I S & C

Image Save And Carry

標準ファイルマネージャ

仕様説明書

（財）医療情報システム開発センター

平成４年８月

目次

はじめに・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 1

1．　サービス関数の機能と一覧・・・・・・・・・・・・・・・・・・・・・・・ 3
1．1　システム変数の初期化・・・・・・・・・・・・・・・・・・・・・ 3
1．2　ボリュームの初期化・・・・・・・・・・・・・・・・・・・・・・ 3
1．3　ボリュームの管理・・・・・・・・・・・・・・・・・・・・・・・ 3
1．4　ファイルの管理・・・・・・・・・・・・・・・・・・・・・・・・ 3
1．5　データの入出力・・・・・・・・・・・・・・・・・・・・・・・・ 4
1．6　ヘッダを順番に読み込む・・・・・・・・・・・・・・・・・・・・ 4
1．7　システムの管理・・・・・・・・・・・・・・・・・・・・・・・・ 4

2．　共通データ構造（メモリ上での管理テーブル）・・・・・・・・・・・・・・ 5
2．1　システム定数・・・・・・・・・・・・・・・・・・・・・・・・・ 6
2．2　エラー発生位置とコード・・・・・・・・・・・・・・・・・・・・ 7
2．3　Ｉ／Ｏバッファの定義・・・・・・・・・・・・・・・・・・・・・ 8
2．4　ボリューム管理情報の定義・・・・・・・・・・・・・・・・・・・ 9
2．5　セクタテーブルの定義・・・・・・・・・・・・・・・・・・・・・ 10
2．6　ゾーンテーブルの定義・・・・・・・・・・・・・・・・・・・・・ 10
2．7　インデックステーブルの定義・・・・・・・・・・・・・・・・・・ 11
2．8　インデックステーブル属性の定義・・・・・・・・・・・・・・・・ 13
2．9　ヘッダコントローラの定義・・・・・・・・・・・・・・・・・・・ 13
2．10　オープンモードの定義・・・・・・・・・・・・・・・・・・・・・ 13
2．11　適用分野を示すフラグ・・・・・・・・・・・・・・・・・・・・・ 13

3．　共通内部関数・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ 14
3．1　ＭＯＤへの書込み・・・・・・・・・・・・・・・・・・・・・・・ 16
3．2　ＭＯＤからの読出し・・・・・・・・・・・・・・・・・・・・・・ 20
3．3　ファイルインデックスを読出す・・・・・・・・・・・・・・・・・ 24
3．4　ファイルインデックスを書込む・・・・・・・・・・・・・・・・・ 28
3．5　ディスクの排出を禁止する・・・・・・・・・・・・・・・・・・・ 32
3．6　ディスクの排出を許す・・・・・・・・・・・・・・・・・・・・・ 35
3．7　新たなゾーンを得る・・・・・・・・・・・・・・・・・・・・・・ 38
3．8　指定した種別のゾーン中の予約領域のポインタを得る・・・・・・・ 42
3．9　指定したゾーン中の予約領域の先頭のポインタを得る・・・・・・・ 45
3．10　指定したポインタの領域を解放する・・・・・・・・・・・・・・・ 49
3．11　Ｂゾーン中にヘッダのポインタを得る・・・・・・・・・・・・・・ 53
3．12　ワードデータ列のコピー・・・・・・・・・・・・・・・・・・・・ 56
3．13　４バイトデータ列のコピー・・・・・・・・・・・・・・・・・・・ 59
3．14　バイトスワップを行う・・・・・・・・・・・・・・・・・・・・・ 62
3．15　ボリューム管理情報の書込み・・・・・・・・・・・・・・・・・・ 65
3．16　ボリューム管理情報その２への書込み・・・・・・・・・・・・・・ 68
3．17　デバイスをオープンする・・・・・・・・・・・・・・・・・・・・ 71
3．18　デバイスをクローズする・・・・・・・・・・・・・・・・・・・・ 72
3．19　データを読み込む・・・・・・・・・・・・・・・・・・・・・・・ 73
3．20　データを書き込む・・・・・・・・・・・・・・・・・・・・・・・ 74
3．21　論理ユニットが動作可能かをチェックする・・・・・・・・・・・・ 75
3．22　論理ユニットの状態をセットする・・・・・・・・・・・・・・・・ 76
3．23　論理ユニットの容量をセットする・・・・・・・・・・・・・・・・ 77

３．24　メディアの排出を許す・・・・・・・・・・・・・・・・・・・・・・・・・７８
３．25　メディアの排出を禁止する・・・・・・・・・・・・・・・・・・・・・・７９
３．26　センスデータをイニシエータへ転送するよう要求する・・・・・・・・・８０
３．27　時間を得る・・・・・・・・・・・・・・・・・・・・・・・・・・・・８１

４．　サービス関数の設計仕様・・・・・・・・・・・・・・・・・・・・・・・８２
４．１　システム変数の初期化・・・・・・・・・・・・・・・・・・・・・・・８３
４．１．１　システム変数を初期化する・・・・・・・・・・・・・・・・・・８３
４．２　ボリュームの初期化・・・・・・・・・・・・・・・・・・・・・・・・８６
４．２．１　ボリュームをフォッマットする・・・・・・・・・・・・・・・・８６
４．３　ボリュームの管理・・・・・・・・・・・・・・・・・・・・・・・・・９４
４．３．１　ボリュームをマウントする・・・・・・・・・・・・・・・・・・９４
４．３．２　ボリュームをディスマウントする・・・・・・・・・・・・・・・９８
４．３．３　ボリュームの使用量を得る・・・・・・・・・・・・・・・・・・１０３
４．４　ファイルの管理・・・・・・・・・・・・・・・・・・・・・・・・・１０７
４．４．１　インデックスのファイルの管理情報を得る・・・・・・・・・・１０７
４．４．２　サイズ固定のファイルを作成する・・・・・・・・・・・・・・１１１
４．４．３　サイズ可変のファイルを作成する・・・・・・・・・・・・・・１２５
４．４．４　ファイル名からファイルＩＤを得る・・・・・・・・・・・・・１３１
４．４．５　ファイルの仮削除する・・・・・・・・・・・・・・・・・・・１３４
４．５　データの入出力・・・・・・・・・・・・・・・・・・・・・・・・・１３９
４．５．１　ファイルをオープンする・・・・・・・・・・・・・・・・・・１３９
４．５．２　ファイルをクローズする・・・・・・・・・・・・・・・・・・１４３
４．５．３　データを読み込む・・・・・・・・・・・・・・・・・・・・・１４６
４．５．４　データを書き込む・・・・・・・・・・・・・・・・・・・・・１５２
４．５．５　ヘッダを読み込む・・・・・・・・・・・・・・・・・・・・・１６５
４．５．６　ヘッダを書き込む・・・・・・・・・・・・・・・・・・・・・１６９
４．５．７　ファイルのサイズを得る・・・・・・・・・・・・・・・・・・１７４
４．５．８　ファイルの名前を得る・・・・・・・・・・・・・・・・・・・１７７
４．５．９　ファイルに名前を付ける・・・・・・・・・・・・・・・・・・１８０
４．６　ヘッダを順番に読み込む・・・・・・・・・・・・・・・・・・・・・１８３
４．６．１　読み込みポインタを先頭に戻す・・・・・・・・・・・・・・・１８３
４．６．２　次のヘッダを読み込む・・・・・・・・・・・・・・・・・・・１８６
４．７　システムの管理・・・・・・・・・・・・・・・・・・・・・・・・・１９０
４．７．１　すべてのファイルの管理情報を得る・・・・・・・・・・・・・１９０
４．７．２　ファイルの実削除をする・・・・・・・・・・・・・・・・・・１９４
４．７．３　ファイルの再登録をする・・・・・・・・・・・・・・・・・・２０１
４．７．４　ファイルを書き込み禁止にする・・・・・・・・・・・・・・・２０５
４．７．５　ファイルを書き込み可能にする・・・・・・・・・・・・・・・２０８
４．７．６　ファイルを読み込み禁止にする・・・・・・・・・・・・・・・２１１
４．７．７　ファイルを読み込み可能にする・・・・・・・・・・・・・・・２１４
４．７．８　ファイルをシステム状態にする・・・・・・・・・・・・・・・２１７
４．７．９　ファイルのシステム状態を解除する・・・・・・・・・・・・・２２０
４．７．10　ファイルをディレクトリ状態にする・・・・・・・・・・・・・２２３
４．７．11　ファイルのディレクトリ状態を解除する・・・・・・・・・・・２２６

５．　サービス関数および内部関数のモジュール構造・・・・・・・・・・・・２２９

６．　ＩＳ＆Ｃファイルマネージャ使用の手引き・・・・・・・・・・・・・・２３５

7．　ＩＳ＆Ｃファイルマネージャの移植手順について・・・・・・・・・・・・・２３８

8．　提供フロッピーのプログラムの説明・・・・・・・・・・・・・・・・・・・２４２

はじめに

本仕様書は、ＩＳ＆Ｃファイルマネージャソフトウェアのプログラム仕様について、モジュールごとの機能、入出力データフォーマット、エラー発生条件、アルゴリズム、検査指針、及びそれらの使用方法を記述したものである。なお、本仕様書に示されているプログラムのモジュール構造とアルゴリズムは、東京工業大学大山助教授によるインプリメンテーションを基本とした。また、アルゴリズムの記述にはＪＩＳフローチャートを、データ構造と手続きの説明にはＫ＆Ｒ（カーニハン＆リッチー）のＣ言語を使用している。

以下に、本仕様書の概要を示す。

１．サービス関数の機能と一覧

ソフトウェア全体のサービス関数の機能と一覧についての説明。

２．共通データ構造（メモリ上での管理テーブル）

本仕様書の中で共通に使用するデータ構造（グローバル変数）についての説明。

３．共通内部関数

本仕様書の中で共通に使用する内部関数についての説明。

４．サービス関数の設計仕様

各サービス関数単位に、以下の項目について記述している。

・サービス関数の名称
・サービス関数の機能
・入出力データ（コーリングシーケンス）のフォーマット
・エラーの発生条件
・境界条件と注意事項の説明
・ＪＩＳフローチャートによるアルゴリズムの記述
・アルゴリズムに対する補足説明
・サービス関数単位の移植における検査指針

5．サービス関数および内部関数のモジュール構造

各ルーチンの参照関係を記述している。

6．ＩＳ＆Ｃファイルマネージャ使用の手引き

本プログラムの使用手順についての記述。

7．ＩＳ＆Ｃファイルマネージャの移植手順について

本プログラムはＳＵＮ　ＯＳを標準に記述しているので、他のＯＳに移植する場合の変更留意点を記述している。

8．提供フロッピーのプログラムの説明

提供されるフロッピーに格納されている各ファイルの内容の説明。

１．サービス関数の機能と一覧

ＩＳ＆Ｃファイルマネージャソフトウェア全体のサービス関数を以下に示す。これらのモジュールはアプリケーションプログラムから直接に呼び出され、ＩＳ＆Ｃ光磁気ディスクに対する各種のサービス処理を実行する。サービス関数はＩＳ＆Ｃ光磁気ディスクに対するサービスの種類から、以下の如く分類される。なお、アプリケーションを開発する場合にこれらのサービス関数をどのように組み合わせて使用するかについては、提供フロッピーに格納されているテストプログラム例を参照していただきたい。

１．１　システム変数の初期化

システム変数を初期化する場合に使用する。

①システム変数を初期化する　　ifm_initial()

１．２　ボリュームの初期化

物理フォーマットされたディスクを、論理的なＩＳＡＣ光磁気ディスクフォーマットとして初期化する場合に使用する。

①ボリュームをフォーマットする　　ifm_format_media()

１．３　ボリュームの管理

ボリュームのマウント、ディスマウント、ボリューム使用量の取得、メモリ上の情報と光磁気ディスクメディア内の情報の統一の各サービス関数が含まれる。

①ボリュームをマウントする　　ifm_mount_media()
②ボリュームをディスマウントする　　ifm_dismount_media()
③ボリュームの使用量を得る　　ifm_get_space()

１．４　ファイルの管理

ファイル管理情報の取得、ファイルの作成と削除、ファイルＩＤの検索の各サービス関数が含まれる。

①ファイルの管理情報を得る　　ifm_get_list()
②サイズ固定のファイルを作成する　　ifm_create_file()
③サイズ可変のファイルを作成する　　ifm_create_file_variable()
④ファイル名からファイルＩＤを得る　　ifm_get_file_id()
⑤ファイルを仮削除する　　ifm_delete_file()

１．５　ファイル単位のデータの入出力

ファイルのオープンとクローズ、データとヘッダの読み書き、ファイルサイズの取得、ファイル名の管理の各サービス関数が含まれる。

①ファイルをオープンする	ifm_open_file()
②ファイルをクローズする	ifm_close_file()
③データを読み込む	ifm_read_data()
④データを書き込む	ifm_write_data()
⑤ヘッダを読み込む	ifm_read_header()
⑥ヘッダを書き込む	ifm_write_header()
⑦ファイルのサイズを得る	ifm_get_file_size()
⑧ファイルの名前を得る	ifm_get_filename()
⑨ファイルの名前を付ける	ifm_put_filename()

１．６　ヘッダを順番に読み込む

ヘッダ情報を、ヘッダゾーンの順番に読み込むための各サービス関数が含まれる。

①読み込みポインタを先頭に戻す	ifm_reset_header_counter()
②次のヘッダを読み込む	ifm_read_header_sequential()

１．７　システムの管理

おもにシステム管理者によって使用されるモジュールで、すべてのファイルの管理情報の取得、削除されているファイルの再登録と実削除、ファイルの属性の変更の各サービス関数が含まれる。

①すべてのファイルの管理情報（ポインタを除く）を得る	ifm_get_list_all()
②ファイルを実削除する	ifm_delete_actually()
③ファイルを再登録する	ifm_recover_temporaly_deleted()
④ファイルを書込み禁止にする	ifm_write_protect_on()
⑤ファイルを書込み可能にする	ifm_write_protect_off()
⑥ファイルを読み込み禁止にする	ifm_read_protect_on()
⑦ファイルを読み込み可能にする	ifm_read_protect_off()
⑧ファイルの属性をシステム状態にする	ifm_system_flag_on()
⑨ファイルの属性をシステム状態から解除する	ifm_system_flag_off()
⑩ファイルの属性をディレクトリ状態にする	ifm_directory_flag_on()
⑪ファイルの属性をディレクトリ状態から解除する	ifm_directory_flag_off()

2．共通データ構造（メモリ上での管理テーブル）

本説明書の中で共通に使用するデータ構造について説明する。本説明書に記載したインプリメンテーション例では特に処理速度を改善するために、メモリ上に各種のテーブルを常駐させている。本説明書では、ある特定のオペレーティングシステムに依存した機能は使用していないので、この部分は実際に使用するオペレーティングシステムの機能に応じて自由に変更して差しつかえない。

２．１　システム定数

```
#define N_VOL       2           /*  最大マウント可能ボリューム数          */
#define N_FIL       2           /*  最大オープン可能ファイル数            */

#define ISAC        "IS&C"      /*  初期化識別子                          */
#define VS          "01.0"      /*  バージョン番号                        */
#define MAXZONE     307         /*  ゾーン数                              */
#define MAX_CONT_SECTOR 32767   /*  連続セクタ数                          */
```

２．２　エラー発生位置とコード

```
extern long  err_loc                /*  エラー発生位置（規格書に示された関数番号と関数
                                        内の位置）                                  */
----  論理エラー  ----
#define ENUNIT      0x0001          /*  存在しないユニット番号を指定した           */
#define EMNTED      0x0002          /*  指定したユニットはマウントされている       */
#define ENBUFF      0x0005          /*  マウントされているボリュームの数が多すぎる
                                        または、オープンされているファイルが多すぎる */
#define ENMNTD      0x0006          /*  指定のボリュームはマウントされていない     */
#define EPARAM      0x0007          /*  関数のパラメータが不正である               */
#define ENINDX      0x0008          /*  インデックス領域に空きがない               */
#define ENDATA      0x0009          /*  データ領域に空きがない                     */
#define EFLNEX      0x000A          /*  指定したファイルが存在しない               */
#define EOPNED      0x000B          /*  指定したファイルはオプーンされている       */
#define ENPERM      0x000C          /*  禁止されているアクセスを行おうとした       */
#define ENOPND      0x000D          /*  指定したファイルはオープンされていない     */
#define ESZOVR      0x000E          /*  開始位置がファイルのデータサイズ以上である */
#define ENDELD      0x000F          /*  指定したファイルは仮削除されていない       */
#define ESYSRD      0x0010          /*  ＡゾーンのＲＥＡＤに失敗し、バックアップゾーン
                                        のデータを使用した                         */
#define ESYSWR      0x0011          /*  Ａゾーンの書き込みに失敗した               */
#define ENHEAD      0x0020          /*  ヘッダ領域に空きがない                     */
#define EBNKCK      0x0030          /*  読み込み中に空白が見つかった               */
#define ELOGRD      0x0040          /*  読み込み時に論理エラーが発生した           */
#define ELOGWR      0x0041          /*  書き込み時に論理エラーが発生した           */
----  オペレータの介入でリカバリ出来るエラー  ----
#define EMWPRT      0x5000          /*  ライトプロテクトメディアに書き込もうとした */
#define EUNRDY      0x5001          /*  ユニットノットレディ                       */
----  メディアの診断を必要とするエラー  ----
#define ENVINF      0x6000          /*  vol_dtの内容が正しいボリューム管理情報ではない*/
#define ENISAC      0x6001          /*  メディアがＩＳ＆Ｃ対象フォーマットではない */
#define EMEDUM      0x6002          /*  ディスクの物理エラーが発生した             */
----  ドライブの診断を必要とするエラー  ----
#define ETMOUT      0x7000          /*  タイムアウト                               */
#define EDRIVE      0x8000          /*  物理エラーが発生した。いずれのエラーにも当ては
                                        まらない場合にのみ発行される。又、下位バイトに
                                        はエラーの状況を通知するエラーセンスコードが入
                                        っている場合もある                         */
#define ESYSIO      0xC000          /*  システム領域アクセス時に物理エラーが発生した */
```

２．３　Ｉ／Ｏバッファの定義

```
#define IOBUF       32              /*  I/O Buffer(32Kﾊﾞｲﾄ)                  */
extern long  SPDAT[IOBUF*1024/4];
extern long  buf_ss,                /*  ﾊﾞｯﾌｧの開始ｾｸﾀ                      */
             buf_es,                /*  ﾊﾞｯﾌｧの終了ｾｸﾀ                      */
             buf_ut;                /*  ﾊﾞｯﾌｧのﾕﾆｯﾄ番号                      */
```

２．４　ボリューム管理情報

```
extern long  flg_v[N_VOL] ;          /*  ボリューム管理情報フラグ                    */

struct datetype {
        short  year ;
        char   month, day, hour, mint ;
} ;
struct volumedatatype {
/*  第０セクタ  */
        char  init[4],               /*  初期化識別子                                */
              vs[4],                 /*  バージョン番号                              */
              appl[16],              /*  適用分野                                    */
              vlnm[32],              /*  ボリューム名称                              */
        long  vol_id ;               /*  ボリュームＩＤ                              */
        char  owner[32],             /*  所有者名                                    */
              owner_cd[32] ;         /*  所有者コード                                */
        struct datetype
              in_date ;              /*  初期化日時                                  */
        long  n_zn ;                 /*  ゾーン数                                    */
        short sct_zn,                /*  ゾーン内セクタ数                            */
              sct_bit ;              /*  セクタ容量                                  */
        long  loc_st,                /*  ゾーンテーブル開始番号                      */
              loc_zt,                /*  セクタテーブル開始番号                      */
              loc_idx ;              /*  インデックステーブル開始番号                */
        short sz_idx ;               /*  インデックスサイズ                          */
/*  第１セクタ  */
        long  n_idx,                 /*  インデックス総数                            */
              n_fil,                 /*  登録ファイル数                              */
              n_tdfl,                /*  仮削除ファイル数                            */
              n_fidx ;               /*  空きインデックス数                          */
        short n_sfil,                /*  システムファイル数                          */
              n_dfil ;               /*  ディレクトリファイル数                      */
        struct datetype
              update ;               /*  更新日時                                    */
        long  f_idx ;                /*  空きインデックス開始番号                    */
        short using ;                /*  ボリューム使用中フラグ                      */
} ;
extern struct volumedatatype  vol_dt[N_VOL] ; /*  ボリューム管理情報テーブル         */
```

２．５　セクタテーブル

```
extern long  flg_s[N_VOL] ;          /*  セクタテーブルフラグ                    */

#define BS_ST MAXZONE                /*  セクタテーブルのバッファサイズ          */
extern long  st[N_VOL][BS_ST][32] ; /*  セクタテーブル                          */
```

２．６　ゾーンテーブル

```
#define UNDEF_ZONE  0
#define A_ZONE      1
#define B_ZONE      2
#define C_ZONE      3
#define D_ZONE      4
#define E_ZONE      5
#define F_ZONE      6
#define G_ZONE      7
#define H_ZONE      8
#define A_ZONE_BU  -1
#define B_ZONE_BU  -2
#define C_ZONE_BU  -3
#define D_ZONE_BU  -4
#define E_ZONE_BU  -5
#define F_ZONE_BU  -6
#define G_ZONE_BU  -7
#define H_ZONE_BU  -8

extern long  flg_z[N_VOL] ;          /*  ゾーンテーブルフラグ                    */

#define BS_ZT MAXZONE                /*  ゾーンテーブルのバッファサイズ          */
struct ztentrytype {
       short z_type,                 /*  ゾーン種別                              */
             fr_blk,                 /*  空きブロック数                          */
             bkup ;                  /*  バックアップゾーン番号                  */
} ;
extern struct ztentrytype  zt[N_VOL][BS_ZT] ; /*  ゾーンテーブル                 */
```

２．７　インデックステーブル

```
extern long  flg_i[N_VOL] ;          /*  インデックステーブルフラグ                  */

struct idxptrtype {
        long  loc ;                  /*  開始セクタ番号                              */
        short n_sct ;                /*  連続セクタ数                                */
} ;
/*  インデックステーブル（親）  */
struct fatheridxtype {
        long  id ;                   /*  ファイルＩＤ                                */
        char  fn[24] ;               /*  ファイル名                                  */
        struct datetype
              crtd,                  /*  作成日時                                    */
              updd ;                 /*  最終変更日時                                */
        long  fsz ;                  /*  ファイルバイト長                            */
        char  attr,                  /*  属性                                        */
              attr1 ;
        struct idxptrtype
              head,                  /*  ヘッダポインタ                              */
              ptr[10] ;              /*  ポインタ                                    */
        char  dummy[8] ;
        long  son_no ;               /*  子のインデックステーブル番号                */
} ;
```

```
/*  インデックステーブル（子）  */
struct sonidxtype {
        long  tblno,                    /*  インデックステーブル番号                */
              fatno ;                   /*  親のインデックステーブル番号            */
        struct idxptrtype
              ptr[17] ;                 /*  ポインタ                                */
        long  nxt_no ;                  /*  次の子のインデックステーブル番号        */
} ;

struct listidxtype {
        long  id ;                      /*  ファイルＩＤ                            */
        char  fn[24] ;                  /*  ファイル名                              */
        struct datetype
              crtd,                     /*  作成日時                                */
              updd ;                    /*  最終変更日時                            */
        long  fsz ;                     /*  ファイルバイト長                        */
        char  attr,                     /*  属性                                    */
              attr1 ;
} ;

struct idxtype {
        long  u ;                       /*  ユニット番号                            */
        long  m ;                       /*  ファイルオープンモード                  */
        struct fatheridxtype f ;        /*  インデックステーブル（親）              */
        struct sonidxtype    s ;        /*  インデックステーブル（子）              */
        long  d ;                       /*  インデックステーブルフラグ              */
} ;
extern struct idxtype  idx[N_FIL] ; /*  インデックステーブル                        */
```

２．８　インデックステーブルのファイル属性

```
#define BLKFILE     0x01        /*  サイズ可変ファイル                  */
#define WRPROTECT   0x02        /*  書き込み禁止ファイル                */
#define RDPROTECT   0x04        /*  読み出し禁止ファイル                */
#define SYSFILE     0x08        /*  システムファイル                    */
#define DIRFILE     0x10        /*  ディレクトリファイル                */
#define WRITING     0x40        /*  書き込み中フラグ                    */
#define TEMPDEL     0x80        /*  仮削除                              */
```

２．９　ヘッダコントローラ

```
extern long  b_pnt[N_VOL] ;     /*  カレントなＢゾーン中のヘッダ読み込み位置    */
```

２．10　オープンモード

```
#define ReadOnly    0x01        /*  読み出し専用                        */
#define WriteOnly   0x02        /*  書き込み専用                        */
#define ReadWrite   0x03        /*  読み書き用                          */
```

２．11　適用分野を示すフラグ

```
extern long  flg_medical[N_VOL]     /*  医療分野か否かを示すフラグ          */
```

3．共通内部関数の説明

本関数群は、ファイルマネージャ内部で共通に使用されるものである。

1. ＭＯＤ*への書込み	iut_write_mod()
2. ＭＯＤ*からの読出し	iut_read_mod()
3. ファイルインデックスを読出す	iut_get_index()
4. ファイルインデックスを書込む	iut_put_index()
5. ディスクの排出を禁止する	iut_lock_mod()
6. ディスクの排出を許す	iut_unlock_mod()
7. 新たなゾーンを得る	iut_define_zone()
8. 指定した種別のゾーン中の予約領域のポインタを得る	iut_get_pointer()
9. 指定したゾーン中の予約領域の先頭のポインタを得る	iut_get_start_sector()
10. 指定したポインタの領域を解放する	iut_release_pointer()
11. Ｂゾーン中にヘッダのポインタを得る	iut_get_header_pointer()
12. ワードデータ列のコピー	iut_copy_word()
13. ４バイトデータ列のコピー	iut_copy_dword()
14. バイトスワップを行う	iut_byte_swap()
15. ボリューム管理情報その１（第０セクタ）への書込み	iut_write_volume_info_0()
16. ボリューム管理情報その２（第１セクタ）への書込み	iut_write_volume_info()
17. デバイスをオープンする	iut_open_devices()

18. デバイスをクローズする	iut_close_devices()
19. データを読み込む	iut_read_data()
20. データを書き込む	iut_write_data()
21. 論理ユニットが動作可能かをチェックする	iut_test_unit_ready()
22. 論理ユニットの状態をセットする	iut_rezero_unit()
23. 論理ユニットの容量をセットする	iut_read_capacity()
24. メディアの排出を許す	iut_unlock_media()
25. メディアの排出を禁止する	iut_lock_media()
26. センスデータをイニシエータへ転送するよう要求する	iut_error_sense()
27. 日時を得る	iut_get_time()

*ＭＯＤ：Ｍａｇｎｅｔｏ－Ｏｐｔｉｃａｌ　Ｄｉｓｋ（光磁気ディスク）

３．１　ＭＯＤへの書込み

1)　long　iut_write_mod(unit, dt, st_sct, len, swp, istat)

long	unit;	（入力）	ユニット番号
char	*dt;	（入力）	データアレイ
long	st_sct;	（入力）	先頭セクタ番号
long	len;	（入力）	大きさ（バイト単位）
long	swp;	（入力）	バイトスワップのフラグ
long	*istat;	（出力）	ドライブからのステータス

return	0	正常終了
	0以外	異常終了

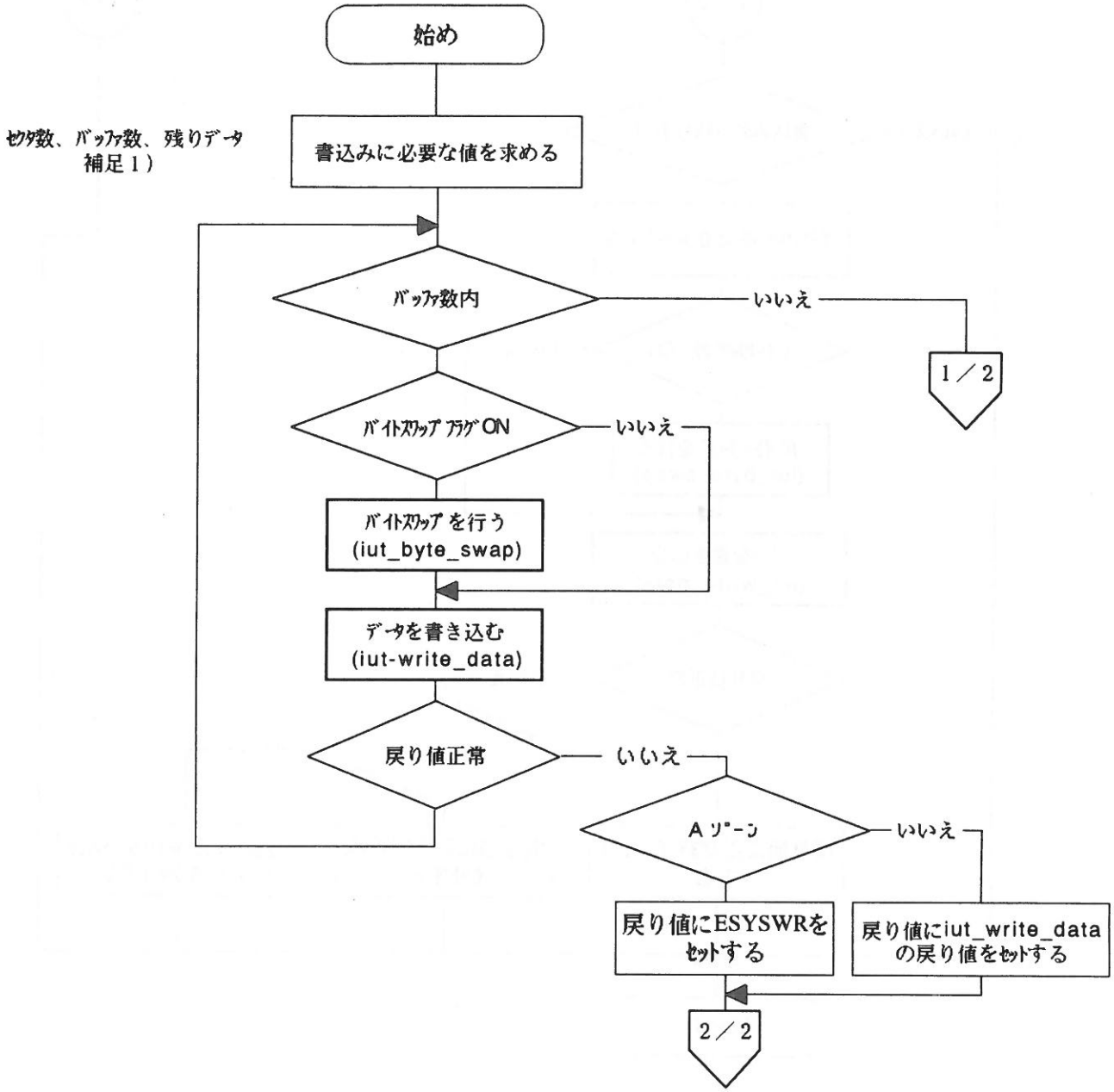
ESYSWR（Ａゾーンの書き込みに失敗した）注

注２．２エラーコード参照のこと

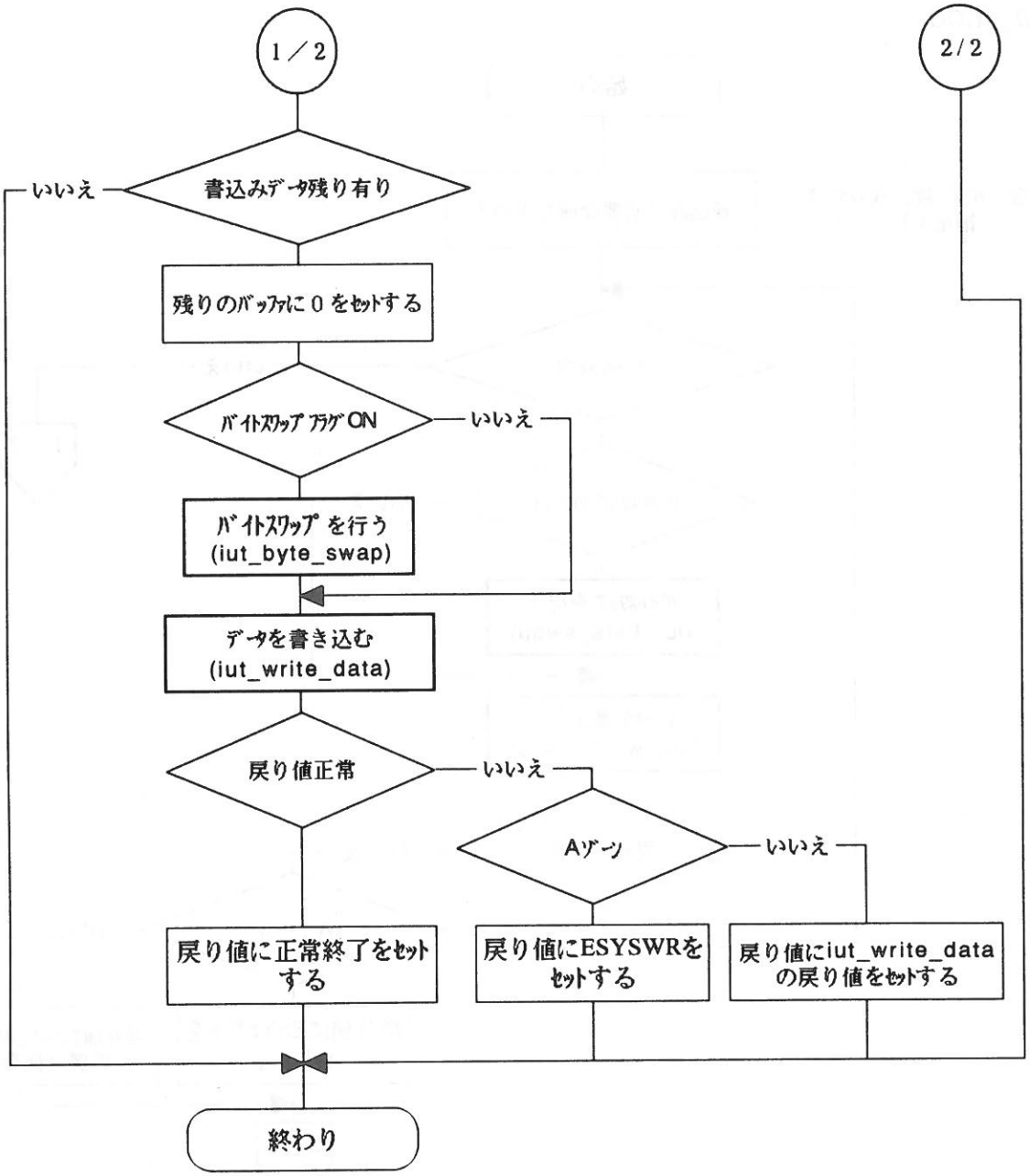
2)　機能

データを書き込む。

iut_write_mod



補足１）セクタ数＝（大きさ(len)＋１０２３）／１０２４
書込みに必要なバッファ数＝セクタ数／３２
書込みデータの残り＝セクタ数％３２


1／2
2/2
書込みデータ残り有り
いいえ
残りのバッファに０をセットする
バイトスワップ フラグ ON
いいえ
バイトスワップ を行う
(iut_byte_swap)
データを書き込む
(iut_write_data)
戻り値正常
いいえ
Aゾーン
いいえ
戻り値に正常終了をセット
する
戻り値にESYSWRを
セットする
戻り値にiut_write_data
の戻り値をセットする
終わり

関数iut_write_mod()の検査指針

関数 iut_write_mod()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査についてはマウントされていることを確認の上、検査を行うこと。

（１）機能に関する検査

1) 関数iut_write_mod() を用いて、MODへの書き込みを行う。
指定されたセクタにデータが書き込まれていることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。

2) Ａゾーンの書き込みに失敗した場合は、ESYSWRが返されることを確認する。

３．２　ＭＯＤからの読出し

1)　long　iut_read_mod(unit, dt, st_sct, len, swp, istat)

long	unit;	（入力）	ユニット番号
char	*dt;	（出力）	データアレイ
long	st_sct;	（入力）	先頭セクタ番号
long	len;	（入力）	大きさ（バイト単位）
long	swp;	（入力）	バイトスワップのフラグ
long	*istat;	（出力）	ドライブからのステータス

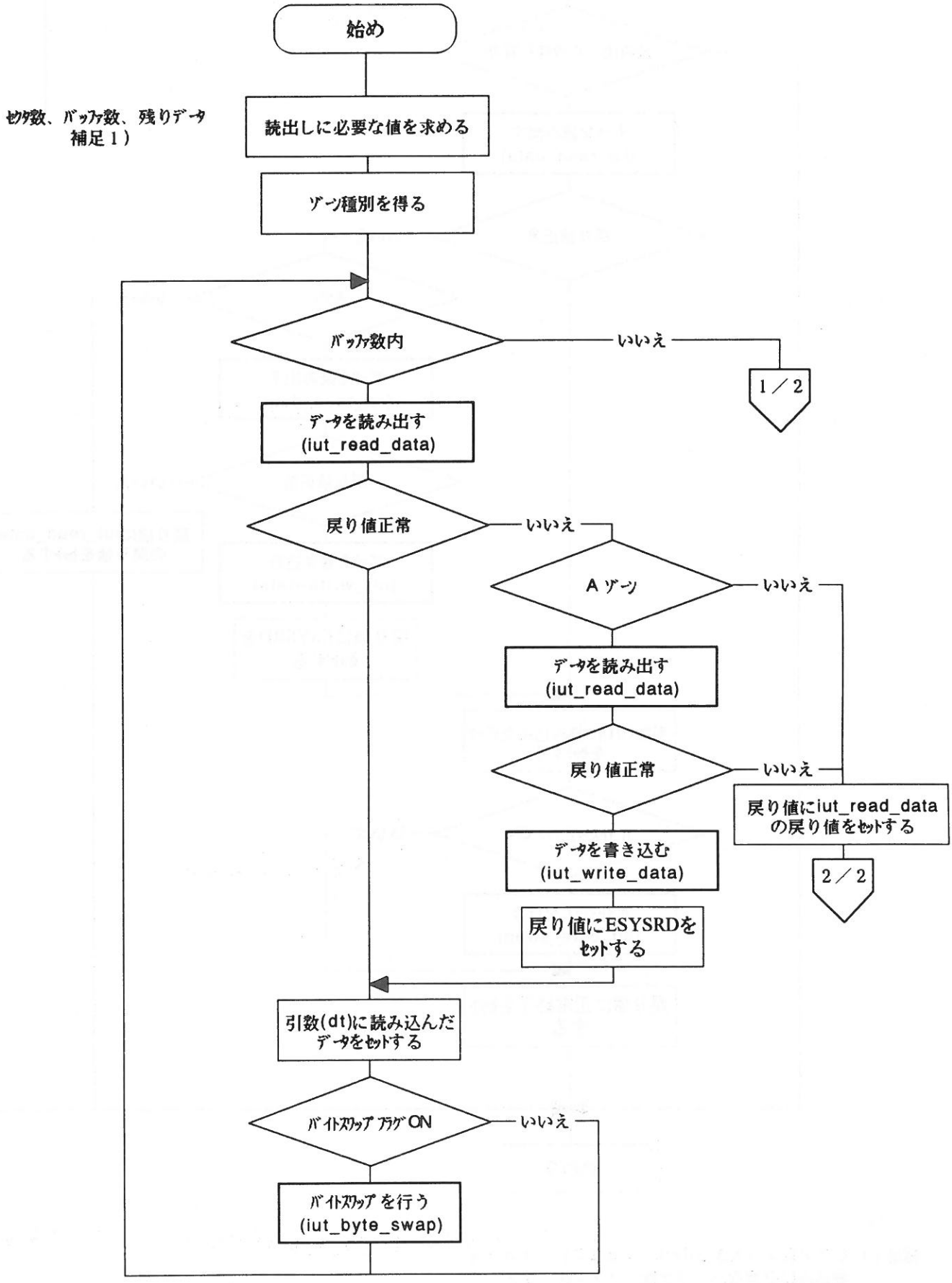
return	0	正常終了
	0以外	異常終了

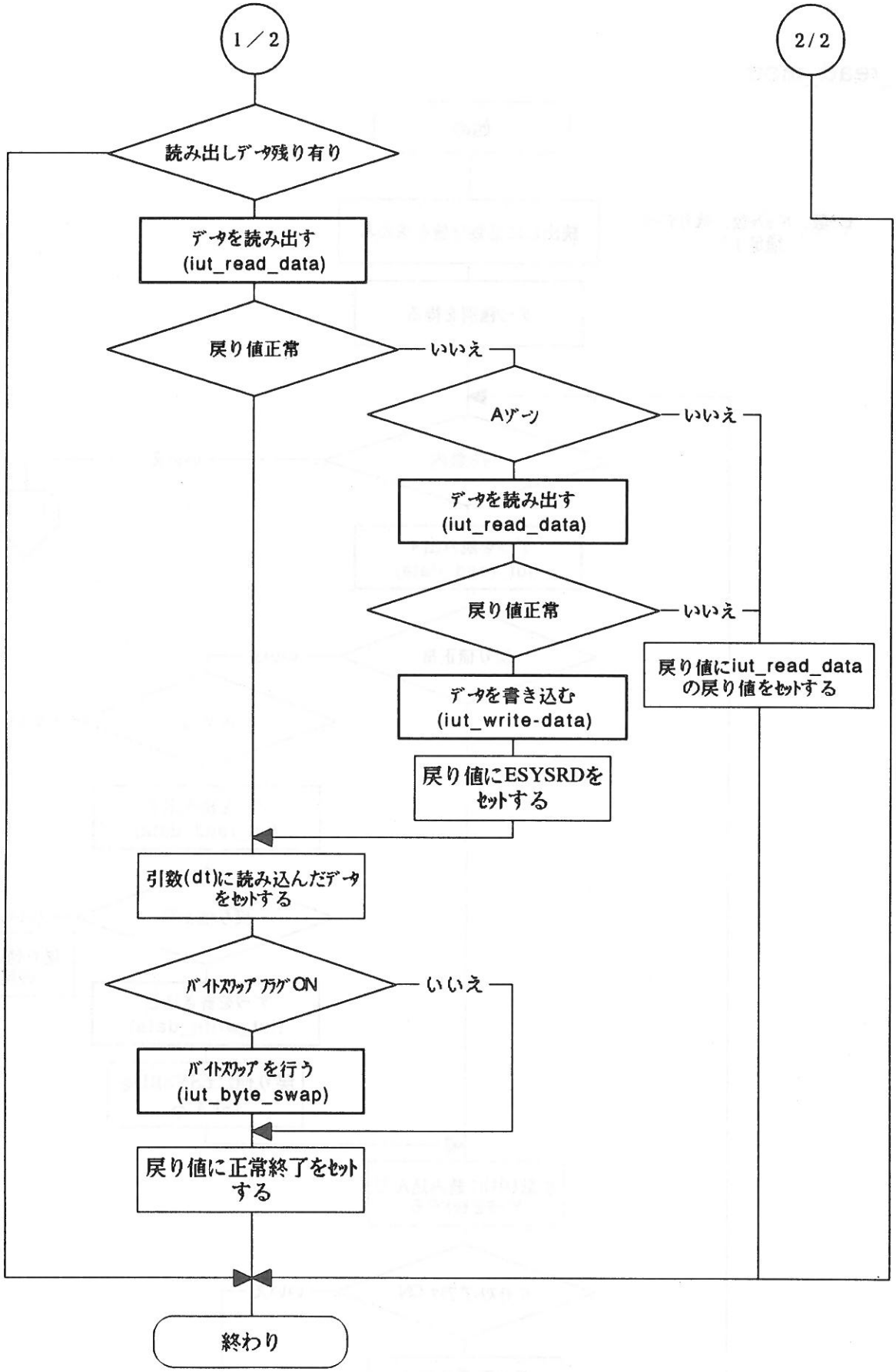
ESYSRD（ＡゾーンのＲＥＡＤに失敗し、バックアップゾーンのデータを使用した。また、Ａゾーンにバックアップゾーンのデータを書き込んだ）

2)　機能

データを読み出す。

iut_read_mod


始め
セクタ数、バッファ数、残りデータ
補足１）
読出しに必要な値を求める
ゾーン種別を得る
バッファ数内
いいえ
1／2
データを読み出す
(iut_read_data)
戻り値正常
いいえ
Aゾーン
いいえ
データを読み出す
(iut_read_data)
戻り値正常
いいえ
戻り値にiut_read_data
の戻り値をセットする
2／2
データを書き込む
(iut_write_data)
戻り値にESYSRDを
セットする
引数(dt)に読み込んだ
データをセットする
バイトスワップフラグON
いいえ
バイトスワップを行う
(iut_byte_swap)



補足１）セクタ数＝（大きさ(len)＋１０２３）／１０２４
書込みに必要なバッファ数＝セクタ数／３２
書込みデータの残り＝セクタ数％３２

関数iut_read_mod()の検査指針

関数iut_read_mod()はボリュームに対して、マウントされている状態の下で正常に動作するものであるため、機能に関する検査についてはマウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_read_mod()を用いて、MODからの読み出しを行う。
指定されたメモリにデータが読み込まれていることを確認する。

（2）戻り値に関する検査

1) 正常終了の場合には、0が返されることを確認する。

2) Aゾーンの読み込みに失敗し、バックアップゾーンのデータを使用した場合には、ESYSRDが返されることを確認する。

３．３　ファイルインデックスを読出す

1)　long　iut_get_index(unit, cid, array, flag)

long　unit;　（入力）　ユニット番号
long　cid;　（入力）　インデックス番号
char　*array;（出力）　データアレイ
long　*flag;　（出力）　インデックス識別フラグ

return　0　　正常終了
　　0以外　　異常終了

EFLNEX（指定されたファイルが存在しない）
EPARAM（関数のパラメータが不正である）

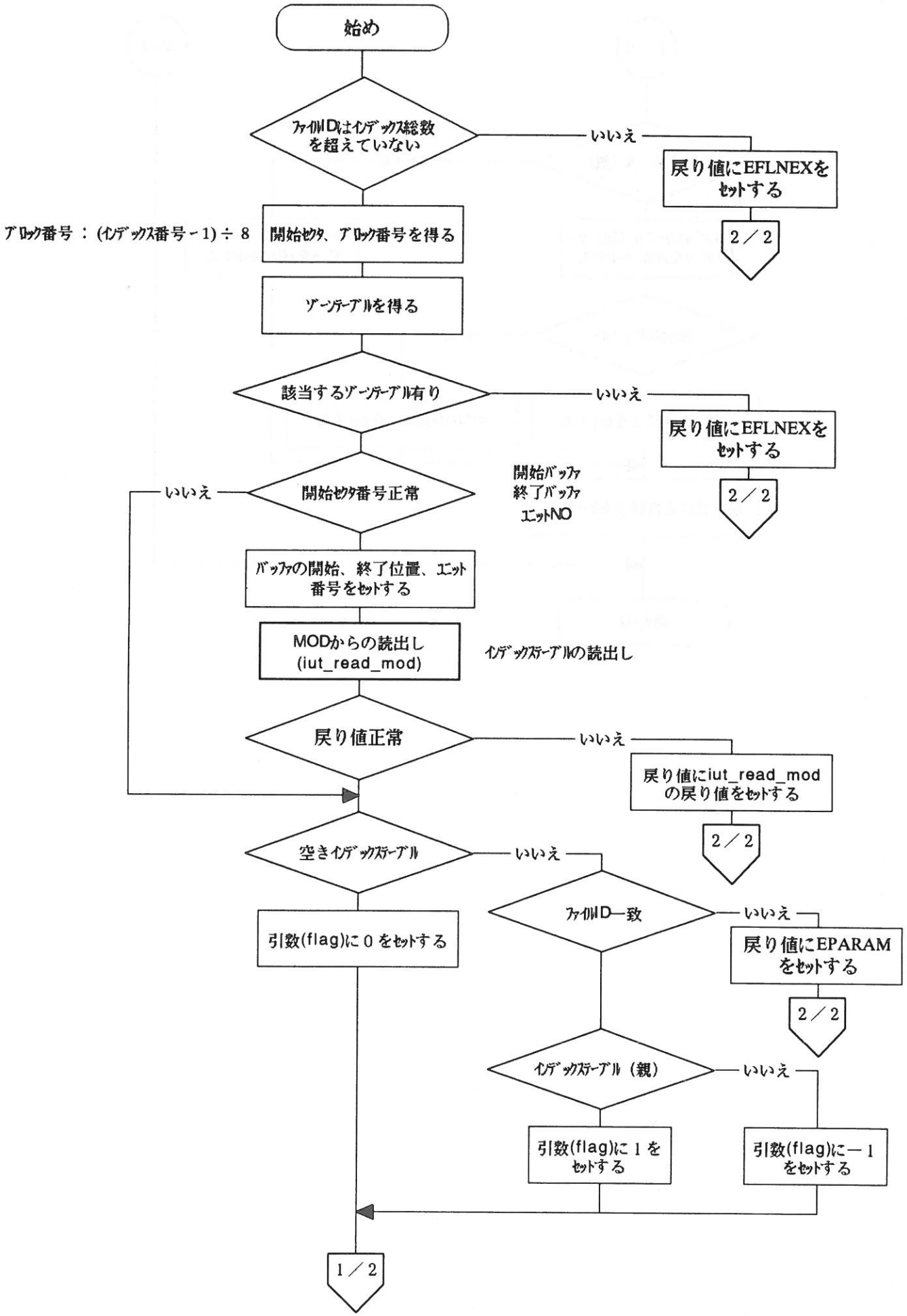
2)　機能

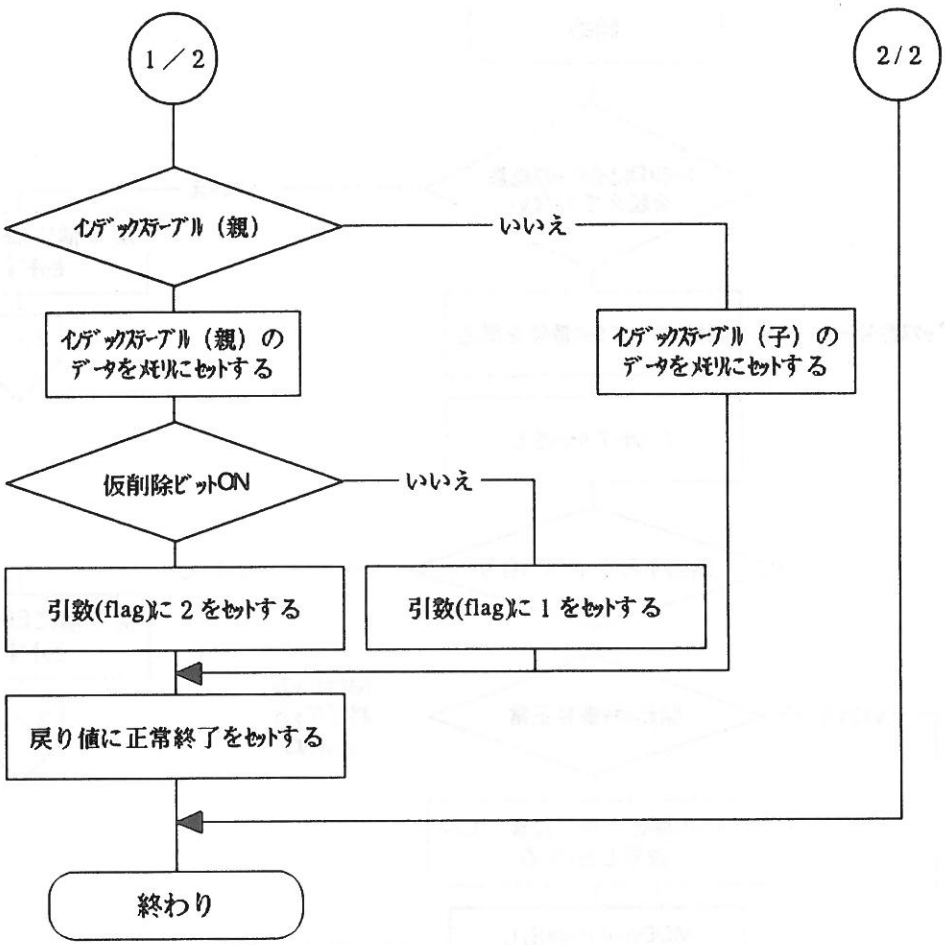
指定されたインデックステーブルをメモリに読み込む。

3)　備考

・arrayには必要に応じて、size of(struct fatheridxtype)又は、size of(sonidxtype)分の領域が確保されていなければならない。
・読み込まれたarrayインデックスデータは、構造体struct fatheridxtype又は、struct sonidxtypeで参照しなくてはならない。
・flagの戻り値
-1: 子のインデックス
0 : このインデックスは空である
1 : ファイルが存在する（親インデックスである）
2 : ファイルが仮削除されている

iut_get_index




1／2
2/2
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）
いいえ
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）のデｰﾀをﾒﾓﾘにｾｯﾄする
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）のデｰﾀをﾒﾓﾘにｾｯﾄする
仮削除ﾋﾞｯﾄON
いいえ
引数(flag)に 2 をｾｯﾄする
引数(flag)に 1 をｾｯﾄする
戻り値に正常終了をｾｯﾄする
終わり

