupFileSize(UInt32 inum, UInt32 vtype, UInt32 size)
パラメータ	inum アイコンの枚数 vtype ビジュアルタイプ size アプリケーションセーブデータのバイト数
戻り値	正 バックアップファイルの使用ブロック数 BUD_ERR_INVALID_PARAM パラメータが不正
機能	・Dreamcast バックアップファイル形式イメージサイズを計算します。
参照	buAnalyzeBackupFileImage(), buMakeBackupFileImage()
備考	
例	buMakeBackupFileImage() の例を参照してください。

buAnalyzeBackupFileImage		ユーティリティ関数	TYPE A
書式	Sint32 buAnalyzeBackupFileImage(BUS_BACKUPFILEHEADER* hdr, void* buf)		
パラメータ	hdr	バックアップファイルヘッダアドレス	
	buf	解析したいファイルを読み込んだアドレス(4バイト境界)	
戻り値	BUD_ERR_OK	正常終了	
	BUD_ERR_BUPFILE_ILLEGAL	正しい形式のファイルではない	
	BUD_ERR_BUPFILE_CRC	CRC が異なっている	
機能	・メモリ上のデータを解析し、BUS_BACKUPFILEHEADER 構造体を作成します。		
参照	buMakeBackupFileInfo(),buCalcBackupFileSize()		
備考	・データアドレスは必ず4バイト境界を指定してください。		
例	<pre>Sint32 ret, nblock; extern Uint8 buf[]; BUS_BACKUPFILEHEADER hdr; ret = buLoadFile(BUD_DRIVE_A1, "SAVEDATA_001", buf, 0); if (ret != BUD_ERR_OK) return NG; while (1) { if (buStat(BUD_DRIVE_A1) == BUD_STAT_READY) break; } ret = buAnalyzeBackupFileImage(&hdr, buf); switch (ret) { case BUD_ERR_OK: return OK; default: return NG; }</pre>		

・構造体

BUS_BACKUPFILEHEADER		構造体
定義	<pre>typedef struct { char vms_comment[18]; char btr_comment[34]; Uint8 game_name[16]; void* icon_palette; void* icon_data; Uint16 icon_num; Uint16 icon_speed; void* visual_data; Uint16 visual_type; Uint16 reserved; void* save_data; Uint32 save_size; } BUS_BACKUPFILEHEADER;</pre>	
説明	バックアップファイルイメージ作成 / 解析用の情報を格納した構造体です	
メンバ	vms_comment	VMS 用コメント
	btr_comment	BOOT ROM 用コメント
	game_name	ゲーム名(ソートアイテム)
	icon_palette	アイコンパレットアドレス
	icon_data	アイコンデータアドレス
	icon_num	アイコン枚数
	icon_speed	アイコンアニメーションスピード
	visual_data	ビジュアルデータアドレス
	visual_type	ビジュアルタイプ

reserved	予約
save_data	アプリケーションセーブデータアドレス
save_size	アプリケーションセーブデータサイズ

参照 buMakeBackupFileImage(),buAnalyzeBackupFileImage()

6.5 VMS ボリュームアイコン

これまで解説したバックアップファイルとは全く別に、「VMS ボリュームアイコン」と呼ばれるファイルをバックアップ RAM に保存することができます。

以下の形式に従ったファイルを、「ICONDATA_VMS」というファイル名で保存すると、BOOT ROM のファイル管理画面で、デフォルトアイコンの代わりに表示されます。これにより、アプリケーション独自のアイコンを自由に付けることができます。

ボリュームアイコンは、VMS 等の液晶を持つバックアップ RAM の液晶画面に表示するためのモノクロアイコンと、TV 画面に表示するためのカラーアイコンをそれぞれ1枚ずつ持つことができます。

ボリュームアイコンのファイルフォーマットは下図のようになっています。なお、図中のワード、ロングワードデータはリトルエンディアンとします。

VMS ボリュームアイコンのファイルは、BOOT ROM のファイル管理画面では他のファイルと異なり、ファイル一覧に表示されませんので、コピー、削除等はできません。