関数iut_get_index()の検査指針

関数 iut_get_index()は予め作成されているファイルのインデックステーブルの読み込みを行うものである。よって本関数の動作を確認する以前にファイル作成関数ifm_create_file() 等での動作確認、ボリュームのマウント関数ifm_mount_media() の動作確認をしておかなければならない。

（1）機能に関する検査

1) 関数 iut_get_index()を用いてインデックステーブルの読み込みを行う。
指定したインデックス番号のテーブルを指定エリアに読み込むことを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 指定されたファイルが存在しない場合には、EFLNEXが返されることを確認する。

3) ファイルIDの値が不正の場合には、EPARAMが返されることを返されることを確認する。

4) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

３．４　ファイルインデックスを書込む

1)　long　iut_put_index(unit, cid, array)

long　unit;　（入力）　ユニット番号
long　cid;　（入力）　インデックス番号
char　*array;（入力）　データアレイ

return　0　　正常終了
　　0以外　　異常終了

EFLNEX（指定されたファイルが存在しない）

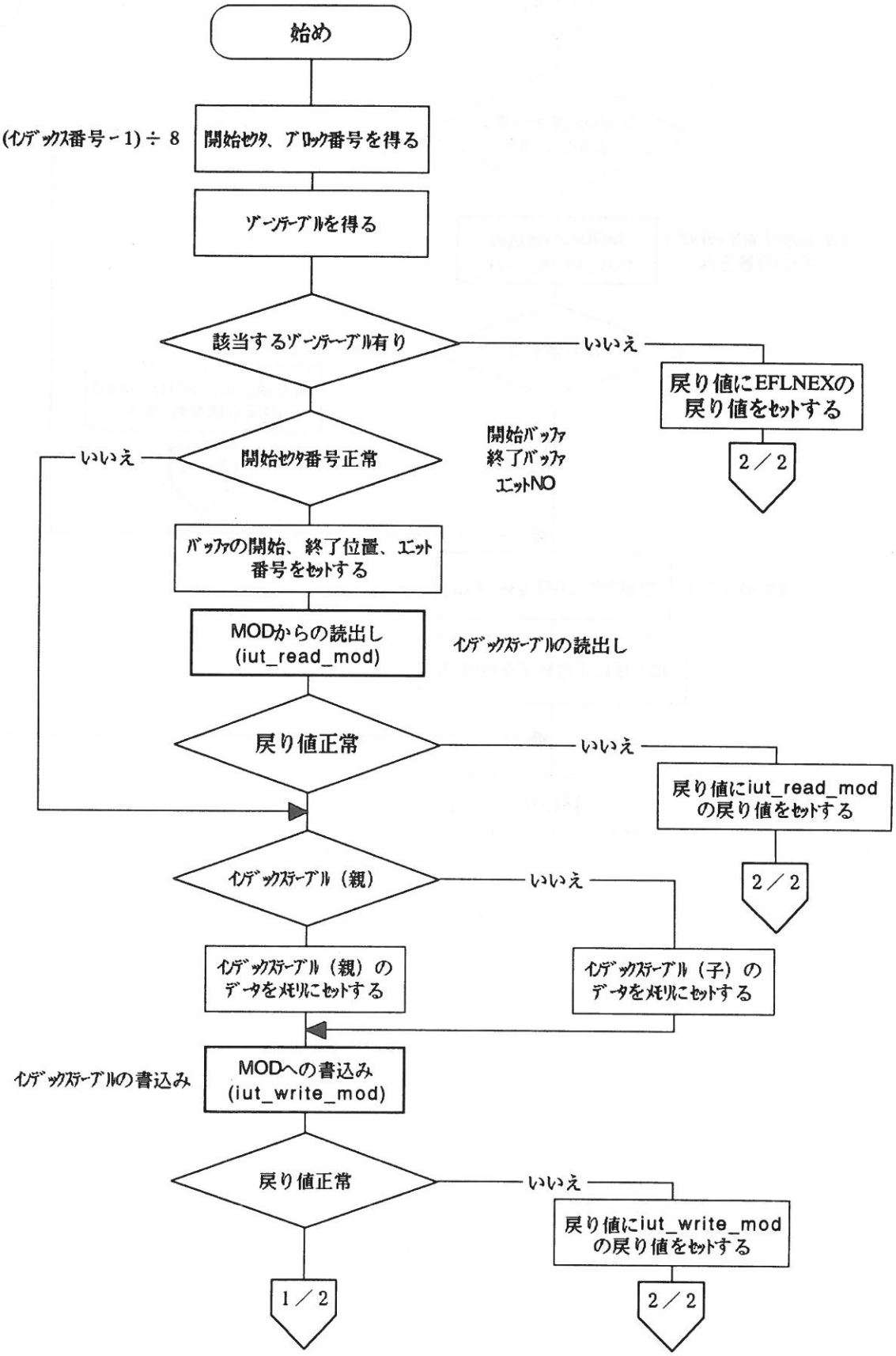
2)　機能

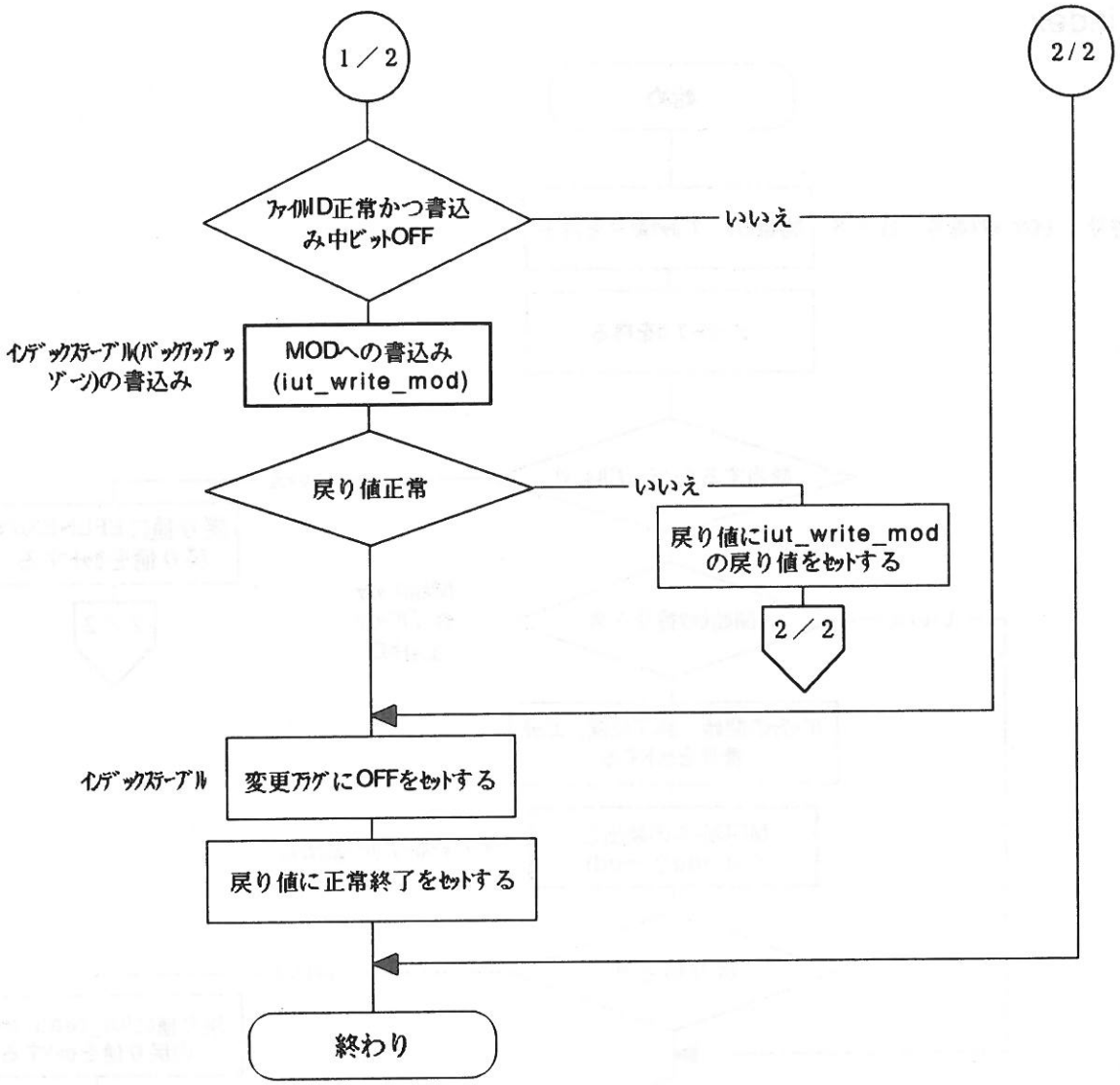
指定されたデータをインデックステーブルに書き込む。

3)　備考

・arrayは、struct fatheridxtypeまたは、struct sonidxtype型のテーブルへのポインタでなければならない。

iut_put_index




1／2
2/2
ファイルID正常かつ書込み中ビットOFF
いいえ
インデックステーブル(バックアップゾーン)の書込み
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
2／2
インデックステーブル
変更フラグにOFFをセットする
戻り値に正常終了をセットする
終わり

関数iut_put_index() の検査指針

関数 iut_put_index()は予め作成されているファイルのインデックステーブルの書き出しを行うものである。よって本関数の動作を確認する以前にファイル作成関数ifm_create_file() 等での動作確認、ボリュームのマウント関数ifm_mount_media() の動作確認をしておかなければならない。

（1）機能に関する検査

1) 関数 iut_put_index()を用いてインデックステーブルの書き出しを行う。
指定したインデックスデータを指定したインデックス番号へ書き出すことを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 指定されたファイルが存在しない場合には、EFLNEXが返されることを確認する。

3) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

３．５　ディスクの排出を禁止する

1)　long　iut_lock_mod(unit)

long　unit;　　（入力）　ユニット番号

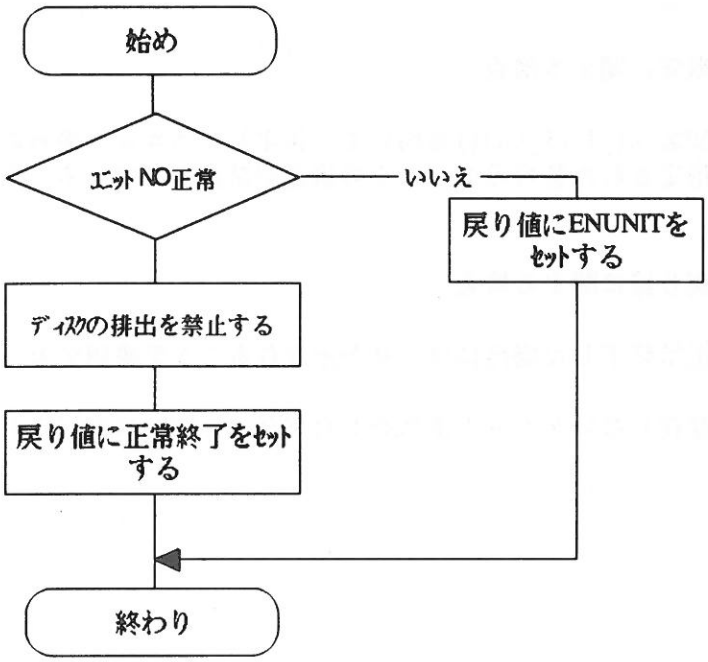
return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

ディスクの排出を禁止する。

iut_lock_mod


始め
ユニットNO正常
いいえ
戻り値にENUNITをセットする
ディスクの排出を禁止する
戻り値に正常終了をセットする
終わり

関数iut_lock_mod()の検査指針

関数 iut_lock_mod() はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_lock_mod()を用いて、指定したユニット番号のディスクの排出を禁止する。
指定された番号のディスクの排出が禁止されていることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 存在しないユニット番号をした場合は、ENUNITが返されることを確認する。

３．６　ディスクの排出を許す

1)　long　iut_unlock_mod(unit)

long　unit;　　（入力）　ユニット番号

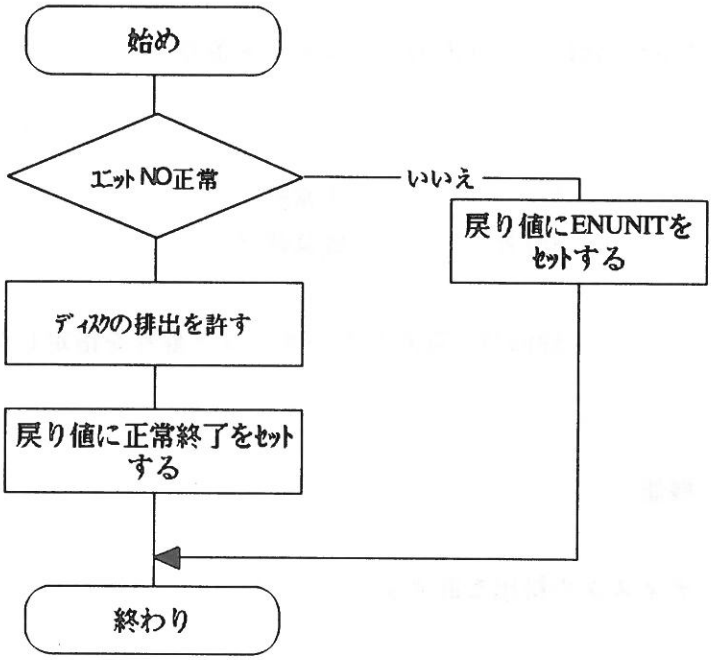
return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

ディスクの排出を許す。

iut_unlock_mod



関数iut_unlock_mod()の検査指針

関数 iut_unlock_mod() はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_unlock_mod()を用いて、指定したユニット番号のディスクの排出を許す。
指定された番号のディスクの排出が許されていることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 存在しないユニット番号をした場合は、ENUNITが返されることを確認する。

３．７　新たなゾーンを得る

1)　long　iut_define_zone(unit, zone)

long　unit;　　（入力）　ユニット番号
long　zone;　　（入力）　ゾーンの種別

return　0　　正常終了
　　0以外　　異常終了

EPARAM（関数のパラメータが不正である）
ENDATA（データ領域に空きがない）
ENINDX（インデックス領域に空きがない）

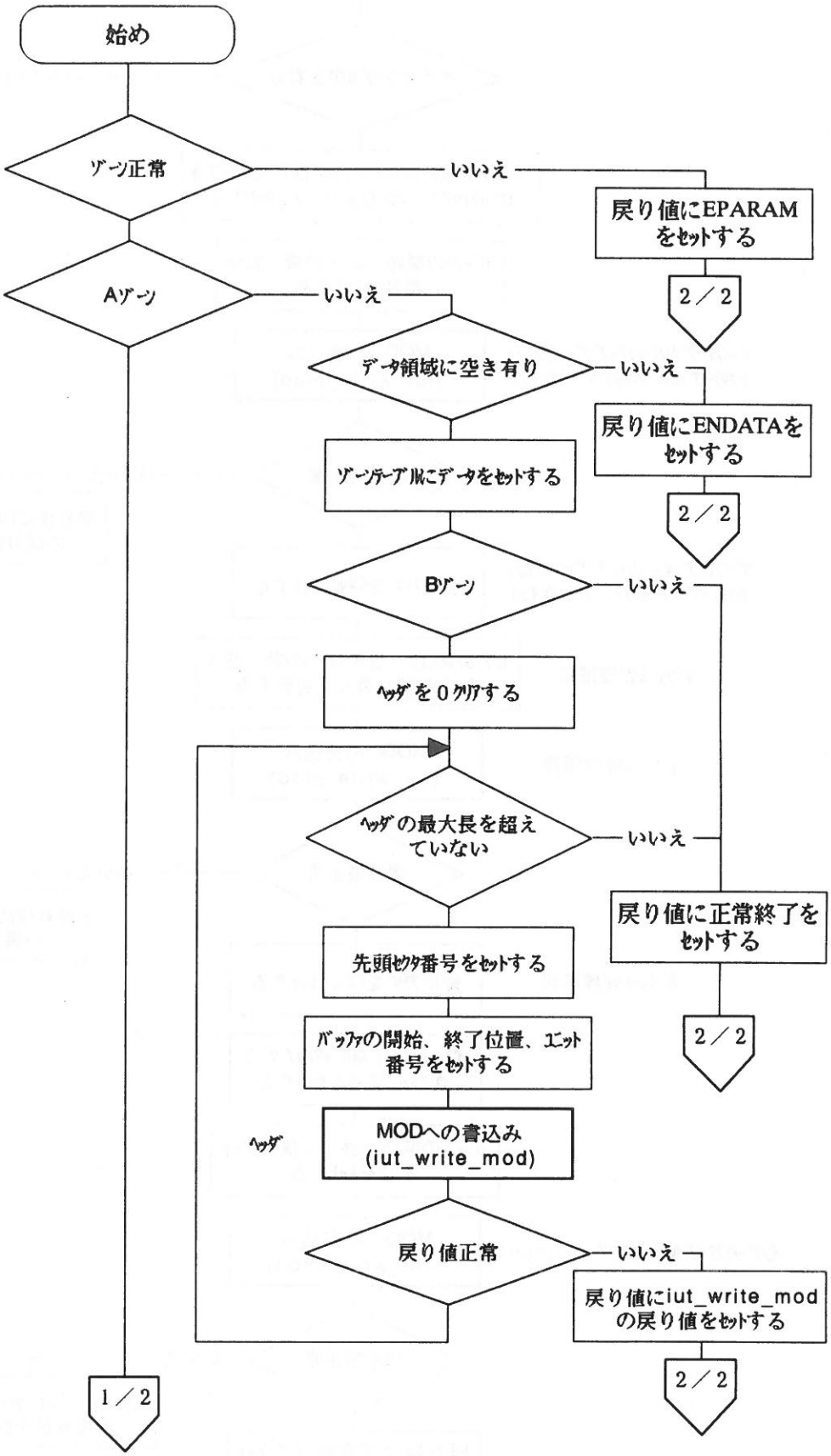
2)　機能

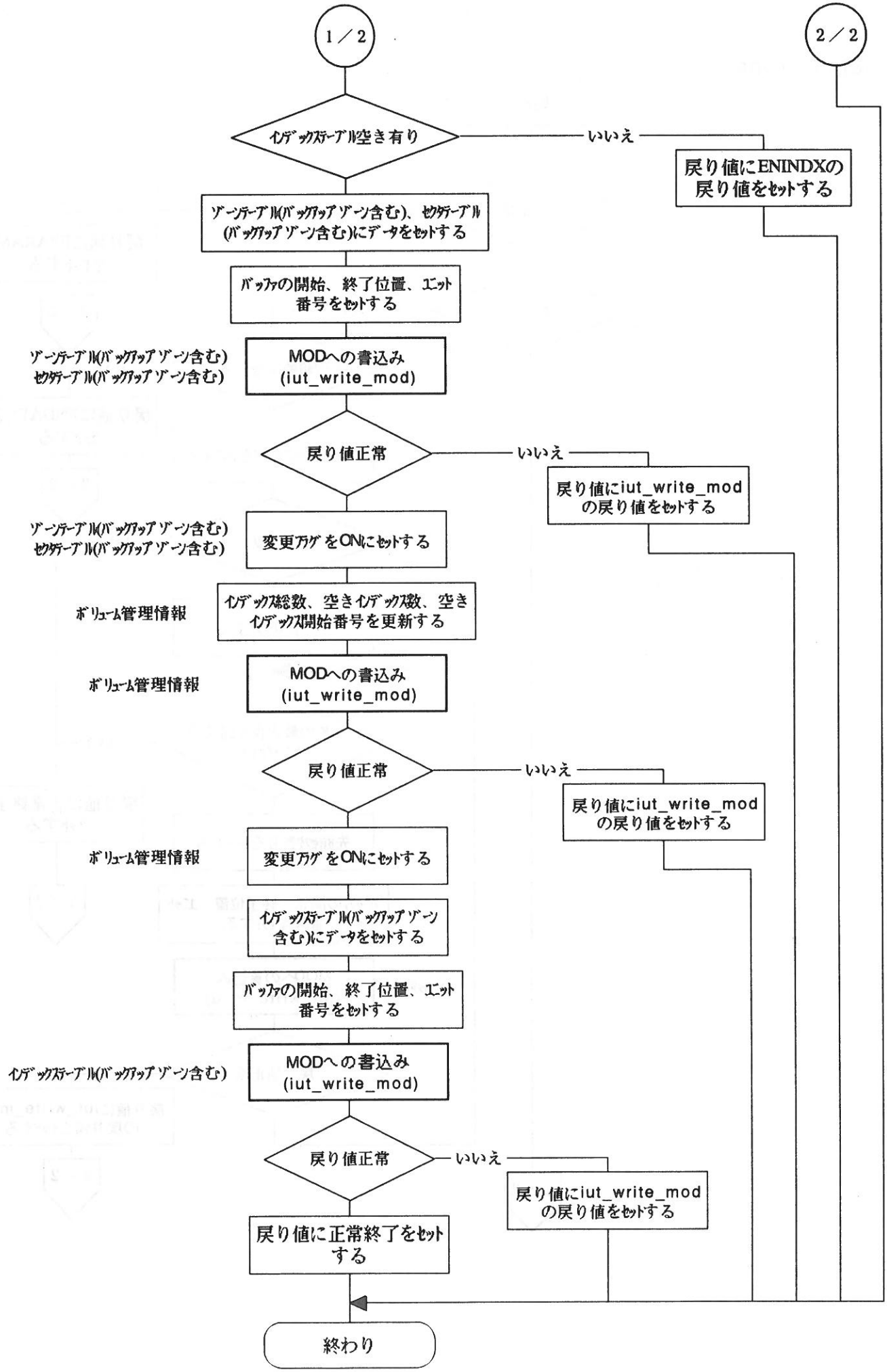
指定されたゾーン種別（Ａゾーン～Ｈゾーン）の新たなゾーンを得る。

3)　備考

・ゾーン種別
１→→Ａゾーン
２→→Ｂゾーン
３→→Ｃゾーン
４→→Ｄゾーン
５→→Ｅゾーン
６→→Ｆゾーン
７→→Ｇゾーン
８→→Ｈゾーン

iut_define_zone




1／2
2／2
インデックステーブル空き有り
いいえ
戻り値にENINDXの戻り値をセットする
ゾーンテーブル(バックアップゾーン含む)、セクタテーブル(バックアップゾーン含む)にデータをセットする
バッファの開始、終了位置、ユニット番号をセットする
ゾーンテーブル(バックアップゾーン含む)
セクタテーブル(バックアップゾーン含む)
MODへの書込み(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
ゾーンテーブル(バックアップゾーン含む)
セクタテーブル(バックアップゾーン含む)
変更フラグをONにセットする
ボリューム管理情報
インデックス総数、空きインデックス数、空きインデックス開始番号を更新する
ボリューム管理情報
MODへの書込み(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
ボリューム管理情報
変更フラグをONにセットする
インデックステーブル(バックアップゾーン含む)にデータをセットする
バッファの開始、終了位置、ユニット番号をセットする
インデックステーブル(バックアップゾーン含む)
MODへの書込み(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
戻り値に正常終了をセットする
終わり

関数iut_define_zone()の検査指針

関数 iut_define_zone()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数 iut_define_zone()を用いて、新たなゾーンを得る。
指定されたゾーン（A～H）種別を基に、新たなゾーンを得ることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) ゾーン種別に0を指定して、EPARAMが返されることを確認する。

3) ゾーン種別に1以外を指定、最大ゾーン数分ゾーンを確保して、ENDATAが返されることを確認する。

4) ゾーン種別に1を指定、最大ゾーン数分ゾーンを確保して、ENINDXが返されることを確認する。

３．８　指定した種別のゾーン中の予約領域のポインタを得る

1)　long　iut_get_pointer(unit, zone, pn)

long	unit;	（入力）	ユニット番号
long	zone;	（入力）	ゾーンの種別
struct idxptrtype	*pn;	（出力）	ポインタ

return	0	正常終了
	0以外	異常終了

EPARAM（関数のパラメータが不正である）

2)　機能

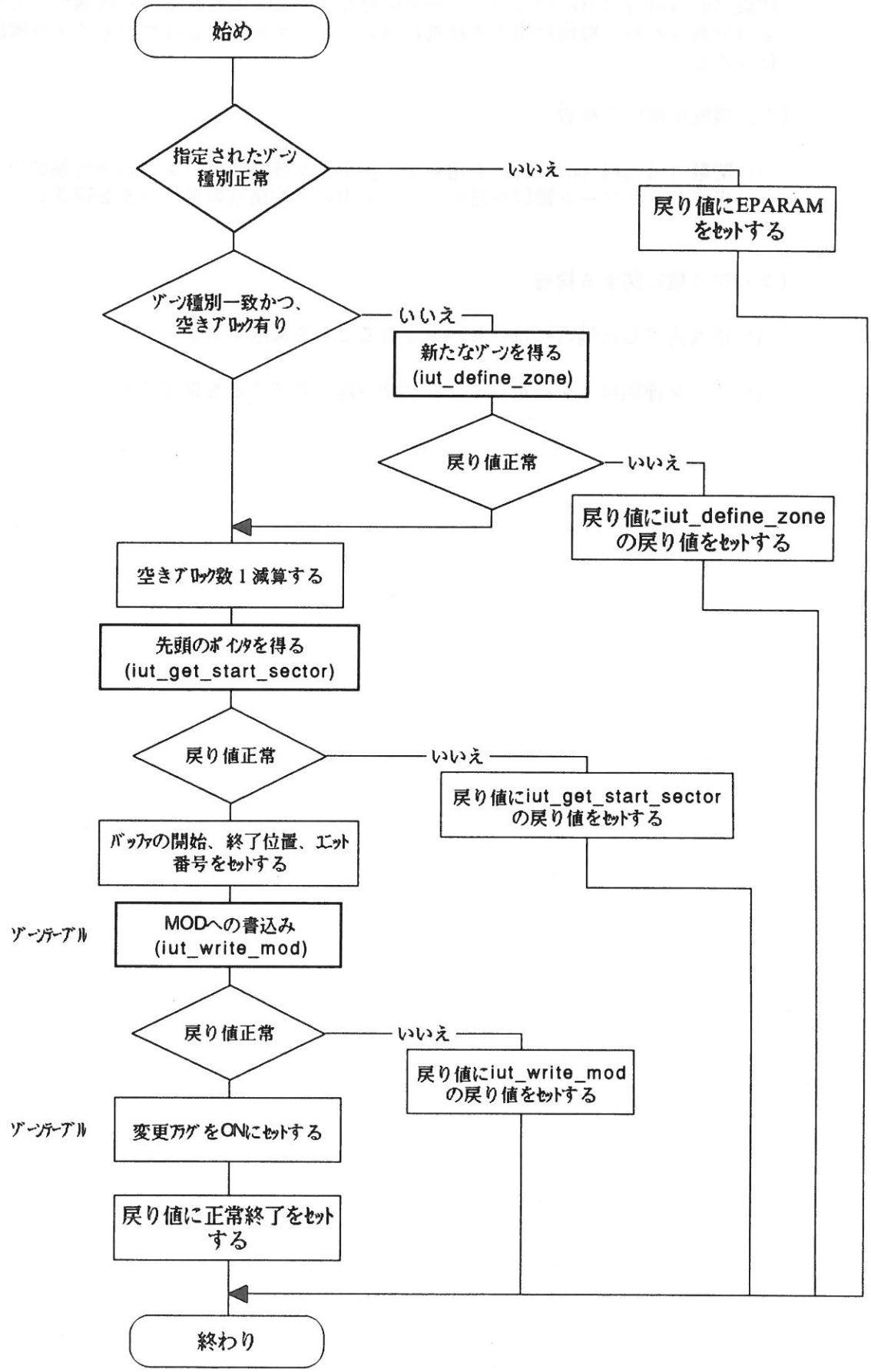
指定されたゾーン種別（Ｃゾーン～Ｈゾーン）のポインタを得る。

3)　備考

・ゾーン種別

３→→Ｃゾーン
４→→Ｄゾーン
５→→Ｅゾーン
６→→Ｆゾーン
７→→Ｇゾーン
８→→Ｈゾーン

iut_get_pointer



関数iut_get_pointer()の検査指針

関数 iut_get_pointer()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数 iut_get_pointer()を用いて、指定した種別のゾーン中の予約領域のポインタを得る。指定されたゾーン種別を基に、ゾーン中の予約領域のポインタを得ることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) ゾーン種別に1を指定して、EPARAMが返されることを確認する。

3．9　指定したゾーン中の予約領域の先頭のポインタを得る

1)　long　iut_get_start_sector(unit, znn, ptr)

```
long                unit;    (入力)   ユニット番号
long                znn;     (入力)   ゾーン番号
struct idxptrtype   *ptr;    (出力)   先頭セクタ番号
```

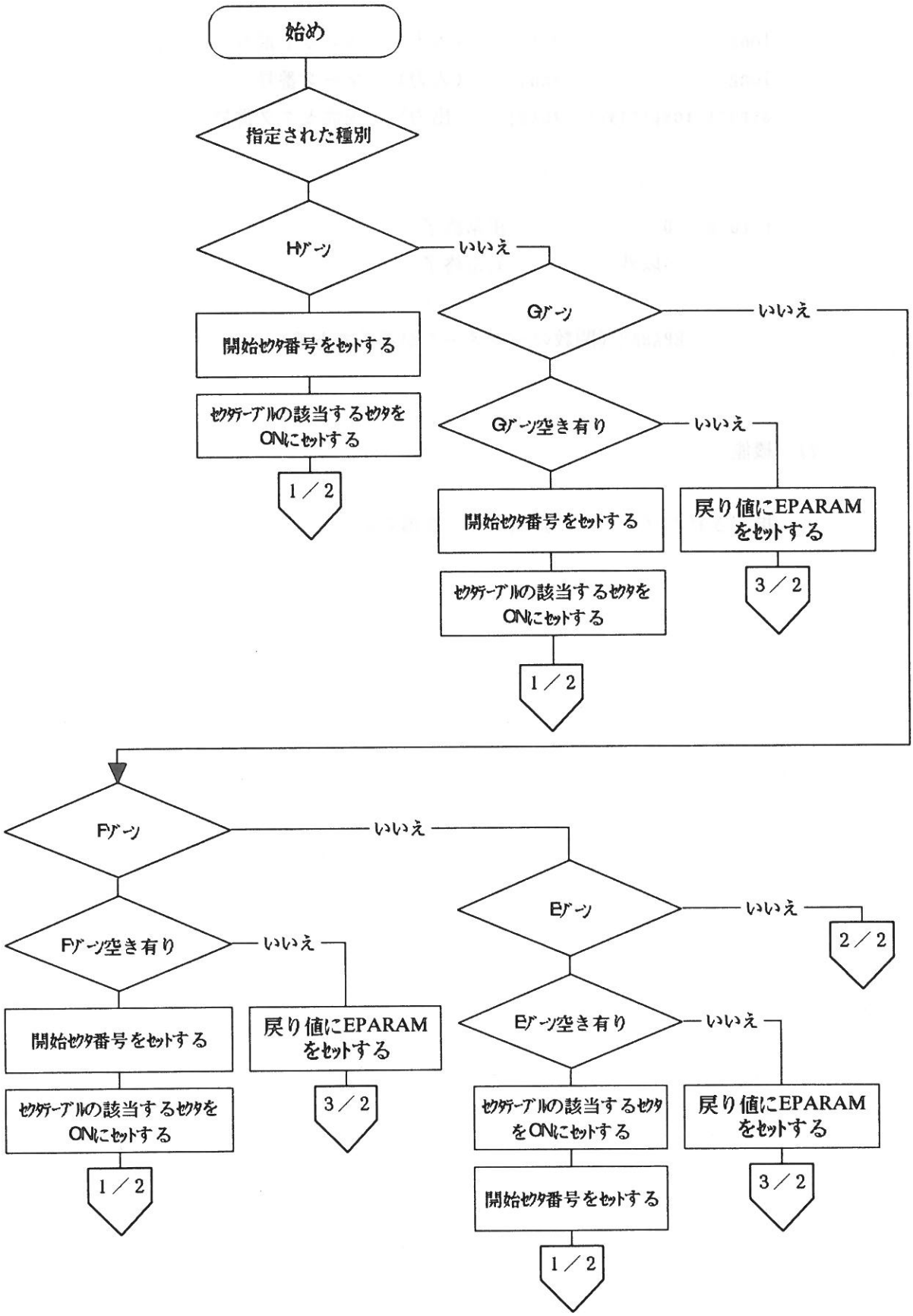
```
return   0          正常終了
         0以外      異常終了
```

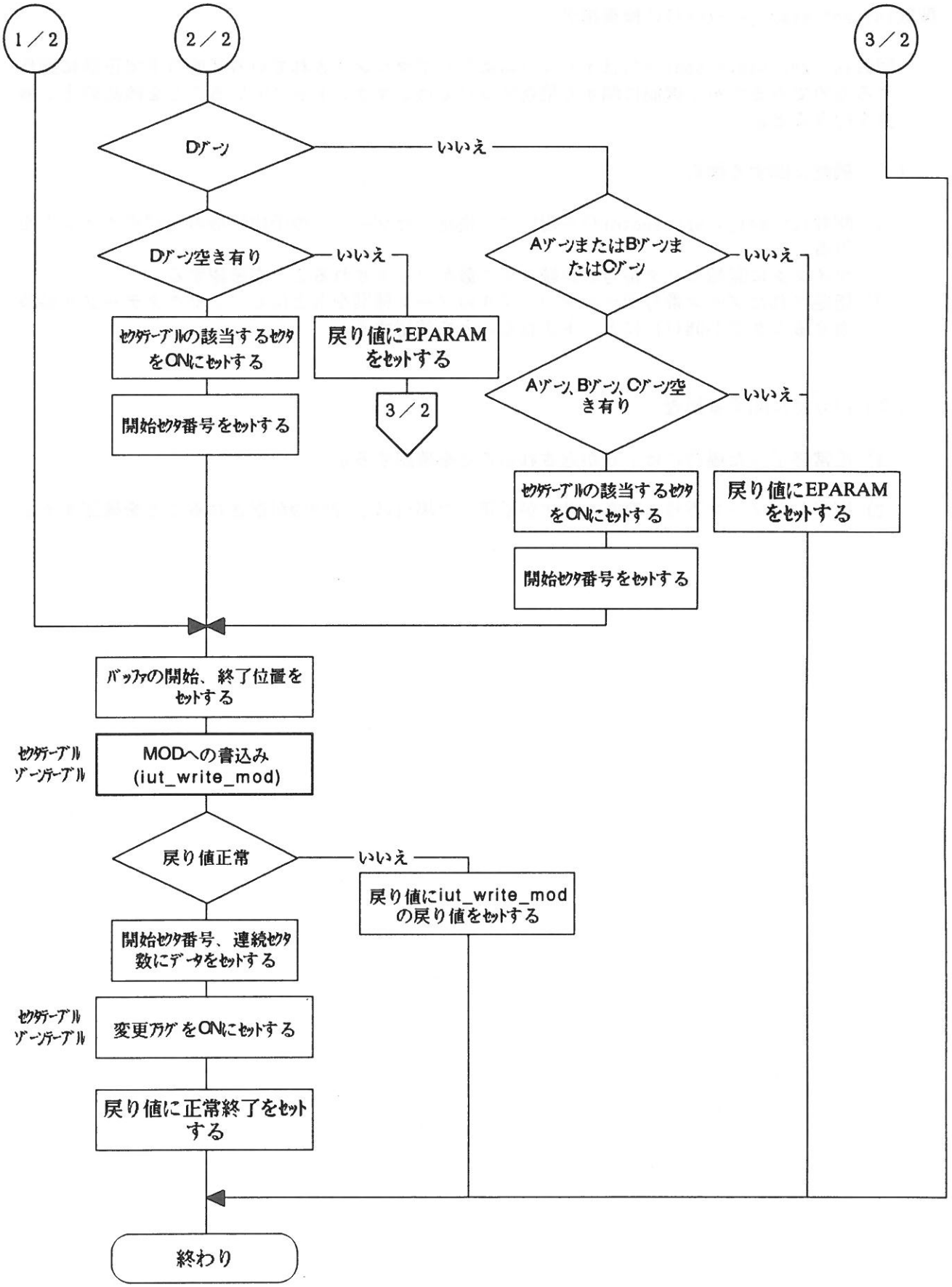
EPARAM（関数のパラメータが不正である）

2)　機能

指定されたゾーン中の先頭セクタを得る。

iut_get_start_sector




1／2
2／2
3／2
Dゾーン
いいえ
Dゾーン空き有り
いいえ
AゾーンまたはBゾーンまたはCゾーン
いいえ
セクタテーブルの該当するセクタをONにセットする
戻り値にEPARAMをセットする
3／2
開始セクタ番号をセットする
Aゾーン、Bゾーン、Cゾーン空き有り
いいえ
セクタテーブルの該当するセクタをONにセットする
戻り値にEPARAMをセットする
開始セクタ番号をセットする
バッファの開始、終了位置をセットする
セクタテーブル
ゾーンテーブル
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
開始セクタ番号、連続セクタ数にデータをセットする
セクタテーブル
ゾーンテーブル
変更フラグをONにセットする
戻り値に正常終了をセットする
終わり

関数iut_get_start_sector()の検査指針

関数iut_get_start_sector()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_get_start_sector()を用いて、指定したゾーン中の予約領域の先頭のポインタを得る。
ポインタに開始セクタ番号、連続セクタ数がセットされることを確認する。
2) 指定されたゾーン番号のゾーンテーブルのゾーン種別をもとにして、セクタテーブルの該当するセクタがON(1) にセットされることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 指定したゾーン番号に空きエリアが不足した場合は、EPARAMが返されることを確認する。

3．10　指定したポインタの領域を解放する

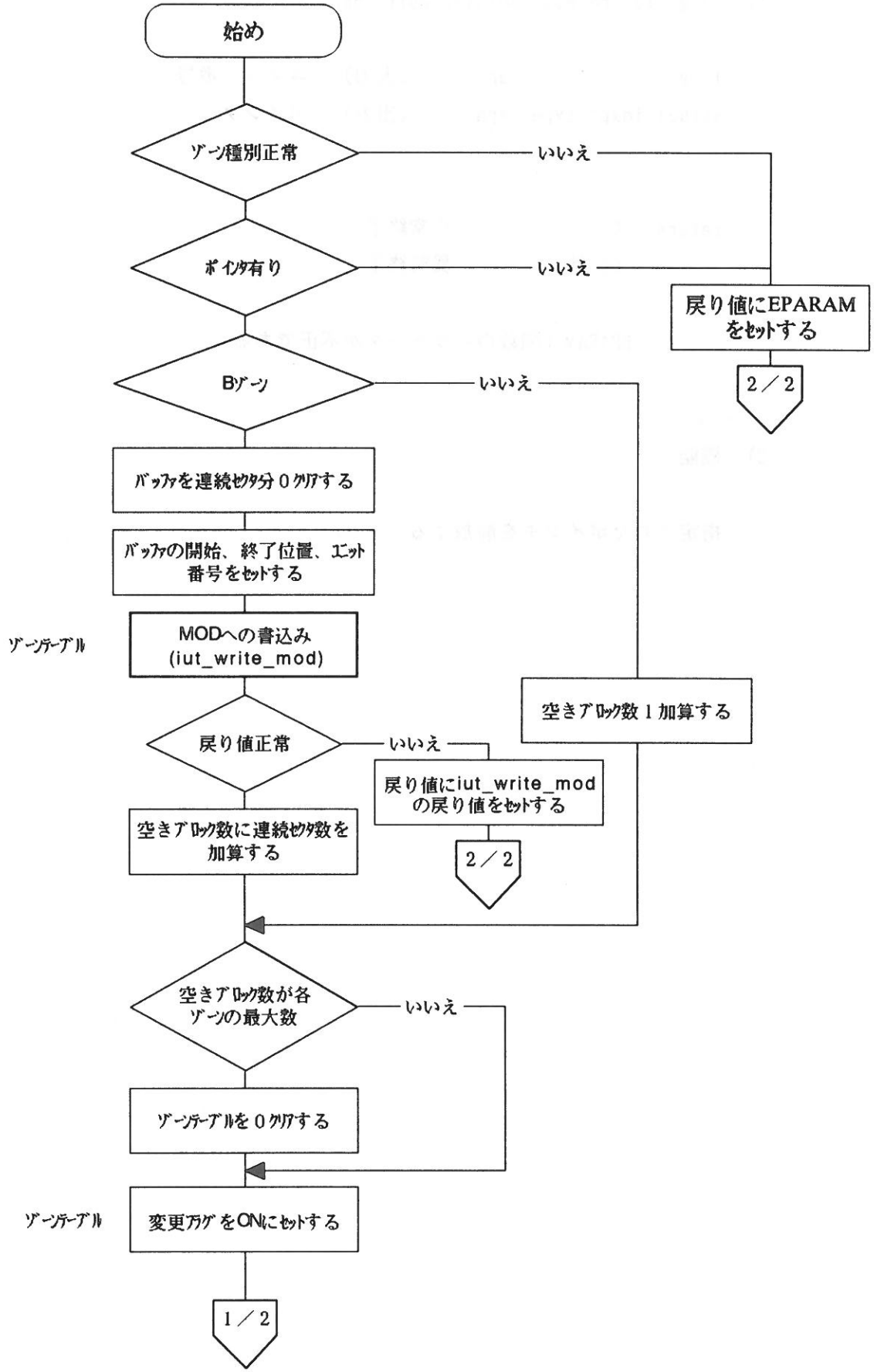
1)　long　iut_release_pointer(unit, pn)

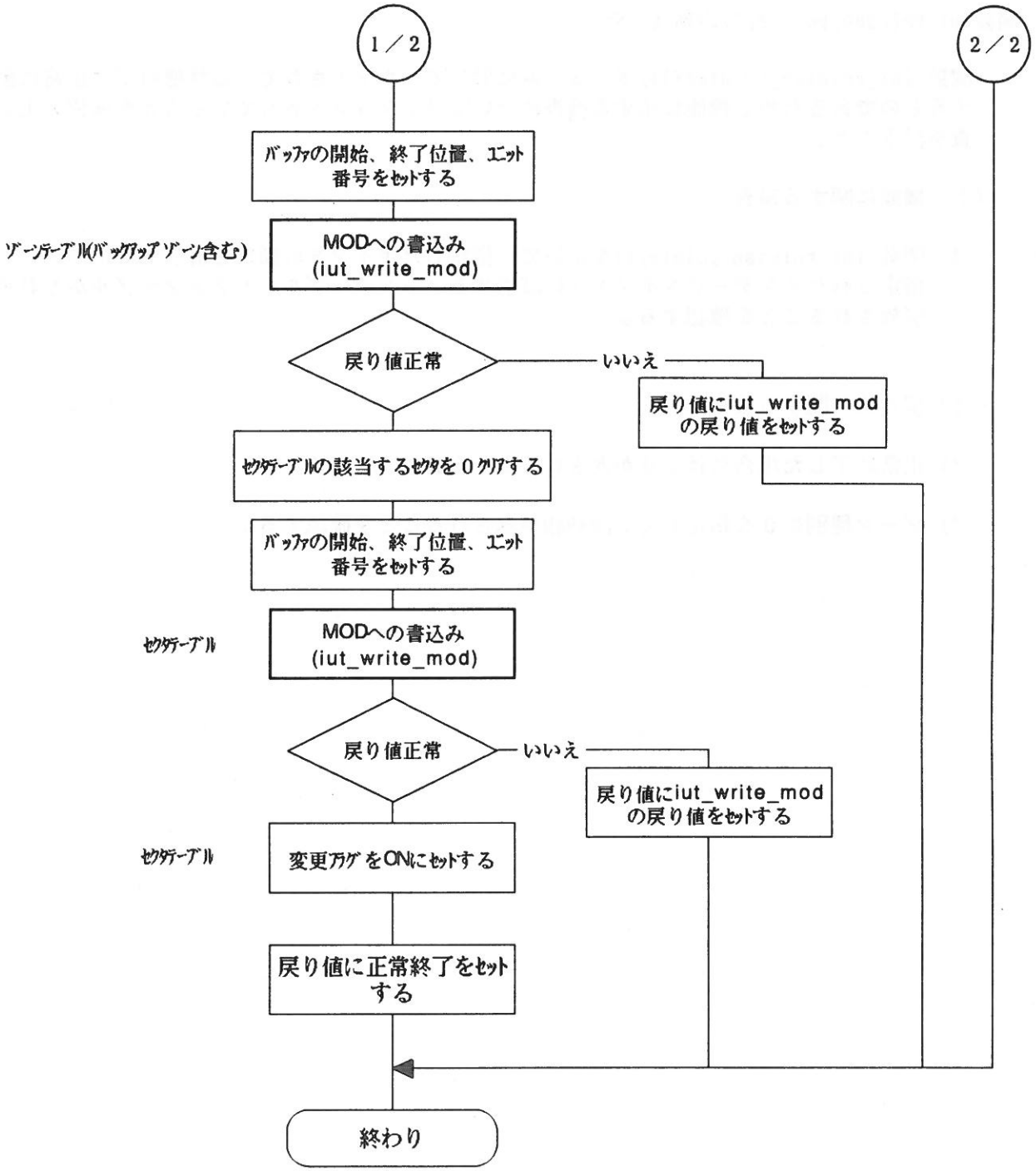
```
long                unit;    （入力）　ユニット番号
struct idxptrtype   *pn;     （出力）　ポインタ


return    0                  正常終了
          0以外              異常終了

          EPARAM（関数のパラメータが不正である）
```

2)　機能

指定されたポインタを解放する。

iut_release_pointer




1／2
2／2
バッファの開始、終了位置、ユニット番号をセットする
ゾーンテーブル(バックアップゾーン含む)
MODへの書込み (iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_mod の戻り値をセットする
セクタテーブルの該当するセクタを０クリアする
バッファの開始、終了位置、ユニット番号をセットする
セクタテーブル
MODへの書込み (iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_mod の戻り値をセットする
セクタテーブル
変更フラグをONにセットする
戻り値に正常終了をセットする
終わり

関数iut_release_pointer()の検査指針

関数 iut_release_pointer()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数 iut_release_pointer()を用いて、指定したポインタの領域を解放する。
指定されたインデックスポインタに該当するゾーンテーブル、セクタテーブルがそれぞれ解放されることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) ゾーン種別に0を指定して、EPARAMが返されることを確認する。

3．11　Bゾーン中にヘッダのポインタを得る

1)　long　iut_get_header_pointer(unit, nsct, pnt)

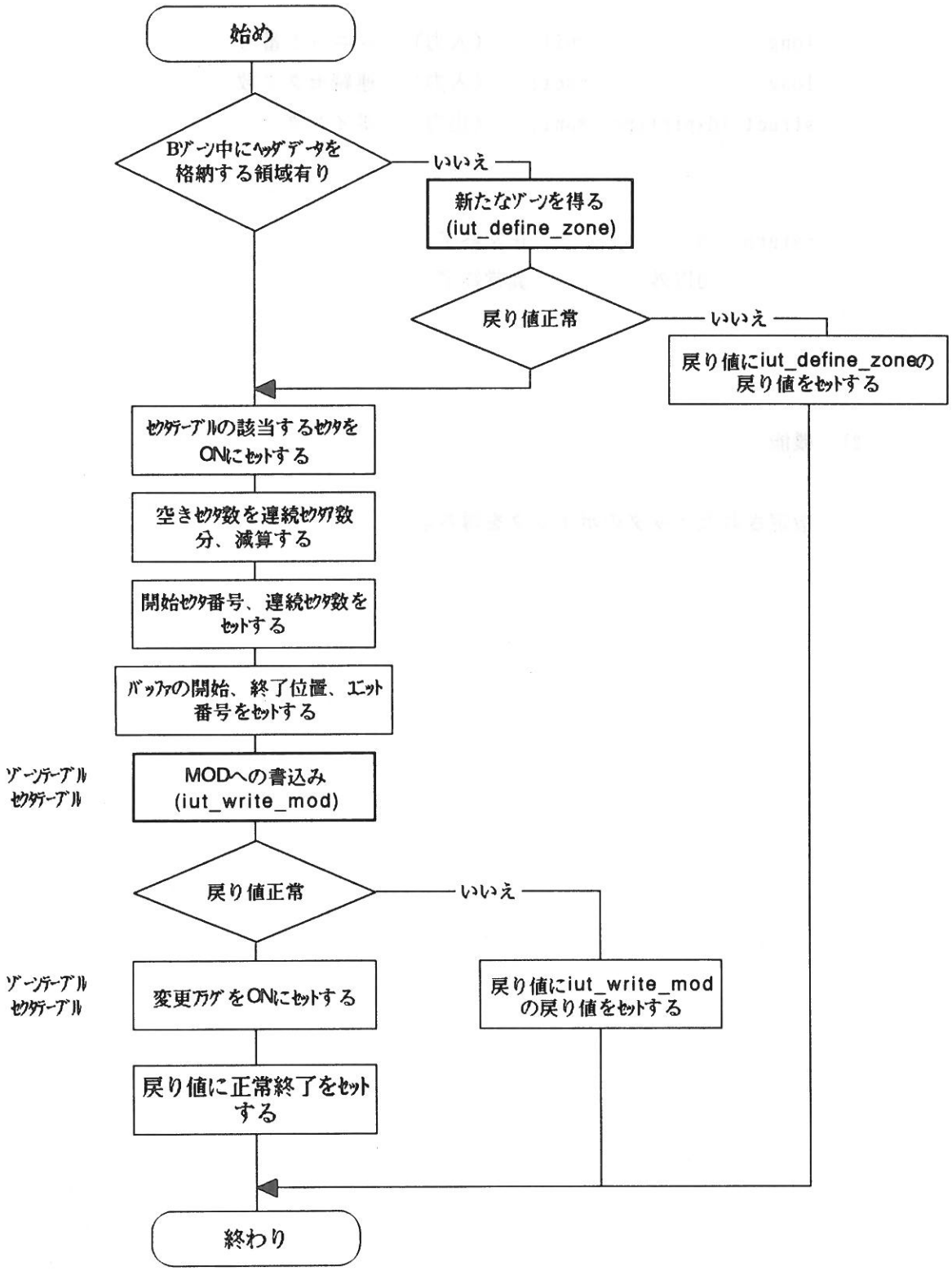
```
long                unit;    (入力)   ユニット番号
long                nsct;    (入力)   連続セクタ数
struct idxptrtype   *pnt;    (出力)   ポインタ
```

```
return   0          正常終了
         0以外      異常終了
```