	+00	+01	+02	+03	+04	+05	+06	+07	+08	+09	+0A	+0B	+0C	+0D	+0E	+0F
+0000	VMS コメント(16 バイト)															
+0010	モノクロアイコン オフセット				カラーアイコン オフセット				(予約 00H)							
+0020	モノクロアイコンデータ															
+00A0	カラーアイコンパレットデータ															
+00C0	カラーアイコンピクセルデータ															

図 6-5. VMS ボリュームアイコンファイルフォーマット

6.5.1 VMS コメント

バックアップファイルの VMS コメントに準じます。

6.5.2 モノクロアイコンオフセット

00000020H を必ず格納してください。

6.5.3 カラーアイコンオフセット

000000A0H を必ず格納してください。

6.5.4 モノクロアイコンデータ

BOOT ROM のファイル管理画面で、バックアップ RAM の液晶に表示されるモノクロのアイコンです。

項目	内容
サイズ	横 32x 縦 32 ピクセル
格納方式	ピクセルデータを左上から右下への情報を連続したデータとして格納する。 1バイトで8ピクセルを表す。LSB 側が右、MSB 側が左となる。 1が黒、0が下地の色になる。
サイズ	128バイト

表 6-8.モノクロアイコンデータフォーマット

6.5.4 カラーアイコン(パレットデータ/ピクセルデータ)

BOOT ROM のファイル管理画面で、TV 画面に表示されるカラーアイコンです。

項目	内容
サイズ	横 32x 縦 32 ピクセル
カラー形式	4ビット(16色パレット)
パレット形式	ARGB4444 形式 x16色 2バイト(16ビット)で1色を表現 リトルエンディアン形式
格納方式	パレットデータを16色分格納する。続いてピクセルデータを左上から右下への情報を連続したデータとして格納する。 1バイトで2ピクセルを表す。上位4ビットが左のドット、下位4ビットが右のドットとなる。
サイズ	544バイト

表 6-9.アイコンデータフォーマット

第7章

新機能「動的ワーク確保」

これまで解説した通常の方法では、バックアップライブラリの動作にはかなりの容量のワークバッファが必要となります。特に 1M バイトもの大容量バックアップ RAM に8 個対応するとすると、400K バイト以上のワークを、固定的に確保する必要があります。

この章で解説する新機能「動的ワーク確保」の機能を使用することにより、必要なバックアップ RAM のみを、必要なときだけマウントすることができるようになり、ワーク容量の大幅な削減が可能です。

7.1 動的ワーク確保の使用方法

7.1.1 初期化

初期化関数 `buInit()` に指定するワークアドレスに `NULL` を指定することにより、動的ワーク確保モードとなり、新機能が利用可能になります。以下のように初期化してください。
なお、第2パラメータの、対応する最大容量は無視されます。

```
void BackupInit(void)
{
    buInit(BUD_USE_DRIVE_ALL, BUD_CAPACITY_128KB, NULL, init_callback);
}
```

7.1.2 バックアップRAM の接続

バックアップ RAM が接続されると、完了コールバック関数が呼び出されます。
その際のパラメータは

drive	ドライブ番号(0 ~ 7)
op	BUD_OP_CONNECT(新コールバックオペレーションコード)
stat	接続されたバックアップ RAM の容量フラグ
	BUD_CAPACITY_128KB 容量 128KB
	BUD_CAPACITY_256KB 容量 256KB
	BUD_CAPACITY_512KB 容量 512KB
	BUD_CAPACITY_1MB 容量 1MB
param	ユーザーパラメータ

となっています。この `BUD_OP_CONNECT` コールバックは、動的ワーク確保モードの場合にのみ発生するコールバックです。

7.1.3 ワークの確保とドライブのマウント

`BUD_OP_CONNECT` コールバックが発生した後は、そのドライブをいつでもマウントすることができます。

`BUD_OP_CONNECT` コールバックに渡された容量フラグと `BUM_WORK_SIZE` マクロを利用

して必要ワーク容量を計算し、アプリケーションでワークを確保してください。以下は、syMalloc()関数を利用してワークを確保する例です。

```
/* stat で示される容量のワーク1ドライブ分を確保する */
work = syMalloc(BUM_WORK_SIZE(stat, 1));
if (work == NULL) return NG;
```