2)　機能

指定されたヘッダのポインタを得る。

iut_get_header_pointer



関数iut_get_header_pointer()の検査指針

関数iut_get_header_pointer()はボリュームに対してマウントされている状態の下で正常に動作するものであるため、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_get_header_pointer()を用いて、Bゾーン中にヘッダのポインタを得る。
指定された連続セクタ数分の領域が確保され、必要な情報がセットされることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

3．12　ワードデータ列のコピー

1)　long　iut_copy_word(src, dst, n)

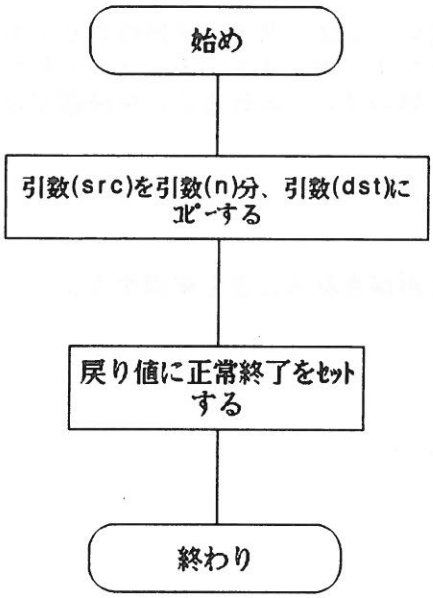
short *src;　　(入力)　ソースデータアレイ
short *dst;　　(出力)　ディスティネーションデータアレイ
long　n;　　(入力)　コピーするワード数

return　0　　正常終了

2)　機能

指定されたソースデータをコピーするワード数分、ディスティネーションデータにコピーする。

iut_copy_word



関数iut_copy_word()の検査指針

（１）機能に関する検査

1) 関数iut_copy_word() を用いて、ワードデータ列のコピーを行う。
指定されたソースデータアレイ、ディスティネーションデータアレイ、コピーするワード数を用いて、ワードデータ列のコピーされることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

4．13　4バイトデータ列のコピー

1)　long　iut_copy_dword(src, dst, n)

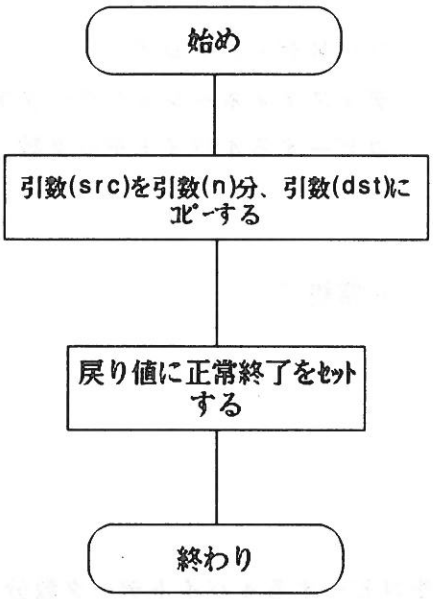
long　*src;	（入力）	ソースデータアレイ
long　*dst;	（出力）	ディスティネーションデータアレイ
long　n;	（入力）	コピーする4バイトデータ数

return　0　　正常終了

2)　機能

指定されたソースデータをコピーする4バイトデータ数分、ディスティネーションデータにコピーする。

iut_copy_dword



関数iut_copy_dword()の検査指針

（１）機能に関する検査

1) 関数iut_copy_dword()を用いて、４バイトデータ列のコピーを行う。
指定されたソースデータアレイ、ディスティネーションデータアレイ、コピーする４バイトデータ数を用いて、４バイトデータ列のコピーされることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。

3．14　バイトスワップを行う

1)　long　iut_byte_swap(arr, len)

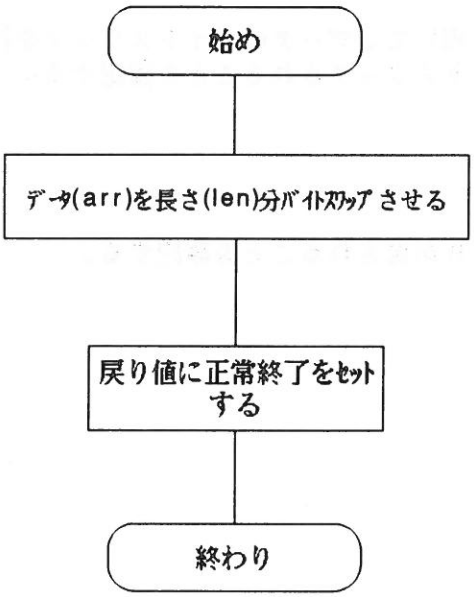
char	*arr;	（入出力）	データアレイ
long	len;	（入力）	スワップする長さ（偶数）

return　0　　正常終了

2)　機能

指定されたデータをスワップする長さ分、バイトスワップを行う。

iut_byte_swap



関数iut_byte_swap()の検査指針

（１）機能に関する検査

1) 関数iut_byte_swap() を用いて、データのバイトスワップを行う。
指定されたデータがバイトスワップされることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。

3．15　ボリューム管理情報の書込み

1)　long　iut_write_volume_info_0(unit)

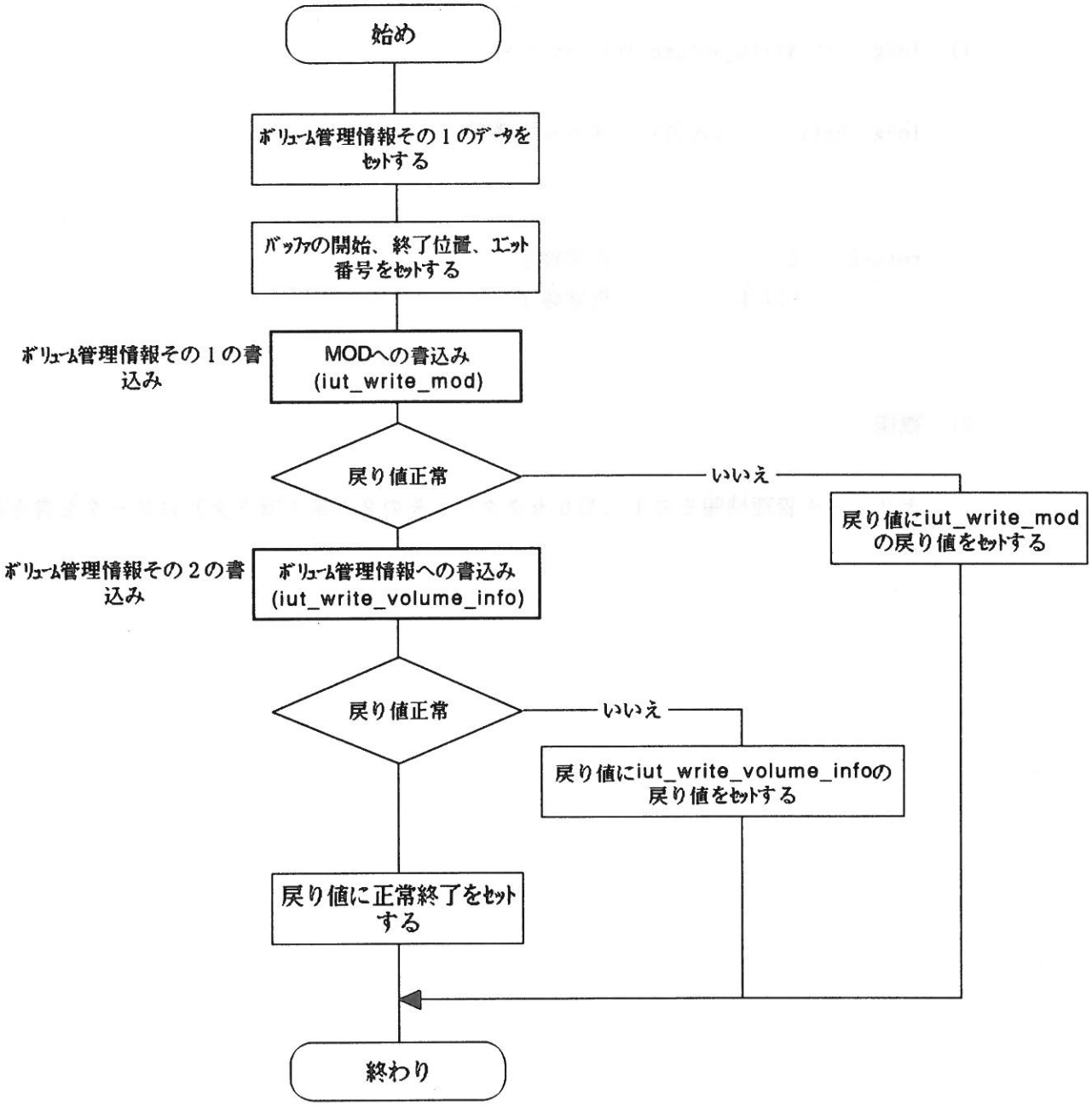
long　unit;　　（入力）　ユニット番号

return　0　　正常終了
　　0以外　　異常終了

2)　機能

ボリューム管理情報その１（第０セクタ）、その２（第１セクタ）にデータを書き込む。

iut_write_volume_info_0



関数iut_write_volume_info_0()の検査指針

（１）機能に関する検査

1) 関数iut_write_volume_info_0() を用いて、ボリューム管理情報その１（NO.0セクタ）、ボリューム管理情報その２（NO.1セクタ）の書き込みを行う。
ボリューム管理情報その１（NO.0セクタ）、ボリューム管理情報その２（NO.1セクタ）がバッファにセットされることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。

3．16　ボリューム管理情報その２（第１セクタ）への書込み

1)　long　iut_write_volume_info(unit)

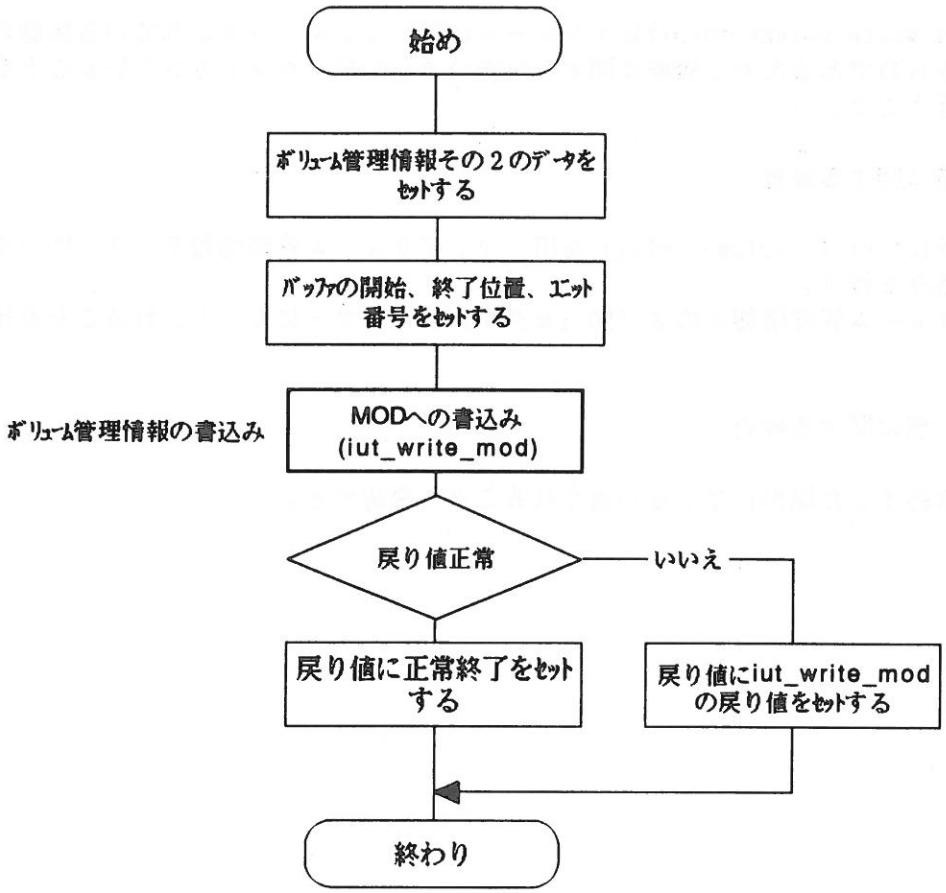
long　unit;　　（入力）　ユニット番号

return　0　　正常終了
　　0以外　　異常終了

2)　機能

ボリューム管理情報その２（第１セクタ）にデータを書き込む。

iut_write_volume_info



関数iut_write_volume_info()の検査指針

関数 iut_write_volume_info()はボリュームに対して、マウントされている状態の下で正常に動作するものであるため、機能に関する検査についてはマウントされていることを確認の上、検査を行うこと。

（1）機能に関する検査

1) 関数iut_write_volume_info() を用いて、ボリューム管理情報その2（NO.1セクタ）の書き込みを行う。
ボリューム管理情報その2（NO.1セクタ）がバッファにセットされることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

3．17　デバイスをオープンする

1)　long　iut_open_devices(iut)

long　iut;　　（入力）　ユニット番号

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）
EMNTED（指定したユニットはマウントされている）

2)　機能

デバイスのオープンを行う。

3．18　デバイスをクローズする

1)　long　iut_close_devices(iut)

long　iut;　　（入力）　ユニット番号

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）
ENMNTD（指定のボリュームはマウントされていない）
EDRIVE（ドライブからのエラーステータス）

2)　機能

デバイスのクローズを行う。

3．19　データを読み込む

1)　long　iut_read_data(iuti, arr, stb, n_sct, istat)

long	iuti;	（入力）	ユニット番号
char	*arr;	（出力）	データアレイ
long	stb;	（入力）	先頭セクタ番号
long	n_sct;	（入力）	大きさ（バイト単位）
long	*istat;	（出力）	ドライブからのエラーステータス

return	0	正常終了
	0以外	異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

ターゲットがイニシエータにデータを送ることを要求する。

3．20　データを書き込む

1)　long　iut_write_data(iuti, arr, stb, n_sct, istat)

long	iuti;	（入力）	ユニット番号
char	*arr;	（入力）	データアレイ
long	stb;	（入力）	先頭セクタ番号
long	n_sct;	（入力）	大きさ（バイト単位）
long	*istat;	（出力）	ドライブからのエラーステータス

return	0	正常終了
	0以外	異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

イニシエータから転送されるデータを、ターゲットが媒体に書き込むことを要求する。

3．21　論理ユニットが動作可能かをチェックする

1)　long　iut_test_unit_ready(iuti, istat)

long　iuti;　　（入力）　ユニット番号
long　*istat;　（出力）　ドライブからのエラーステータス

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

論理ユニットが動作可能であるかをチェックする。

3．22　論理ユニットの状態をセットする

1)　long　iut_rezero_unit(iuti, istat)

long　iuti;　　（入力）　ユニット番号
long　*istat;　（出力）　ドライブからのエラーステータス

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

ターゲットに対して論理ユニットをヘッドの論理アドレスが零の指定状態にセットすることを要求する。

3．23　論理ユニットの容量をセットする

1)　long　iut_read_capacity(iuti, dsize, bsize, istat)

long　iuti;　　(入力)　ユニット番号
long　*dsize;　(出力)　全容量？
long　*bsize;　(出力)　ブロックサイズ？
long　*istat;　(出力)　ドライブからのエラーステータス

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

イニシエータが論理ユニットの容量に関する情報を請求する。

3．24　メディアの排出を許す

1)　long　iut_unlock_media(iuti, istat)

long　iuti;　　（入力）　ユニット番号
long　*istat;　（出力）　ドライブからのエラーステータス

return　0　　正常終了
　　0以外　　異常終了

ENUNIT（存在しないユニット番号を指定した）

2)　機能

メディアの排出を許す。

3．25　メディアの排出を禁止する

1)　long　iut_lock_media(iuti, istat)

```
long  iuti;     (入力)   ユニット番号
long  *istat;   (出力)   ドライブからのエラーステータス
```

```
return    0              正常終了
          0以外          異常終了
```

ENUNIT（存在しないユニット番号を指定した）

2)　機能

メディアの排出を禁止する。

3．26　センスデータをイニシエータへ転送するよう要求する

1)　long　iut_error_sense(iuti, istat)

long　iuti;　　(入力)　ユニット番号
long　*istat;　(出力)　ドライブからのエラーステータス

return　0　　正常終了
　　0以外　　異常終了

EUNRDY（ユニットノットレディ）
EMEDUM（ディスクの物理エラーが発生した）
EDRIVE（ドライブからのエラーステータス）
EPARAM（関数のパラメータが不正である）
EMWPRT（ライトプロテクトメディアに書き込もうとした）

2)　機能

ターゲットにセンスデータをイニシエータへ転送するよう要求する。

3．27　日時を得る

1)　long　iut_get_time(dt)

struct datetype　*dt;　（出力）　日時

return　0　　正常終了
　　0以外　　異常終了

2)　機能

１９××年××月××日　○○時△△分　形式で日時の獲得を行う。

4．サービス関数の設計仕様

各サービス関数単位に、以下の項目について記述する。

・サービス関数の名称
・サービス関数の機能
・入出力データ（コーリングシーケンス）のフォーマット
・エラーの発生条件（｜はＯＲのこと）
　ex. ＥＤＲＩＶＥ｜ドライブからのエラーステータスはＥＤＲＩＶＥとドライブからのエラーステータスのｂｉｔのＯＲ
・境界条件と注意事項の説明
・ＪＩＳフローチャートによるアルゴリズムの記述
・アルゴリズムに対する補足説明
・サービス関数単位の移植における検査指針

４．１　システム変数の初期化

４．１．１　システム変数を初期化する

long　ifm_initial()

機能

　システム変数を初期化する。

引数

　なし

戻り値

　正常終了　　0

備考

・ＩＳ＆Ｃサービス関数を使用する前に、１回だけ呼び出す。

・初期化されるシステム変数を以下に示す。

　ボリューム管理情報フラグ

　セクタテーブルフラグ

　ゾーンテーブルフラグ

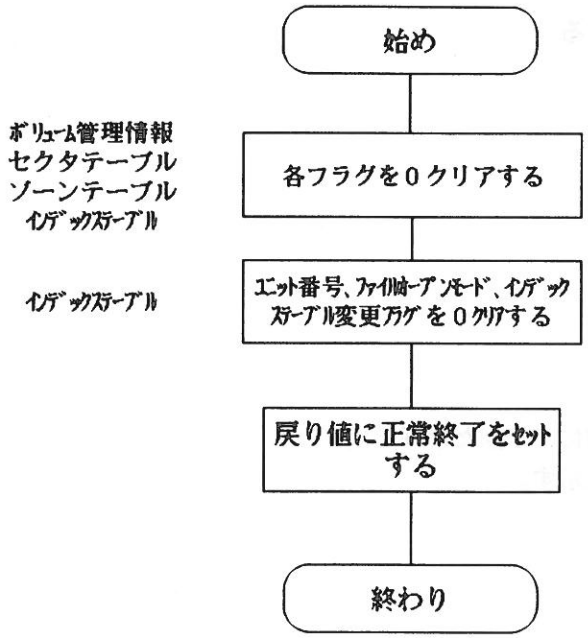
　インデックステーブルフラグ

　インデックステーブルのユニット番号

　インデックステーブルのファイルオープンモード

　インデックステーブルのインデックステーブル変更フラグ

ifm_initial


始め
ボリューム管理情報
セクタテーブル
ゾーンテーブル
インデックステーブル
各フラグを０クリアする
インデックステーブル
ユニット番号、ファイルオープンモード、インデックステーブル変更フラグを０クリアする
戻り値に正常終了をセットする
終わり

関数ifm_initial()の検査指針

（1）機能に関する検査

1）関数ifm_initial() を用いて、システム変数の初期化を行う。
システム変数の値がゼロクリアされていることを確認する。

（2）戻り値に関する検査

1）正常終了した場合には、0が返されることを確認する。

４．２　ボリュームの初期化

４．２．１　ボリュームをフォーマットする

```
long  ifm_format_media(unit, voldata)
long  unit ;
struct volumedatatype  *voldata ;
```

機能

物理フォーマットされたディスクに対し、IS&Cボリュームの管理情報を第０、第１セクタに書き込み、IS&Cフォーマットで初期化する。

引数

unit　（入力）光磁気ディスクメディアのユニット番号。

voldata（入力）第０、第１セクタに書き込むボリューム管理情報が入っている、struct volumedatatype型へのポインタ。

戻り値

正常終了　０

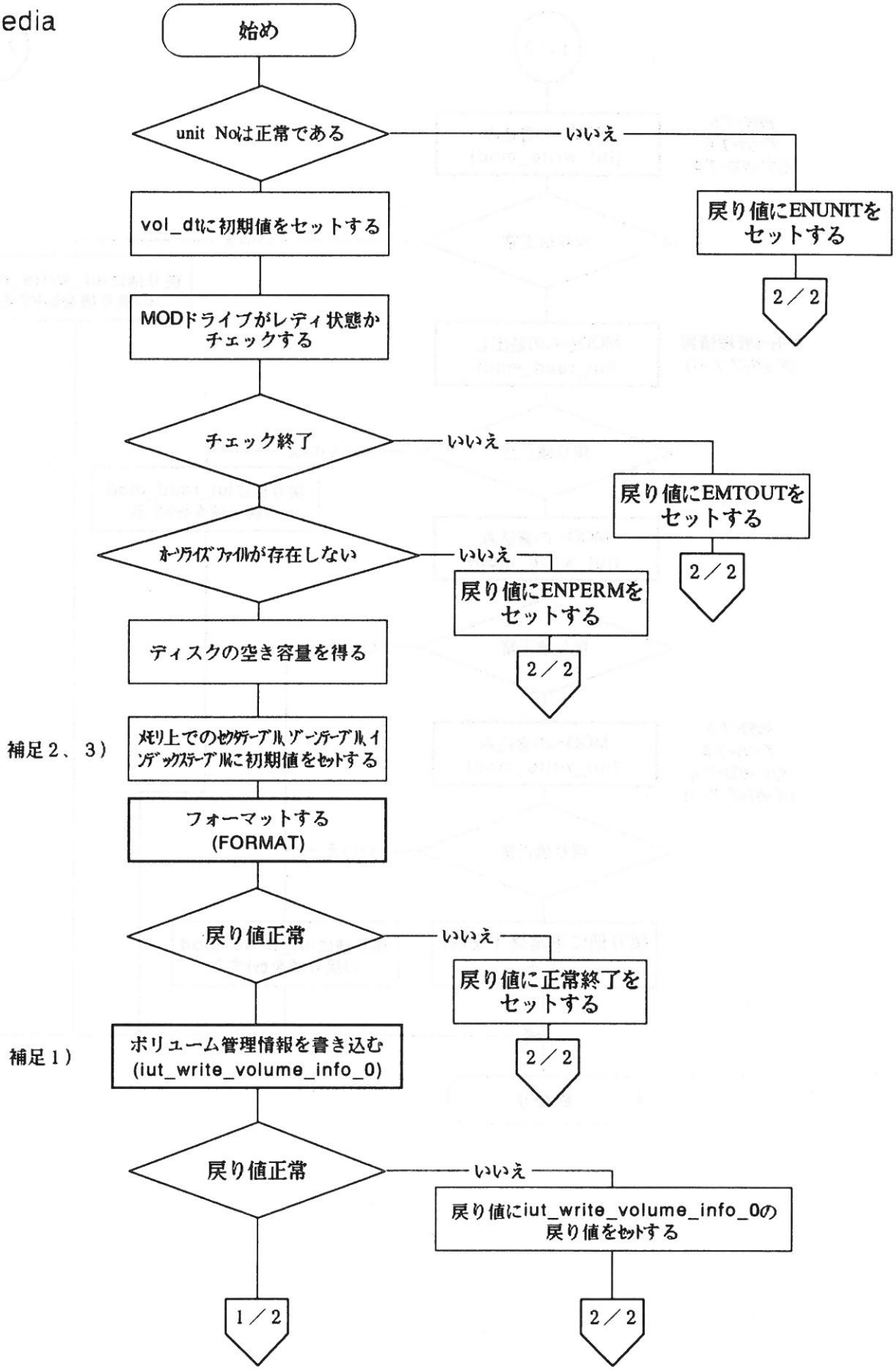
論理エラー　ENUNIT（存在しないユニット番号を指定した）
ENTOUT（タイムアウト）
ENVINF（vol_dtの内容が正しいボリューム管理情報ではない）
ENPERM（禁止されているアクセスを行おうとした）

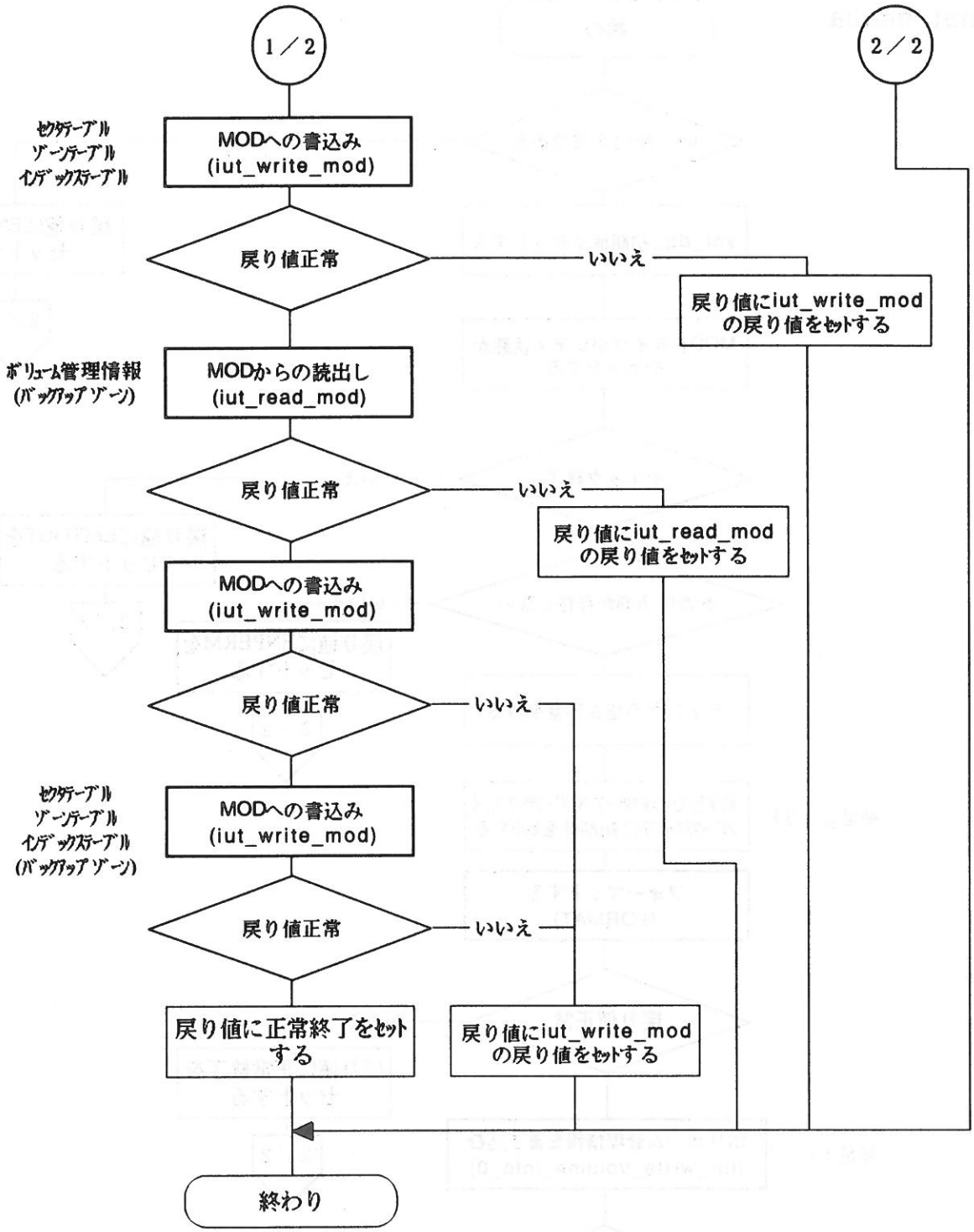
物理エラー　EDRIVE｜ドライブからのエラーステータス

備考

・指定したメディアの内容はすべて消去されるので、十分注意して使用しなければならない。
・IS&C規格書P７０の表１、ユーザ指定の部分は、voldata内でユーザが指定できる。他はこの関数内で表１のように指定される。但し、ゾーン数は３０７になっている。

ifm_format_media




1／2
2／2
セクタテーブル
ゾーンテーブル
インデックステーブル
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_mod
の戻り値をセットする
ボリューム管理情報
(バックアップゾーン)
MODからの読出し
(iut_read_mod)
戻り値正常
いいえ
戻り値にiut_read_mod
の戻り値をセットする
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
セクタテーブル
ゾーンテーブル
インデックステーブル
(バックアップゾーン)
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
戻り値に正常終了をセット
する
戻り値にiut_write_mod
の戻り値をセットする
終わり

関数ifm_format_media()の補足説明

補足１　　ボリューム管理情報部の初期化

１）　初期化識別子が”ＩＳ＆Ｃ”であること。
２）　バージョン番号が”０１．０”であること。
３）　ゾーン内論理セクタ数および論理セクタ容量が１０２４であること。
４）　インデックスサイズが１２８であること。
５）　各テーブル（ボリューム管理情報部、ゾーンテーブル、セクタテーブル）のアドレスの重なりがないこと。
６）　ゾーン数がＭＡＸＺＯＮＥ以内であること。

補足２　　セクタテーブルの初期化

１）　テーブルの初期化
Ａゾーンに対応するセクタテーブルのすべてのビットを１にする。
他の使用可能領域のビットをすべて０にする。

１・・・・１	０００・・・・・・・・・・０００	１・・・・１
Ａゾーン		Ａ’ゾーン

２）　セクタテーブルをファイルに書き込む。（バックアップを含む）

補足３　　インデックステーブルの初期化

１）　テーブルの初期化
インデックスの総セクタ数を求めてすべて０のデータを書き込む。ただし、子のインデックステーブル番号（１２４～１２７バイトアドレス）については、２、３、４～７８４８、－１の継続番号を書き込む。
インデックスの総セクタ数　＝　（インデックス総数＋７）／８の商

２）　インデックステーブルをファイルに書き込む。（バックアップを含む）

関数ifm_format_media()の検査指針

ディスクをドライブにセットし、レディー状態にしてから以下の検査をする。

（１）　機能に関する検査

１）フォーマッティング
関数ifm_format_media()を用いてフォーマッティングを行い、関数から正常復帰することを確認する。異常復帰した場合はハードウェアのトラブルが予想されるので、ドライブからのエラーステータスにより適切な処置を行ってから再度検査する。

２）ボリューム状態の確認

・ゾーンテーブル

項　　目	ｿﾞｰﾝ種別	空きﾌﾞﾛｯｸ数	ﾊﾞｯｸｱｯﾌﾟｿﾞｰﾝ番号
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ1	1	－１	３０７
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ2	0	0	0
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ3	0	0	0
・			
・			
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ307	－１	－１	1

・セクタテーブル
セクタテーブルのＡゾーン（第１ゾーン）とそのバックアップゾーン（第３０７ゾーン）に対応するセクタは全てＯＮ（１：使用済）になり、残りの全てはＯＦＦ（０：未使用）になることを確認する。

・インデックステーブル
各々のインデックスについて、ファイルＩＤ（０ ～ ３バイトアドレス）にｎｕｌｌ（００Ｈ）を埋め、子のインデックステーブル番号（１２４ ～ １２７バイトアドレス）に２、３、４、 ～ 、７８４８、－１の継続番号になることを確認する。

（２）　戻り値に関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニットを指定して、ＥＮＵＮＩＴが返されることを確認する。

３）禁止されているアクセスを行おうとした場合は、ＥＮＰＥＲＭが返されることを確認する。

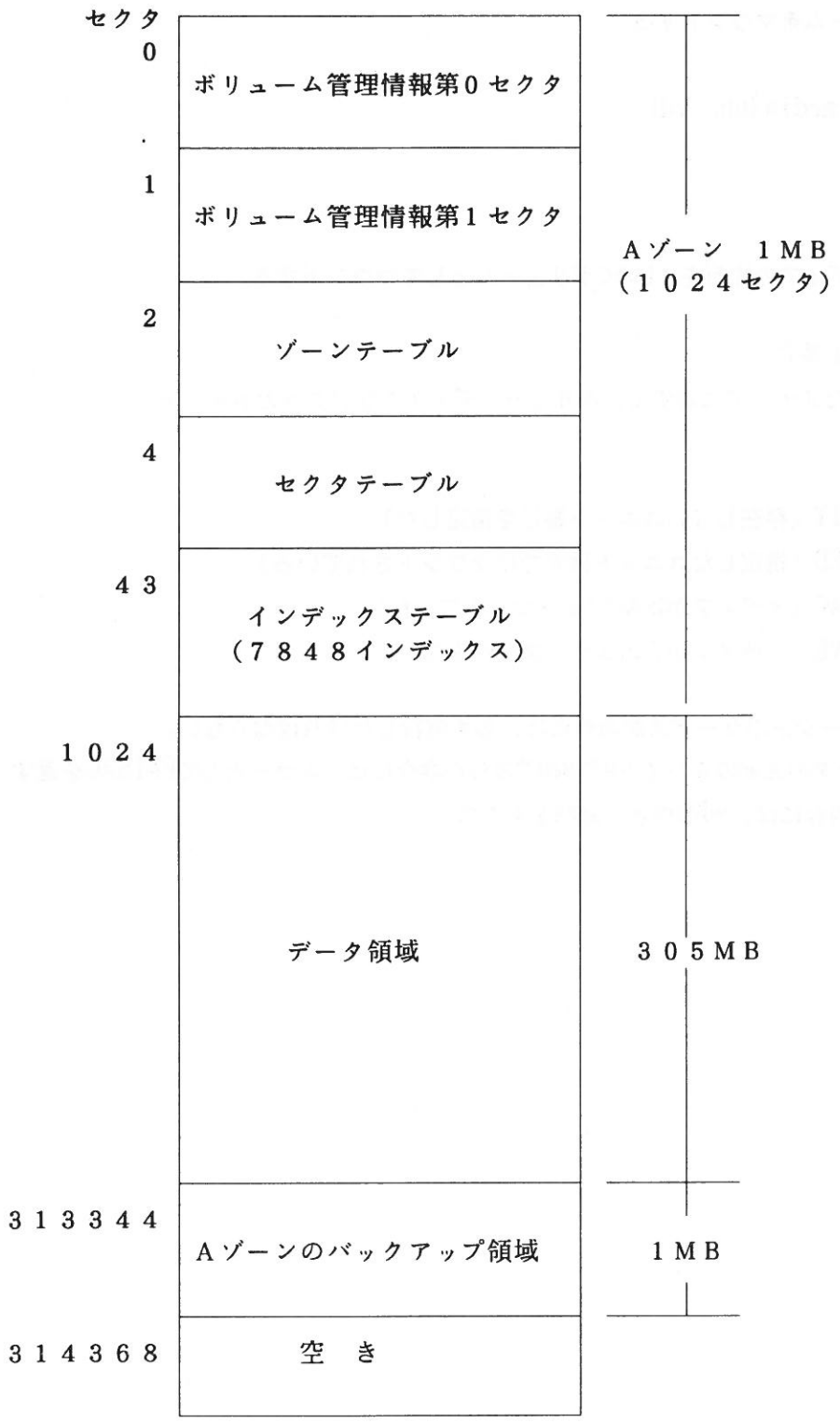
４）ドライブをレディー状態にしない場合には、ＥＴＭＯＵＴが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

ボリューム状態

ボリューム管理情報部	初期値
管理情報（第０セクタ）	
初期化識別子	"IS&C"
バージョン番号	"01.0"
ボリューム名称	任意
ボリュームＩＤ	任意
所有者名	任意
所有者コード	任意
初期化日時	日時
ゾーン数	３０７
ゾーン内論理セクタ	１０２４
論理セクタ容量	１０２４
ｿﾞｰﾝﾃﾌﾞﾙ開始ｾｸﾀ番号	2
ｾｸﾀﾃｰﾌﾞﾙ開始ｾｸﾀ番号	4
ｲﾝﾃﾞｯｸｽ開始ｾｸﾀ番号	４３
インデックスサイズ	１２８
予約領域	0
管理情報（第１セクタ）	
インデックス総数	７８４８
登録ファイル数	0
仮削除ファイル数	0
空きインデックス数	７８４８
システムファイル数	0
ディレクトリファイル数	0
更新日時	日時
空きインデックス開始番号	1
ボリューム使用中フラグ	0
予約領域	0
システム予約領域	0

ボリューム構成


セクタ
0
ボリューム管理情報第0セクタ
1
ボリューム管理情報第1セクタ
2
ゾーンテーブル
4
セクタテーブル
43
インデックステーブル
（7848インデックス）
1024
データ領域
313344
Aゾーンのバックアップ領域
314368
空　き
Aゾーン　1MB
（1024セクタ）
305MB
1MB

４．３　ボリュームの管理

４．３．１　ボリュームをマウントする

```
long   ifm_mount_media(un,  vd)
long   un ;
long   *vd ;
```

機能

光磁気ディスクメディアの片面を、IS&Cボリュームとしてマウントする。

引数

un（入力）ユニット番号。

vd（出力）指定したメディアに対する、ボリュームディスクリプタへのポインタ。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）

EMNTED（指定したユニットはすでにマウントされている）

ENISAC（メディアがISACフォーマットではない）

物理エラー　EDRIVE｜ドライブからのエラーステータス

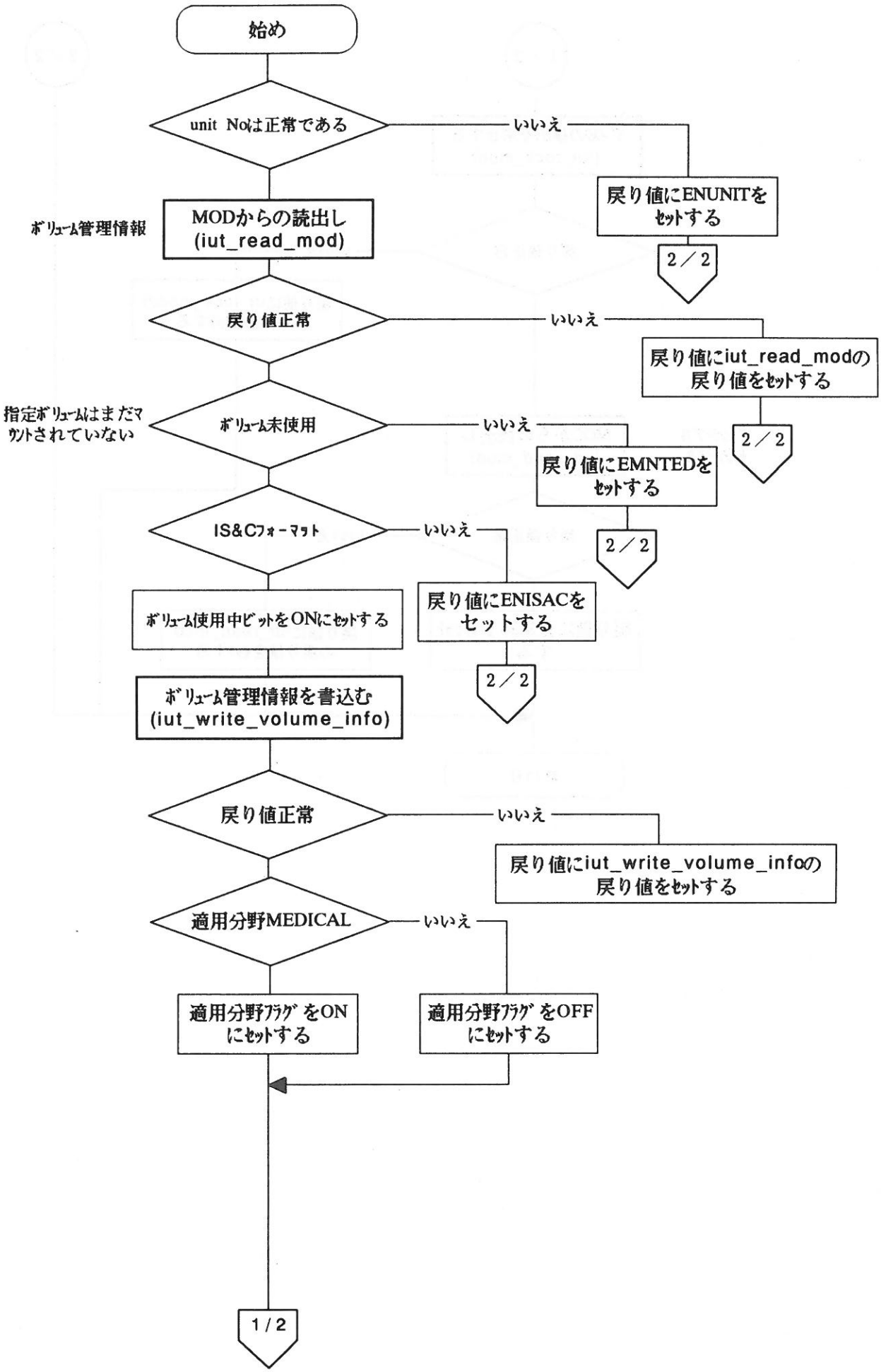
備考

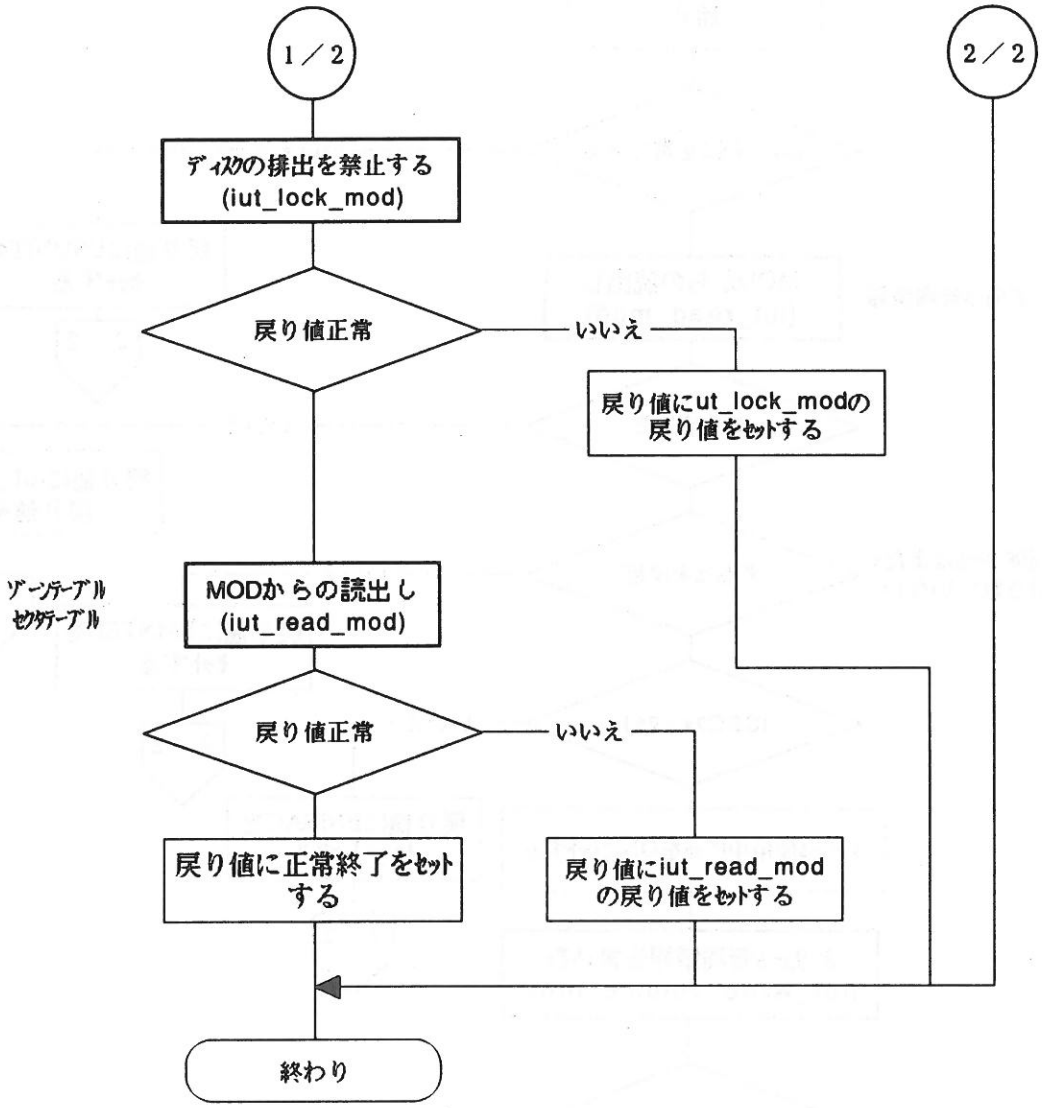
・IS&Cファイルマネージャのサービス開始時には、必ず実行しなければならない。

・メディアの第０セクタの先頭の４バイトが"IS&C"以外の場合には、エラーとしてENISACを返す。

・エラーが発生した場合には、vdの内容は意味をもたない。

ifm_mount_media




1／2
2／2
ディスクの排出を禁止する
(iut_lock_mod)
戻り値正常
いいえ
戻り値にut_lock_modの
戻り値をセットする
ゾーンテーブル
セクタテーブル
MODからの読出し
(iut_read_mod)
戻り値正常
いいえ
戻り値に正常終了をセット
する
戻り値にiut_read_mod
の戻り値をセットする
終わり

関数ifm_mount_media()の検査指針

関数 ifm_format_media()を用いてフォーマッティングしたメディアをユニットにセットし、レディー状態にしてから以下の検査をする。

（1）機能に関する検査

1）指定したユニットのボリューム情報（システム領域のボリューム管理情報、ゾーンテーブル、セクタテーブル）がメモリ上のシステムテーブルに読み込まれることを確認する。

（2）戻りに関する検査

1）正常終了した場合には、0が返されることを確認する。

2）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

3）すでにマウントされているユニットを指定して、ＥＭＮＴＥＤが返されることを確認する。

4）ＩＳ＆Ｃフォーマットでないメディアをセットし、ＥＮＩＳＡＣが返されることを確認する。

・フォーマッティングされていないメディア
・ＩＳ＆Ｃフォーマット以外のメディア

5）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

4．3．2　ボリュームをディスマウントする

```
long   ifm_dismount_media(vd)
long   vd ;
```

機能

IS&Cボリュームをディスマウントする。

引数

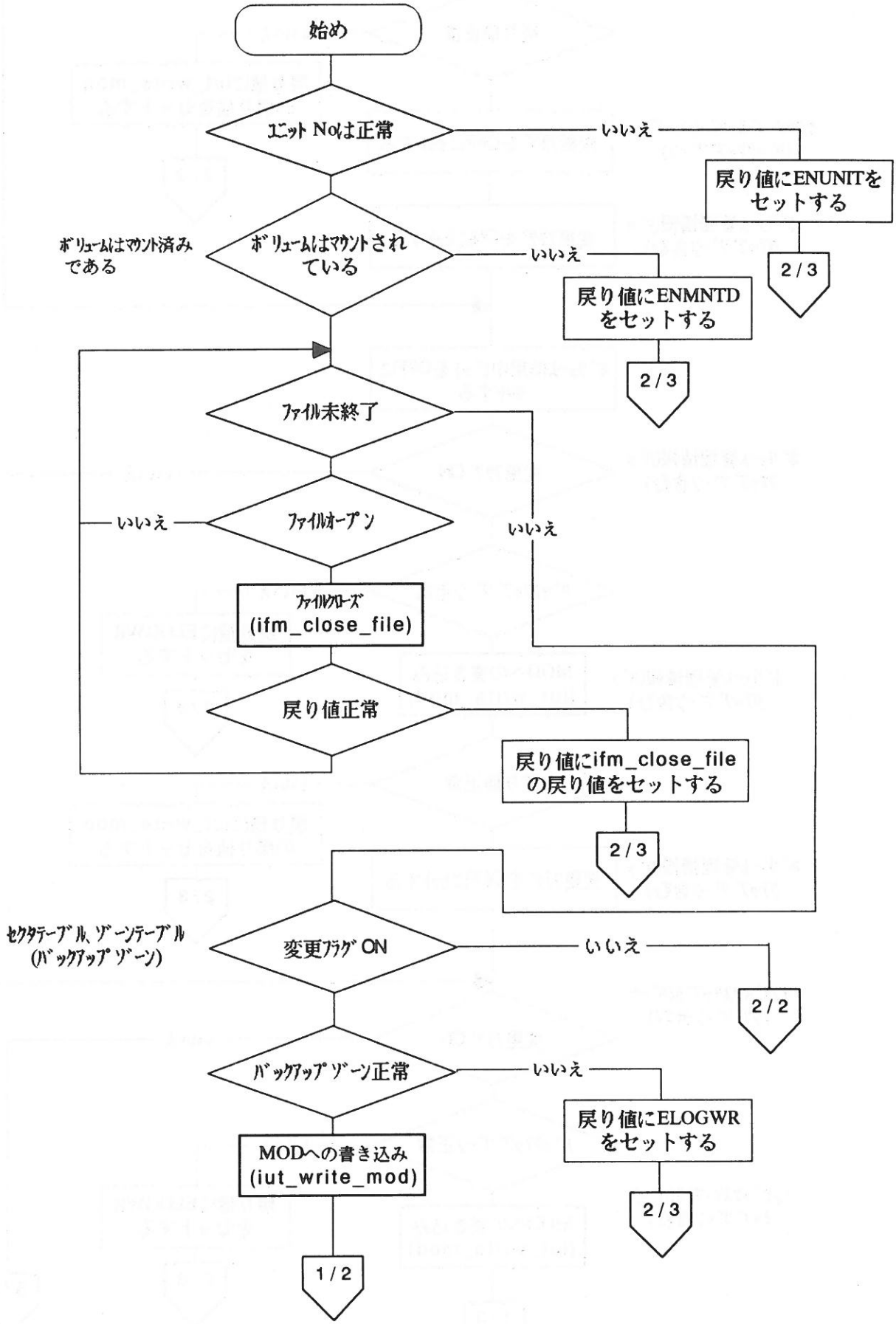
vd（入力）ボリュームディスクリプタ。

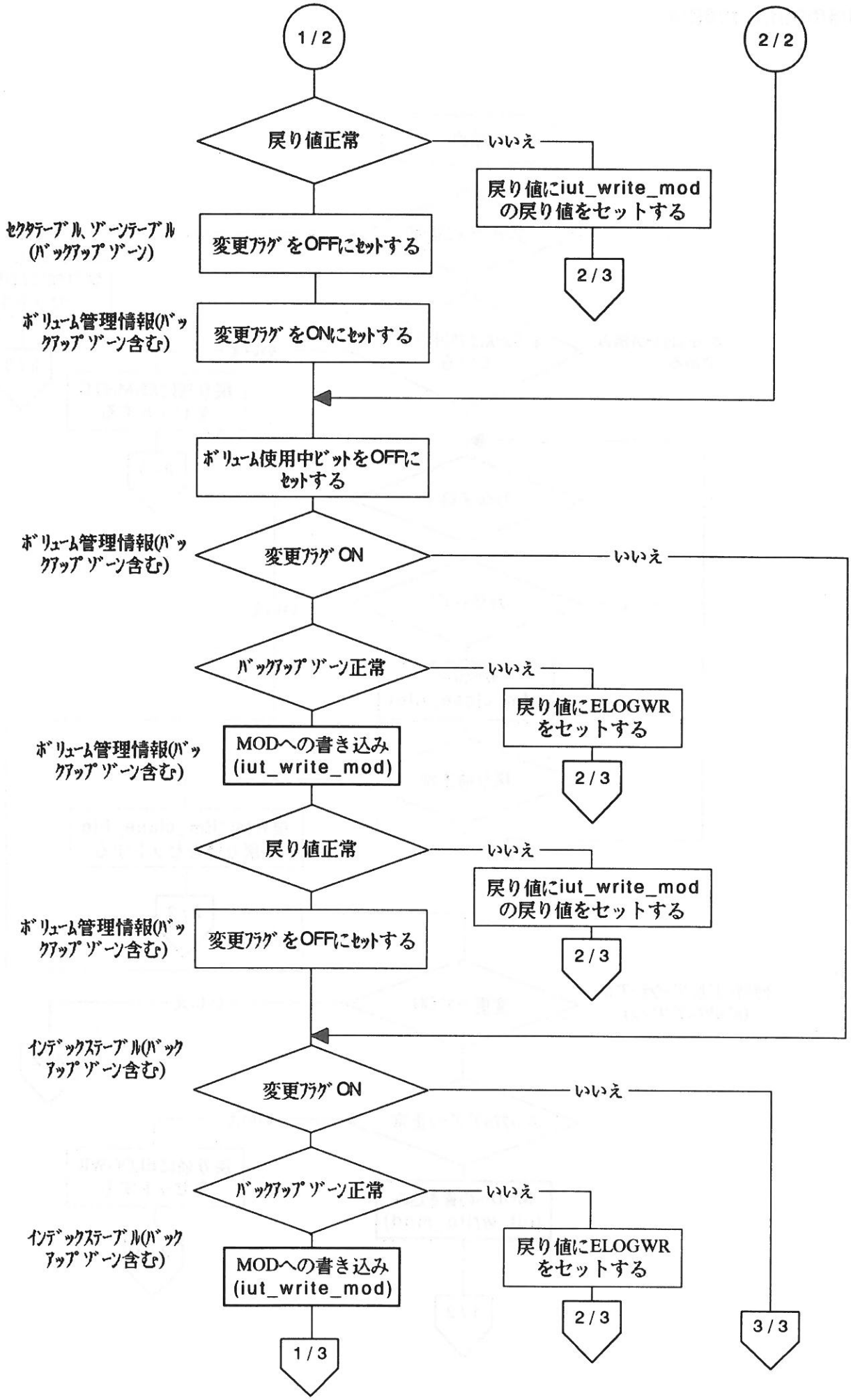
戻り値

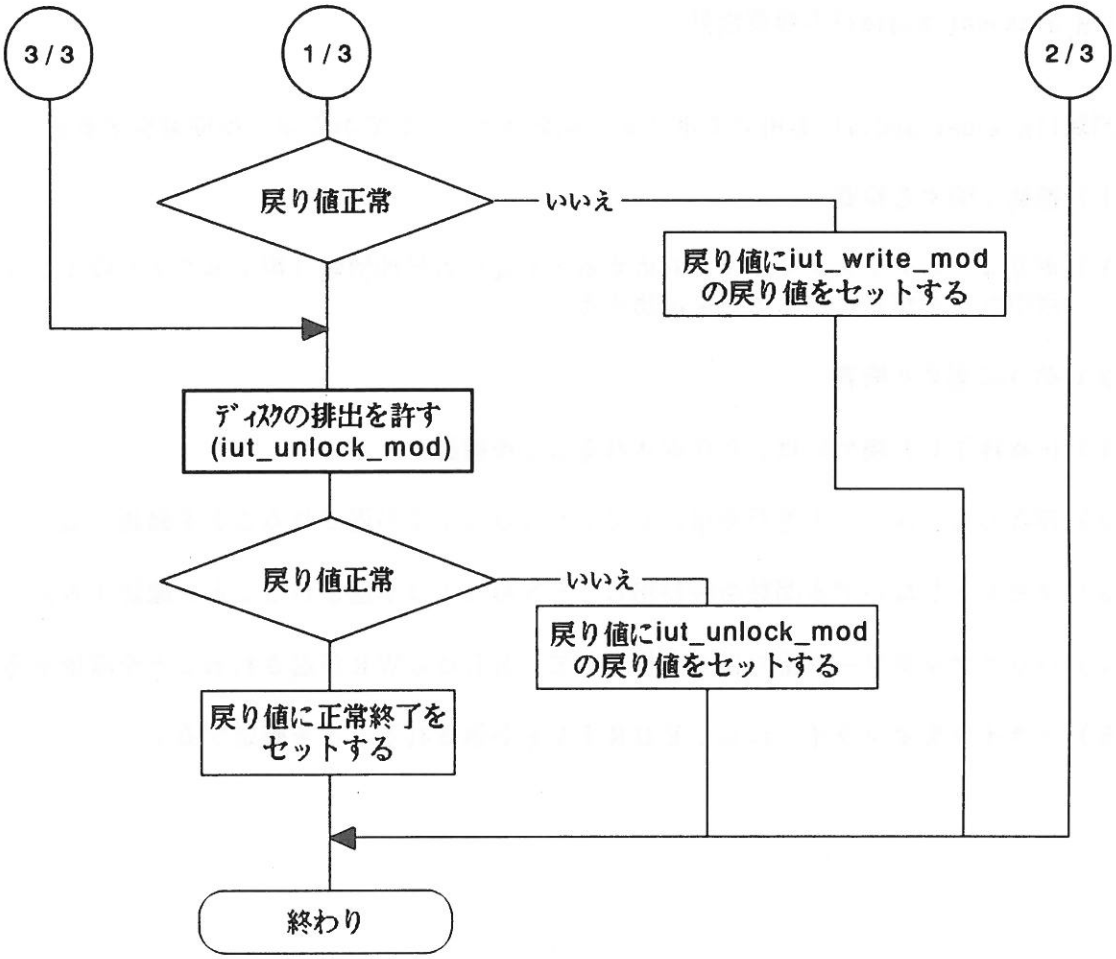
正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ELOGWR（書込み時に論理エラーが発生した）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・IS&Cファイルマネージャのサービス終了時には、必ず実行しなければならない。
・指定したボリューム内でオープンされているファイルは、すべてクローズされる。


始め
ユニット Noは正常
いいえ
戻り値にENUNITを
セットする
2/3
ボリュームはマウント済み
である
ボリュームはマウントされ
ている
いいえ
戻り値にENMNTD
をセットする
2/3
ファイル未終了
いいえ
ファイルオープン
いいえ
ファイルクローズ
(ifm_close_file)
戻り値正常
戻り値にifm_close_file
の戻り値をセットする
2/3
セクタテーブル、ゾーンテーブル
(バックアップゾーン)
変更フラグ ON
いいえ
2/2
バックアップゾーン正常
いいえ
戻り値にELOGWR
をセットする
2/3
MODへの書き込み
(iut_write_mod)
1/2


1/2
2/2
戻り値正常
いいえ
戻り値にiut_write_mod
の戻り値をセットする
2/3
セクタテーブル、ゾーンテーブル
(バックアップゾーン)
変更フラグをOFFにセットする
ボリューム管理情報(バック
アップゾーン含む)
変更フラグをONにセットする
ボリューム使用中ビットをOFFに
セットする
ボリューム管理情報(バック
アップゾーン含む)
変更フラグ ON
いいえ
バックアップゾーン正常
いいえ
戻り値にELOGWR
をセットする
2/3
ボリューム管理情報(バック
アップゾーン含む)
MODへの書き込み
(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_mod
の戻り値をセットする
2/3
ボリューム管理情報(バック
アップゾーン含む)
変更フラグをOFFにセットする
インデックステーブル(バック
アップゾーン含む)
変更フラグ ON
いいえ
バックアップゾーン正常
いいえ
インデックステーブル(バック
アップゾーン含む)
MODへの書き込み
(iut_write_mod)
戻り値にELOGWR
をセットする
2/3
1/3
3/3


3/3
1/3
2/3
戻り値正常
いいえ
戻り値にiut_write_mod
の戻り値をセットする
ﾃﾞｨｽｸの排出を許す
(iut_unlock_mod)
戻り値正常
いいえ
戻り値にiut_unlock_mod
の戻り値をセットする
戻り値に正常終了を
セットする
終わり

関数ifm_dismount_media()の検査指針

関数ifm_mount_media()を用いてボリュームをマウントしてから以下の検査をする。

（１）機能に関する検査

１）ボリュームディスクリプタに対応するボリューム管理情報（第１セクタ）のボリューム使用中フラグが０になることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

３）マウントしないで本関数を呼び出し、ＥＮＭＮＴＤが返されることを確認する。

４）バックアップゾーン番号に０を指定して、ＥＬＯＧＷＲが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．３．３　ボリュームの使用量を得る

```
long  ifm_get_space(vd, msize, usize)
long  vd ;
long  *msize ;
long  *usize ;
```

機能

ボリュームの総容量と、使用量を得る。

引数

vd　（入力）ボリュームディスクリプタ。

msize（出力）ボリュームの総容量をキロバイト単位で表した値へのポインタ。

usize（出力）ボリュームの使用量をキロバイト単位で表した値へのポインタ。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）

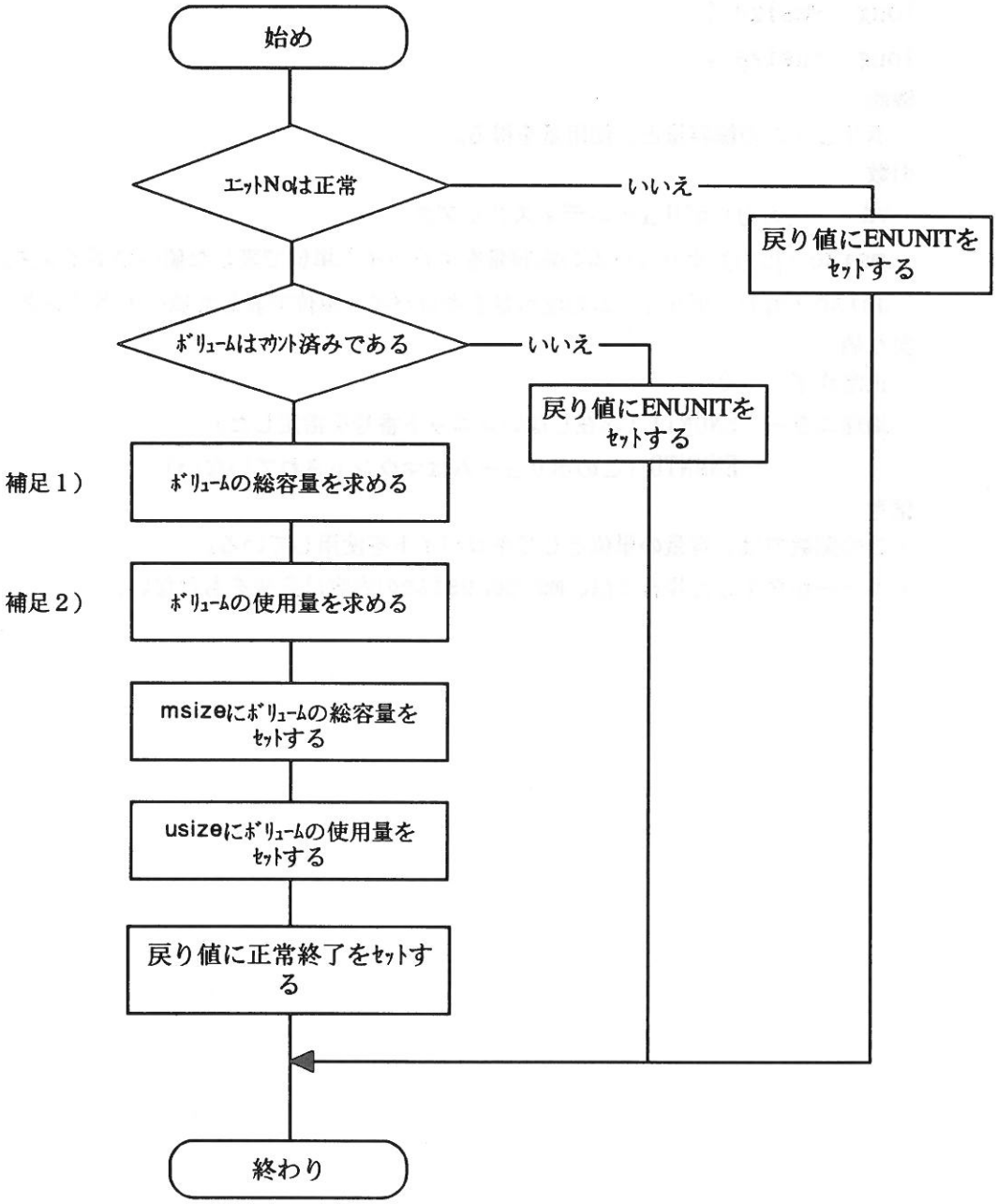
ENMNTD（このボリュームはマウントされていない）

備考

・この関数では、容量の単位としてキロバイトを使用している。

・エラーが発生した場合には、msize, usizeの内容は意味をもたない。

ifm_get_space



関数ifm_get_space()の補足説明

補足１）ボリュームの総容量

ゾーン数×ゾーン内論理セクタ×論理セクタ容量÷１０２４

補足２）ボリュームの使用量を求めるアルゴリズム

```
for  ボリューム管理情報第0セクタのゾーン数内

     ゾーンテーブルのゾーン種別の絶対値を求める

     if  未定義ゾーン？
         continue

     if  Aゾーン？
         usize  =  usize  +  1024

     else
         usize  =  usize  +
             （1024  −  空きブロック数  ×  各ゾーンのブロックサイズ）

end  for
```

関数ifm_get_space()の検査指針

関数ifm_mount_media()を用いてボリュームをマウントしてから以下の検査をする。

（１）機能に関する検査

１）ボリュームディスクリプタで指定したボリュームの総容量がｍｓｉｚｅに返されることを確認する。

３１４３６８（３０７ＭＢ＊１０２４セクタ）

２）ボリュームディスクリプタで指定したボリュームの使用量がｕｓｉｚｅに返されることを確認する。

・フォーマッティング直後、３１４３６８
・フォーマッティングしてから１００ＫＢのデータを書き込んだ直後、３１４２６８

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

３）マウントしないで本関数を呼び出し、ＥＮＭＮＴＤが返されることを確認する。

４．４　ファイルの管理

４．４．１　インデックスのファイルの管理情報を得る

```
long  ifm_get_list(vd, id, array, req_size, act_size)
long  vd ;
long  id ;
char  *array ;
long  req_size ;
long  *act_size ;
```

機能

指定したファイルＩＤ以上のＩＤをもつファイルの管理情報を順番に得る。

引数

vd　　（入力）ボリュームディスクリプタ。

id　　（入力）管理情報を得るファイルＩＤの開始値。

array　（出力）ファイルの管理情報を格納する、各size of(struct listidxtype)バイトのstruct listidxtypeストラクチャの配列へのポインタ（データポインターを除く６４バイトのもの）。

req_size（入力）arrayの配列の個数。最大req_size個のファイルの管理情報が得られる。

act_size（出力）実際に得られた管理情報の個数。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）

ENMNTD（このボリュームはマウントされていない）

物理エラー　EDRIVE｜ドライブからのエラーステータス

備考

・指定したファイルＩＤに対応するファイルが存在しない場合には、つぎに大きいＩＤをもつファイルから順番に管理情報が得られる。なお、ファイルＩＤの最小値は１なので、ボリュームの先頭から管理情報を得る場合には１を指定する。

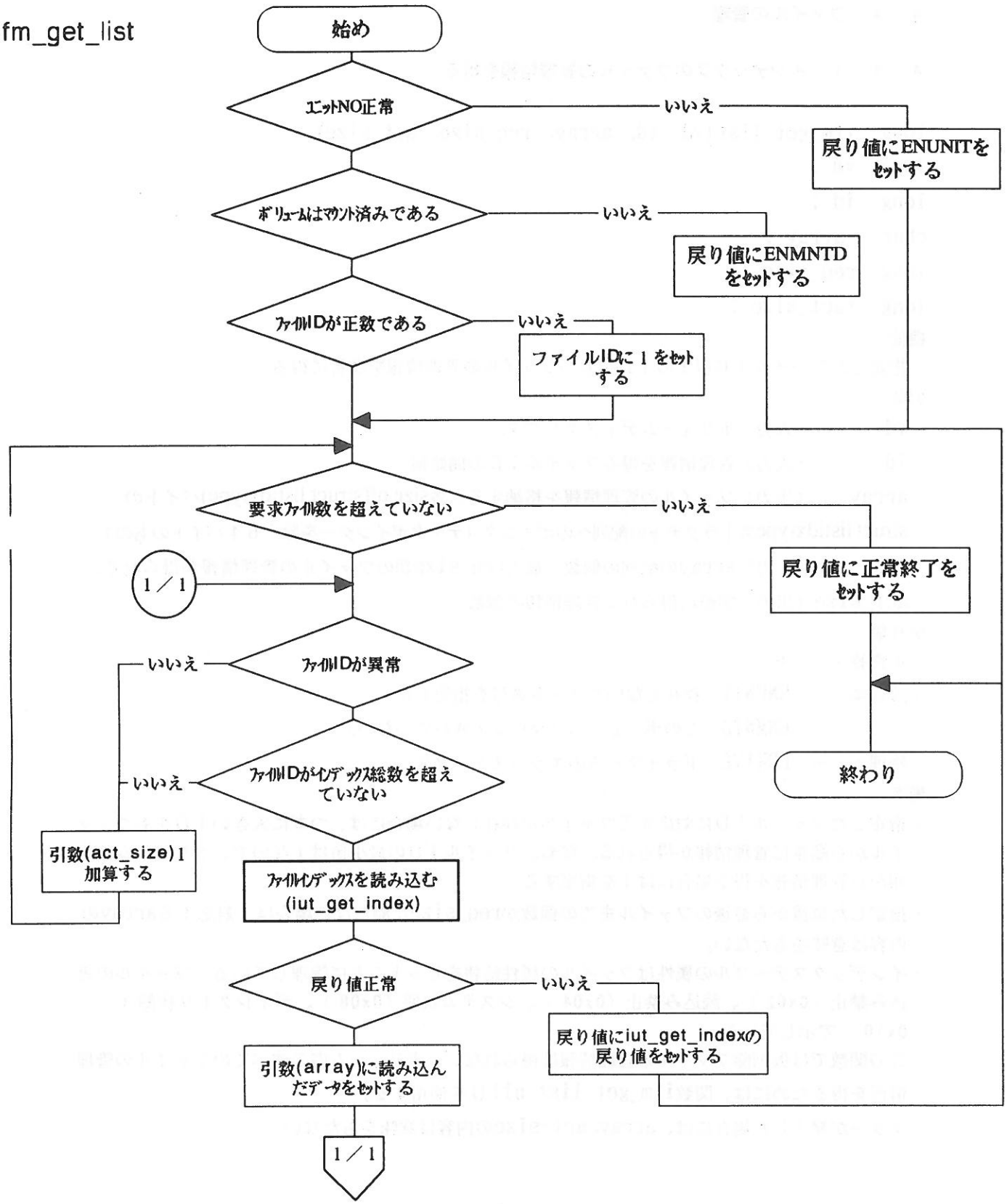
・指定した位置から最後のファイルまでの個数がreq_sizeに満たない場合は、対応するarrayの内容は意味をもたない。

・インデックステーブルの属性はファイルの属性情報をビットごとに管理している。ファイルの書込み禁止（0x02）、読込み禁止（0x04）、システム状態（0x08）、ディレクトリ状態（0x10）で示している。

・この関数では仮削除ファイルの管理情報は得られない。ボリューム内のすべてのファイルの管理情報を得るためには、関数ifm_get_list_all()を使用する。

・エラーが発生した場合には、array, act-sizeの内容は意味をもたない

ifm_get_list


始め
ユニットNO正常
いいえ
戻り値にENUNITをセットする
ボリュームはマウント済みである
いいえ
戻り値にENMNTDをセットする
ファイルIDが正数である
いいえ
ファイルIDに 1 をセットする
要求ファイル数を超えていない
いいえ
戻り値に正常終了をセットする
1／1
ファイルIDが異常
いいえ
ファイルIDがインデックス総数を超えていない
いいえ
終わり
引数(act_size)1加算する
ファイルインデックスを読み込む(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をセットする
引数(array)に読み込んだデータをセットする
1／1

関数ifm_get_list()の検査指針

関数ifm_format_media(),ifm_mount_media() を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（1）機能に関する検査

1）関数 ifm_create_file()で１ＫＢのファイルを１０個作成してから、ｉｄとｒｅｑ＿ｓｉｚｅを以下の3通り変えたときのａｃｔ＿ｓｉｚｅとａｒｒａｙの内容を確認する。

ｉｄ	1	5	9
ｒｅｑ＿ｓｉｚｅ	２０	2	3
ａｃｔ＿ｓｉｚｅ	１０	2	2
ａｒｒａｙ　ファイルＩＤ	1	5	9
ファイルＩＤ	2	6	１０
ファイルＩＤ	3		
ファイルＩＤ	4		
ファイルＩＤ	5		
ファイルＩＤ	6		
ファイルＩＤ	7		
ファイルＩＤ	8		
ファイルＩＤ	9		
ファイルＩＤ	１０		

また、各インデックステーブルの属性にはファイルの属性として０ｘ００がセットされることを確認する。

２）次にファイルの属性を以下のように変えてから１）と同様の確認をする。

・ｉｄが１のファイルを関数ifm_write_protect_on()を用いて書き込み禁止にする。
・ｉｄが５のファイルを関数ifm_read_potect_on()を用いて読み込み禁止にする。
・ｉｄが６のファイルを関数ifm_system_flag_on()を用いてシステムファイルにする。
・ｉｄが９のファイルを関数ifm_delete_file()を用いて仮削除する。

ｉｄ	１	５	９
ｒｅｑ＿ｓｉｚｅ	２０	２	３
ａｃｔ＿ｓｉｚｅ	９	２	１
ａｒｒａｙ　ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ ファイルＩＤ	１ ２ ３ ４ ５ ６ ７ ８ １０	５ ６	１０

また、各インデックステーブルの属性には０ｘ００、０ｘ０４、０ｘ０８、０ｘ８０がセットされることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

３）マウントしないで本関数を呼び出し、ＥＮＭＮＴＤが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．４．２　サイズ固定のファイルを作成する

```
long  ifm_create_file(vd, name, size, flag_z, id)
long  vd ;
char  *name ;
long  size ;
long  flag_z ;
long  *id ;
```

機能

データサイズを指定して、新規にファイルを作成する。

引数

vd　　（入力）ボリュームディスクリプタ。

name　（入力）ファイル名が入っている文字列へのポインタ。

size　（入力）データサイズをバイト単位で表した値。

flag_z（入力）使用できるゾーンを指定するためのフラグ（デフォルトは-1）。下位１バイトが規格書P39の図の様になる。－１は全てのゾーンが使用され得る。

id　　（出力）作成されたファイルのファイルＩＤへのポインタ。

戻り値

正常終了　0

論理エラー　EPARAM（関数のパラメタが不正である）
　　ENMNTD（このボリュームはマウントされていない）
　　ENVINF（vol_dtの内容が正しいボリューム管理情報ではない）
　　ENUNIT（存在しないユニット番号を指定した）

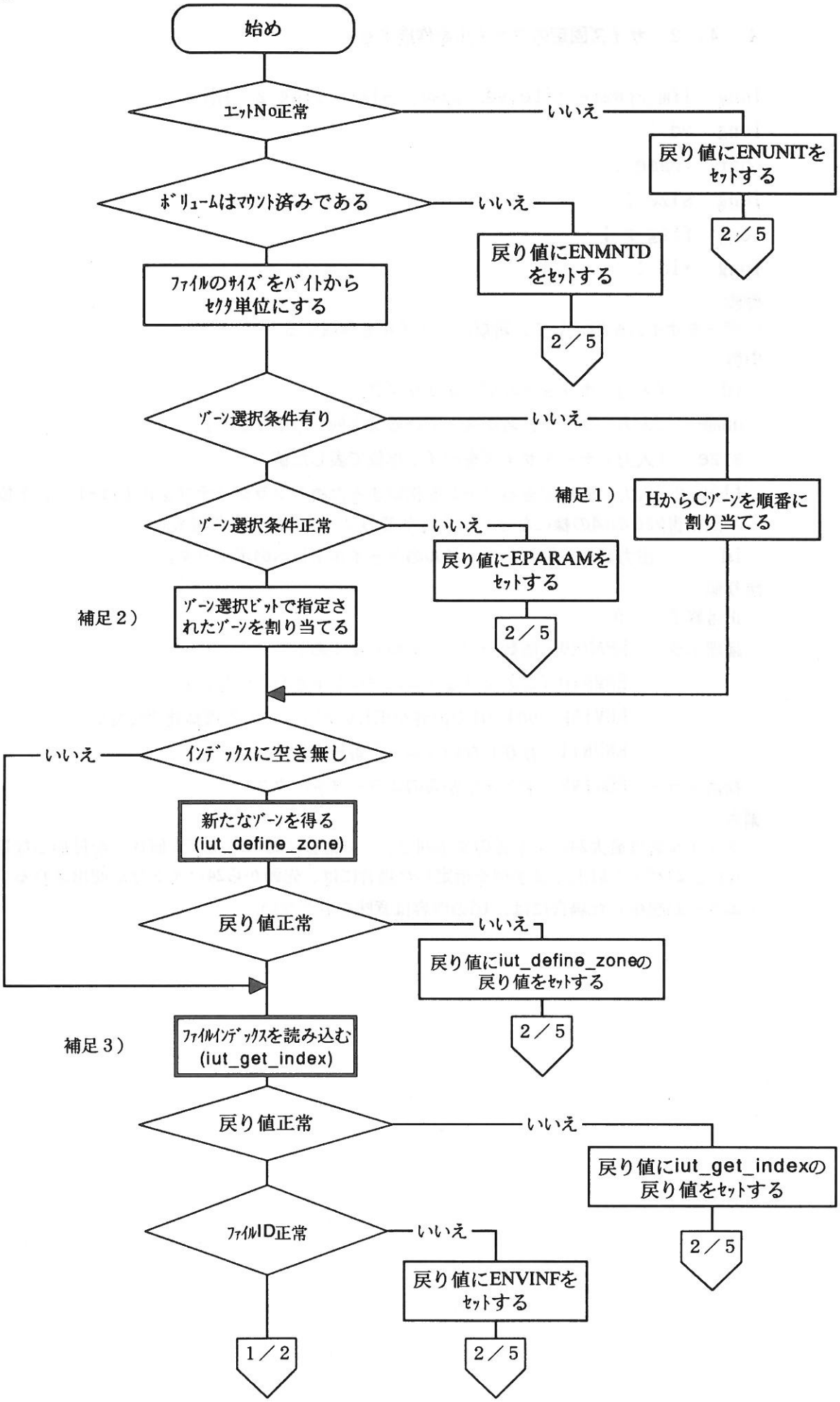
物理エラー　EDRIVE｜ドライブからのエラーステータス

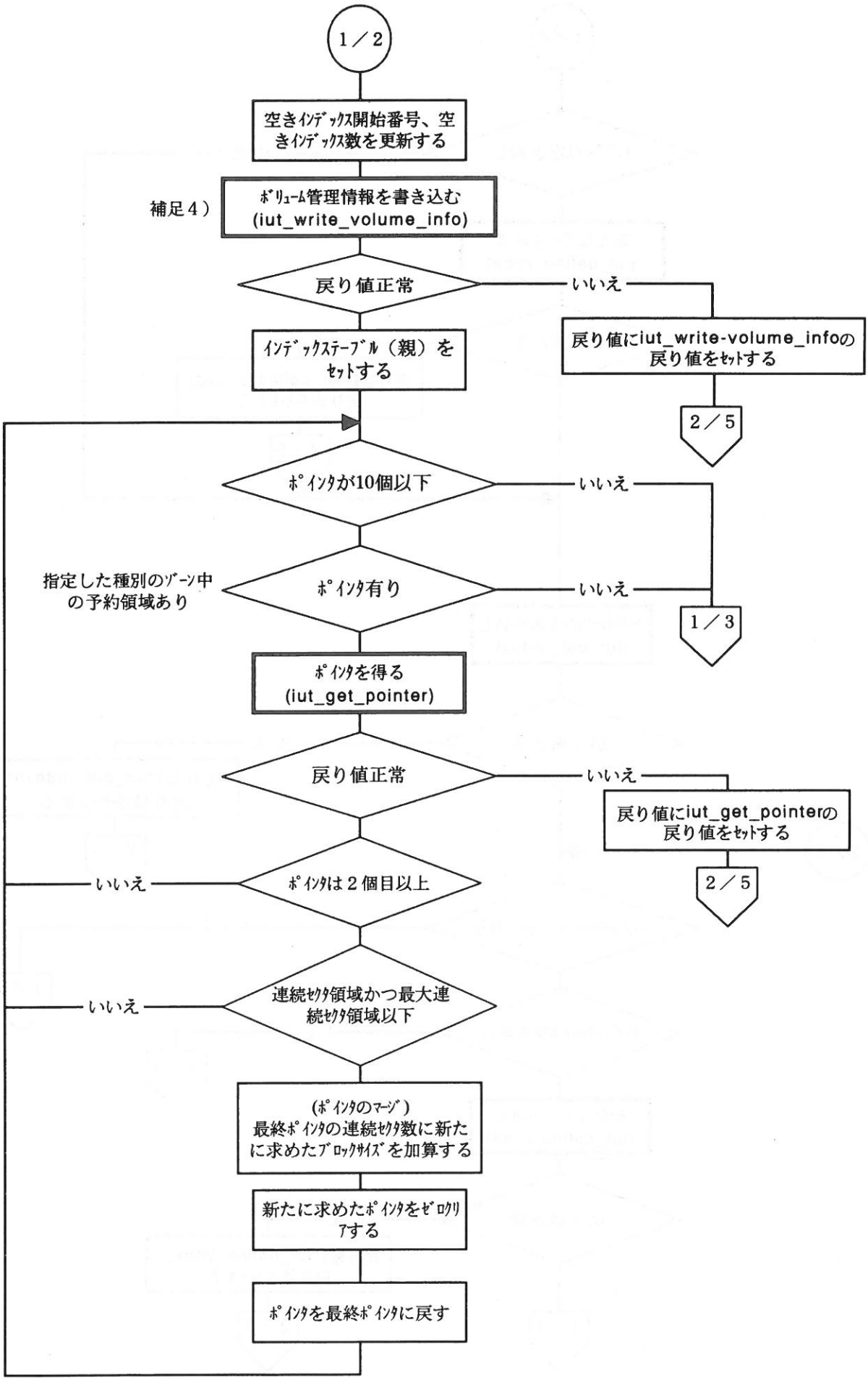
備考

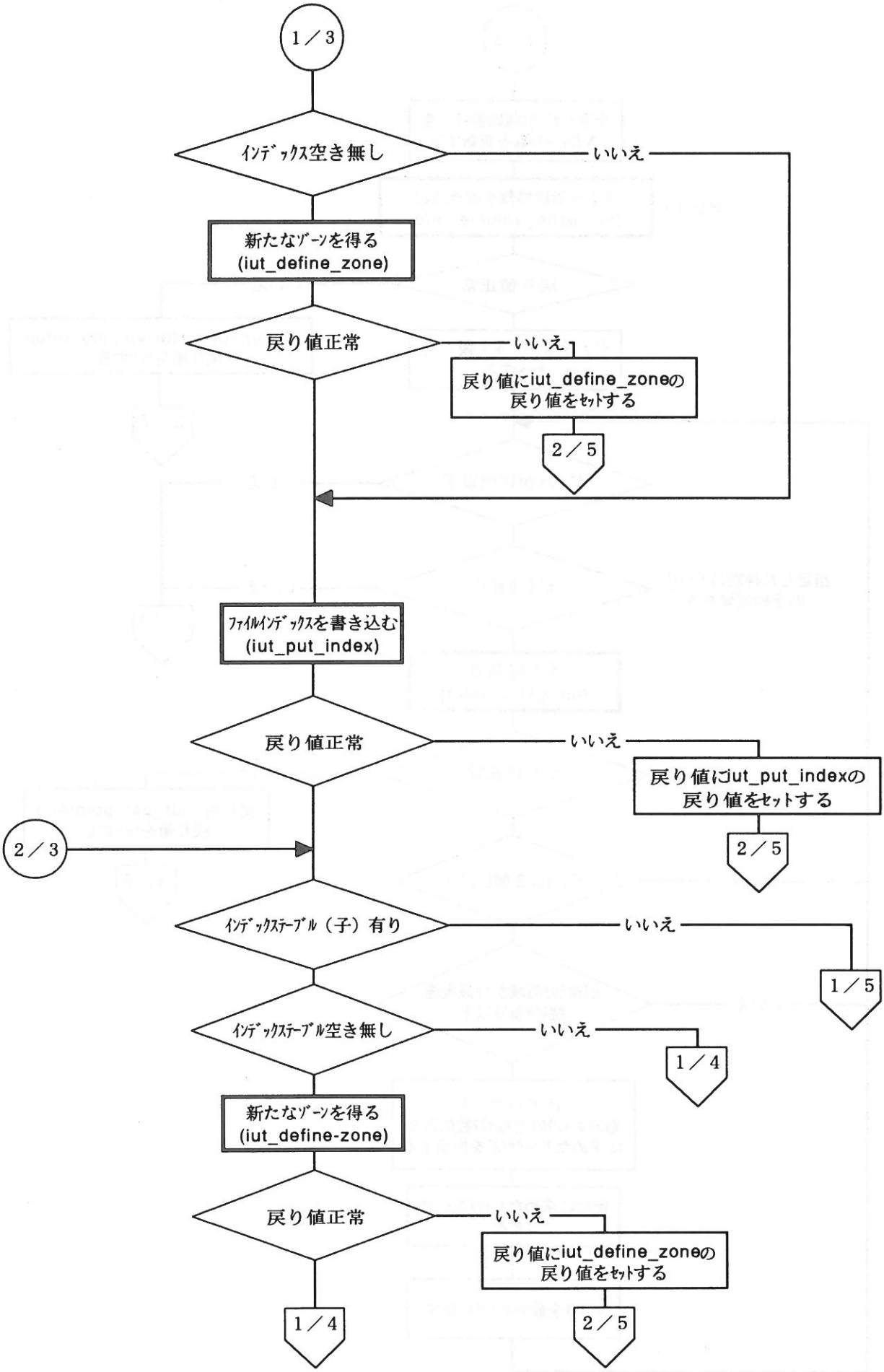
・ファイル名は最大24バイト長の文字列で、その最後にはヌル文字（値０）を付加しなければならない。25バイト以上の文字列を指定した場合には、先頭から24バイト分が使用される。

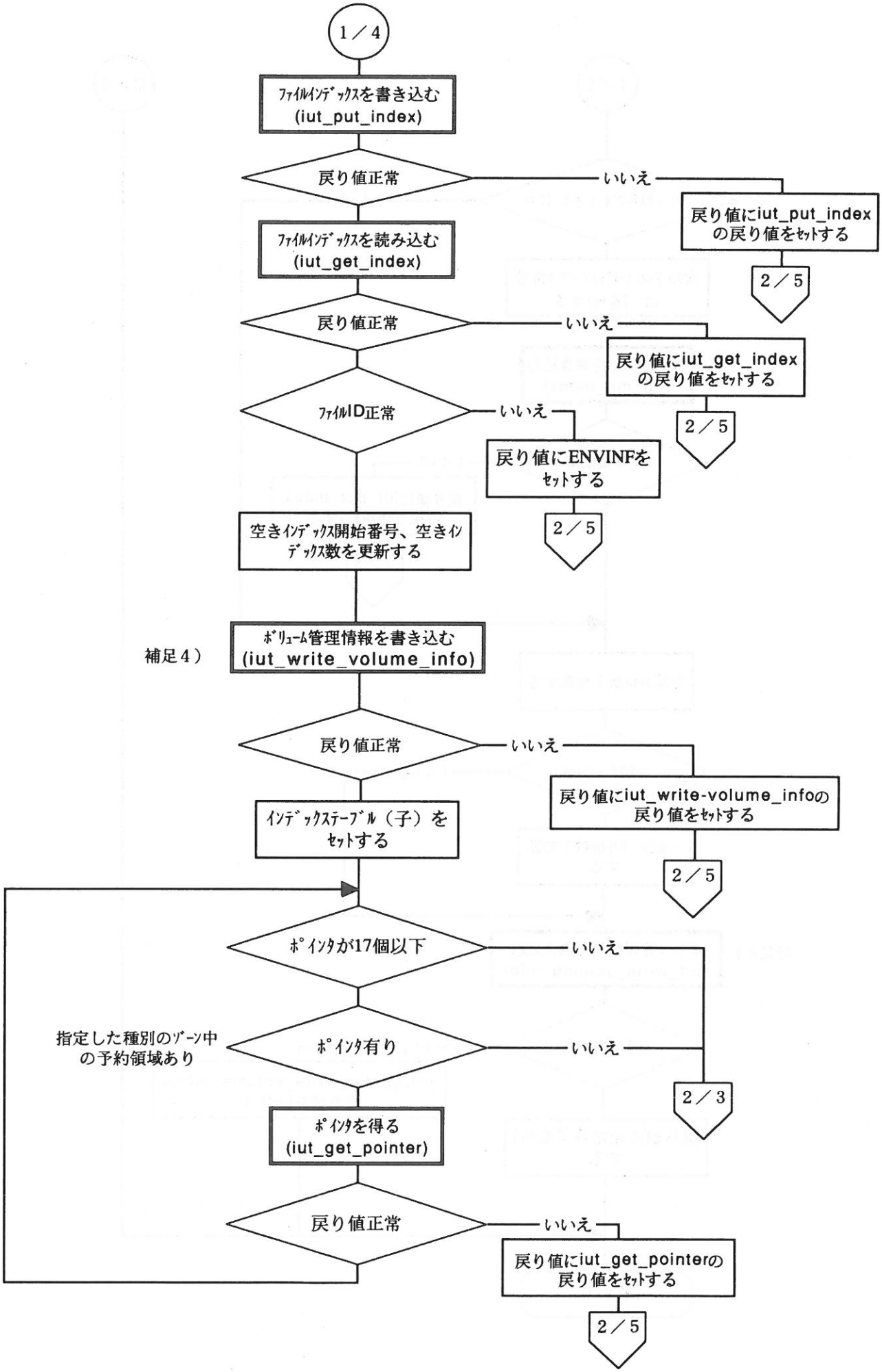
・エラーが発生した場合には、idの内容は意味をもたない。

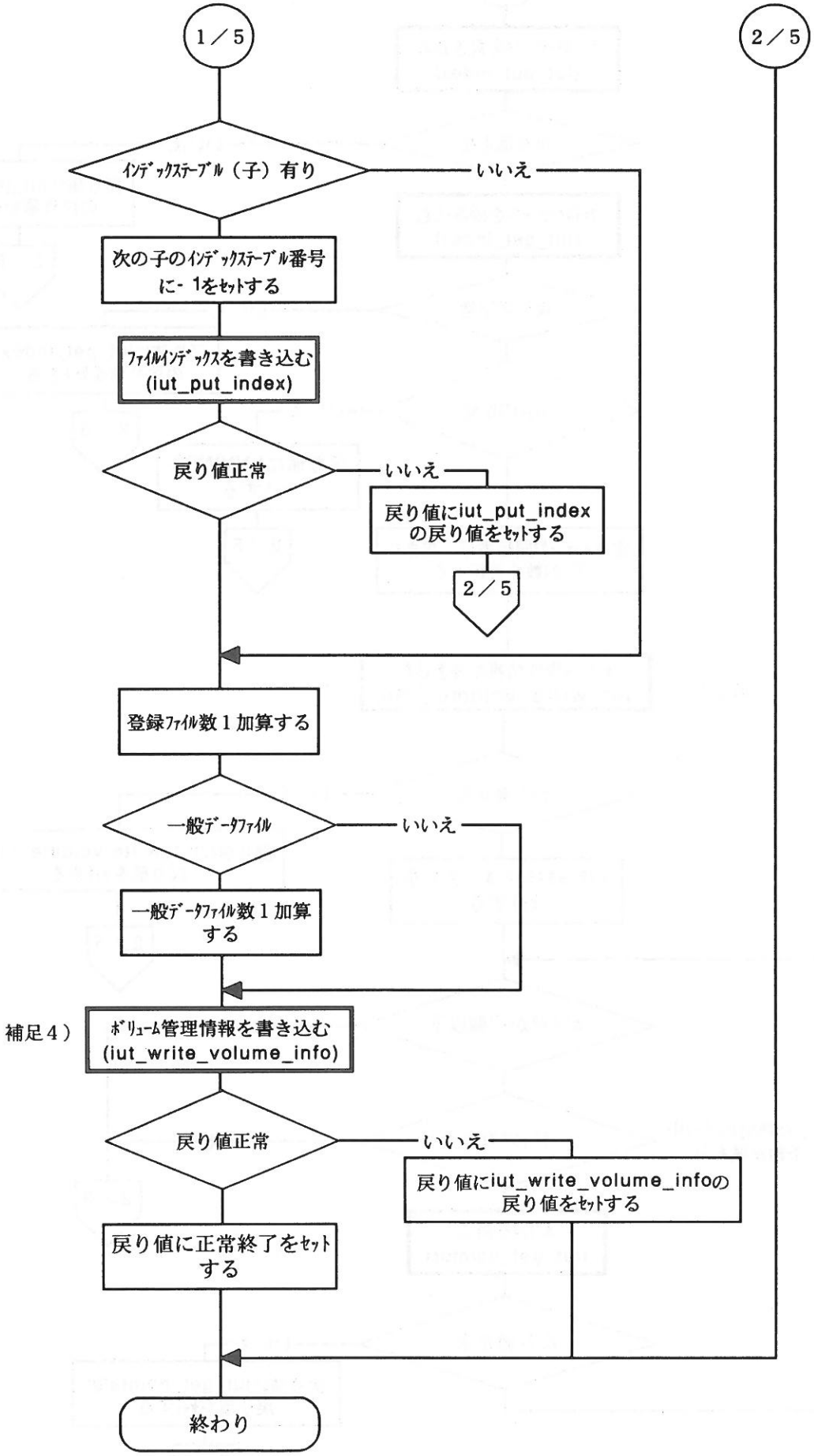
ifm_create_file




1／2
空きｲﾝﾃﾞｯｸｽ開始番号、空きｲﾝﾃﾞｯｸｽ数を更新する
補足４）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write-volume_infoの戻り値をｾｯﾄする
2／5
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）をｾｯﾄする
ﾎﾟｲﾝﾀが10個以下
いいえ
指定した種別のｿﾞｰﾝ中の予約領域あり
ﾎﾟｲﾝﾀ有り
いいえ
1／3
ﾎﾟｲﾝﾀを得る
(iut_get_pointer)
戻り値正常
いいえ
戻り値にiut_get_pointerの戻り値をｾｯﾄする
2／5
いいえ
ﾎﾟｲﾝﾀは２個目以上
いいえ
連続ｾｸﾀ領域かつ最大連続ｾｸﾀ領域以下
(ﾎﾟｲﾝﾀのﾏｰｼﾞ)
最終ﾎﾟｲﾝﾀの連続ｾｸﾀ数に新たに求めたﾌﾞﾛｯｸｻｲｽﾞを加算する
新たに求めたﾎﾟｲﾝﾀをｾﾞﾛｸﾘｱする
ﾎﾟｲﾝﾀを最終ﾎﾟｲﾝﾀに戻す


1／3
ｲﾝﾃﾞｯｸｽ空き無し
いいえ
新たなｿﾞｰﾝを得る
(iut_define_zone)
戻り値正常
いいえ
戻り値にiut_define_zoneの
戻り値をｾｯﾄする
2／5
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にut_put_indexの
戻り値をｾｯﾄする
2／5
2／3
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
1／5
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ空き無し
いいえ
1／4
新たなｿﾞｰﾝを得る
(iut_define-zone)
戻り値正常
いいえ
戻り値にiut_define_zoneの
戻り値をｾｯﾄする
1／4
2／5


1／4
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をｾｯﾄする
2／5
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む
(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_index
の戻り値をｾｯﾄする
2／5
ﾌｧｲﾙID正常
いいえ
戻り値にENVINFを
ｾｯﾄする
2／5
空きｲﾝﾃﾞｯｸｽ開始番号、空きｲﾝ
ﾃﾞｯｸｽ数を更新する
補足４）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write-volume_infoの
戻り値をｾｯﾄする
2／5
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）を
ｾｯﾄする
ﾎﾟｲﾝﾀが17個以下
いいえ
指定した種別のｿﾞｰﾝ中
の予約領域あり
ﾎﾟｲﾝﾀ有り
いいえ
2／3
ﾎﾟｲﾝﾀを得る
(iut_get_pointer)
戻り値正常
いいえ
戻り値にiut_get_pointerの
戻り値をｾｯﾄする
2／5


1／5
2／5
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
次の子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号
に- 1をｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をｾｯﾄする
2／5
登録ﾌｧｲﾙ数１加算する
一般ﾃﾞｰﾀﾌｧｲﾙ
いいえ
一般ﾃﾞｰﾀﾌｧｲﾙ数１加算
する
補足４）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をｾｯﾄする
戻り値に正常終了をｾｯﾄ
する
終わり

関数ifm_create_file()の補足説明

補足１）　ファイルを４のベキ乗単位に切り分ける。

ファイルのサイズを大きいゾーン順（４のベキ乗単位）に切り分けてどの種別のゾーンが何個必要であるかを求め、テーブルにセットする。

テーブルのフォーマット

	個数（４バイト）	ゾーン種別
＋０	1	Ｈゾーン
＋４	1	Ｇゾーン
＋８	3	Ｅゾーン
＋１２	3	Ｃゾーン
＋１６	0	
＋２０	0	

アルゴリズム

４のベキ乗単位に切り分けるアルゴリズムは以下の通り

```
残りのサイズ　＝　ファイルのサイズ（セクタ単位）
テーブルのポインタ　＝　０

ｆｏｒ　（ｎが５から０まで）

        ゾーンタイプ　＝　４のｎ乗
        （残りのサイズ／ゾーンタイプ）の商と余りを求める。

        ｉｆ（ｎ＝０　かつ　余り＞０）
           余り　＝　余り　＋　１
           商　　＝　余り
        ｅｎｄ　ｉｆ

        テーブル［テーブルのポインタ］　＝　商
        残りのサイズ　＝　余り
        テーブルのポインタ　＝　テーブルのポインタ　＋　４

 ｅｎｄ　ｆｏｒ
```

補足２）　ファイルを指定されたゾーン選択ビットに従って切り分ける

ファイルのサイズをゾーン選択ビットに従って切り分けて、どの種別のゾーンが何個必要であるかを求めテーブルにセットする。

7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	0
H	G	F	E	D	C		

テーブルのフォーマット

	個数（４バイト）	ゾーン種別
＋0	1	Hゾーン
＋4	3	Fゾーン
＋8	2	Dゾーン
＋１２	0	
＋１６	0	
＋２０	0	

アルゴリズム

ゾーン選択ビットに従って切り分けるアルゴリズムは以下の通り

```
残りのサイズ　＝　ファイルのサイズ（セクタ単位）
テーブルのポインタ　＝　0
ｆｏｒ　（ｎが５から０まで）
        ゾーンタイプ　＝　ゾーン選択ビット（Ｈゾーン～Ｃゾーン）
        （残りのサイズ／ゾーンタイプ）の商と余りを求める。

        ｉｆ（ｎ＝０　かつ　余り＞０）
           余り　＝　余り　＋　１
           商　　＝　余り
        ｅｎｄ　ｉｆ

        テーブル［テーブルのポインタ］　＝　商
        残りのサイズ　＝　余り
        テーブルのポインタ　＝　テーブルのポインタ　＋　４
 ｅｎｄ　ｆｏｒ
```

補足３）

インデックステーブルの作成

固定ファイルをクリエイトしたときにインデックステーブルを確保すると共に親インデックステーブルの内容を下記のようにセットする。

0	ファイル ID	インデックステーブル番号をセットする。
4	ファイル名	引き数にて与えられたファイル名を24文字セットする。24文字に満たない場合は後ろにヌル文字を詰め、24文字を越える場合は最初の24文字を採用する。
28	作成年月日	ファイルをクリエイトした日付をセットする。
32	作成時刻	ファイルをクリエイトした時刻をセットする。
34	最終変更年月日	初期値としてファイルをクリエイトした日付をセットする。
38	最終変更時刻	初期値としてファイルをクリエイトした時刻をセットする。
40	ファイルバイト長	引き数で与えられたファイルのサイズ(size)をセットする。
44	属性	$(0000)_{16}$ をセットする。
58	ヘッダポインタ	初期値として(0,0)をセットする。
64	#1 ポインタ	最初に確保したエリアの開始セクタとセクタ数をセットする。
70	#2 ポインタ	2番目に確保したエリアの開始セクタとセクタ数をセットする。
76	#3 ポインタ	以下#2 ポインタと同様。

118	#10 ポインタ	#2 ポインタと同様。
124	子のインデックステーブル番号	子インデックスがあれば子インデックスのテーブル番号、無ければ-1をセットする。

サイズ固定ファイルをクリエイトしたとき、空きエリアを確保したポインタのテーブルのサイズにより、インデックス１つでは不足の場合、子インデックスを下記のように作成して親インデックスに子インデックスのテーブル番号をセットする。

ポインタの個数とインデックス数の関係は次の通りである。

```
if(ポインタの個数が10以下)
  then  インデックス数=1
  else  インデックス数=(ポインタの個数－11)/17の商+2
end if
```

子インデックスの内容は以下の通りである。

0	インデックステーブル番号	自分のインデックステーブル番号の2の補数をセットする。
4	親のインデックステーブル番号	親インデックスのテーブル番号をセットする
8	予備	０を詰める。（10バイト分）
22	#1　ポインタ	親インデックスに入り切らなかったポインタをセットする。
28	#2　ポインタ	#1　ポインタと同様
118	#17 ポインタ	#1　ポインタと同様
124	次の子のインデックステーブル番号	自分の子のインデックスが存在する場合そのインデックステーブル番号をセットする。（存在しない場合は－１をセットする。）

例）　親インデックス＝３、第１子＝４、第２子＝６、第３子＝８の場合を示す。

テーブル番号	3	4	5	6	7	8
テーブル番号の2の補数	ID=3	-4	ID=5	-6	ID=7	-8
親のインデックス番号		3		3		3
次の子のインデックス番号	4	6	-1	8	-1	-1
	親	第1子		第2子		第3子

補足４）
ボリューム管理情報の更新

ファイルをクリエイトしたときに、ボリューム内のファイル数およびインデックステーブル数が増したことを管理情報に残すため、ディスク上、主記憶上いずれもボリューム管理情報を更新する。

（１）　ボリューム管理情報その１（０セクタ）については何も更新しない。

（２）　ボリューム管理情報その２（１セクタ）については下記のように更新する。

0	インデックス総数	更新せず。
4	登録ファイル数	１増加する。
8	仮削除ファイル数	更新せず。
12	空きインデックス数	今回作成したインデックス数だけ減算する。
16	システムファイル数	更新せず。
18	ディレクトリファイル数	更新せず。
20	更新日時	更新時の日時。
26	空きインデックス開始番号	今回作成したインデックス数だけ増加する。
30	ボリューム使用中フラグ	更新せず。
32	予約領域	更新せず。
1008	システム予約領域	作成されたファイルが一般データファイルの場合のみ一般データファイル数に１増加する。

関数ifm_create_file（）の検査指針

関数 ifm_create_file()はボリュームに対してマウントされている状態のもとで正常に動作するので、機能に関する検査についてはマウントされていることを確認の上、検査を行うこと。