このワークには、アプリケーション責任において必ず確実に利用可能なメモリ領域を必要分確保し、指定してください。ドライブへのアクセス中にワークを開放したり、ワークの内容を破壊した場合の動作は保証されません。

ワークの確保に成功したら、ドライブをマウントします。

```
Sint32 err;

err = buMountDisk(drive, work, BUM_WORK_SIZE(stat, 1));
if (err != BUD_ERR_OK) return NG;
```

これにより、従来バックアップRAM 接続時に自動的に行われていたマウント処理が行われ、マウントが完了したら BUD_OP_MOUNT コールバックが発生します。コールバック発生後は、通常どおりのファイルアクセスが可能です。

7.1.4 ドライブのアンマウントとワークの開放

必要なファイルアクセスが終了し、そのドライブにアクセスする必要がなくなったら、アンマウントすることができます。アンマウント後には確保したワークを開放することができ、メモリを効率よく利用することができます。

```
Sint32 err;

err = buUnmount();
if (err != BUD_ERR_OK) return NG;
```

buUnmount()関数は即時完了復帰関数のため、関数呼び出しから戻ればアンマウントは完了しています。ここでワークを開放することができます。この例では先ほど syMalloc()関数でワークを確保したので、syFree()関数でワークを開放します。

```
syFree(work);
```

なお、どのドライブにバックアップRAM がいつ接続され、取り外されるかは当然のことながらまったく予測が付きません。syMalloc()、syFree()あるいは malloc()、free()を利用してワークの確保、開放をする場合は、それらメモリ管理関数のアルゴリズムを十分理解した上、使い方によってはメモリのフラグメンテーションが発生する可能性があることを念頭において使用してください。

7.1.5 バックアップRAM の切断時

すでにマウントしているバックアップRAM が取り外された場合従来どおり BUD_OP_UNMOUNT コールバックが発生します。この場合は自動的にアンマウントの処理が行われているため

コールバック発生時点でワークを開放しても問題ありません。

```
switch (op) {  
    case BUD_OP_UNMOUNT:  
        if (work) syFree(work);  
        break;  
    :  
}
```

7.2 動的ワーク確保に関する関数の説明

関数	機能	タイプ
buMountDisk	バックアップ RAM をマウントする	B
buUnmount	バックアップ RAM アンマウントする	A

表 7-1.関数一覧

buMountDisk		動的ワーク関数	TYPE B
書式	Sint32 buMountDisk(Sint32 drive, void* work, Sint32 size)		
パラメータ	drive	ドライブ番号	
	work	ワークアドレス(4バイト境界)	
	size	ワークサイズ	
戻り値	BUD_ERR_OK	処理要求を受け付けた	
	BUD_ERR_INVALID_PARAM	接続されたバックアップ RAM に対してワークサイズが少ない	
機能 参照 備考 例	・ドライブをマウントし、ファイルアクセス可能にします。		
	buUnmount()		
	・		
	Sint32 err; err = buMountDisk(BUD_DRIVE_A1, work, BUM_WORK_SIZE(BUD_CAPACITY_1MB, 1)); if (err != BUD_ERR_OK) return NG;		
buUnmount		動的ワーク関数	TYPE A
書式	Sint32 buUnmount(Sint32 drive)		
パラメータ	drive	ドライブ番号	
戻り値	BUD_ERR_OK	正常終了	
	BUD_ERR_BUSY	何らかの TYPE B 関数が処理中である	
機能	・ドライブをアンマウントします。		
	・TYPE A 関数であるためコールバックはありません。		
	・この関数を呼び出し後はそのドライブに対してアクセスできません。		
	・そのドライブで使用していたワークは、この関数呼び出し後いつでも開放することができます。		
参照 備考 例	・再びドライブに対してアクセスする場合は buMountDisk()関数を使用してください。		
	buMountDisk()		
	・		
例	buUnmount(BUD_DRIVE_A1);		