（１）機能に関する検査

1) ファイルの存在しない状態の下で、２１Ｋバイトのファイルが作成されることを確認する。

確認項目は以下の通り

- ゾーンテーブルが３つ作成され、
  Ｅゾーン、Ｄゾーン、Ｃゾーンのテーブルが作成される。
  それぞれの空きブロック数に６３、２５５、１０２３がセットされる
  それぞれのバックアップゾーンには０がセットされる。
- それぞれのゾーンに対応するセクタテーブルに
  それぞれ１６ビット、４ビット、１ビットがＯＮになる。
- インデックステーブルが１つ作成され、
  ファイルＩＤに１がセットされる。
  指定したファイル名が２４文字以内でセットされる。
  作成年月日と最終変更年月日、作成時刻と最終変更時刻にそれぞれ同じ日付と時間がセットされる。
  ファイルバイト長にファイルのサイズがバイト単位でセットされる。
  属性に（００００）$_{16}$がセットされる。
  ヘッダポインタに（０、０）がセットされる。
  ＃１ポインタにＥゾーンの開始セクタとセクタサイズ１６がセットされる。
  ＃２ポインタにはＤゾーンの開始セクタとセクタサイズ４がセットされる。
  ＃３ポインタにはＣゾーンの開始セクタとセクタサイズ１がセットされる。
  ＃４ポインタ以降は（０、０）がセットされる。
  子のインデックステーブル番号には、－１がセットされる。

2) ファイルのサイズを２０Ｍにして、1と同様の検査を行う。
3) ファイルのサイズを１Ｋにして、1と同様の検査を行う。
4) 連続するエリアのとれない状態の下で、1と同様の検査を行う。
5) インデックスまたがりの場合について、1と同様の検査を行う。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。
2) 存在しないユニット番号を指定した場合は、ＥＮＵＮＩＴが返されることを確認する。
3) マウントされていない状態の場合には、ＥＮＭＮＴＤが返されることを確認する。
4) ゾーン選択ビットが０の場合は、ＥＰＡＲＡＭが返されることを確認する。
5) ｖｏｌ＿ｄｔの値が不正の場合は、ＥＮＶＩＮＦが返されることを確認する。
6) 光磁気ディスクの電源を切り、ＥＤＲＩＶＥが返されることを確認する。

（３）境界条件に関する検査

1) エリアの空きが１０Ｋのみの状態の下で、サイズ１０Ｋのファイルを作成する検査を行い、各テーブルの内容が正しいことを確認する。
2) エリアの空きが９Ｋのみの状態の下で、サイズ１０Ｋのファイルを作成する検査を行い、戻り値に iut_define_zoneの戻り値（ＥＮＤＡＴＡ）が返されることを確認する。
3) インデックスの空きが１件のみの状態で、１Ｋのファイルを作成する検査を行い、開始セクタとセクタサイズが正しく返されることを確認する。
4) インデックスの空きが１件のみの状態で、インデックスまたがりするファイルを作成する検査を行い、戻り値に iut_define_zoneの戻り値（ＥＮＩＮＤＸ）が返されることを確認する。

４．４．３　サイズ可変のファイルを作成する

```
long  ifm_create_file_variable(vd, name, cell, id)
long  vd ;
char  *name ;
long  cell ;
long  *id ;
```

機能

セルサイズを指定して、新規にファイルを作成する。

引数

vd　（入力）ボリュームディスクリプタ。

name（入力）ファイル名が入っている文字列へのポインタ。

cell（入力）セルサイズをバイト単位で表した値。1024の正の整数倍で指定する（1024*1024,256*1024,64*1024,16*1024,4*1024,1024バイトのいずれかを指定）。

id　（出力）作成されたファイルのファイルＩＤへのポインタ。

戻り値

正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ENVINF（vol_dtの内容が正しいボリューム管理情報ではない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

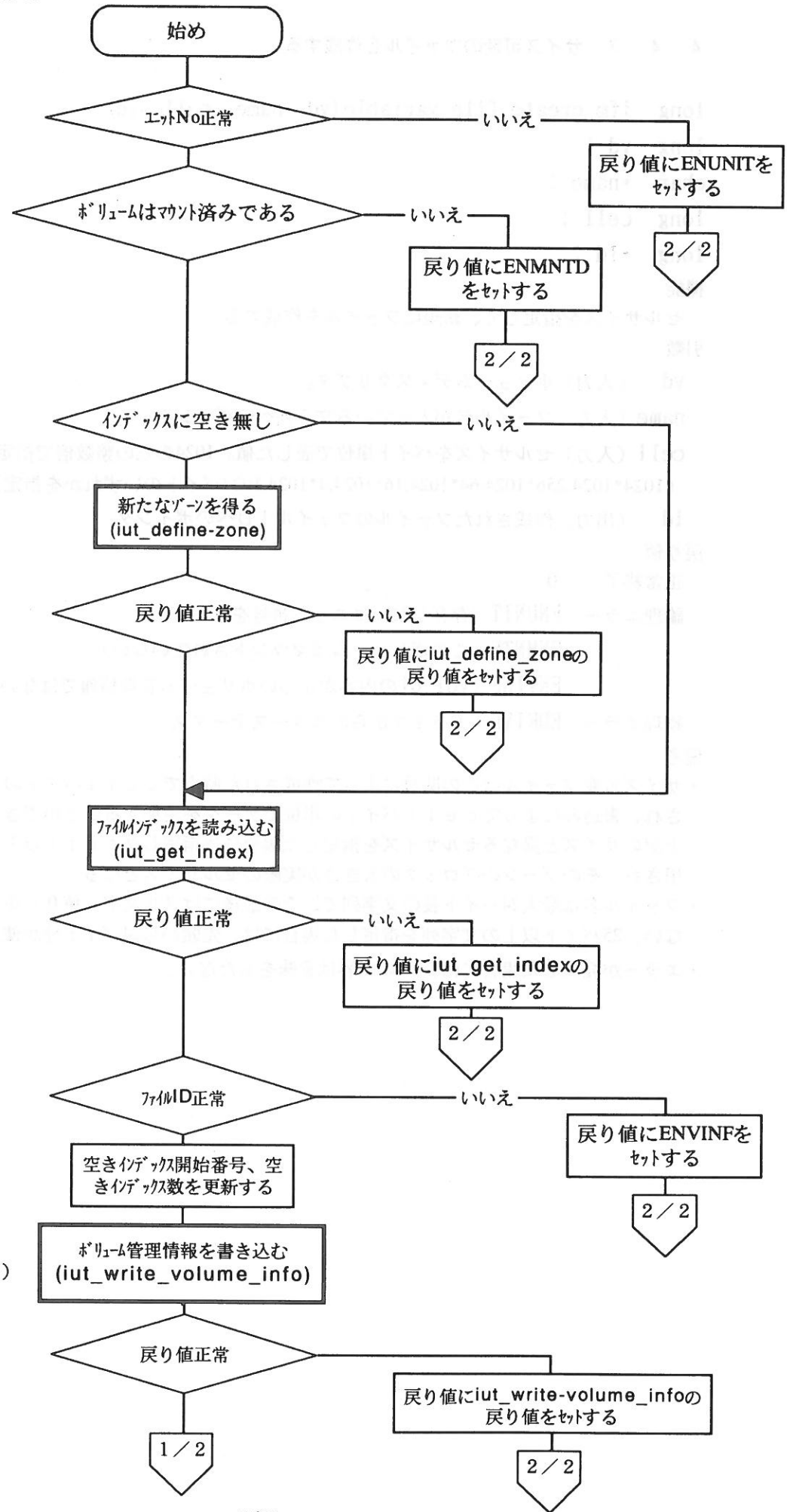
・サイズ可変ファイルはこの関数によって作成された時点でｃｅｌｌバイトの大きさの領域が確保され、書込みによってｃｅｌｌバイトの単位でデータを追加することができる。

・上記のサイズと異なるセルサイズを指定した場合には指定したｃｅｌｌ以下の最大のゾーンが使用され、そのゾーンのブロックの大きさが実際のセルサイズとなる。

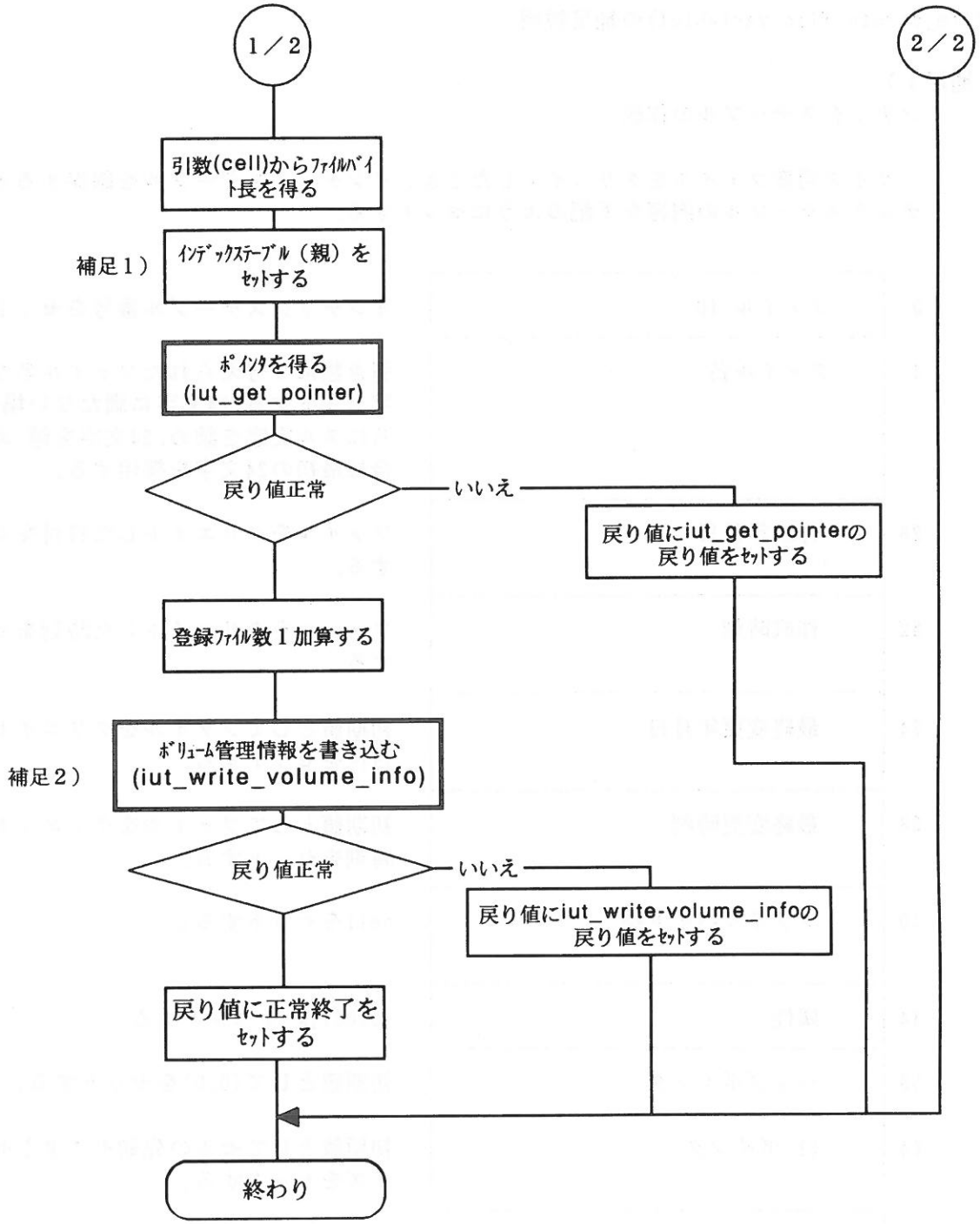
・ファイル名は最大24バイト長の文字列で、その最後にはヌル文字（値０）を付加しなければならない。25バイト以上の文字列を指定した場合には、先頭から24バイト分が使用される。

・エラーが発生した場合には、idの内容は意味をもたない。

ifm_create_file_variable



補足２）


1／2
2／2
引数(cell)からﾌｧｲﾙﾊﾞｲﾄ長を得る
補足１）
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）をｾｯﾄする
ﾎﾟｲﾝﾀを得る
(iut_get_pointer)
戻り値正常
いいえ
戻り値にiut_get_pointerの戻り値をｾｯﾄする
登録ﾌｧｲﾙ数１加算する
補足２）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write-volume_infoの戻り値をｾｯﾄする
戻り値に正常終了をｾｯﾄする
終わり

関数ifm_create_file_variable()の補足説明

補足１）
インデックステーブルの作成

サイズ可変ファイルをクリエイトしたとき、インデックステーブルを確保すると共にインデックステーブルの内容を下記のようにセットする。

0	ファイル ID	インデックステーブル番号をセットする。
4	ファイル名	引き数にて与えられたファイル名を24文字セットする。24文字に満たない場合は後ろにヌル文字を詰め、24文字を越 える場合は最初の24文字を採用する。
28	作成年月日	ファイルをクリエイトした日付をセットする。
32	作成時刻	ファイルをクリエイトした時刻をセットする。
34	最終変更年月日	初期値としてファイルをクリエイトした日付をセットする。
38	最終変更時刻	初期値としてファイルをクリエイトした時刻をセットする。
40	ファイルバイト長	cellをセットする。
44	属性	$(0100)_{16}$ をセットする。
58	ヘッダポインタ	初期値として(0,0)をセットする。
64	#1 ポインタ	初期値としてセルの先頭セクタとセルサイズをセットする。
70	#2 ポインタ	開始セクタとセクタ数として（0,0）をセットする。
76	#3 ポインタ	以下#2 ポインタと同様。

118	#10 ポインタ	#2 ポインタと同様。
124	子のインデックステーブル番号	初期値として（-1）をセットする。

補足２）
ボリューム管理情報の更新

ファイルをクリエイトしたときに、ボリューム内のファイル数およびインデックステーブル数が増したことを管理情報に残すため、ディスク上、主記憶上いずれもボリューム管理情報を更新する。

（１）　ボリューム管理情報その１（０セクタ）については何も更新しない。

（２）　ボリューム管理情報その２（１セクタ）については下記のように更新する。

0	インデックス総数	更新せず。
4	登録ファイル数	１増加する。
8	仮削除ファイル数	更新せず。
12	空きインデックス数	今回作成したインデックス数だけ減算する。
16	システムファイル数	更新せず。
18	ディレクトリファイル数	更新せず。
20	更新日時	更新時の日時。
26	空きインデックス開始番号	今回作成したインデックス数だけ増加する。
30	ボリューム使用中フラグ	更新せず。
32	予約領域	更新せず。
1008	システム予約領域	作成されたファイルが一般データファイルの場合のみ一般データファイル数に１増加する。

関数ifm_create_file_variable()の検査指針

関数ifm_create_file_variable()はボリュームに対してマウントされている状態の下で正常に動作するので、機能に関する検査については、マウントされていることを確認の上、検査を行うこと。

（１）機能に関する検査

1) ファイルの存在しない状態の下で、サイズ可変ファイルが作成されることを確認する。

確認項目は以下の通り

・インデックステーブルが１つ作成され、ファイルＩＤに１がセットされる。
・指定したファイル名が２４文字以内でセットされる。
・作成年月日と最終変更年月日、作成時刻と最終変更時刻にそれぞれ同じ日付と時間がセットされる。
・ファイルバイト長に使用されるゾーンのブロックサイズがセットされる。
・属性に（０１００）$_{16}$がセットされる。
・ヘッダポインタに（０、０）がセットされる。
・＃１ポインタに開始セクタとセクタサイズと、使用されるゾーンのブロックサイズがセットされる。
・＃２ポインタには開始セクタとセクタサイズ（０、０）がセットされる。
・＃３ポインタ以降は（０、０）がセットされる。
・子のインデックステーブル番号には、－１がセットされる。

2) ファイルが複数存在する状態の下で、1と同様の検査を行う。
3) cellを１Ｋにして、1と同様の検査を行う。
4) cellを１Ｍにして、1と同様の検査を行う。

（２）戻り値に関する検査

1) 正常終了した場合には、０が返されることを確認する。
2) 存在しないユニット番号を指定した場合は、ＥＮＵＮＩＴが返されることを確認する。
3) マウントされていない状態の場合には、ＥＮＭＮＴＤが返されることを確認する。
4) ｖｏｌ＿ｄｔの値が不正の場合は、ＥＮＶＩＮＦが返されることを確認する。
5) 光磁気ディスクの電源を切り、ＥＤＲＩＶＥが返されることを確認する。

（３）境界条件に関する検査

1) インデックスの空きが１件のみの状態の下で、ファイルを作成する検査を行い、正常に動作することを確認する。
2) インデックスの空きがない状態の下で、ファイルを作成する検査を行い、戻り値にiut_define_zoneの戻り値（ＥＮＩＮＤＸ）が返されることを確認する。
3) 関数ifm_create_file_variable()でファイルを作成し、そのファイルを指定した場合、fsize=セルサイズ、hsize=０になることを確認する。

4．4．4　ファイル名からファイルＩＤを得る

```
long    ifm_get_file_id(vd,  name,  id,  req_size,  act_size)
long    vd ;
char    *name ;
long    id[ ] ;
long    req_size ;
long    *act_size ;
```

機能

ファイル名を指定して、同一名称のファイルのファイルＩＤを得る。

引数

vd　　（入力）ボリュームディスクリプタ。

name　　（入力）ファイル名が入っている文字列へのポインタ。

id　　（入出力）ファイルＩＤを格納する配列へのポインタ。＊、？のwild cardが使用可。id[0]に入力されたid番号以上のid番号が検索される。

req_size（入力）idの配列の個数。最大req_size個のファイルのファイルＩＤが得られる。

act_size（出力）idに得られたファイルＩＤの個数である。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）

ENMNTD（このボリュームはマウントされていない）

物理エラー　EDRIVE｜ドライブからのエラーステータス

備考

・読み込み禁止ファイル、システムファイル、削除されたファイルのファイルＩＤは得られない。さらにこれらのファイルは、act_sizeの個数の中にも含まれない。

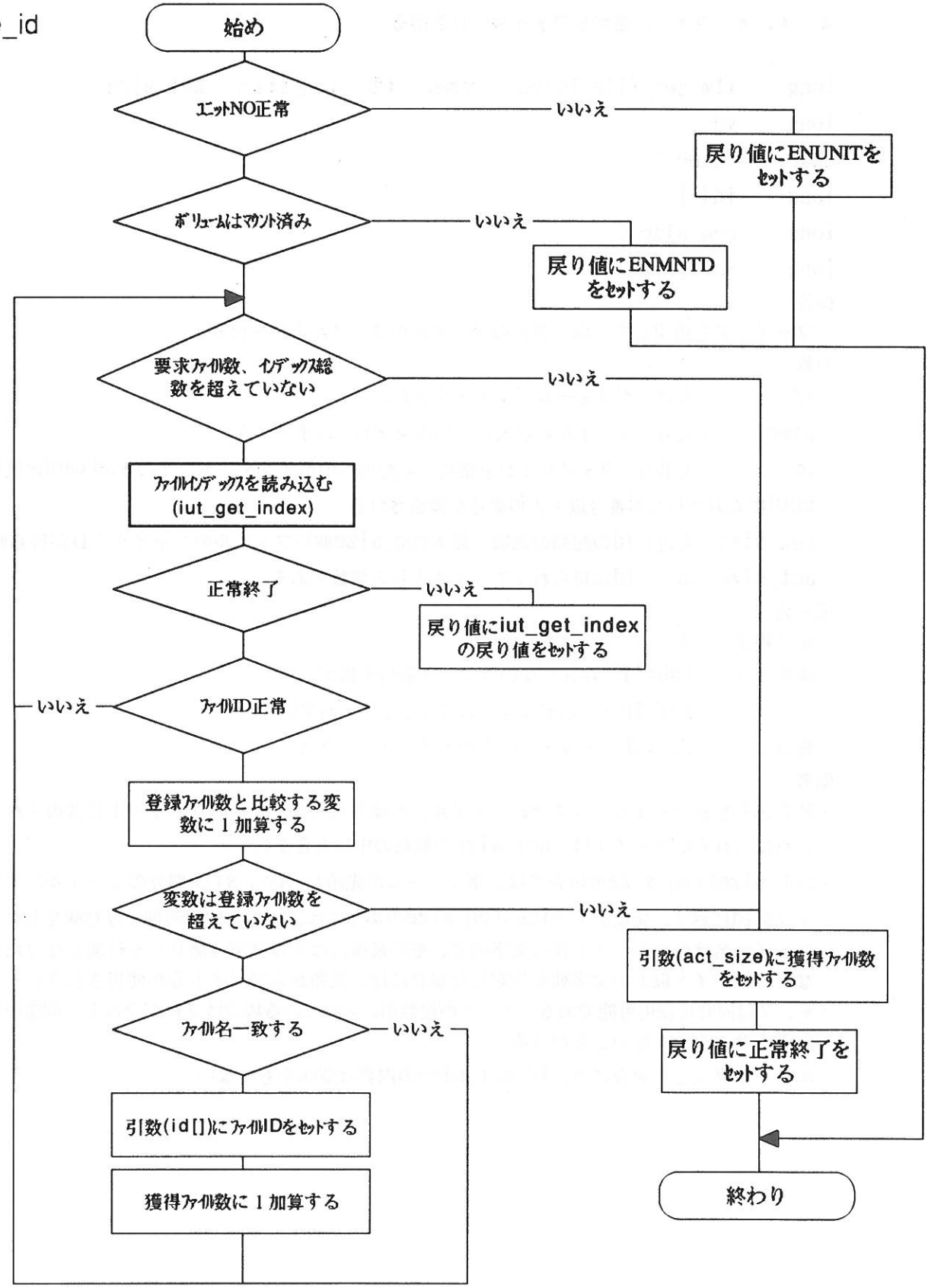
・act_size〉req_sizeの場合には、ボリュームの先頭からreq_size個分のファイルのファイルＩＤをidに返す。またact_size〈req_sizeの場合には、idの未使用の部分は意味をもたない。

・ファイル名は最大24バイト長の文字列で、その最後にはヌル文字（値０）を付加しなければならない。25バイト以上の文字列を指定した場合には、先頭から24バイト分が使用される。

・＊、？は同時に使用可能である。＊、？が複数用いられている場合はプログラム上、問題があり、上手くサーチできないことがある。

・エラーが発生した場合には、id, act_sizeの内容は意味をもたない。

ifm_get_file_id



関数ifm_get_file_id()の検査指針

関数ifm_format_media(), ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

関数 ifm_create_file()を用いて以下のファイル名のファイルを作成し１）と２）の確認をする。ｉｄ［０］＝１を代入すること。

”ＦＩＬＥ－０１”	――	ｉｄ＝１
”ＦＩＬＥ－０２”	――	ｉｄ＝２
”ＦＩＬＥ－０３”	――	ｉｄ＝３
”ＦＩＬＥ－０１”	――	ｉｄ＝４
”ＦＩＬＥ－０１”	――	ｉｄ＝５
”ＦＩＬＥ－０１”	――	ｉｄ＝６
”ＦＩＬＥ－０２”	――	ｉｄ＝７
”ＦＩＬＥ－０３”	――	ｉｄ＝８

次に、ファイルの属性を以下のように変更してから３）の確認をする。

・ｉｄが１のファイルを関数ifm_read_protect_on()を用いて読み込み禁止にする。
・ｉｄが４のファイルを関数ifm_system_flag_on()を用いてシステムファイルにする。
・ｉｄが６のファイルを関数ifm_delete_file()を用いて仮削除する。

１）ｎａｍｅが”ＦＩＬＥ－０１”，ｒｅｑ＿ｓｉｚｅ＝１０のとき
ａｃｔ＿ｓｉｚｅが４で、ｉｄには１、４、５、６がセットされることを確認する。

２）ｎａｍｅが”ＦＩＬＥ－０１”，ｒｅｑ＿ｓｉｚｅ＝３のとき
ａｃｔ＿ｓｉｚｅが３で、ｉｄには１、４、５がセットされることを確認する。

３）ｎａｍｅが”ＦＩＬＥ－０１”，ｒｅｑ＿ｓｉｚｅ＝１０のとき
ａｃｔ＿ｓｉｚｅが１で、ｉｄには１、４、５がセットされることを確認する。

４）ｎａｍｅが”ＦＩＬＥ－０１０１”，ｒｅｑ＿ｓｉｚｅ＝１０のときａｃｔ＿ｓｉｚｅが０になることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

３）マウントしないで本関数を呼び出し、ＥＮＭＮＴＤが返されることを確認する。

４）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．４．５　ファイルを仮削除する

```
long   ifm_delete_file(vd, id)
long   vd ;
long   id ;
```

機能

ファイルＩＤを指定して、ファイルを仮削除する。

引数

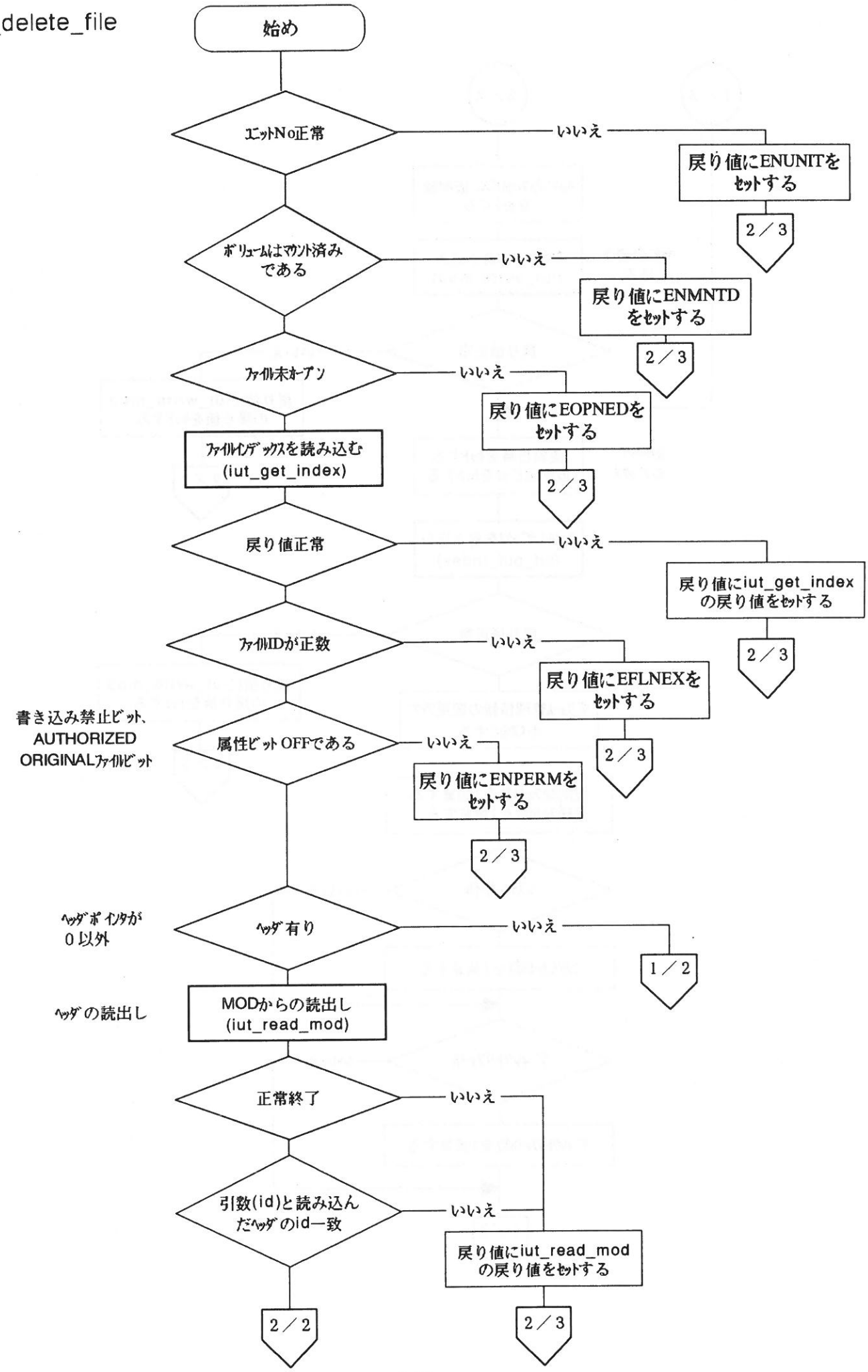
vd（入力）ボリュームディスクリプタ。

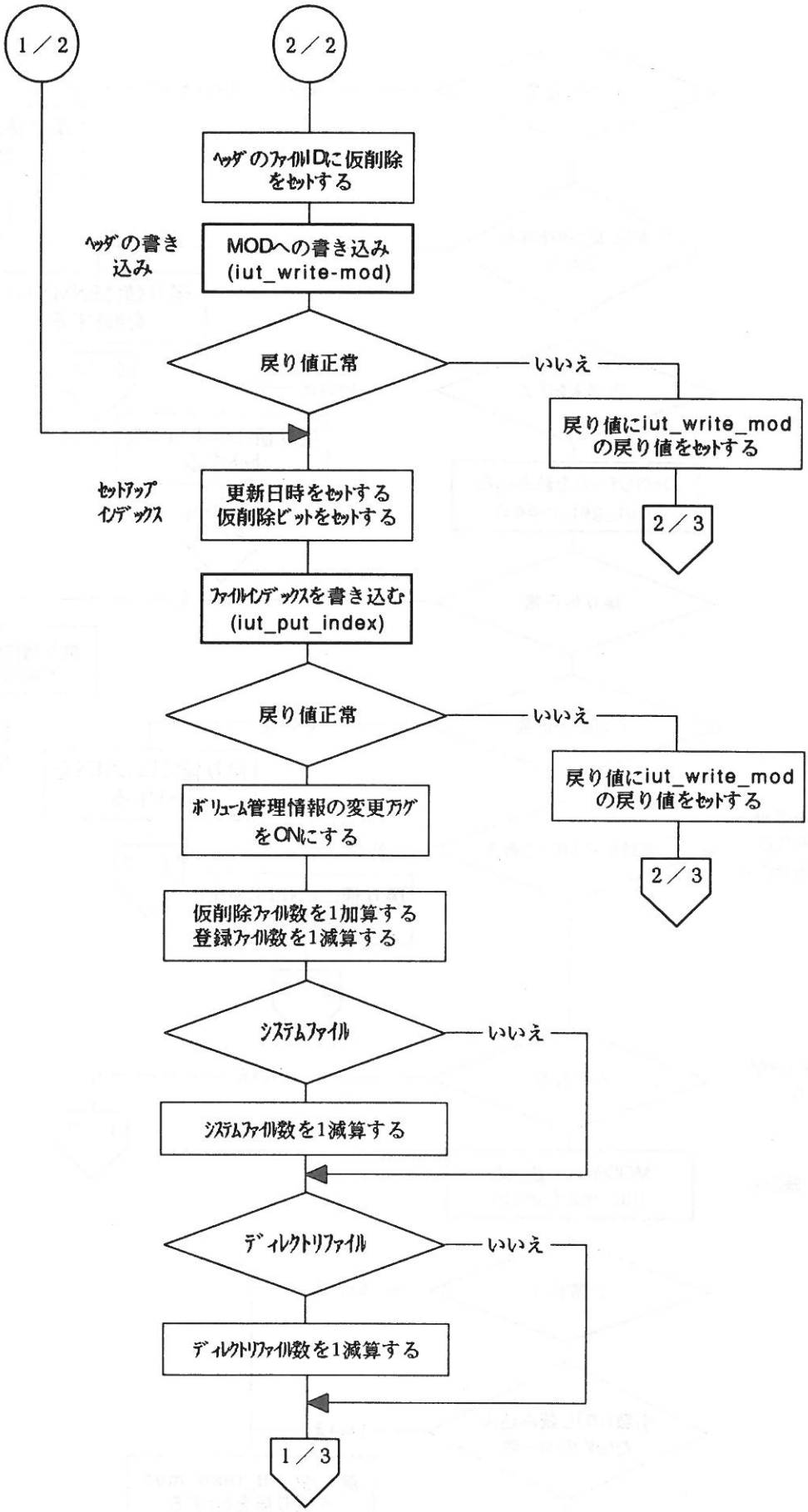
id（入力）仮削除するファイルのファイルＩＤ。

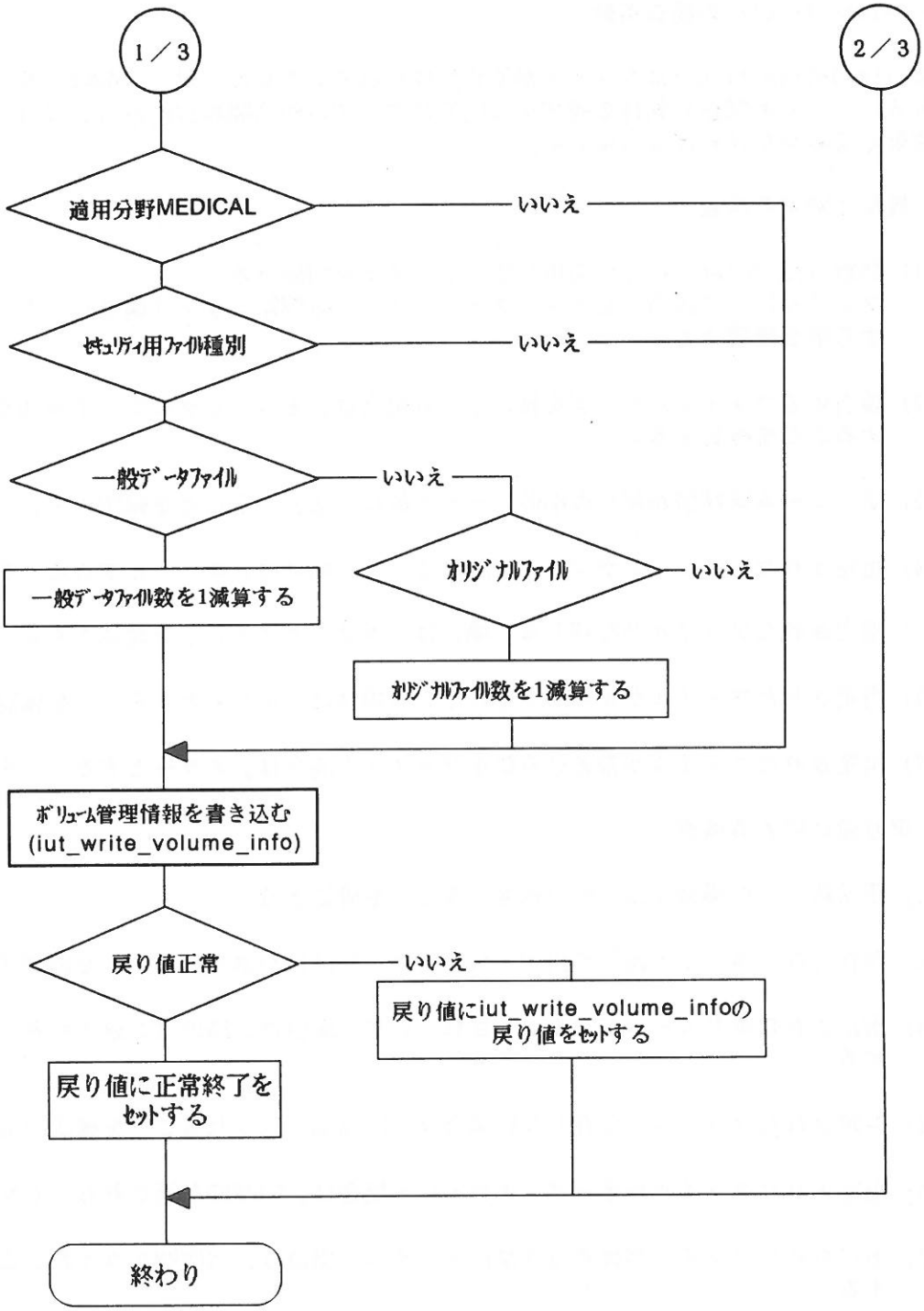
戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENUNIT（存在しないユニット番号を指定した）
	EOPNED（指定したファイルはオープンされている）
	ENPERM（指定したファイルは書き込み禁止ファイルである）
	EFLNEX（存在しないファイルを指定した）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_delete_file




1／2
2／2
ﾍｯﾀﾞのﾌｧｲﾙIDに仮削除をｾｯﾄする
ﾍｯﾀﾞの書き込み
MODへの書き込み (iut_write-mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をｾｯﾄする
2／3
ｾｯﾄｱｯﾌﾟｲﾝﾃﾞｯｸｽ
更新日時をｾｯﾄする 仮削除ﾋﾞｯﾄをｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む (iut_put_index)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をｾｯﾄする
2／3
ﾎﾞﾘｭｰﾑ管理情報の変更ﾌﾗｸﾞをONにする
仮削除ﾌｧｲﾙ数を1加算する 登録ﾌｧｲﾙ数を1減算する
ｼｽﾃﾑﾌｧｲﾙ
いいえ
ｼｽﾃﾑﾌｧｲﾙ数を1減算する
ﾃﾞｨﾚｸﾄﾘﾌｧｲﾙ
いいえ
ﾃﾞｨﾚｸﾄﾘﾌｧｲﾙ数を1減算する
1／3


1／3
2／3
適用分野MEDICAL
いいえ
ｾｷｭﾘﾃｨ用ﾌｧｲﾙ種別
いいえ
一般ﾃﾞｰﾀﾌｧｲﾙ
いいえ
ｵﾘｼﾞﾅﾙﾌｧｲﾙ
いいえ
一般ﾃﾞｰﾀﾌｧｲﾙ数を1減算する
ｵﾘｼﾞﾅﾙﾌｧｲﾙ数を1減算する
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をｾｯﾄする
戻り値に正常終了を
ｾｯﾄする
終わり

関数ifm_delete_file() の検査指針

関数 ifm_delete_file()はファイルが予め作成されているものに対し，削除（仮）を行うものである。よって本関数の動作を確認する以前にファイル作成関数ifm_create_file() 等の動作確認をしておかなければならない。

（１）機能に関する検査

1) 関数 ifm_delete_file()を用いて，ファイルを削除する。
ファイルIDにて該当するインデクスファイルの仮削除ビット（属性ビット）をONにする事を確認する。

2) 該当するファイルがヘッダを持っている場合は，そのヘッダのファイルIDを２の補数にすることを確認する。

3) ボリューム管理情報部の仮削除ファイル数に１加算することを確認する。

4) 指定されたボリュームがマウントされていない場合は，エラーとすることを確認する。

5) 指定されたファイルが存在しない場合は，エラーとすることを確認する。

6) 指定されたファイルがオープンされている場合は，エラーとすることを確認する。

7) 指定されたファイルが書き込み禁止ファイルの場合は，エラーとすることを確認する。

（２）戻り値に関する検査

1) 正常終了した場合には，0 が返されることを確認する。

2) 存在しないユニット番号を指定した場合は、ENUNITが返されることを確認する。

3) 指定されたボリュームがマウントされていない場合は，ENMNTDが返されることを確認する。

4) 指定されたファイルが存在しない場合は，EFLNEXが返されることを確認する。

5) 指定されたファイルがオープンされている場合は，EOPNEDが返されることを確認する。

6) 指定されたファイルが書き込み禁止ファイルの場合は，ENPERMが返されることを確認する。

7) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

4．5　データの入出力

4．5．1　ファイルをオープンする

```
long  ifm_open_file(vd, id, mode, fd)
long  vd ;
long  id ;
long  mode ;
long  *fd ;
```

機能

ファイルＩＤを指定して、ファイルをオープンする。

引数

vd　（入力）ボリュームディスクリプタ。

id　（入力）オープンするファイルのファイルＩＤ。

mode（入力）オープンのモード（読出し専用、書込み専用、読み書き）。

fd　（出力）指定したファイルに対する、ファイルディスクリプタへのポインタ。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）

ENMNTD（このボリュームはマウントされていない）

EFLNEX（存在しないファイルを指定した）

EOPNED（指定したファイルはすでにオープンされている）

ENPERM（指定したファイルは読み込み禁止ファイル、あるいはシステムファイルである）

ENBUFF（オープンされているファイルの個数が多すぎる）

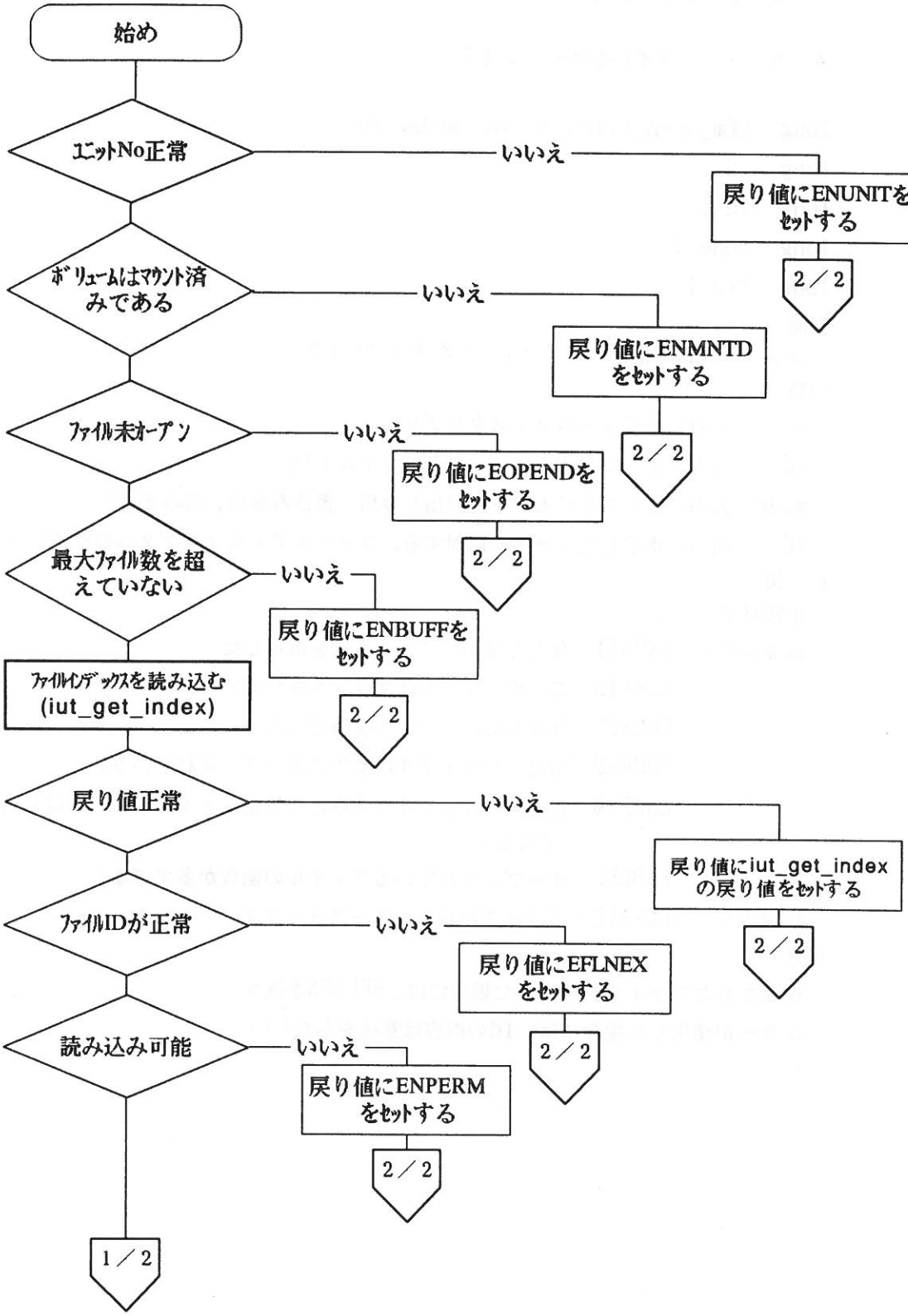
物理エラー　EDRIVE｜ドライブからのエラーステータス

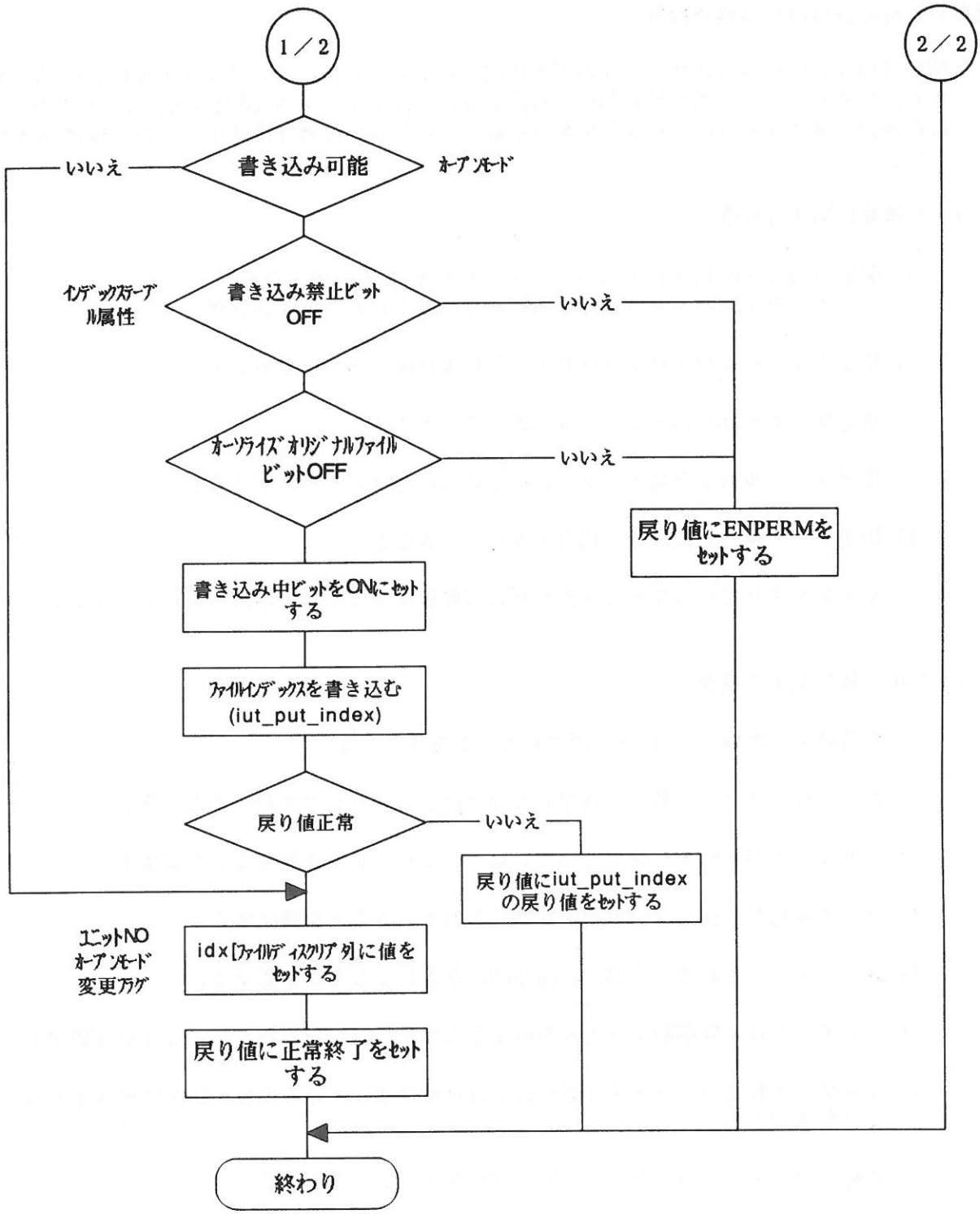
備考

・削除されたファイルを指定した場合には、EFLNEXを返す。

・エラーが発生した場合には、fdの内容は意味をもたない。

ifm_open_file




1／2
2／2
書き込み可能
いいえ
オープンモード
インデックステーブル属性
書き込み禁止ビット OFF
いいえ
オーソライズ オリジナルファイル ビットOFF
いいえ
戻り値にENPERMをセットする
書き込み中ビットをONにセットする
ファイルインデックスを書き込む (iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_indexの戻り値をセットする
ユニットNO オープンモード 変更フラグ
idx[ファイルディスクリプタ]に値をセットする
戻り値に正常終了をセットする
終わり

関数ifm_open_file() の検査指針

関数 ifm_open_file()はファイルが予め作成されているものに対してファイルのオープンを行うものである。よって本関数の動作を確認する以前にファイル作成関数ifm_create_file() の動作確認、ボリュームのマウント関数ifm_mount_media() の動作確認をしておかなければならない。

（１）機能に関する検査

1) 関数 ifm_open_file()を用いて，ファイルID をオープンする。
ファイルディスクリプタとして相対番号が返されることを確認する。

2) 指定ボリュームがマウントされていない場合はエラーとすること。

3) 指定ファイルIDが存在しない場合はエラーとすること。

4) 指定ファイルIDが既にオープンされていた場合はエラーとすること。

5) 指定ファイルIDが仮削除の場合エラーとすること。

6) オープンされているファイル数が既に限界値に達している場合，エラーとする。

（２）戻り値に関する検査

1) 正常終了した場合には、0が返されることを確認する。

2) 存在しないユニット番号を指定した場合は、ENUNITが返されることを確認する。

3) ボリュームがマウントされていない場合，ENMNTDが返されることを確認する。

4) ファイルIDが存在しない場合，EFLNEXが返されることを確認する。

5) 既にオープンされている場合，EOPNEDが返されることを確認する。

6) ファイルＩＤに仮削除ファイルを指定した場合、ENPERMが返されることを確認する。

7) オープンされているファイル数が既に限界値に達している場合，ENBUFFが返されることを確認する。

8) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

4．5．2　ファイルをクローズする

```
long  ifm_close_file(fd)
long  fd ;
```

機能

ファイルディスクリプタを指定して、ファイルをクローズする。

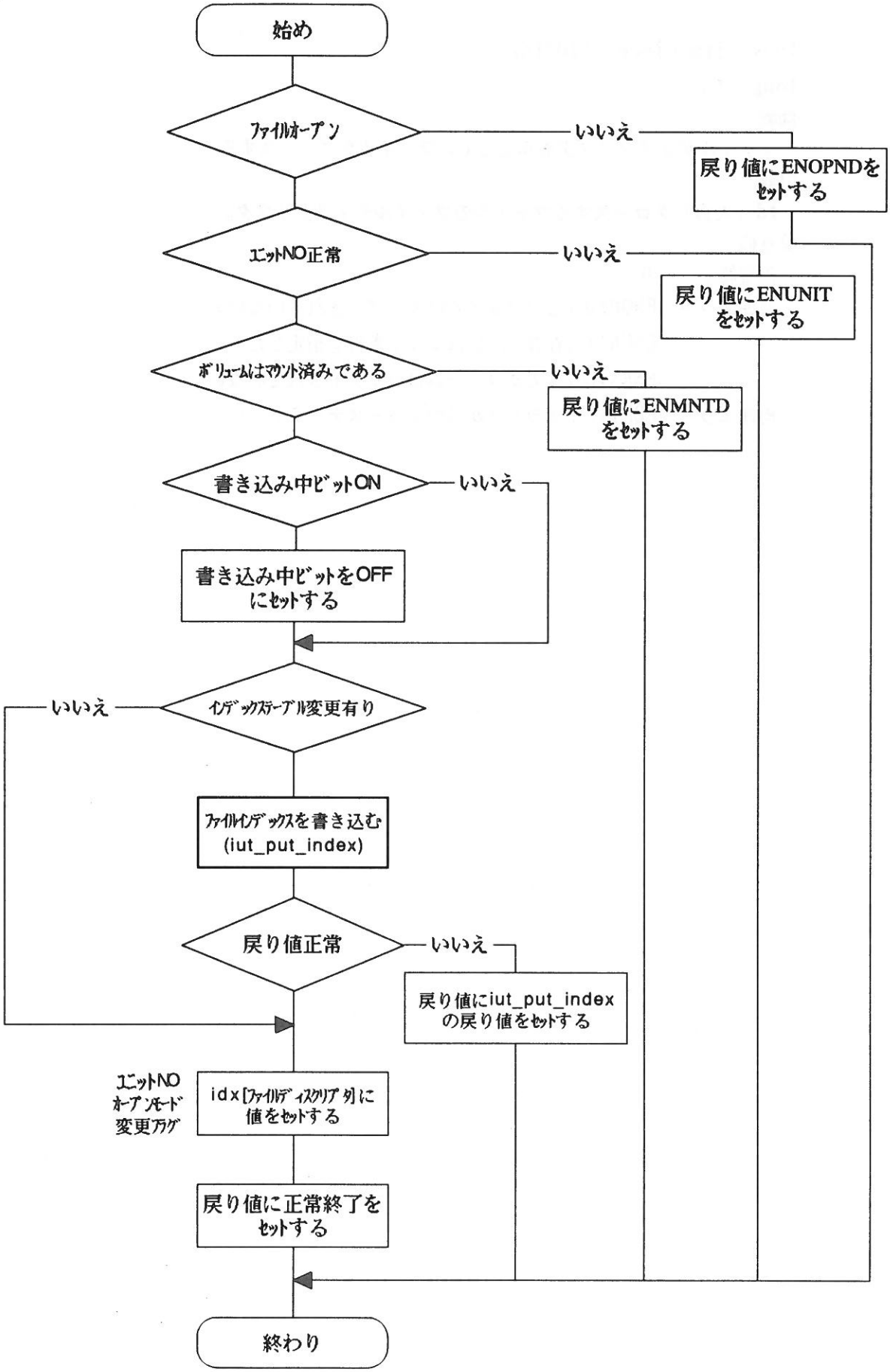
引数

fd（入力）クローズするファイルのファイルディスクリプタ。

戻り値

正常終了	0
論理エラー	ENOPND（このファイルはオープンされていない）
	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_close_file



関数ifm_close_file() の検査指針

関数 ifm_close_file() はファイルを予めオープンしているものに対し，クローズを行うものである。よって本関数の動作を確認する以前にファイルオープン関数ifm_open_file() の動作確認をしておかなければならない。

（1）機能に関する検査

1) 関数ifm_close_file()を用いて，ファイルをクローズする。
ファイルディスクリプタにてidx から該当するファイルIDを未使用にすることを確認する。

2) ファイルIDが存在しない（オープンしていない）場合は，エラーとすること。

（2）戻り値に関する検査

1) 正常終了した場合には、0 が返されることを確認する。

2) 存在しないユニット番号を指定した場合は、ENUNITが返されることを確認する。

3) マウントされていないボリュームを指定した場合は、ENMNTDが返されることを確認する。

4) オープンされていない場合、ENOPNDが返されることを確認する。

5) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

４．５．３　データを読み込む

```
long  ifm_read_data(fd, buf, st, req_len, act_len, swp)
long  fd ;
char  *buf ;
long  st ;
long  req_len ;
long  *act_len ;
long  swp;
```

機能

ファイルの先頭からの位置を指定して、ファイルからデータを読み込む。

引数

fd　　（入力）ファイルディスクリプタ。

buf　　（出力）データを格納するバッファへのポインタ。

st　　（入力）読み込みの開始位置をファイルの先頭からのバイト数で表した値。1024の非負の整数倍で指定する。

req_len（入力）読み込むデータのサイズをバイト単位で表した値。1024の正の整数倍で指定する。

act_len（出力）実際に読み込んだデータのサイズをバイト単位で表した値へのポインタ。

swp　　（入力）バイトスワップを行うか否かのフラグ。

戻り値

正常終了　0

論理エラー　ENUNIT（存在しないユニット番号を指定した）
ENMNTD（このボリュームはマウントされていない）
EPARAM（関数のパラメタが不正である）
ENOPND（このファイルはオープンされていない）
ESZOVR（指定した開始位置がファイルのデータサイズ以上である）
ENPERM（指定したファイルは書き込み禁止ファイルである）
ELOGRD（読み込み時に論理エラーが発生した）

物理エラー　EDRIVE｜ドライブからのエラーステータス

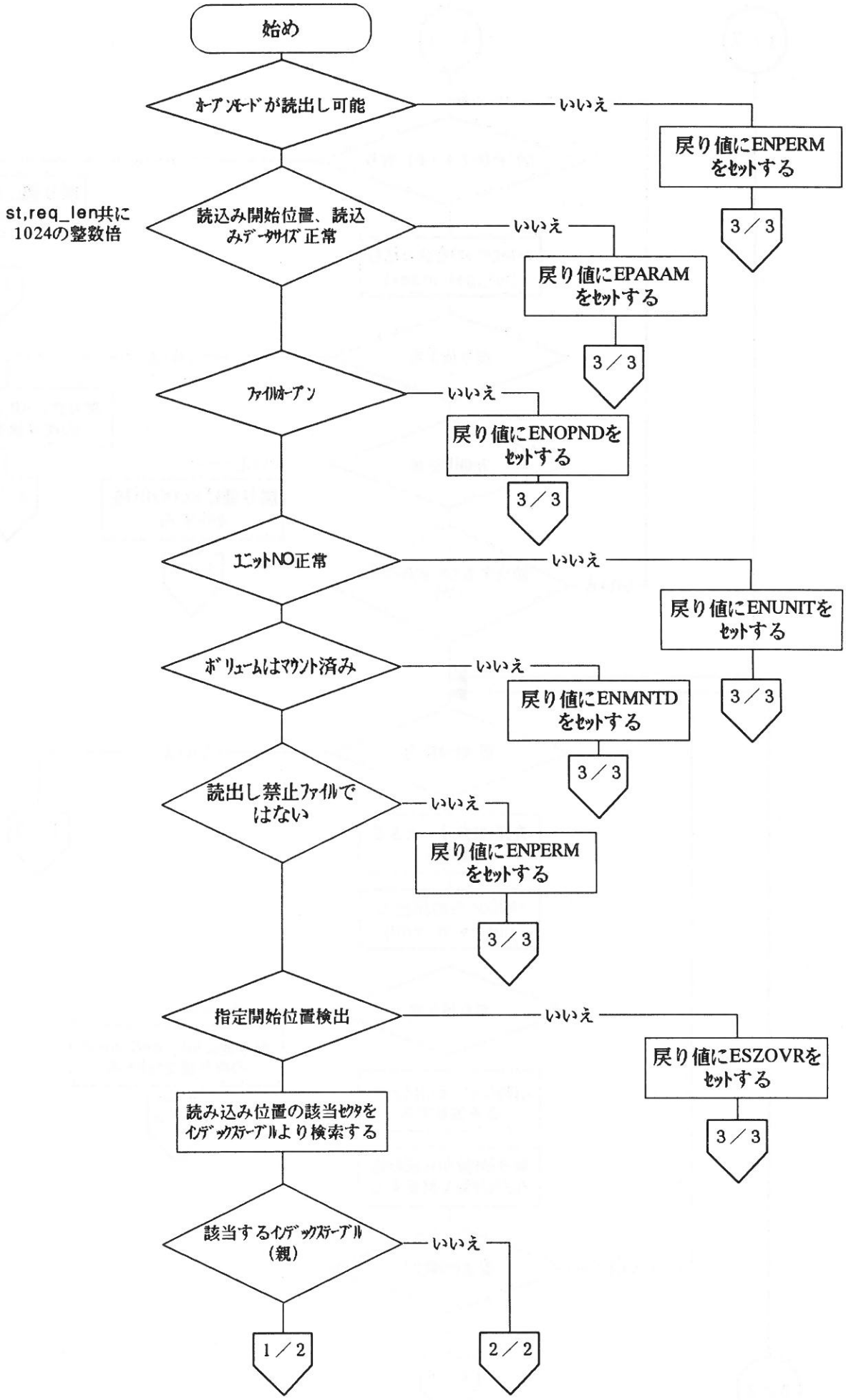
備考

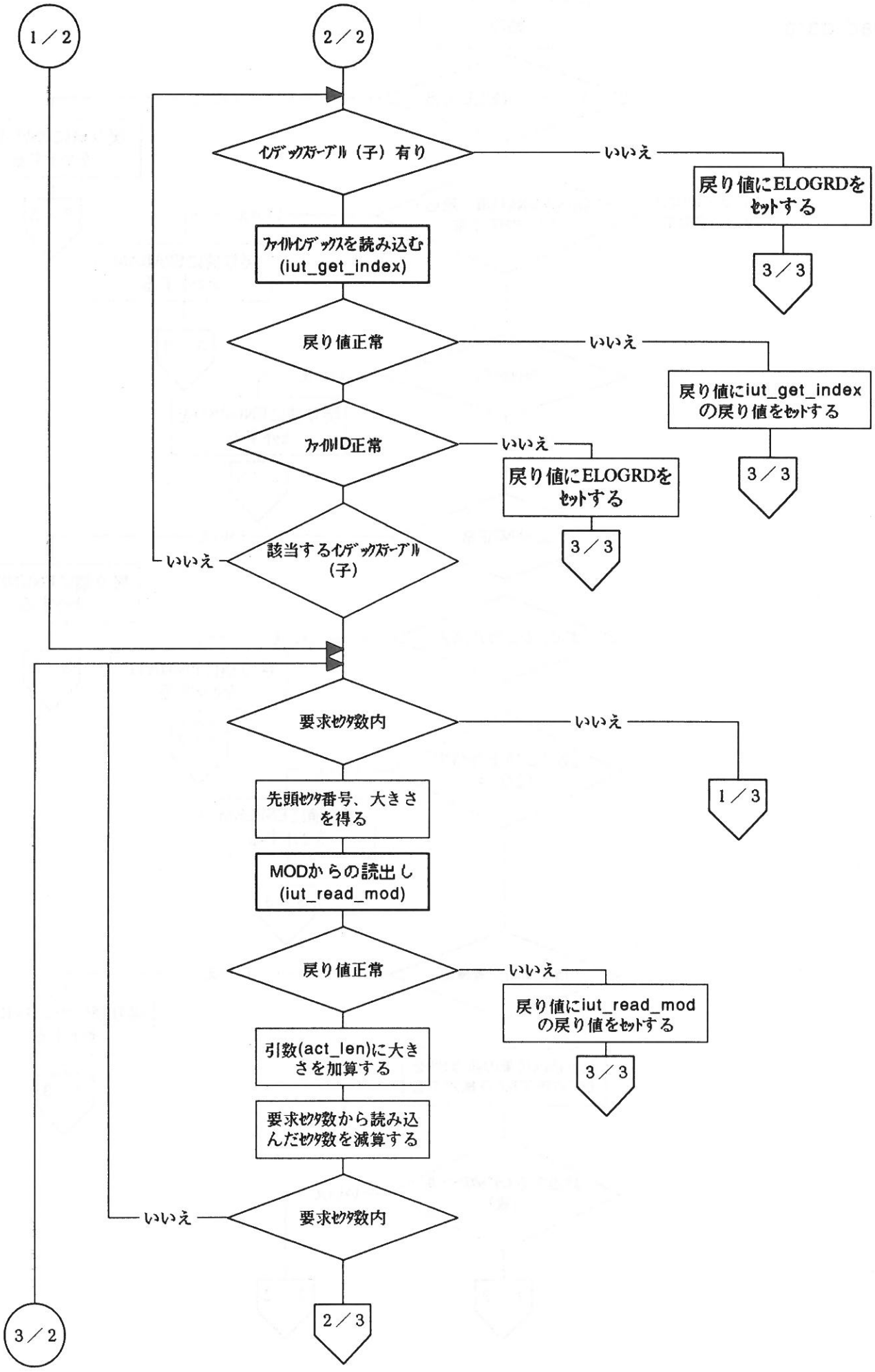
・req_lenの値がファイルのデータサイズよりも大きい場合には、stの位置からデータサイズまでのデータをバッファに読み込んで正常終了する。この場合、bufのact_len+1バイト目以降の内容は意味をもたない。

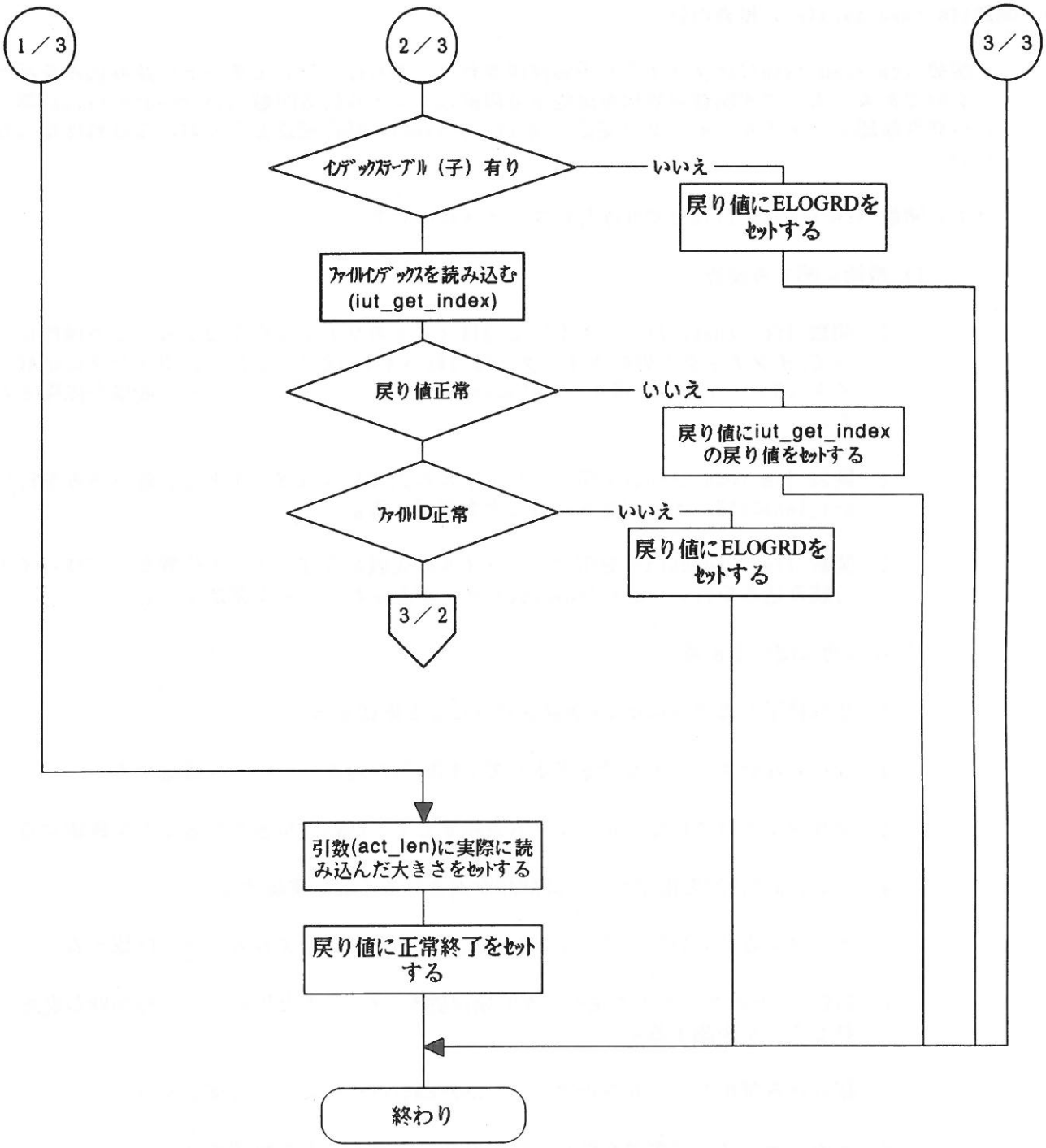
・req_lenが１０２４の整数倍でない場合に切り上げられる。従ってデータを格納しているバッファは、切り上げた分の大きさがなければならない。

・エラーが発生した場合には、buf, act_lenの内容は意味をもたない。

ifm_read-data




1／2
2／2
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
戻り値にELOGRDをｾｯﾄする
3／3
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む
(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をｾｯﾄする
3／3
ﾌｧｲﾙID正常
いいえ
戻り値にELOGRDをｾｯﾄする
3／3
該当するｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ(子)
いいえ
要求ｾｸﾀ数内
いいえ
1／3
先頭ｾｸﾀ番号、大きさを得る
MODからの読出し
(iut_read_mod)
戻り値正常
いいえ
戻り値にiut_read_modの戻り値をｾｯﾄする
3／3
引数(act_len)に大きさを加算する
要求ｾｸﾀ数から読み込んだｾｸﾀ数を減算する
要求ｾｸﾀ数内
いいえ
2／3
3／2


1／3
2／3
3／3
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
戻り値にELOGRDをｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をｾｯﾄする
ﾌｧｲﾙID正常
いいえ
戻り値にELOGRDをｾｯﾄする
3／2
引数(act_len)に実際に読み込んだ大きさをｾｯﾄする
戻り値に正常終了をｾｯﾄする
終わり

関数ifm_read_data() の検査指針

関数 ifm_read_data()はファイルが予め作成されているものに対してデータの読み込みを行うものである。よって本関数の動作を確認する以前にファイル作成関数 ifm_create_file()等での動作確認、ファイルのオープン関数 ifm_open_file()の動作確認をしておかなければならない。

（1）関数 ifm_create_file()で作成したファイルについて

1) 機能に関する検査

1. 関数 ifm_create_file()を用いて 21Kバイトのファイルを作成する。この操作によって,インデックスの#1ポインタには 16Kバイト（Eゾーン）、#2ポインタには4Kバイト（Dゾーン）、#3ポインタには1Kバイト（Cゾーン）の各データ領域が確保される。

2. 関数 ifm_read_data()を用いてファイルの先頭から 21Kバイト分，読み込みを行い，act_lenに21Kバイトが返されることを確認する。

3. 関数 ifm_read_data()を用いてファイルの先頭から1Kバイトの位置から 21Kバイト分読み込みを行い，act_lenに20Kバイトが返されることを確認する。

2) 戻り値に関する検査

1. 正常終了した場合には、0が返されることを確認する。

2. 存在しないユニット番号を指定して、ENUNITが返されることを確認する。

3. マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。

4. req_lenに1023を指定して、EPARAMが返されることを確認する。

5. オープンされていないファイルを指定してENOPNDが返されることを確認する。

6. 21Kバイトのファイルに読み込み開始位置を22Kバイトと指定して、ESZOVRが返されることを確認する。

7. 読み込み禁止ファイルを指定して、ENPERMが返されることを確認する。

8. 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

（２）関数 ifm_create_file_variable() で作成したファイルについて

1）機能に関する検査

1. 関数ifm_write_data()を用いてファイルの先頭から0xAAを3Kバイト分書き込む。
#1ポインタにはデータ領域（Dゾーン ： 4K）が確保されること
インデックステーブル内のファイルバイト長は4Kが格納されていること。

2. 関数ifm_read_data()を用いてファイルの先頭から3Kバイト分読み込み，act_lenに3Kバイトが返されることを確認する。

2）戻り値に関する検査

1. 正常終了した場合には、0が返されることを確認する。

2. 存在しないユニット番号を指定して、ENUNITが返されることを確認する。

3. マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。

4. req_lenに1023を指定して、EPARAMが返されることを確認する。

5. オープンされていないファイルを指定してENOPNDが返されることを確認する。

6. 関数ifm_write_data()を用いてファイルの先頭から0xAAを3Kバイト分書き込み
#1ポインタにはデータ領域（Dゾーン ： 4K）が確保されている
インデックステーブル内のファイルバイト長は3Kが格納されている。
このファイルに対して，読み込み開始位置をファイルの先頭から3Kバイト，サイズを1Kバイトとした場合，ESZOVRが返されることを確認する。

7. 読み込み禁止ファイルを指定して、ENPERMが返されることを確認する。

8. インデックステーブル（子）が存在するが、子のインデックステーブル番号が存在しない場合は、ELOGRDが返されることを確認する。

9. 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

４．５．４　データを書き込む

```
long  ifm_write_data(fd, buf, st, req_len, act_len, swp)
long  fd ;
char  *buf ;
long  st ;
long  req_len ;
long  *act_len ;
long  swp ;
```

機能

ファイルの先頭からの位置を指定して、ファイルにデータを書き込む。

引数

fd　　（入力）ファイルディスクリプタ。

buf　　（入力）データが格納されているバッファへのポインタ。

st　　（入力）書き込みの開始位置をファイルの先頭からのバイト数で表した値。1024の非負の整数倍で指定する。

req_len（入力）書き込むデータのサイズをバイト単位で表した値。1024の正の整数倍で指定する。

act_len（出力）実際に書き込んだデータのサイズをバイト単位で表した値へのポインタ。

swp　　（入力）２バイトデータの上位下位の反転。１：反転する、０：反転しない。

戻り値

正常終了　０

論理エラー　ENUNIT（存在しないユニット番号を指定した）
ENMNTD（このボリュームはマウントされていない）
EPARAM（関数のパラメタが不正である）
ENOPND（このファイルはオープンされていない）
ENPERM（このファイルは書き込み禁止ファイルである）
ESZOVR（指定した開始位置がファイルのデータサイズ以上である）
ELOGWR（書込み時に論理エラーが発生した）

物理エラー　EDRIVE｜ドライブからのエラーステータス

備考１（関数ifm_create_file()で作成したファイルについて）

・stで指定した開始位置がファイルのデータサイズ以上である場合には、エラーコードESZOVRを返す。

・st＋req_lenの値がファイルのデータサイズよりも大きい場合には、stの位置からデータサイズまでのデータをファイルに書き込んで正常終了する。

・req_lenが１０２４の整数倍でない場合に切り上げられる。従ってデータを格納しているバッファは、切り上げた分の大きさがなければならない。

・エラーが発生した場合には、act_lenの内容は意味をもたない。

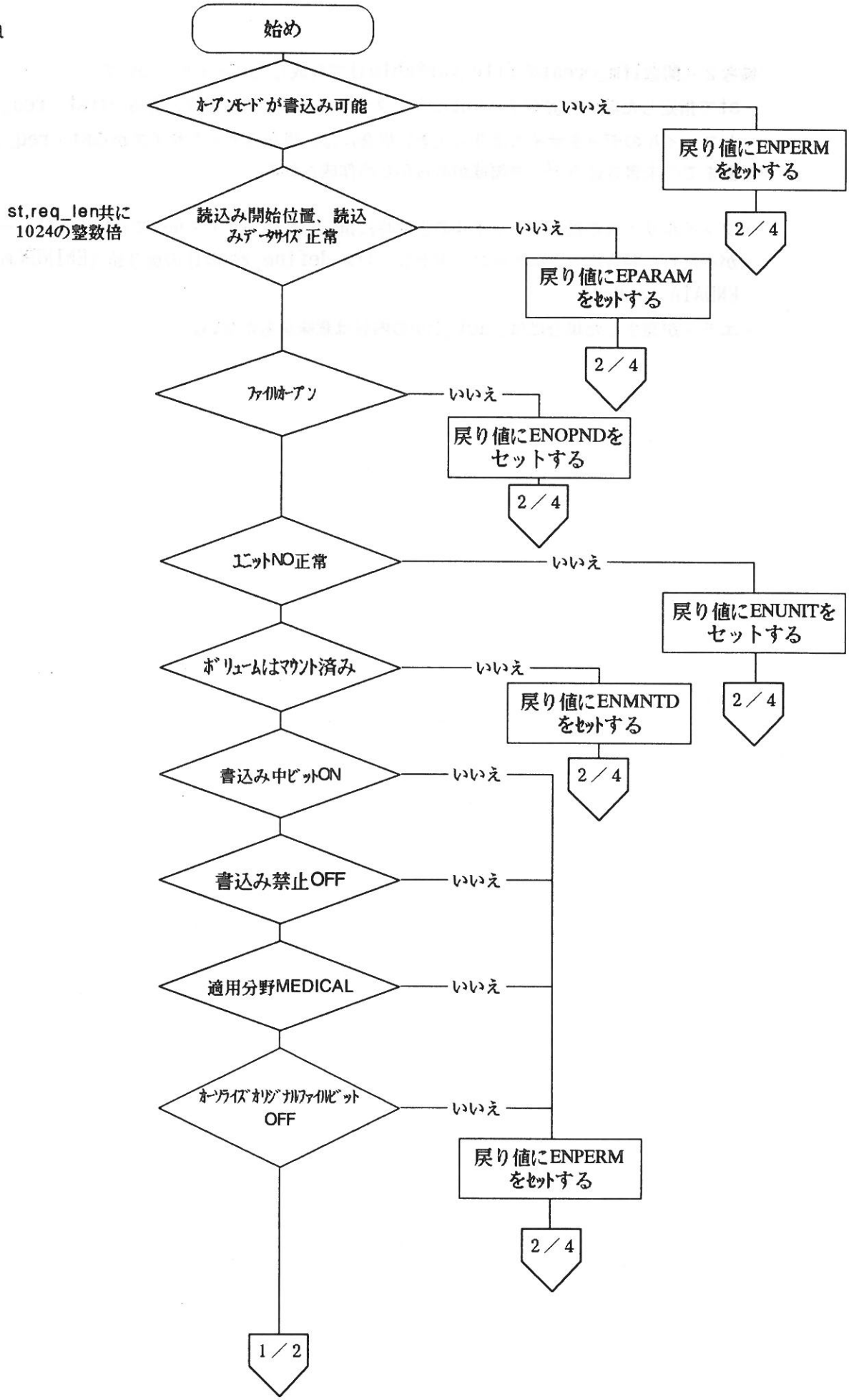
備考２（関数ifm_create_file_variable()で作成したファイルについて）

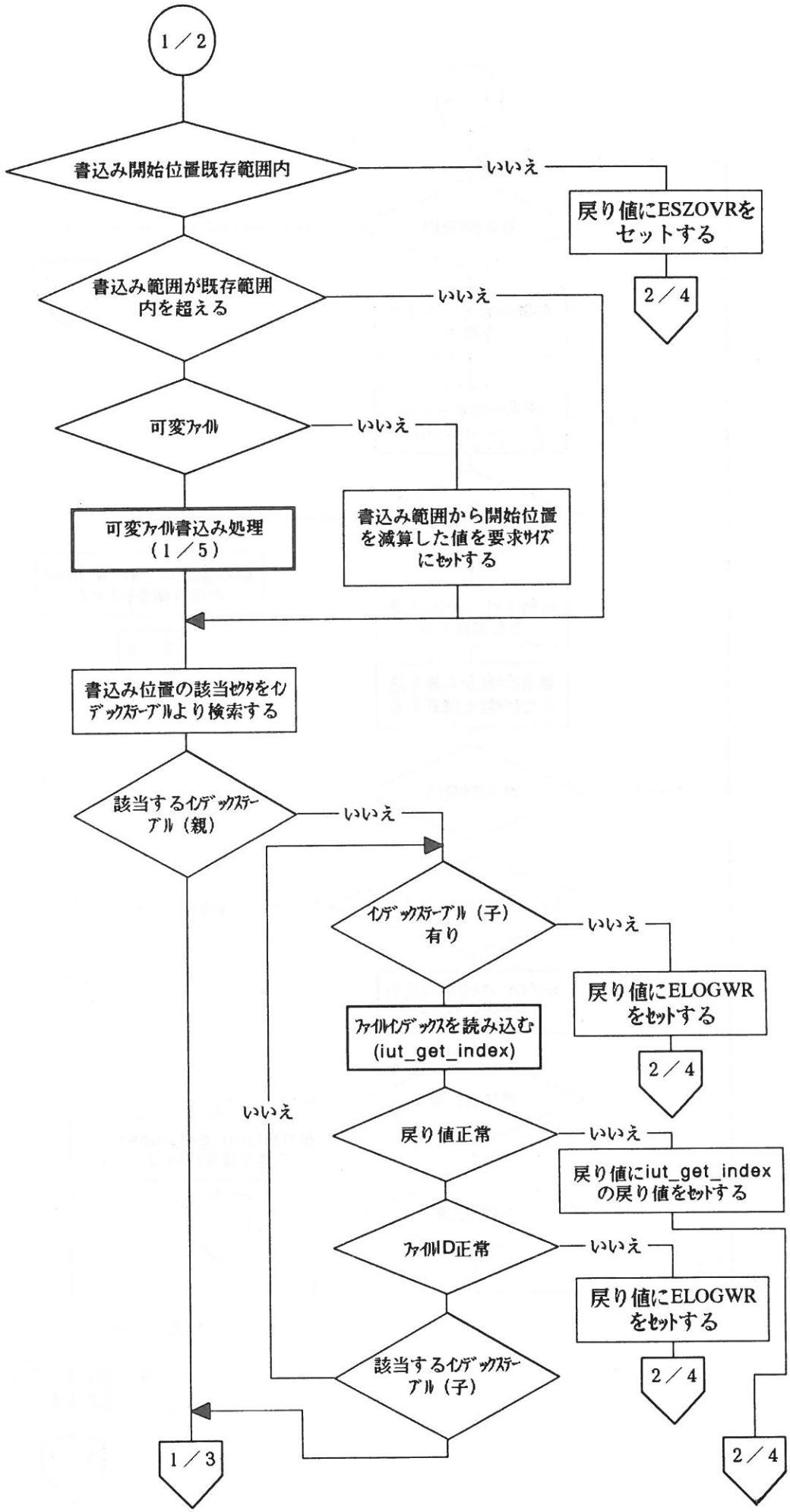
・stで指定した開始位置がファイルのデータサイズ以上である場合、あるいはst＋req_lenの値がファイルのデータサイズよりも大きい場合には、現在のデータサイズからst＋req_lenの位置までの未書き込みデータ領域があらかじめ作成される。

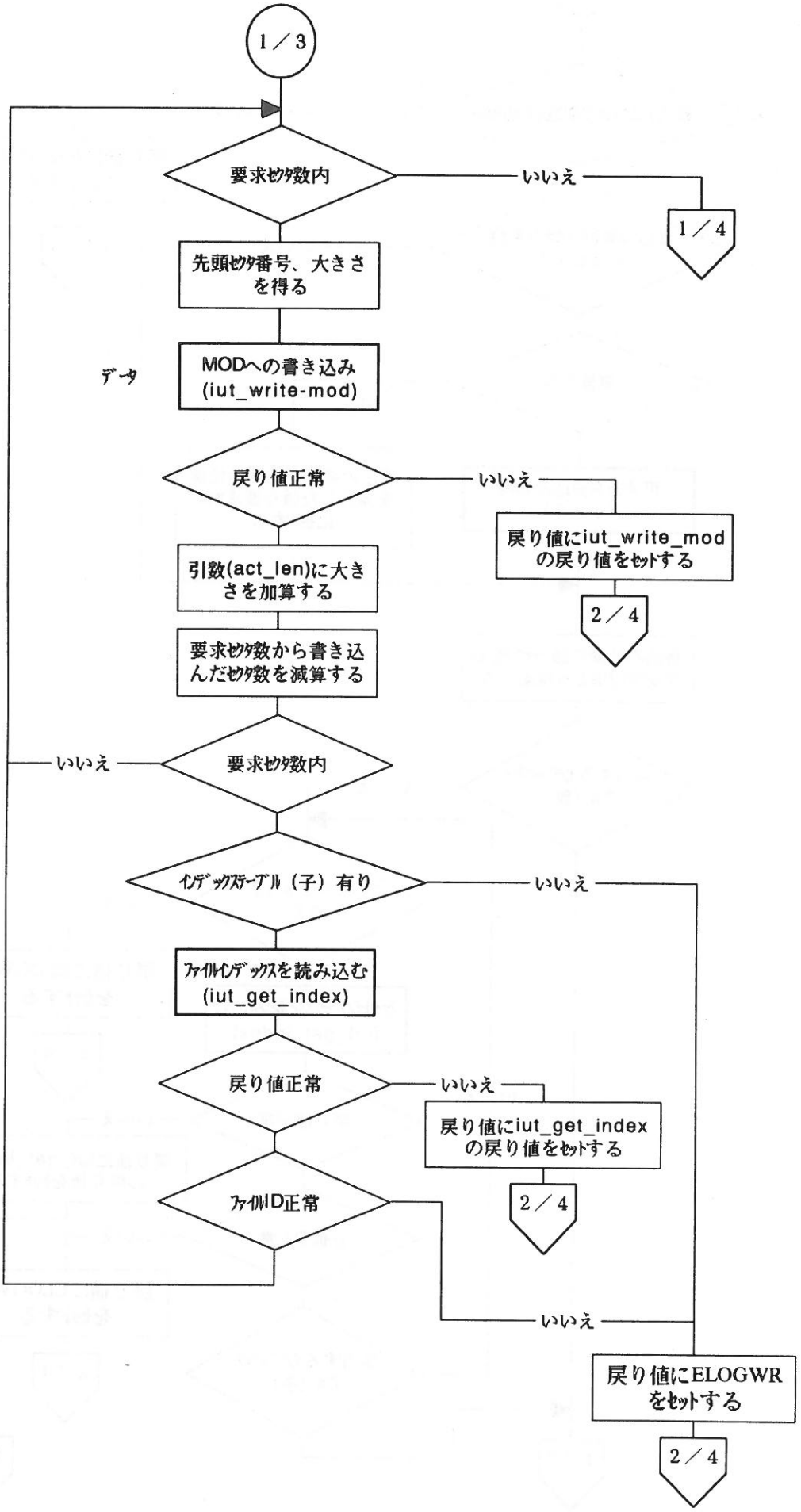
・ファイルサイズの拡張はファイルの生成時に指定したセルサイズ単位で行い、ボリュームの容量が不足した場合には、エラーコードとしてifm_define_zone()の戻り値（ENINDXあるいはENDATA）を返す。

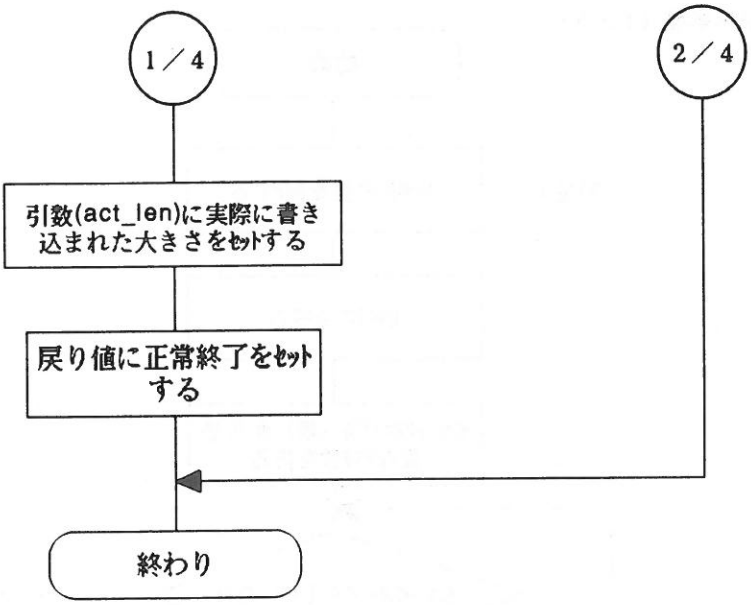
・エラーが発生した場合には、act_lenの内容は意味をもたない。

ifm_write_data

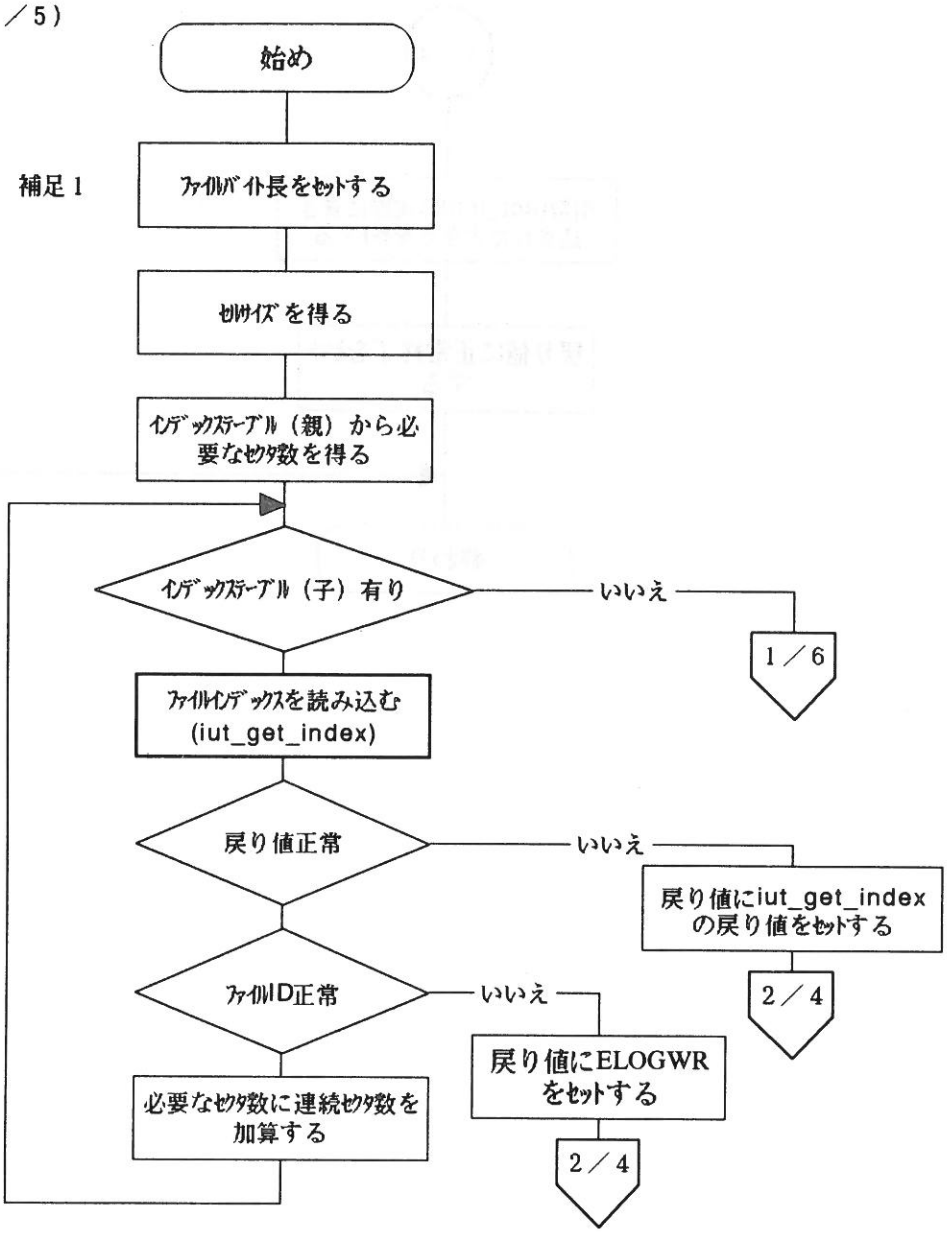


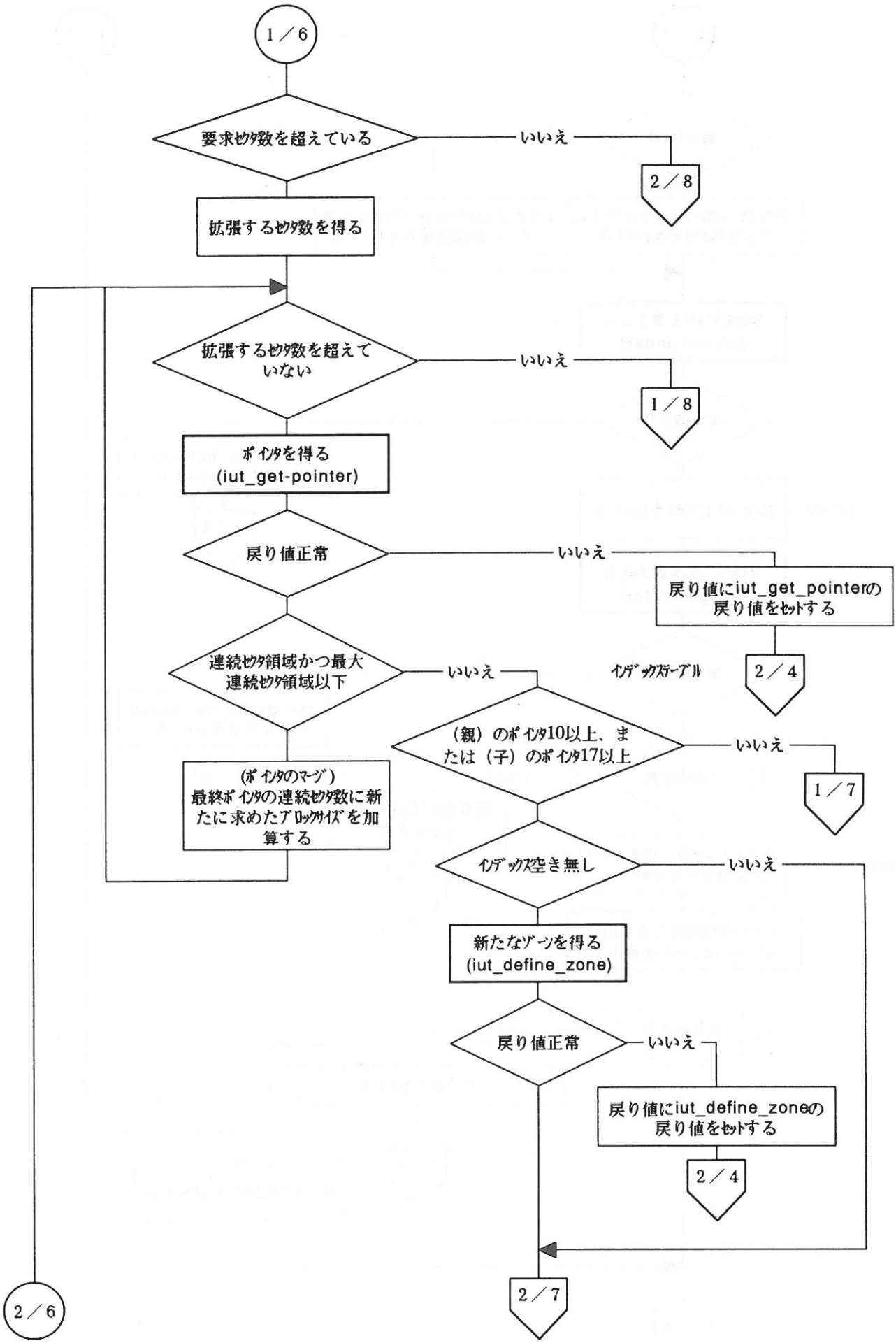

1／2
書込み開始位置既存範囲内
いいえ
戻り値にESZOVRをセットする
2／4
書込み範囲が既存範囲内を超える
いいえ
可変ﾌｧｲﾙ
いいえ
可変ﾌｧｲﾙ書込み処理（1／5）
書込み範囲から開始位置を減算した値を要求ｻｲｽﾞにｾｯﾄする
書込み位置の該当ｾｸﾀをｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙより検索する
該当するｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）
いいえ
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
戻り値にELOGWRをｾｯﾄする
2／4
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む(iut_get_index)
いいえ
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をｾｯﾄする
ﾌｧｲﾙID正常
いいえ
戻り値にELOGWRをｾｯﾄする
2／4
該当するｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）
1／3
2／4

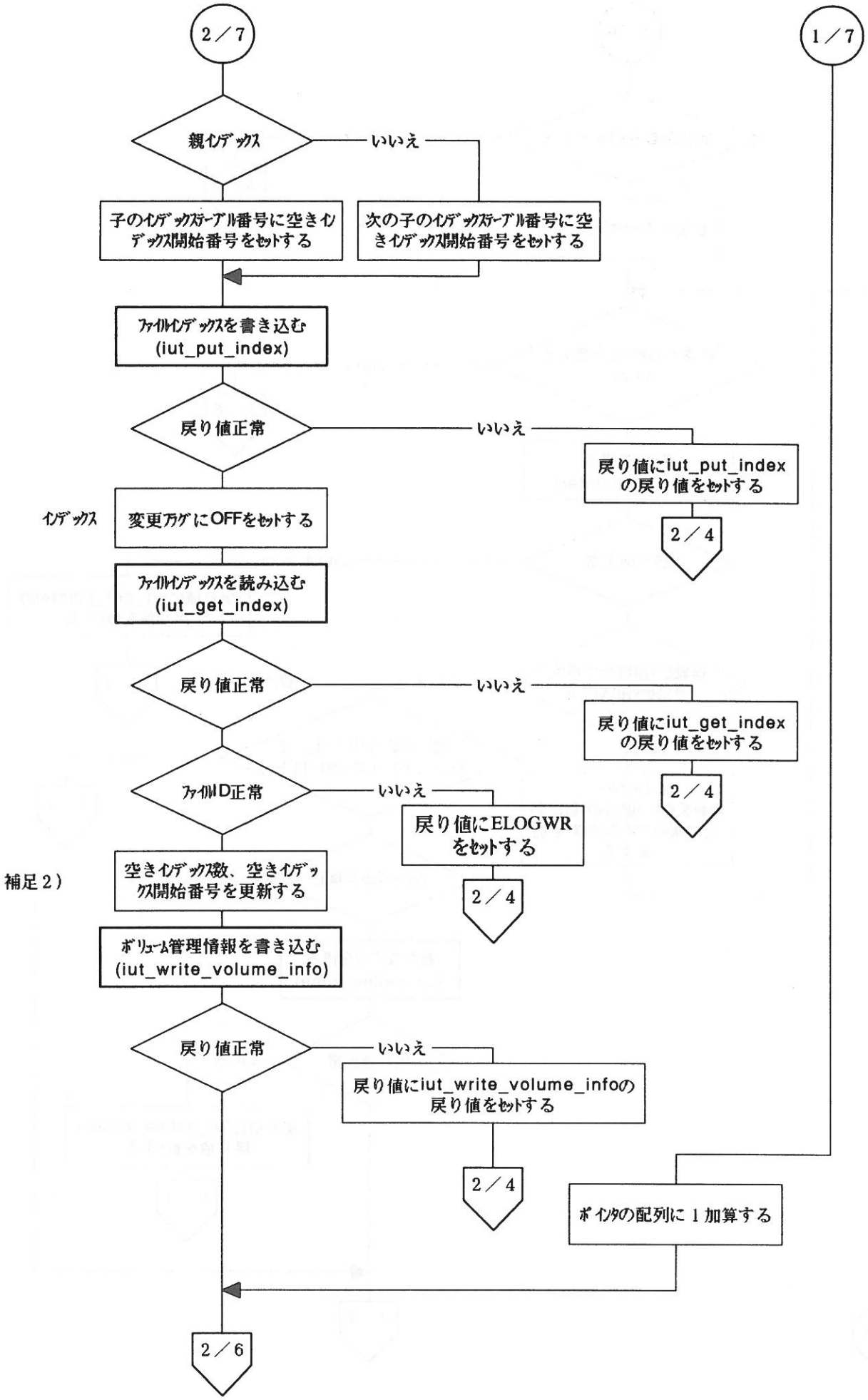

1／3
要求ｾｸﾀ数内
いいえ
1／4
先頭ｾｸﾀ番号、大きさを得る
データ
MODへの書き込み (iut_write-mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をｾｯﾄする
2／4
引数(act_len)に大きさを加算する
要求ｾｸﾀ数から書き込んだｾｸﾀ数を減算する
いいえ
要求ｾｸﾀ数内
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（子）有り
いいえ
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む (iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をｾｯﾄする
2／4
ﾌｧｲﾙID正常
いいえ
戻り値にELOGWRをｾｯﾄする
2／4

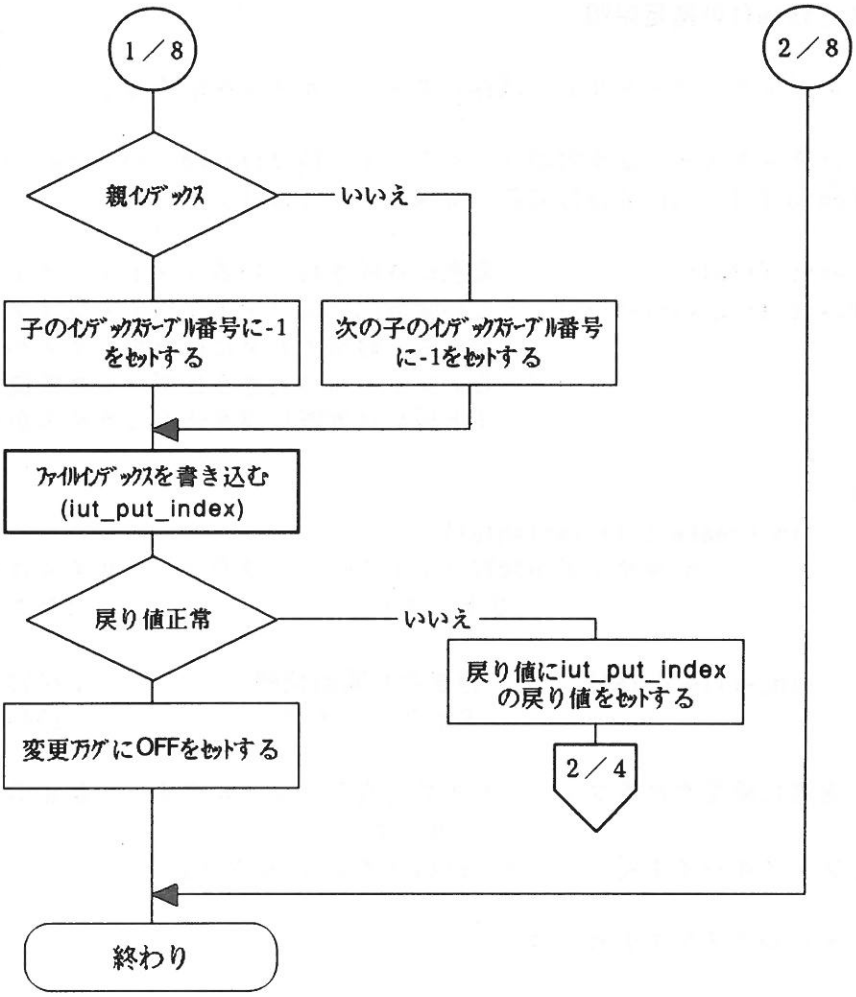

1／4
2／4
引数(act_len)に実際に書き込まれた大きさをｾｯﾄする
戻り値に正常終了をｾｯﾄする
終わり

可変ファイル書込み処理（1／5）




1／6
要求ｾｸﾀ数を超えている
いいえ
2／8
拡張するｾｸﾀ数を得る
拡張するｾｸﾀ数を超えて
いない
いいえ
1／8
ﾎﾟｲﾝﾀを得る
(iut_get-pointer)
戻り値正常
いいえ
戻り値にiut_get_pointerの
戻り値をｾｯﾄする
連続ｾｸﾀ領域かつ最大
連続ｾｸﾀ領域以下
いいえ
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ
2／4
(ﾎﾟｲﾝﾀのﾏｰｼﾞ)
最終ﾎﾟｲﾝﾀの連続ｾｸﾀ数に新
たに求めたﾌﾞﾛｯｸｻｲｽﾞを加
算する
（親）のﾎﾟｲﾝﾀ10以上、ま
たは（子）のﾎﾟｲﾝﾀ17以上
いいえ
1／7
ｲﾝﾃﾞｯｸｽ空き無し
いいえ
新たなｿﾞｰﾝを得る
(iut_define_zone)
戻り値正常
いいえ
戻り値にiut_define_zoneの
戻り値をｾｯﾄする
2／4
2／7
2／6


2／7
親ｲﾝﾃﾞｯｸｽ
いいえ
子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号に空きｲﾝﾃﾞｯｸｽ開始番号をｾｯﾄする
次の子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号に空きｲﾝﾃﾞｯｸｽ開始番号をｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をｾｯﾄする
2／4
ｲﾝﾃﾞｯｸｽ
変更ﾌﾗｸﾞにOFFをｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを読み込む
(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_index
の戻り値をｾｯﾄする
2／4
ﾌｧｲﾙID正常
いいえ
戻り値にELOGWR
をｾｯﾄする
2／4
補足２）
空きｲﾝﾃﾞｯｸｽ数、空きｲﾝﾃﾞｯｸｽ開始番号を更新する
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をｾｯﾄする
2／4
1／7
ﾎﾟｲﾝﾀの配列に１加算する
2／6


1／8
2／8
親ｲﾝﾃﾞｯｸｽ
いいえ
子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号に-1をｾｯﾄする
次の子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号に-1をｾｯﾄする
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_indexの戻り値をｾｯﾄする
2／4
変更ﾌﾗｸﾞにOFFをｾｯﾄする
終わり

関数ifm_write_data()の補足説明

補足１）インデックステーブルより既存のファイルサイズを求める。

親のインデックステーブル内のファイルバイト長はifm_create_file()と
ifm_create_file_variable()で若干意味合いが異なる。

ifm_create_file()　　：実際に作成されているファイルのサイズ。
ifm_create_file_variable()　：この関数内ではファイルバイト長はブロックサイズがセットされる。#1ポインタにはセルサイズがセットされるだけである。ファイルの大きさはブロック単位で増えるがファイルバイト長には実際に書き込んだサイズがセットされる。

例）

ifm_create_file_variable()
セルサイズが3072バイトの時 ： 該当ゾーンサイズはDゾーン
（３セクタ）　　（4セクタ）

ifm_write_data()　：書きだし開始位置　　：4096バイト（5セクタ目）
書きだしサイズ　　：2048バイト（2セクタ）

実際に確保されるファイルサイズ（既存ファイルサイズ：＃ｎポインタのサイズ）
：　8セクタ
ファイルバイト長　　：　6144バイト（6セクタ）

既存ファイルサイズの求め方は

1)　親のインデックステーブルのファイルバイト長を調べ、これがゼロなら既存のファイルサイズもゼロである。

2)　ファイルバイト長がゼロ以外なら＃１ポインタのサイズより該当するゾーン種別を求め１増加単位のセクタ数を得る。次にファイルバイト長が納まる増加単位の最小整数倍のセクタサイズが既存のファイルサイズである。

補足２）　ボリューム管理情報部の更新

1)　インデックステーブルを新たに作成した場合、空きインデックス数は作成インデックス数を引いた値、空きインデックス開始番号は作成インデックス数を足した値にする。

関数ifm_write_data() の検査指針

関数 ifm_write_data() はファイルが予め作成されているものに対してデータの書き込みを行うものである。よって本関数の動作を確認する以前にファイル作成関数 ifm_create_file()又はifm_create_file_variable()の動作確認、ファイルのオープン関数 ifm_open_file()の動作確認をしておかなければならない

（1）関数 ifm_create_file() で作成したファイルについて

1) 機能に関する検査

1. 関数ifm_create_file()を用いて21Kバイトのファイルを作成する。この操作によって,インデックスの#1ポインタには 16Kバイト（Eゾーン）、#2ポインタには4Kバイト（Dゾーン）、#3ポインタには1Kバイト（Cゾーン）の各データ領域が確保される。

2. 関数 ifm_write_data()を用いてファイルの先頭から0xAAを21Kバイト分書き込み、act_lenに21Kバイトが返され、上記の３つのデータ領域がすべて0xAAで満たされていることを確認する。

3. 関数ifm_write_data()を用いてファイルの先頭から1Kバイトの位置からデータ0x55を16Kバイト分書き込み、Eゾーンの先頭から 1Kバイトの位置から15Kバイト分とDゾーンの先頭から 1Kバイト分が0x55で満たされ、その他の領域は0xAAのままであることを確認する

4. 関数ifm_write_data()を用いてファイルの先頭から1Kバイトの位置からデータ0x55を21Kバイト分書き込み、Eゾーンの先頭から1Kバイトの位置からCゾーン全てが0x55で満たされ、act_lenは20Kがセットされることを確認する。

2) 戻り値に関する検査

1. 正常終了した場合には、0が返されることを確認する。
2. 存在しないユニット番号を指定して、ENUNITが返されることを確認する。
3. マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。
4. stに1025を指定して、EPARAMが返されることを確認する。
5. オープンされていないファイルを指定してENOPNDが返されることを確認する。
6. 書き込み禁止ファイルを指定して、ENPERMが返されることを確認する。
7. 作成ファイルサイズが21Kバイトのファイルに対し，書き込み開始位置を22Kバイト目と指定して、ESZOVRが返されることを確認する。
8. 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

（２）関数 ifm_create_file_variable() で作成したファイルについて

1）機能に関する検査

1. 関数ifm_create_file_variable()を用いて3Kバイトのファイルを作成する。この操作によって,インデックスの#1ポインタには3Kバイト（Dゾーン）のサイズのみが格納される。

2. 関数ifm_write_data()を用いてファイルの先頭から0xAAを3Kバイト分書き込む。act_lenに 3Kバイトが返され、#1ポインタにはデータ領域（Dゾーン : 4K）が確保されること,またその内容は先頭から0xAAで満たされていることを確認する。
インデックステーブル内のファイルバイト長は3Kが格納されていることを確認する。

3. 関数ifm_write_data()を用いてファイルの先頭から4Kバイトの位置からデータ0x55を2Kバイト分書き込み， act_lenに2Kバイトがかえされることを確認する。
#2ポインタにはデータ領域（Dゾーン）が確保され先頭から2Kバイト分だけが0x55で満たされることを確認する
インデックステーブル内のファイルバイト長は6Kが格納されていることを確認する。

4. 関数ifm_write_data()を用いてファイルの先頭から6Kバイトの位置からデータ0x55を 1Kバイト分書き込み，act_lenに1Kバイトがかえされることを確認する。
#2ポインタにて示されるデータ領域（Dゾーン）は，先頭から 3Kバイト分だけ0x55で満たされることを確認する。
インデックステーブル内のファイルバイト長は7Kが格納されていることを確認する。

5. 関数ifm_write_data()を用いてファイルの先頭の位置からデータ0x55を8Kバイト分書き込み、2つのDゾーンは全てが0x55で満たされ、 act_lenは8Kがセットされることを確認する。また#1,#2ポインタには，なんら変化がないことを確認する。
インデックステーブル内のファイルバイト長は8Kが格納されていることを確認する。

2）戻り値に関する検査

1. 正常終了した場合には、0が返されることを確認する。
2. 存在しないユニット番号を指定して、ENUNITが返されることを確認する。
3. マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。
4. stに1025を指定して、EPARAMが返されることを確認する。
5. オープンされていないファイルを指定してENOPNDが返されることを確認する。
6. 書き込み禁止ファイルを指定して、ENPERMが返されることを確認する。
7. 未使用ゾーンも含め、新たな書き込みに必要なブロックの空きがない場合は、iut_define_zone()の戻り値（ENDATA）が返されることを確認する。
8. インデックス領域に空きがない状態で，インデックステーブルエリアが１個必要となる指定を行った時、 iut_define_zone()の戻り値（ENINDX）が返されることを確認する。
9. 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

4．5．5　ヘッダを読み込む

```
long  ifm_read_header(fd, buf, req_len, act_len)
long  fd ;
char  *buf ;
long  req_len ;
long  *act_len ;
```

機能

ファイルからヘッダを読み込む。

引数

fd	（入力）ファイルディスクリプタ。
buf	（出力）ヘッダを格納するバッファへのポインタ。
req_len	（入力）読み込むヘッダのサイズをバイト単位で表した値。
act_len	（出力）実際に読み込んだヘッダのサイズをバイト単位で表した値へのポインタ。

戻り値

正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ENOPND（このファイルはオープンされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	EMEDUM（ディスクの物理エラーが発生した）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

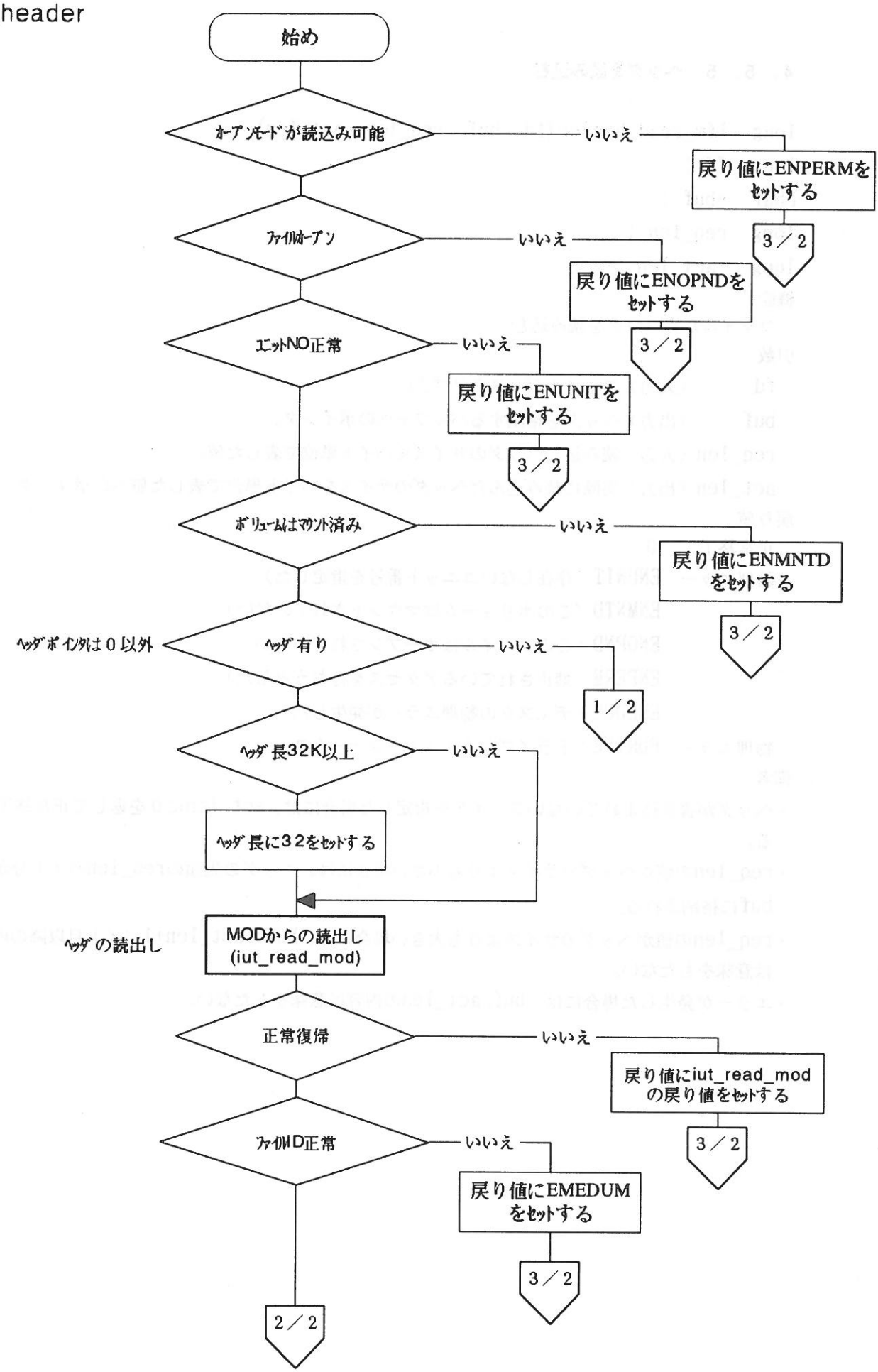
・ヘッダが書き込まれていないファイルを指定した場合には、act_lenに0を返して正常終了する。

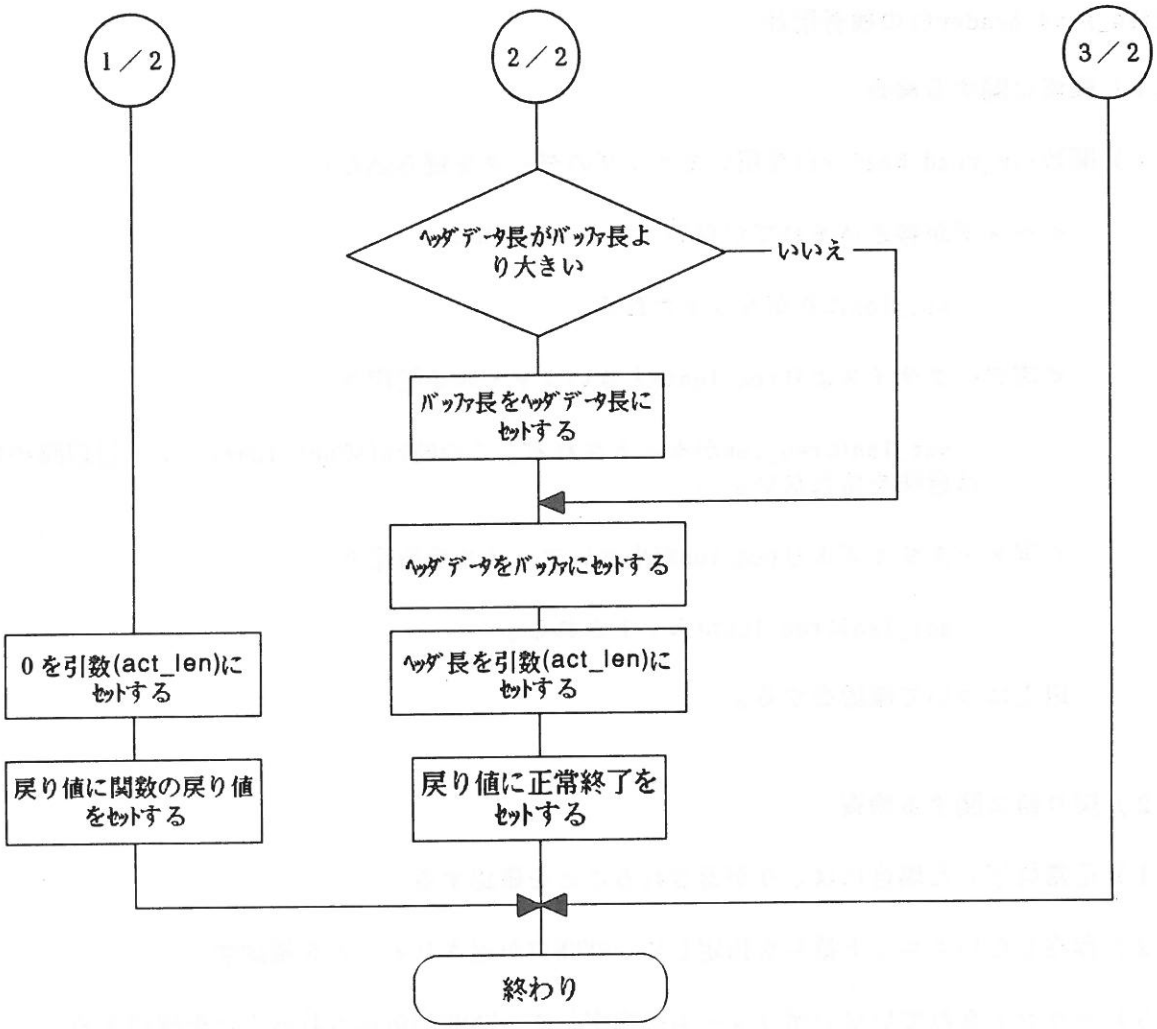
・req_lenの値がヘッダのサイズよりも小さい場合には、ヘッダの先頭のreq_lenバイト分がbufに格納される。

・req_lenの値がヘッダのサイズよりも大きい場合には、bufのact_len+1バイト目以降の内容は意味をもたない。

・エラーが発生した場合には、buf, act_lenの内容は意味をもたない。

ifm_read_header




1／2
2／2
3／2
ﾍｯﾀﾞﾃﾞｰﾀ長がﾊﾞｯﾌｧ長より大きい
いいえ
ﾊﾞｯﾌｧ長をﾍｯﾀﾞﾃﾞｰﾀ長にｾｯﾄする
ﾍｯﾀﾞﾃﾞｰﾀをﾊﾞｯﾌｧにｾｯﾄする
0を引数(act_len)にｾｯﾄする
ﾍｯﾀﾞ長を引数(act_len)にｾｯﾄする
戻り値に関数の戻り値をｾｯﾄする
戻り値に正常終了をｾｯﾄする
終わり

関数ifm_read_header()の検査指針

（１）機能に関する検査

１）関数ifm_read_header()を用いてヘッダのデータを読み込む。

＜ヘッダが書き込まれていないファイルを指定＞

act_lenに０がセットされる。

＜実データサイズよりreq_lenが大きいファイルを指定＞

act_lenにreq_lenがセットされる。この時bufのact_len+1バイト目以降の内容は意味を持たない。

＜実データサイズよりreq_lenが小さいファイルを指定＞

act_lenにreq_lenがセットされる。

以上について確認をする。

（２）戻り値に関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ENUNITが返されることを確認する。

３）マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。

４）オープンされていないファイルを指定して、ENOPNDが返されることを確認する。

５）読み込み禁止のファイルを指定して、ENPERMが返されることを確認する。

６）読み込まれたファイルＩＤがインデックステーブルのファイルＩＤと異なる場合は、EMEDUMが返されることを確認する。

７）光磁気ディスクの電源を切り、EDRIVEが返されることを確認する。

4．5．6　ヘッダを書き込む

```
long  ifm_write_header(fd, buf, req_len, act_len)
long  fd ;
char  *buf ;
long  req_len ;
long  *act_len ;
```

機能

ファイルにヘッダを書き込む。

引数

fd	（入力）ファイルディスクリプタ。
buf	（入力）ヘッダが格納されているバッファへのポインタ。
req_len	（入力）書き込むヘッダのサイズをバイト単位で表した値。
act_len	（出力）実際に書き込んだヘッダのサイズをバイト単位で表した値へのポインタ。

戻り値

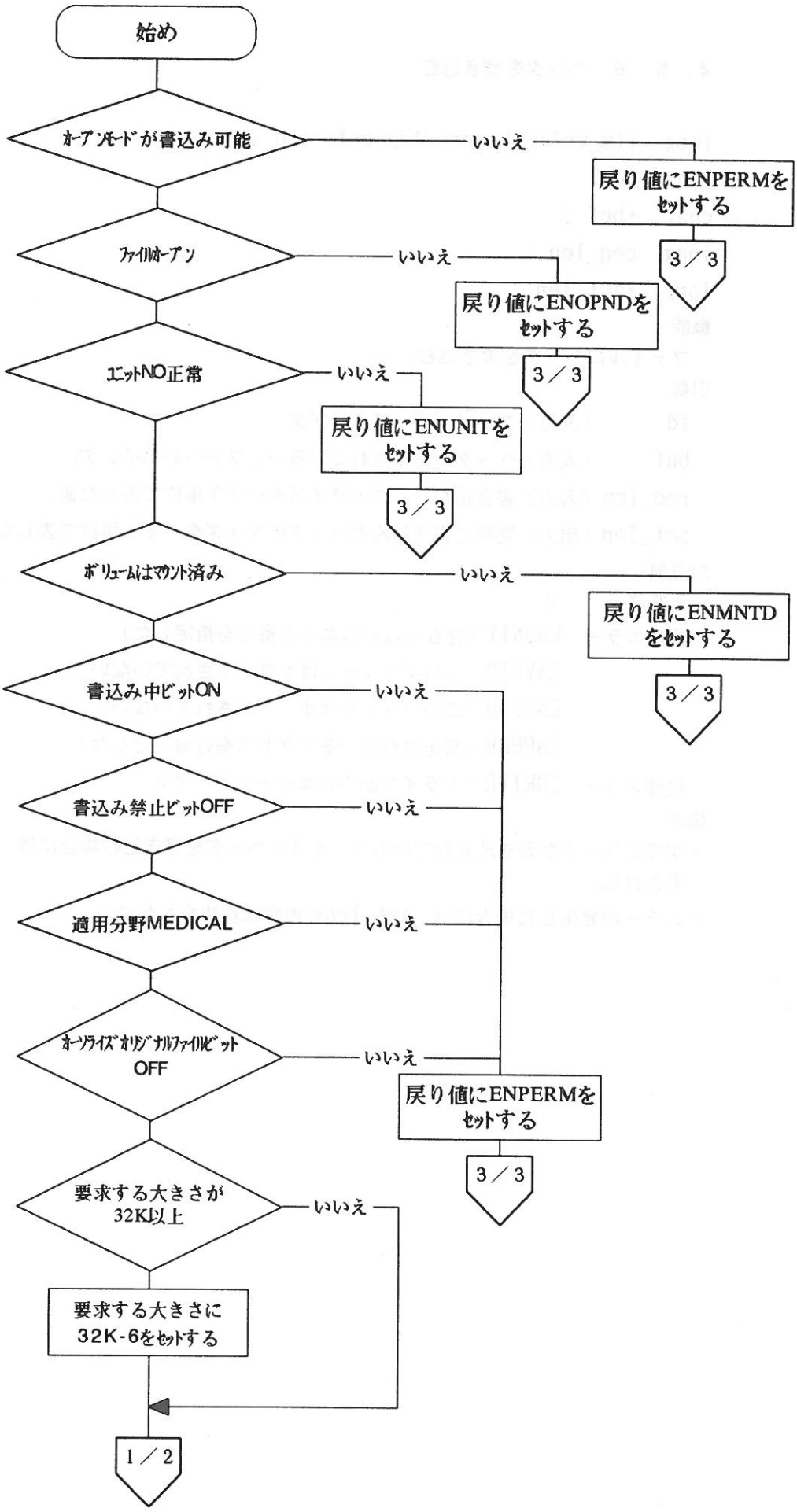
正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ENOPND（このファイルはオープンされていない）
	ENPERM（禁止されているアクセスを行おうとした）
物理エラー	EDRIVE｜ドライブからのエラーステータス

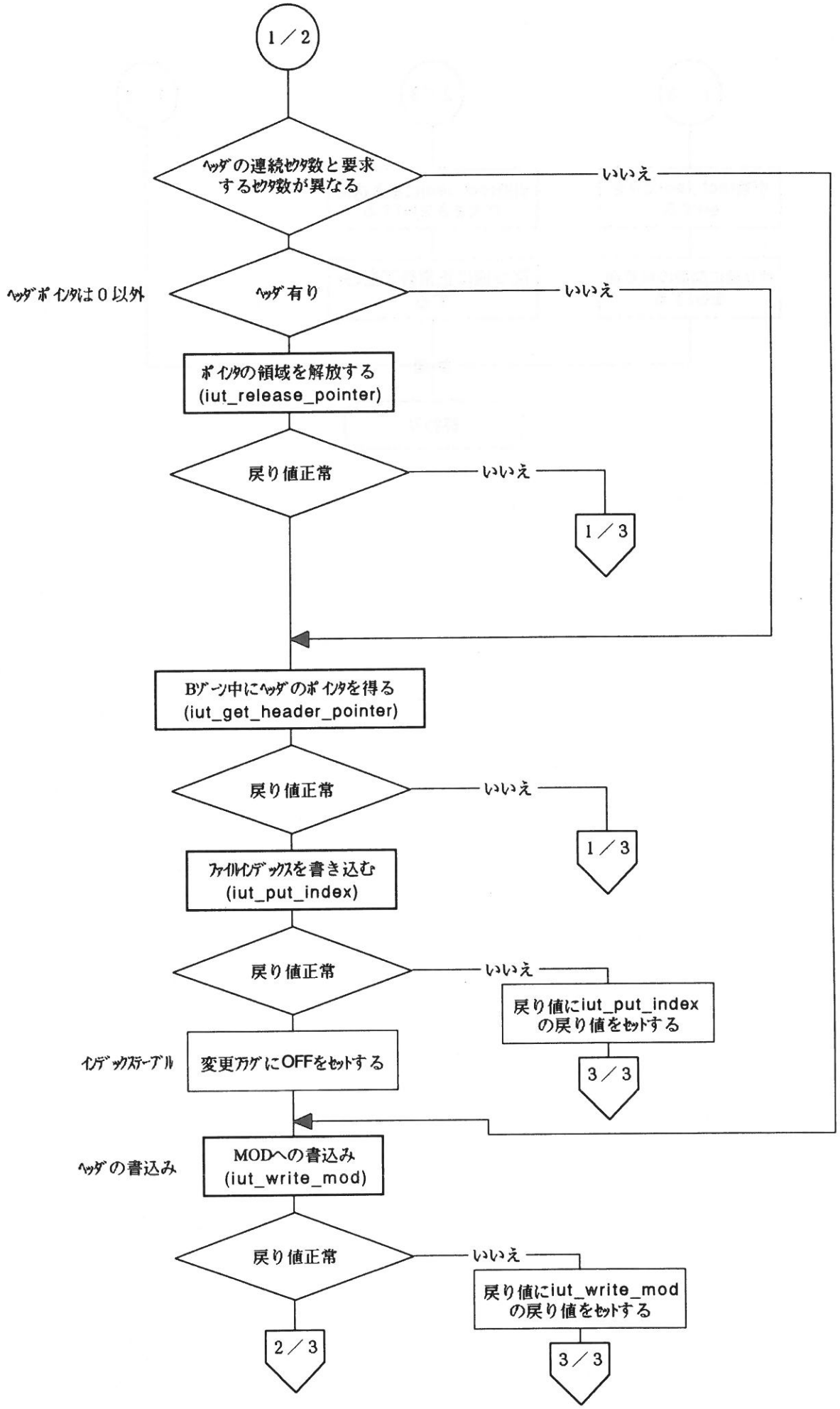
備考

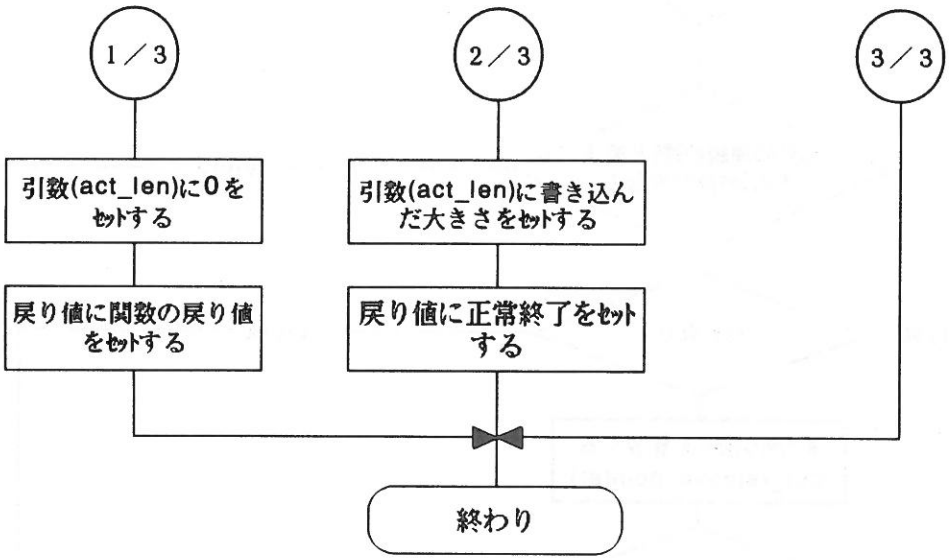
・すでにヘッダが書き込まれているファイルにヘッダを書き込む場合には、古いヘッダの内容は消去される。

・エラーが発生した場合には、act_lenの内容は意味をもたない。

ifm_write_header




1／2
ﾍｯﾀﾞの連続ｾｸﾀ数と要求するｾｸﾀ数が異なる
いいえ
ﾍｯﾀﾞﾎﾟｲﾝﾀは０以外
ﾍｯﾀﾞ有り
いいえ
ﾎﾟｲﾝﾀの領域を解放する
(iut_release_pointer)
戻り値正常
いいえ
1／3
Bﾌﾞｰﾝ中にﾍｯﾀﾞのﾎﾟｲﾝﾀを得る
(iut_get_header_pointer)
戻り値正常
いいえ
1／3
ﾌｧｲﾙｲﾝﾃﾞｯｸｽを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をｾｯﾄする
3／3
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ
変更ﾌﾗｸﾞにOFFをｾｯﾄする
ﾍｯﾀﾞの書込み
MODへの書込み
(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_mod
の戻り値をｾｯﾄする
2／3
3／3


1／3
2／3
3／3
引数(act_len)に0をセットする
引数(act_len)に書き込んだ大きさをセットする
戻り値に関数の戻り値をセットする
戻り値に正常終了をセットする
終わり

関数ifm_write_header()の検査指針

（１）機能に関する検査

１）関数 ifm_write_header()を用いて２０Ｋバイトのデータ0xBBを書き込み、この時act_lenに２０Ｋバイトが返され、ヘッダの先頭６バイトにファイルIDとデータサイズ（２０Ｋバイト）が、６般　デ0xBBで凡気譴討　とを確認する。　　る

２）指定ファイルのインデックステーブルに開始セクタアドレスとセクタ数２０がセットされていることを確認する。

（２）戻り値に関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ENUNITが返されることを確認する。

３）マウントされていないボリュームを指定して、ENMNTDが返されることを確認する。

４）オープンされていないファイルを指定して、ENOPNDが返されることを確認する。

５）書き込み禁止ファイルを指定して、ENPERMが返されることを確認する。

６）光磁気ディスクの電源を切り、EDRIVEが返されることを確認する。

4．5．7　ファイルのサイズを得る

```
long  ifm_get_file_size(fd, fsize, hsize)
long  fd ;
long  *fsize ;
long  *hsize ;
```

機能

ファイルのデータとヘッダのサイズを得る。

引数

fd　（入力）ファイルディスクリプタ。

fsize（出力）データサイズをバイト単位で表した値へのポインタ。

hsize（出力）ヘッダサイズをバイト単位で表した値へのポインタ。

戻り値

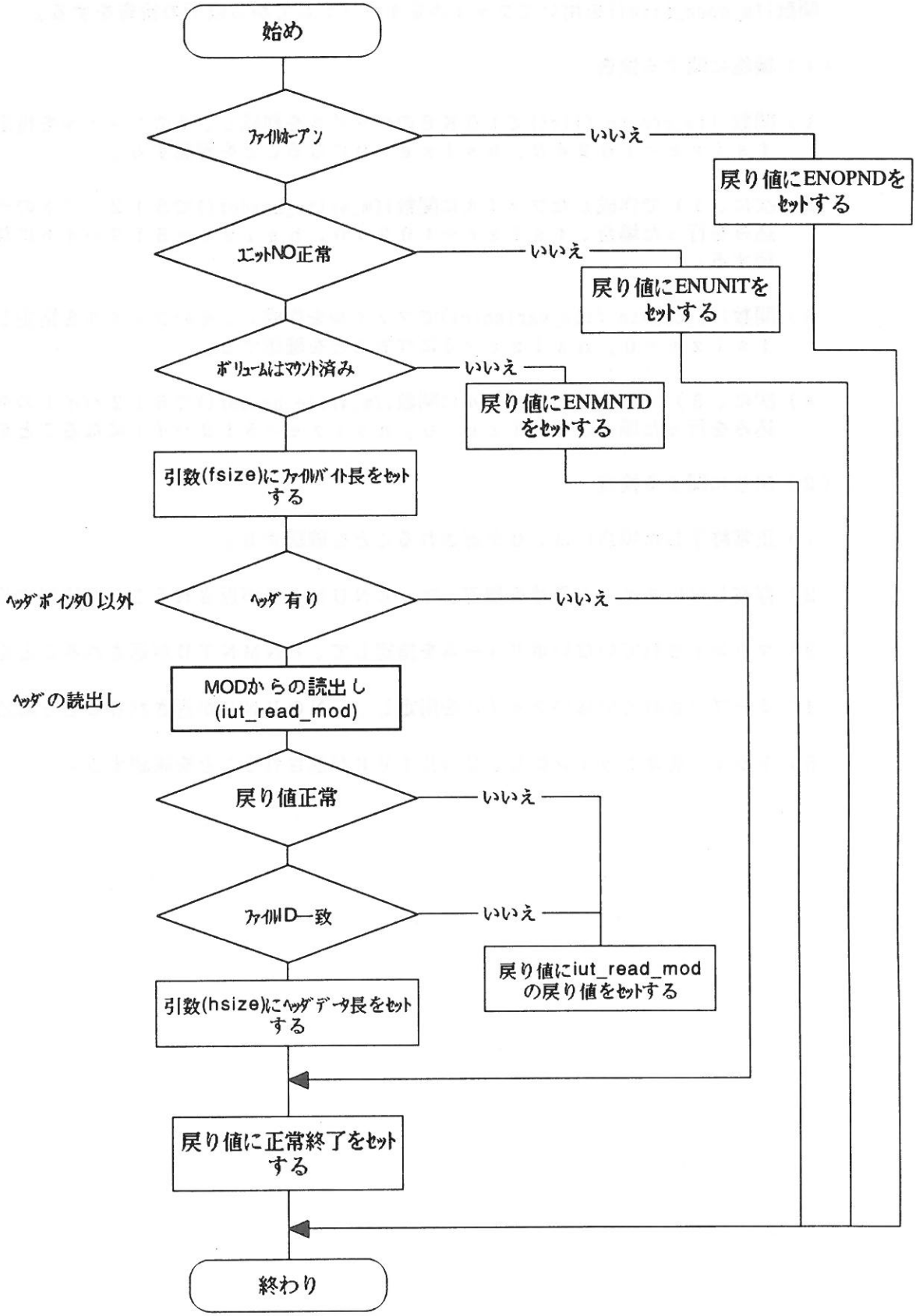
正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ENOPND（このファイルはオープンされていない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・ヘッダが書き込まれていないファイルを指定した場合には、hsizeに0を返す。

・エラーが発生した場合には、fsize, hsizeの内容は意味をもたない。

ifm_get_file_size



関数ifm_get_file_size()の検査指針

関数ifm_open_file()を用いてファイルをオープンしてから以下の検査をする。

（１）機能に関する検査

１）関数 ifm_create_file()で１０ＫＢのファイルを作成し、そのファイルを指定した場合、ｆｓｉｚｅ＝１０２４０，ｈｓｉｚｅ＝０になることを確認する。

２）次に、１）で作成したファイルに関数ifm_write_header()で５１２バイトのデータの書き込みを行った場合、ｆｓｉｚｅ＝１０２４０，ｈｓｉｚｅ＝５１２バイトになることを確認する。

３）関数ifm_create_file_variable()でファイルを作成し、そのファイルを指定した場合、ｆｓｉｚｅ＝０，ｈｓｉｚｅ＝０になることを確認する。

４）次に、３）で作成したファイルに関数ifm_write_header()で５１２バイトのデータの書き込みを行った場合、ｆｓｉｚｅ＝０，ｈｓｉｚｅ＝５１２バイトになることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定して、ＥＮＵＮＩＴが返されることを確認する。

３）マウントされていないボリュームを指定して、ＥＮＭＮＴＤが返されることを確認する。

４）オープンされていないファイルを指定し、ＥＮＯＰＮＤが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

4．5．8　ファイルの名前を得る

```
long  ifm_get_filename(fd, name)
long  fd ;
char  *name ;
```

機能

ファイルディスクリプタを指定して、ファイルの名前を得る。

引数

fd　（入力）ファイルディスクリプタ。

name（出力）ファイル名を格納するバッファへのポインタ。

戻り値

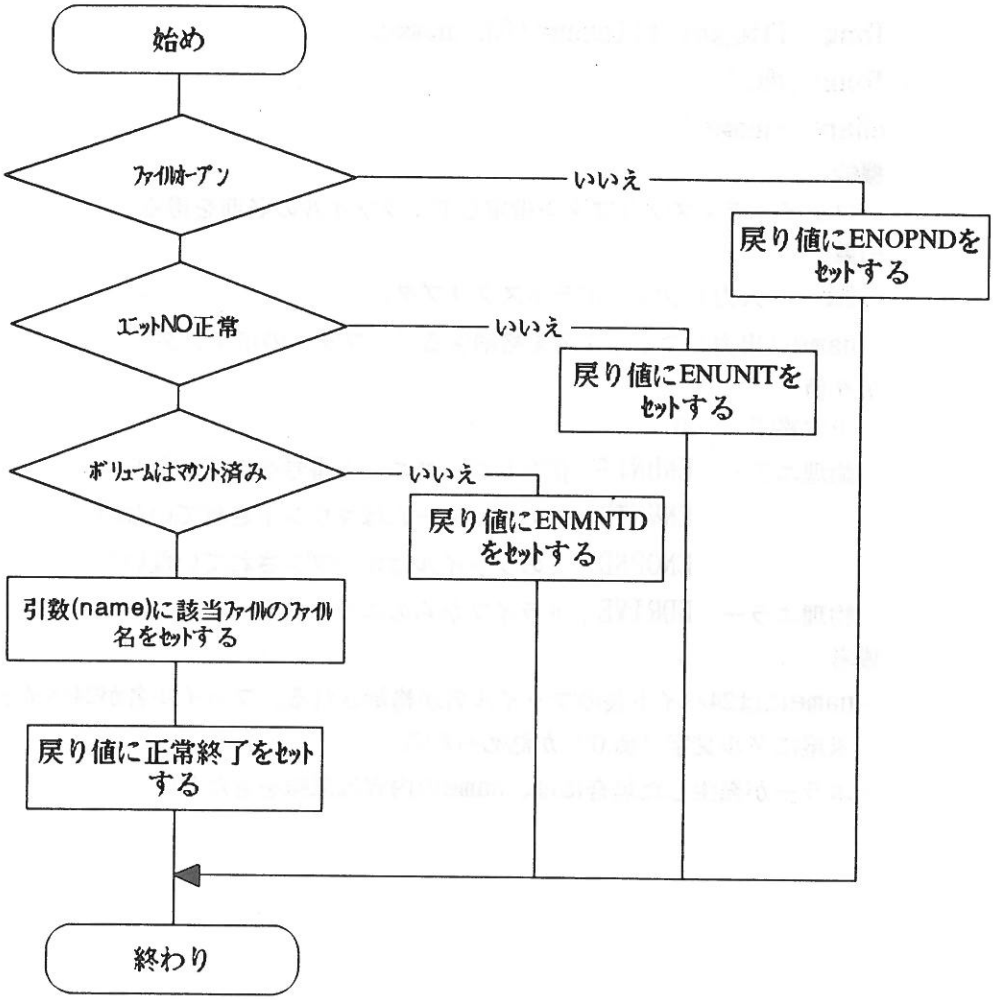
正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	ENOPND（このファイルはオープンされていない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・nameには24バイト長のファイル名が格納される。ファイル名が24バイトに満たない場合には、末尾にヌル文字（値0）が詰められる。

・エラーが発生した場合には、nameの内容は意味をもたない。

ifm_get_filename



関数ifm_get_filename()の検査指針

関数ifm_format_media(), ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）関数ifm_create_file()を用いてファイル名が

”ＦＩＬＥ－０１”
”ＦＩＬＥ－０２”
”ＦＩＬＥ－０３”

のファイルを作成する。

２）次に、関数ifm_open_file()を用いてｉｄが１、２、３のファイルをオープンする。

３）次に、本関数でｉｄ＝１のファイルを指定した場合”ＦＩＬＥ－０１”，ｉｄ＝２のファイルを指定した場合”ＦＩＬＥ－０２”，ｉｄ＝３を指定した場合”ＦＩＬＥ－０３”がファイル名として返されることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定し、ＥＮＵＮＩＴが返されることを確認する。

３）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

４）オープンされていないファイルを指定し、ＥＮＯＰＮＤが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

4．5．9　ファイルに名前を付ける

```
long  ifm_put_filename(fd, name)
long  fd ;
char  *name ;
```

機能

ファイルディスクリプタを指定して、ファイルに名前を付ける。

引数

fd　（入力）ファイルディスクリプタ。

name（入力）ファイル名が入っている文字列へのポインタ。

戻り値

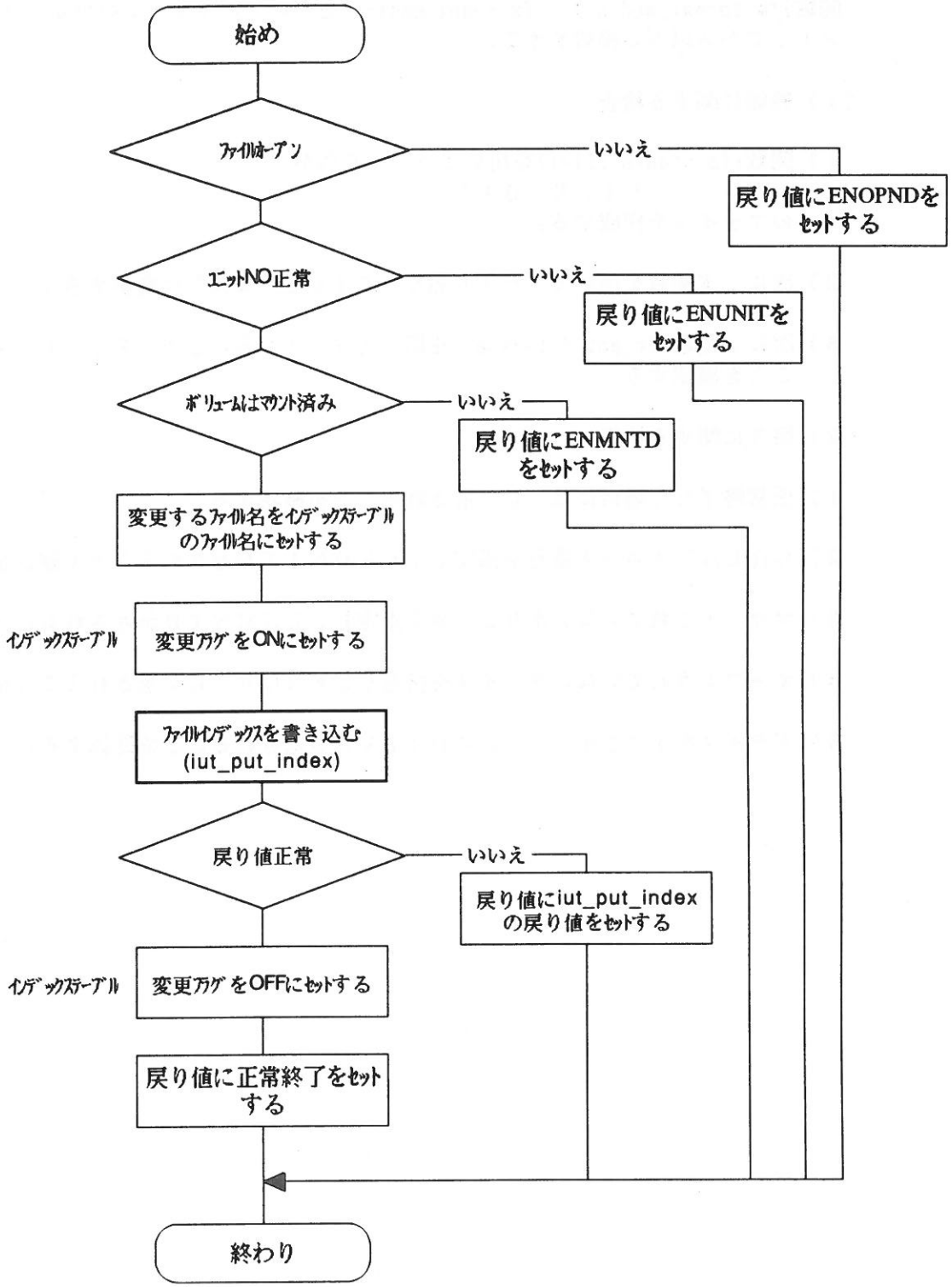
正常終了	0
論理エラー	ENUNIT（存在しないユニット番号を指定した）
	ENMNTD（このボリュームはマウントされていない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・ファイルの名前を変更する場合に使用する。

・ファイル名は最大24バイト長の文字列で、その最後にはヌル文字（値0）を付加しなければならない。25バイト以上の文字列を指定した場合には、先頭から24バイト分が使用される。

ifm_put_filename



関数ifm_put_filename()の検査指針

関数ifm_format_media(), ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）関数ifm_create_file()を用いてファイル名が
”ＦＩＬＥ－０１”
のファイルを作成する。

２）次に、本関数を用いてファイル名を”ＦＩＬＥ－ＸＸ”に変更する。

３）次に、関数ifm_get_filename()を用いてファイル名として”ＦＩＬＥ＿ＸＸ”が返されることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）存在しないユニット番号を指定し、ＥＮＵＮＩＴが返されることを確認する。

３）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

４）オープンされていないファイルを指定し、ＥＮＯＰＮＤが返されることを確認する。

５）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

4．6　ヘッダを順番に読み込む

4．6．1　読み込みポインタを先頭に戻す

```
long  ifm_reset_header_counter(vd)
long  vd ;
```

機能

ヘッダを順番に読み込むために、読み込みポインタをボリュームの先頭に戻す。

引数

vd（入力）ボリュームディスクリプタ。

戻り値

正常終了　0

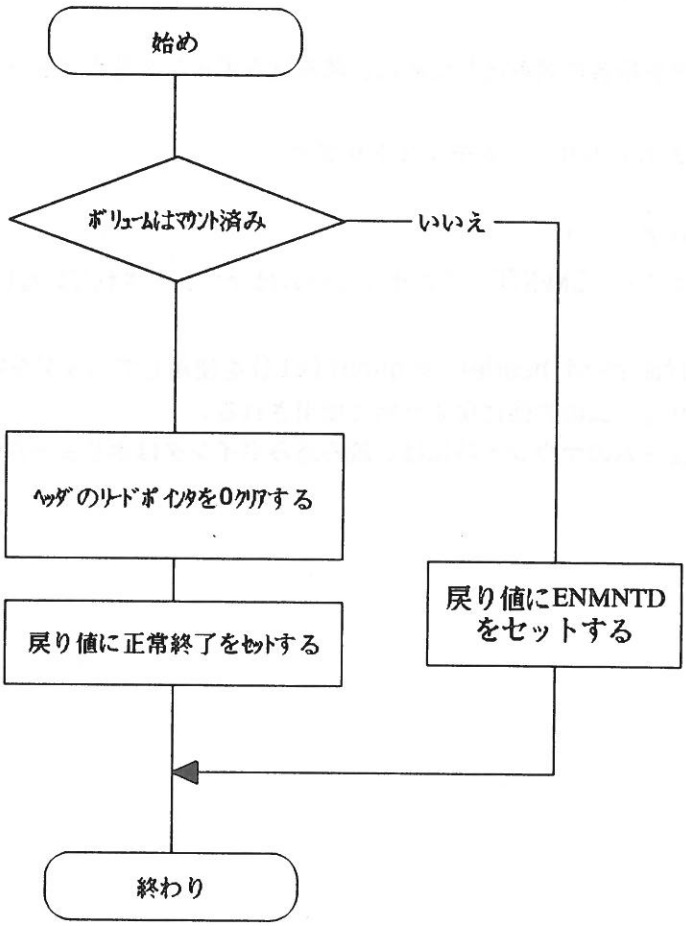
論理エラー　ENMNTD（このボリュームはマウントされていない）

備考

・関数ifm_read_header_sequential()を使用してヘッダを順番に読み込む場合に、ポインタをボリュームの先頭に戻すために使用される。

・ボリュームのマウント時には、読み込みポインタはボリュームの先頭に戻される。

ifm_reset_header_counter



関数ifm_reset_header_counter()の検査指針

（1）機能に関する検査

1）関数ifm_reset_header_counter()を実行し、指定したvdに対応するヘッダのリードポインタが0になることを確認する。

（2）戻り値に関する検査

1）マウントされていないボリュームを指定し、ENMNTDが返されることを確認する。

2）光磁気ディスクの電源を切り、EDRIVEが返されることを確認する。

４．６．２　順番にヘッダを読み込み

```
long  ifm_read_header_sequential(vd, id, buf, bsz, vlen)
long  vd ;
long  *id ;
char  *buf ;
long  bsz ;
long  *vlen ;
```

機能

次のファイルのヘッダを読み込む。

引数

vd　（入力）ボリュームディスクリプタ。

id　（出力）読み込んだファイルのファイルＩＤ値へのポインタ。

buf　（出力）ヘッダを格納するバッファへのポインタ。

bsz　（入力）読み込むヘッダのサイズをバイト単位で表した値。

vlen（出力）実際に読み込んだヘッダのサイズをバイト単位で表した値へのポインタ。

戻り値

正常終了　0

論理エラー　ENMNTD（このボリュームはマウントされていない）

EFLNEX（存在しないファイルを指定した）

物理エラー　EDRIVE｜ドライブからのエラーステータス

備考

・仮削除されたファイル、ヘッダの書き込まれていないファイルは読みとばされる。

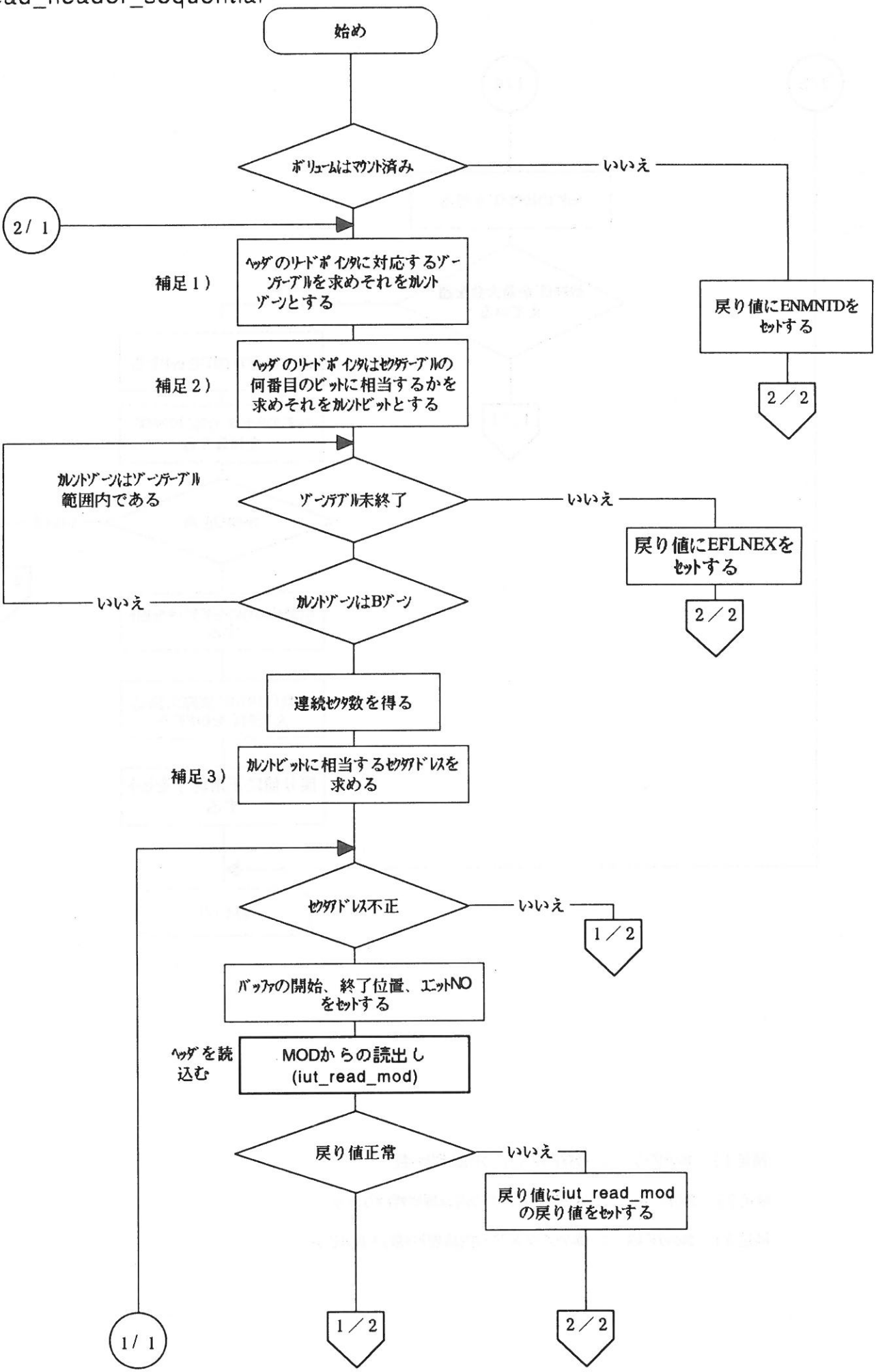
・この関数はボリューム内にヘッダが格納されている順にヘッダを読み込むので、得られるヘッダの順番はファイルＩＤの順番とは異なる場合がある。

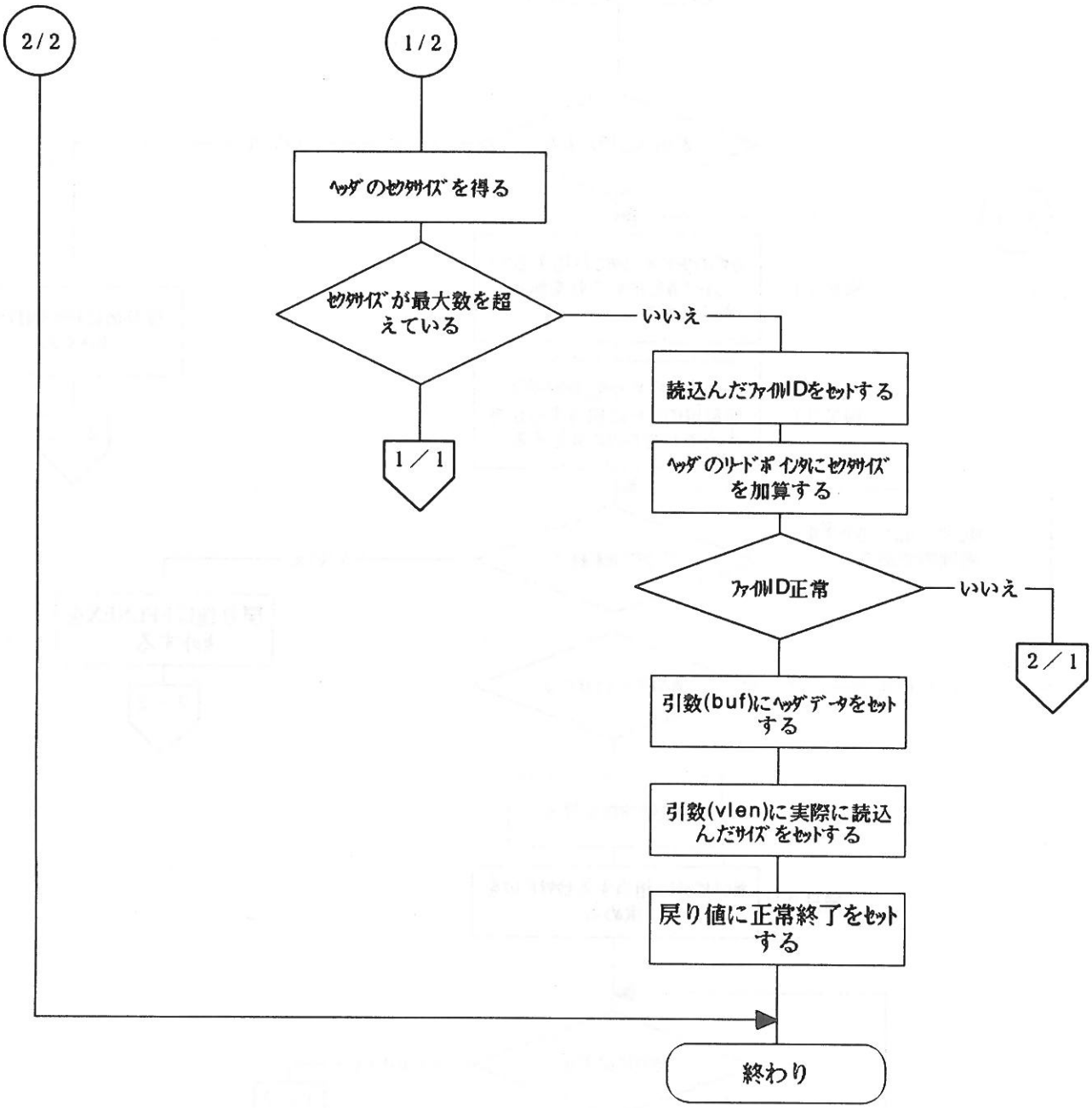
・bszの値がヘッダのサイズよりも小さい場合には、ヘッダの先頭のbszバイト分がbufに格納される。

・bszの値がヘッダのサイズよりも大きい場合には、bufのvlen+1バイト目以降の内容は意味をもたない。

・エラーが発生した場合には、id, buf, vlenの内容は意味をもたない。

ifm_read_header_sequential





補足１）　ｶﾚﾝﾄｿﾞｰﾝ　：　ｾｸﾀｱﾄﾞﾚｽ÷ｿﾞｰﾝ内論理ｾｸﾀ数

補足２）　ｶﾚﾝﾄﾋﾞｯﾄ　：　(ｾｸﾀｱﾄﾞﾚｽ÷ｿﾞｰﾝ内論理ｾｸﾀ数)の余り

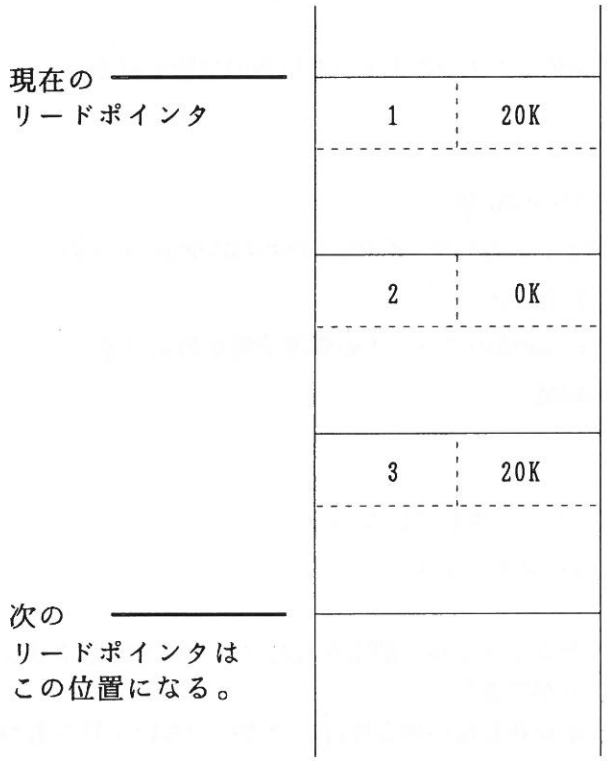
補足３）　ｶﾚﾝﾄｱﾄﾞﾚｽ　：　(ｶﾚﾝﾄｿﾞｰﾝ×ｿﾞｰﾝ内論理ｾｸﾀ数)＋ｶﾚﾝﾄﾋﾞｯﾄ

関数ifm_read_header_sequential()の検査指針

（１）機能に関する検査

１）ifm_reset_header_counter()を行う。

２）関数ifm_write_header()を用いて２０Ｋバイトの、ファイル属性が仮削除であるヘッダを作成する。次に、同サイズの領域を持っているが実データサイズが０のヘッダ、同サイズでデータ０ｘ１１で満たされているヘッダを作成する。この操作により以下の様なデータ領域が確保される。



３）関数ifm_read_header_sequential()を用いてヘッダを１０Ｋバイト分読み込むと、idには３（１と２は読み飛ばされる）、vlenには１０Ｋバイトが返され、bufはvlen分のデータ０ｘ１１で満たされ、リードポインタがファイルID３のヘッダのセクタアドレス＋２０を指していることを確認する。

４）また、１）で作成したヘッダのIDが３のファイルの属性を削除ファイルとし、関数ifm_read_header_sequential()を実行した場合には、vlenに０が返され正常終了することを確認する。

（２）戻り値に関する検査

１）正常終了した場合には、０が返されることを確認する。
２）マウントされていないボリュームを指定した場合は、ENMNTDが返されることを確認する。
３）存在しないファイルを指定した場合は、EFLNEXが返されることを確認する。
４）光磁気ディスクの電源を切り、EDRIVEが返されることを確認する。

4．7　システムの管理

4．7．1　すべてのファイルの管理情報を得る

```
long  ifm_get_list_all(vd, id, array, size, gt_sz)
long  vd ;
long  id ;
char  *array ;
long  size ;
long  *gt_sz ;
```

機能

id(〉0)で指定したファイルのファイルインデックス中のポインタ以外の情報を得る。

引数

vd　（入力）ボリュームディスクリプタ。

id　（入力）管理情報を得るファイルＩＤの開始値。

array（出力）ファイルの管理情報を格納する、各size of(struct listidxtype)バイトのstruct listidxtypeストラクチャの配列へのポインタ。

size　（入力）arrayの配列の個数。最大size個のファイルの管理情報が得られる。

gt_sz（出力）実際に得られた管理情報の個数。

戻り値

正常終了　0

論理エラー　ENMNTD（このボリュームはマウントされていない）

物理エラー　EDRIVE｜ドライブからのエラーステータス

備考

・この関数では読み込み禁止ファイル、システムファイル、削除されたファイルを含むボリューム内のすべてのファイルの管理情報を得ることができる。

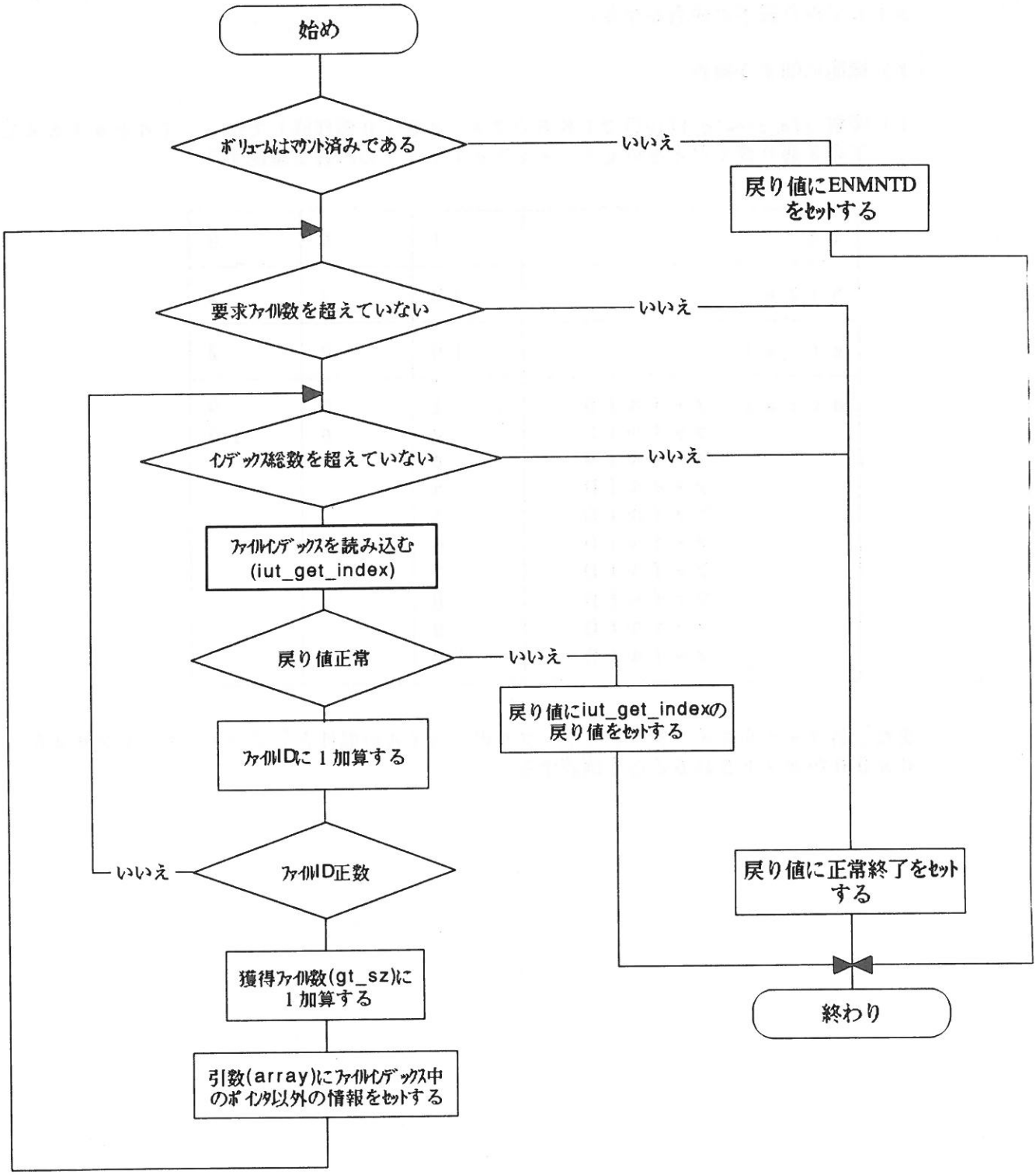
・指定したファイルＩＤに対応するファイルが存在しない場合には、つぎに大きいＩＤをもつファイルから順番に管理情報が得られる。なお、ファイルＩＤの最小値は１なので、ボリュームの先頭から管理情報を得る場合には１を指定する。

・指定した位置から最後のファイルまでの個数がsizeに満たない場合には、対応するarrayの内容は意味をもたない。

・インデックステーブルの属性はファイルの属性情報をビットごとに管理している。ファイルの書込み禁止（0x02）、読込み禁止（0x04）、システム状態（0x08）、ディレクトリ状態（0x10）で示している。

・エラーが発生した場合には、array, gt_szの内容は意味をもたない。

ifm_get_list_all



関数ifm_get_list_all()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）関数 ifm_create_file()で１ＫＢのファイルを１０個作成してから、ｉｄとｓｉｚｅを以下の３通り変えたときのｇｔ＿ｓｚとａｒｒａｙの内容を確認する。

ｉｄ	1	5	9
ｓｉｚｅ	２０	2	3
ｇｔ＿ｓｚ	１０	2	2
ａｒｒａｙ　ファイルＩＤ	1	5	9
ファイルＩＤ	2	6	１０
ファイルＩＤ	3		
ファイルＩＤ	4		
ファイルＩＤ	5		
ファイルＩＤ	6		
ファイルＩＤ	7		
ファイルＩＤ	8		
ファイルＩＤ	9		
ファイルＩＤ	１０		

また、各ファイルのインデックステーブルのファイルの属性としてバイトアドレス４４から０ｘ００がセットされることを確認する。

2）次にファイルの属性を以下のように変えてから１）と同様の確認を行いsizeと arrayが１）と同じか確認する。

・ｉｄが１のファイルを関数ifm_write_protect_on()を用いて書き込み禁止にする。
・ｉｄが３のファイルを関数ifm_read_protect_on()を用いて読み込み禁止にする。
・ｉｄが５のファイルを関数ifm_system_flag_on()を用いてシステムファイルにする。
・ｉｄが７のファイルを関数ifm_directory_flag_on()を用いてディレクトリファイルにする。

また、各ファイルのインデックステーブルのファイルの属性がセットされていることを確認する。

ａｒｒａｙの属性が”０２Ｈ”（書き込み禁止）
属性が”００Ｈ”
属性が”０４Ｈ”（読み込み禁止）
属性が”００Ｈ”
属性が”０８Ｈ”（システムファイル）
属性が”００Ｈ”
属性が”１０Ｈ”（ディレクトリファイル）
属性が”００Ｈ”
属性が”００Ｈ”
属性が”００Ｈ”

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントしないで本関数を呼び出し、ＥＮＭＮＴＤが返されることを確認する。

３）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．２　ファイルを実削除する

```
long   ifm_delete_actually(vd, id)
long   vd ;
long   id ;
```

機能

仮削除されているファイルをボリューム内から消去する。

引数

vd（入力）ボリュームディスクリプタ。

id（入力）実削除するファイルのファイルＩＤ。

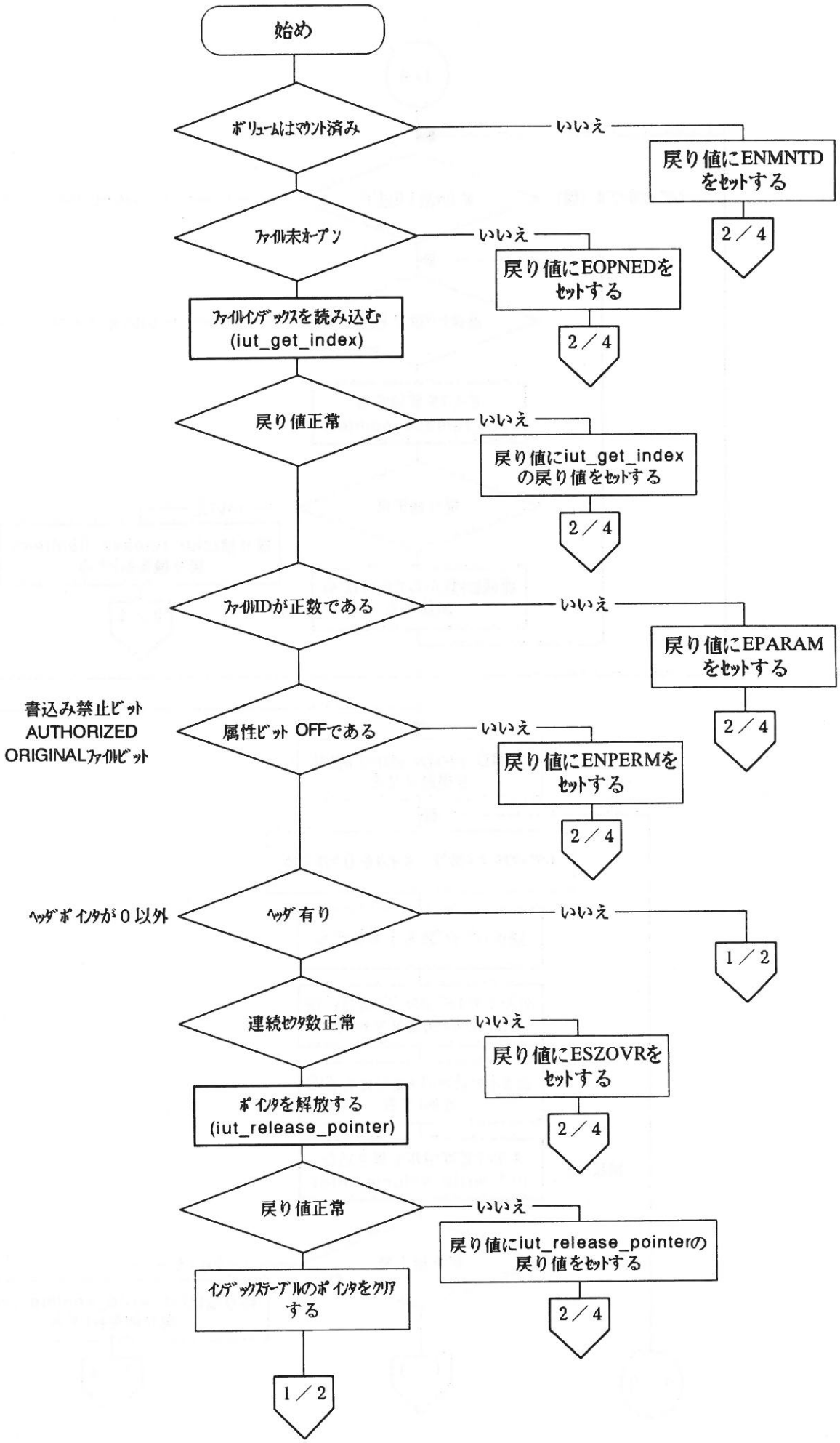
戻り値

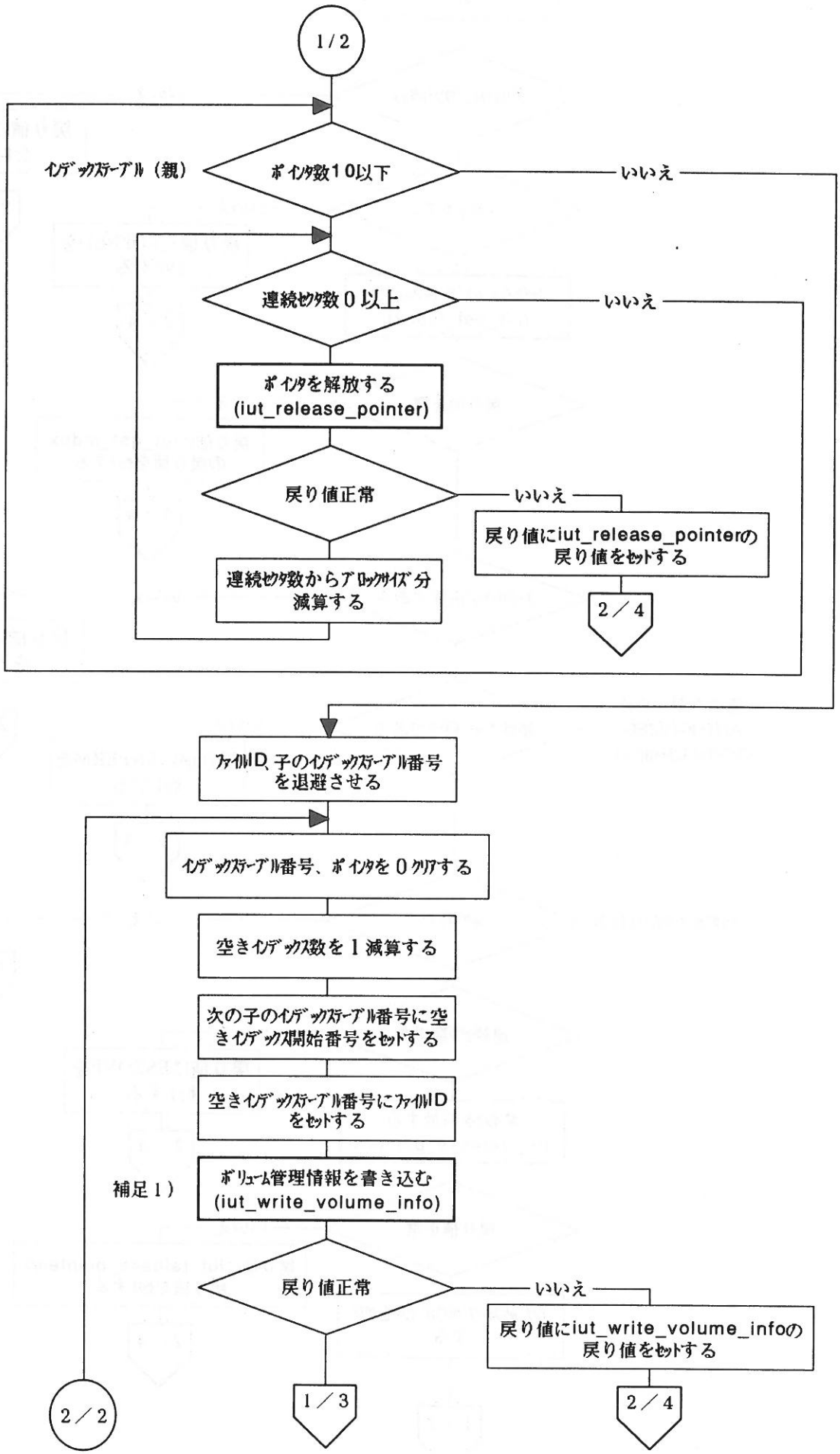
正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	EOPNED（指定したファイルはオープンされている）
	EPARAM（関数のパラメタが不正である）
	ESZOVR（開始位置がファイルのデータサイズ以上である）
	ENPERM（禁止されているアクセスを行おうとした）
物理エラー	EDRIVE｜ドライブからのエラーステータス

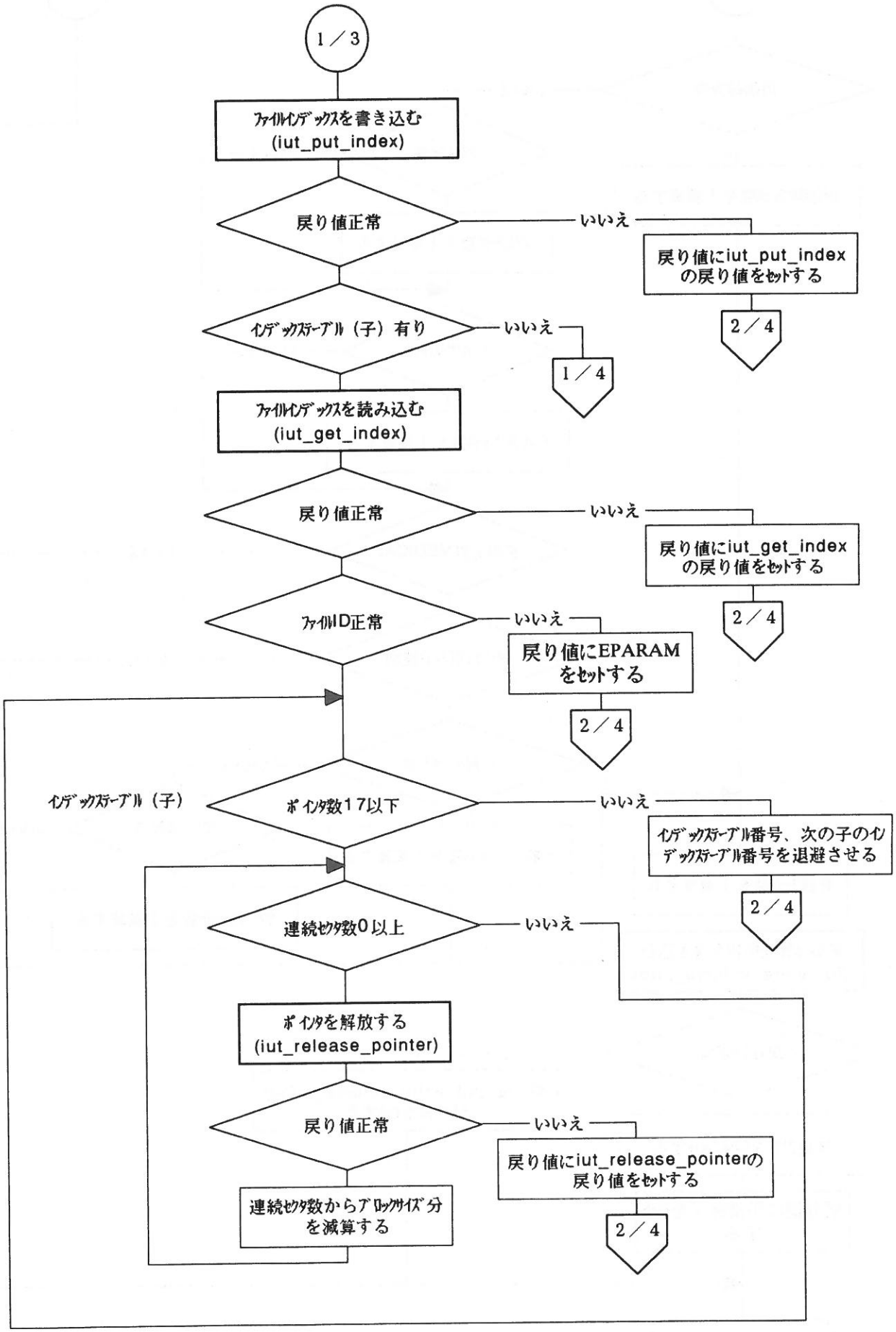
備考

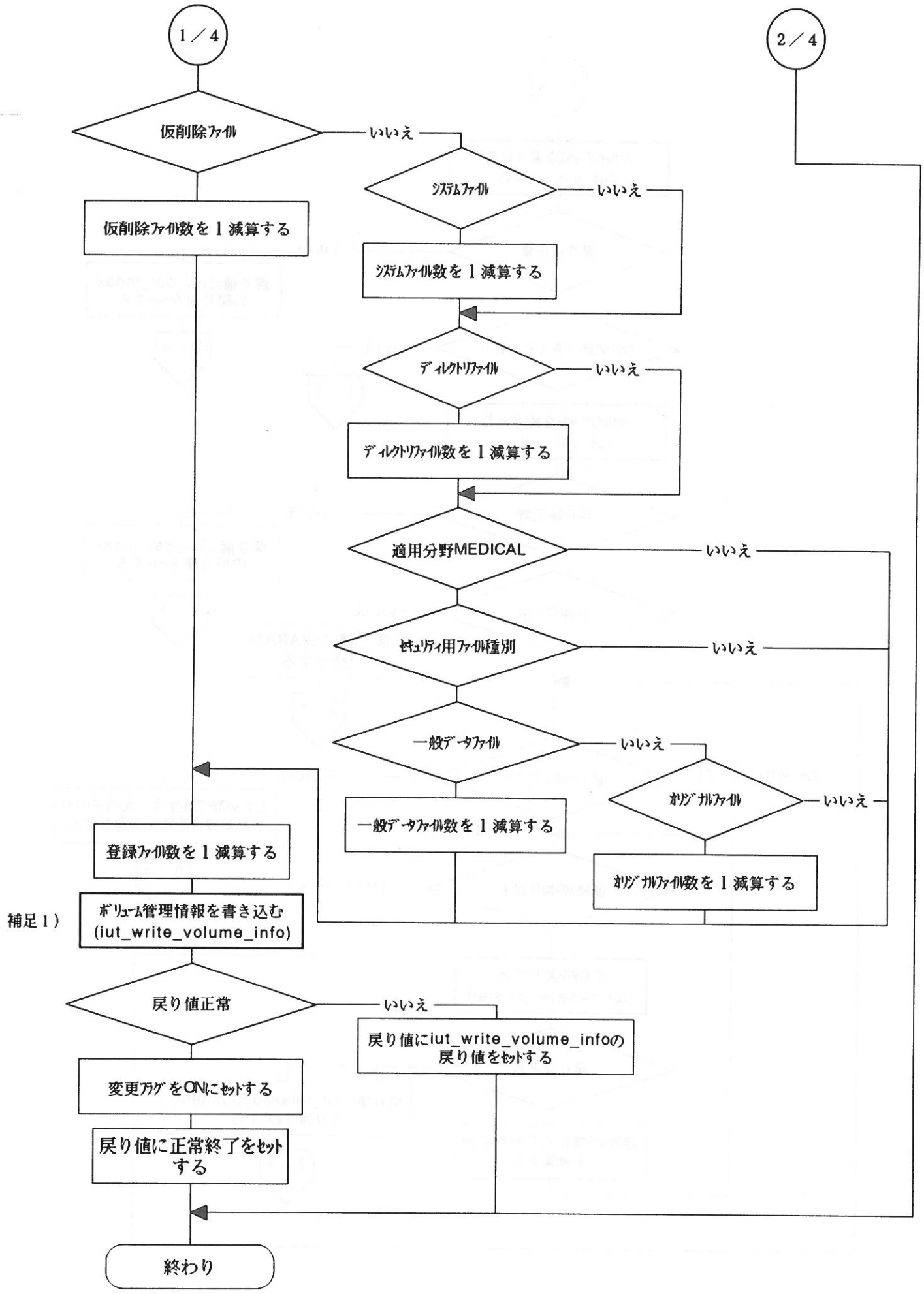
・指定したファイルの内容はすべて消去されるので、十分注意して使用しなければならない。

ifm_delete_actually




1/2
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ（親）
ﾎﾟｲﾝﾀ数10以下
いいえ
連続ｾｸﾀ数０以上
いいえ
ﾎﾟｲﾝﾀを解放する
(iut_release_pointer)
戻り値正常
いいえ
戻り値にiut_release_pointerの
戻り値をｾｯﾄする
2／4
連続ｾｸﾀ数からﾌﾞﾛｯｸｻｲｽﾞ分
減算する
ﾌｧｲﾙID、子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号
を退避させる
ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号、ﾎﾟｲﾝﾀを０ｸﾘｱする
空きｲﾝﾃﾞｯｸｽ数を１減算する
次の子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号に空
きｲﾝﾃﾞｯｸｽ開始番号をｾｯﾄする
空きｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号にﾌｧｲﾙID
をｾｯﾄする
補足１）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をｾｯﾄする
2／2
1／3
2／4


1／3
ファイルインデックスを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をセットする
2／4
インデックステーブル（子）有り
いいえ
1／4
ファイルインデックスを読み込む
(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_index
の戻り値をセットする
2／4
ファイルID正常
いいえ
戻り値にEPARAM
をセットする
2／4
インデックステーブル（子）
ポインタ数17以下
いいえ
インデックステーブル番号、次の子のインデックステーブル番号を退避させる
2／4
連続セクタ数0以上
いいえ
ポインタを解放する
(iut_release_pointer)
戻り値正常
いいえ
戻り値にiut_release_pointerの
戻り値をセットする
2／4
連続セクタ数からブロックサイズ分
を減算する


1／4
2／4
仮削除ﾌｧｲﾙ
いいえ
ｼｽﾃﾑﾌｧｲﾙ
いいえ
仮削除ﾌｧｲﾙ数を１減算する
ｼｽﾃﾑﾌｧｲﾙ数を１減算する
ﾃﾞｨﾚｸﾄﾘﾌｧｲﾙ
いいえ
ﾃﾞｨﾚｸﾄﾘﾌｧｲﾙ数を１減算する
適用分野MEDICAL
いいえ
ｾｷｭﾘﾃｨ用ﾌｧｲﾙ種別
いいえ
一般ﾃﾞｰﾀﾌｧｲﾙ
いいえ
ｵﾘｼﾞﾅﾙﾌｧｲﾙ
いいえ
一般ﾃﾞｰﾀﾌｧｲﾙ数を１減算する
ｵﾘｼﾞﾅﾙﾌｧｲﾙ数を１減算する
登録ﾌｧｲﾙ数を１減算する
補足１）
ﾎﾞﾘｭｰﾑ管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をｾｯﾄする
変更ﾌﾗｸﾞをONにｾｯﾄする
戻り値に正常終了をｾｯﾄ
する
終わり

関数ifm_delete_actually()の補足説明

補足１）

実削除とは，仮削除の状態で媒体上に存在しているデータを媒体上から消去することである。従って次のことをすべて行なわなければならない。

１）当該ファイルのインデックステーブルを解放して他のファイルのために使用可能とすること。子のインデックステーブルが存在する場合はそれも解放すること。
２）当該ファイルが使用していたヘッダ領域を解放して，他のファイルのために使用可能とすること。
３）当該ファイルが使用していたデータ領域を解放して，他のファイルのために使用可能とすること。
４）　３）に於いて，今回のデータ領域解放により，そのデータが含まれていたゾーンの全てのブロックが解放される結果となる場合には，当該ゾーン自体を未使用かつ未定義のゾーンとして，将来のゾーン確保時に使用可能とすること。
５）「ボリューム管理情報その２」中の「仮削除ファイル数」「空インデックス数」を，更新する。

そこでボリューム管理情報の更新値を下記に示す。

１）　仮削除ファイル数　　：　仮削除ファイル数　－　１

２）　空きインデックス数　：　空きインデックス数　＋　実削除したインデックス数

３）　更新日時　　：　更新日時を格納する

関数ifm_delete_actually() の検査指針

関数 ifm_delete_actually()は予め削除（仮）されているファイルに対して、実削除を行うものである。よって本関数の動作を確認する以前にファイル削除関数ifm_delete_file() での動作確認をしておかなければならない。

（1）機能に関する検査

1) 関数 ifm_delete_actually()を用いて，ファイルを実削除する。
ファイルが該当するボリューム内から消去される事を確認する。

2) 該当するファイルがヘッダを持っている場合は，そのヘッダが消去することを確認する。

3) 該当ファイルのインデックステーブルが，解放されることを確認する。

4) 該当ファイルが使用していたデータ領域が，解放されることを確認する。

5) ボリューム管理情報部の仮削除ファイル数が1減算されることを確認する。

6) ボリューム管理情報部の空きインデックスファイル数から，実削除したインデックス数が加算されることを確認する。

7) 指定されたボリュームがマウントされていない場合は，エラーとすることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には，0 が返されることを確認する。

2) 指定されたボリュームがマウントされていない場合は，ENMNTDが返されることを確認する。

3) オープンされているファイルを指定した場合は、EOPNEDが返されることを確認する。

4) 読み込まれたインデックステーブルのファイルＩＤが不正だった場合は、EPARAMが返されることを確認する。

5) 解放するヘッダの開始位置がファイルのデータサイズ以上の場合は、ESZOVRが返されることを確認する。

6) 書き込み禁止ファイルを指定した場合は、ENPERMが返されることを確認する。

7) オーソライズオリジナルファイルを指定した場合は、ENPERMが返されることを確認する。

8) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

４．７．３　ファイルを再登録する

```
long   ifm_recover_temporaly_deleted(vd, id)
long   vd ;
long   id ;
```

機能

削除されているファイルをボリューム内に再登録する。

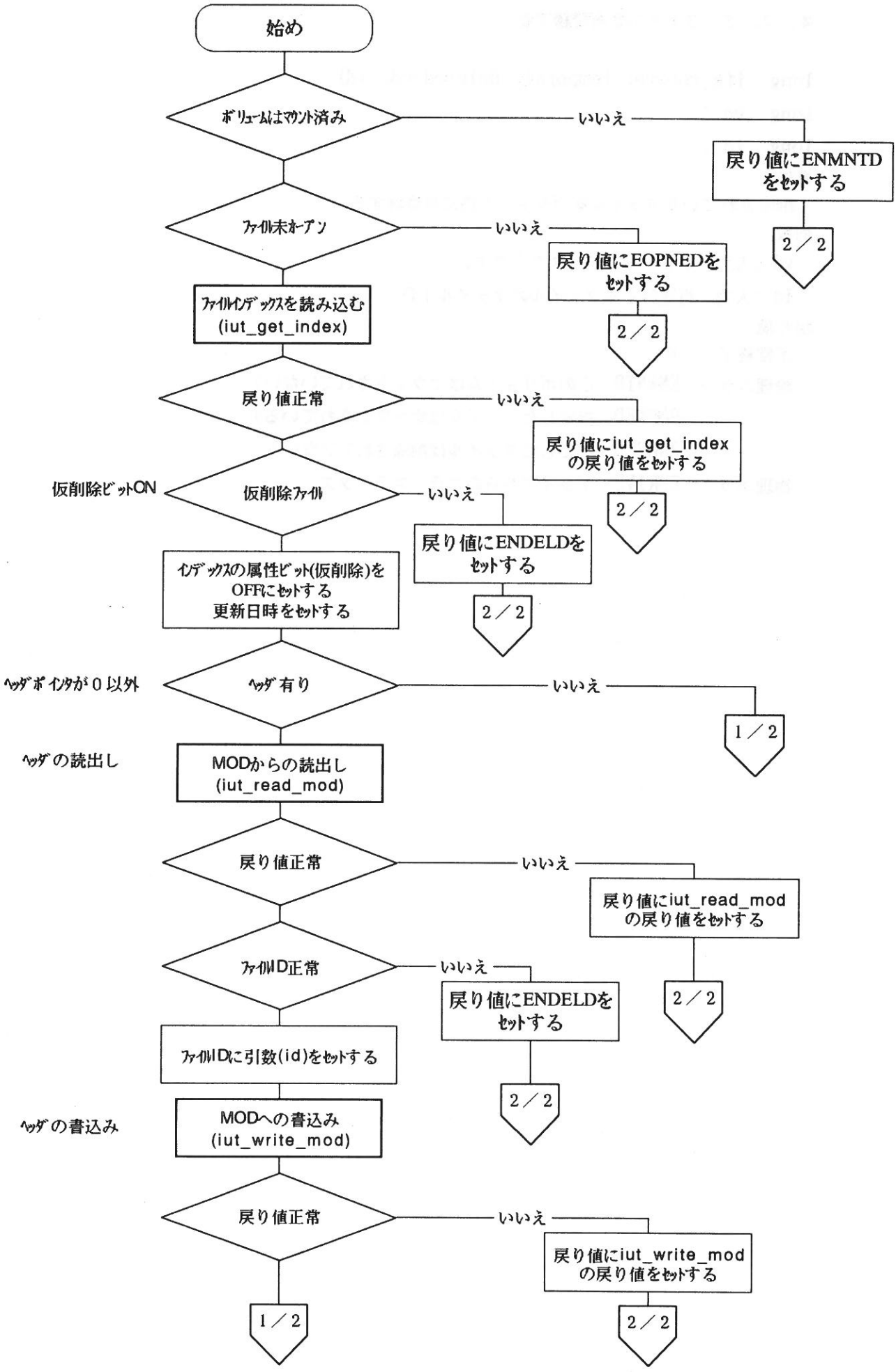
引数

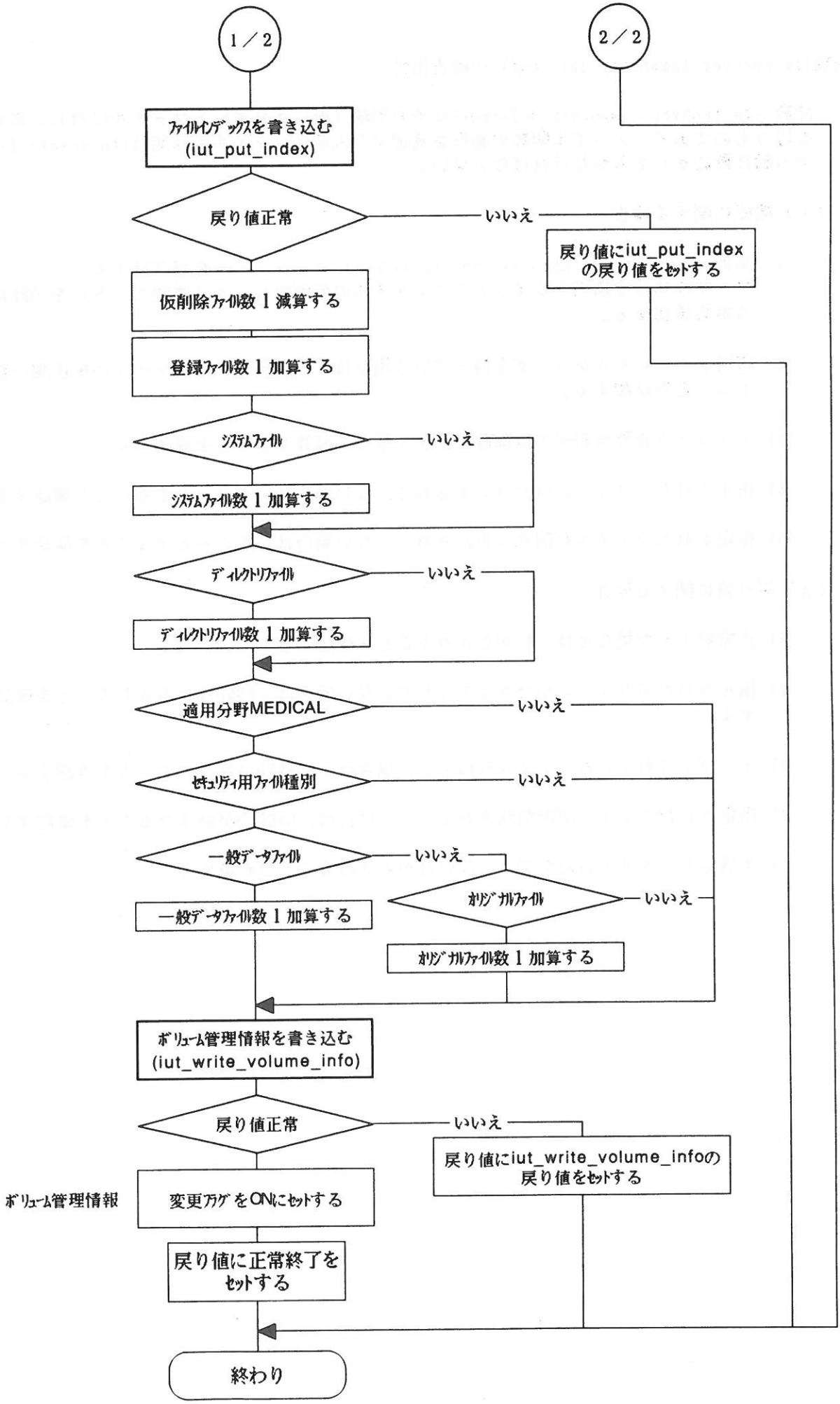
vd（入力）ボリュームディスクリプタ。

id（入力）再登録するファイルのファイルＩＤ。

戻り値

正常終了	0	
論理エラー	ENMNTD	（このボリュームはマウントされていない）
	EOPNED	（指定したファイルはオープンされている）
	ENDELD	（指定したファイルは削除されていない）
物理エラー	EDRIVE	ドライブからのエラーステータス


始め
ボリュームはマウント済み
いいえ
戻り値にENMNTDをセットする
2／2
ファイル未オープン
いいえ
戻り値にEOPNEDをセットする
2／2
ファイルインデックスを読み込む(iut_get_index)
戻り値正常
いいえ
戻り値にiut_get_indexの戻り値をセットする
2／2
仮削除ビットON
仮削除ファイル
いいえ
戻り値にENDELDをセットする
2／2
インデックスの属性ビット(仮削除)をOFFにセットする
更新日時をセットする
ヘッダポインタが0以外
ヘッダ有り
いいえ
1／2
ヘッダの読出し
MODからの読出し(iut_read_mod)
戻り値正常
いいえ
戻り値にiut_read_modの戻り値をセットする
2／2
ファイルID正常
いいえ
戻り値にENDELDをセットする
2／2
ファイルIDに引数(id)をセットする
ヘッダの書込み
MODへの書込み(iut_write_mod)
戻り値正常
いいえ
戻り値にiut_write_modの戻り値をセットする
2／2
1／2


1／2
2／2
ファイルインデックスを書き込む
(iut_put_index)
戻り値正常
いいえ
戻り値にiut_put_index
の戻り値をセットする
仮削除ファイル数 1 減算する
登録ファイル数 1 加算する
システムファイル
いいえ
システムファイル数 1 加算する
ディレクトリファイル
いいえ
ディレクトリファイル数 1 加算する
適用分野MEDICAL
いいえ
セキュリティ用ファイル種別
いいえ
一般データファイル
いいえ
オリジナルファイル
いいえ
一般データファイル数 1 加算する
オリジナルファイル数 1 加算する
ボリューム管理情報を書き込む
(iut_write_volume_info)
戻り値正常
いいえ
戻り値にiut_write_volume_infoの
戻り値をセットする
ボリューム管理情報
変更フラグをONにセットする
戻り値に正常終了を
セットする
終わり

関数ifm_recover_temporaly_deleted() の検査指針

関数 ifm_recover_temporaly_deleted()は予め削除（仮）されているファイルに対し、再登録を行うものである。よって本関数の動作を確認する以前にファイル削除関数ifm_delete_file()での動作確認をしておかなければならない。

（1）機能に関する検査

1) 関数 ifm_recover_temporaly_deleted()を用いて，ファイルを再登録する。
ファイルIDにて該当するインデクスファイルの仮削除ビット（属性ビット）を OFFにする事を確認する。

2) 該当するファイルがヘッダを持っている場合は，そのヘッダのファイルIDを正規の値にすることを確認する。

3) ボリューム管理情報部の仮削除ファイル数を1減算することを確認する。

4) 指定されたボリュームがマウントされていない場合は，エラーとすることを確認する。

5) 指定されたファイルが削除（仮）されていない場合は，エラーとすることを確認する。

（2）戻り値に関する検査

1) 正常終了した場合には，0 が返されることを確認する。

2) 指定されたボリュームがマウントされていない場合は，ENMNTDが返されることを確認する。

3) オープンされているファイルを指定した場合は、EOPNEDが返されることを確認する。

4) 指定されたファイルが仮削除されていない場合は，ENDELDが返されることを確認する。

5) 光磁気ディスクの電源を切り、EDRIVEが返されることを確認する

４．７．４　ファイルを書き込み禁止にする

```
long  ifm_write_protect_on(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルを書き込み禁止にする。

引数

vd（入力）ボリュームディスクリプタ。

id（入力）書き込み禁止にするファイルのファイルＩＤ。

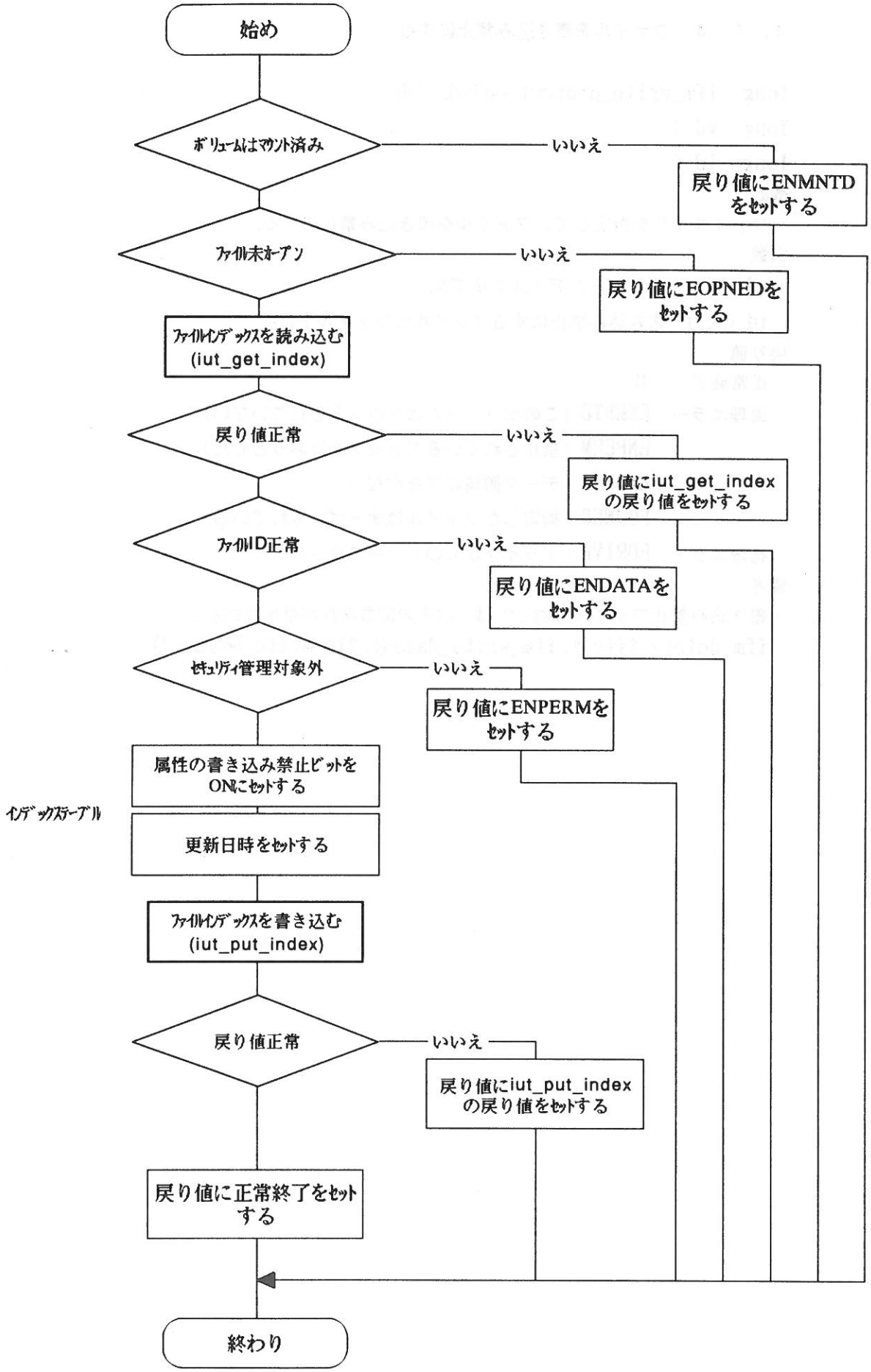
戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・書き込み禁止ファイルに対しては、以下の関数操作が禁止される。
ifm_delete_file(), ifm_write_data(), ifm_write_header()

ifm_write_protect_on



関数ifm_write_protect_on()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルの書き込み禁止ビット（インデックステーブルの属性）がＯＮになることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．５　ファイルを書き込み可能にする

```
long  ifm_write_protect_off(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルを書き込み可能にする。

引数

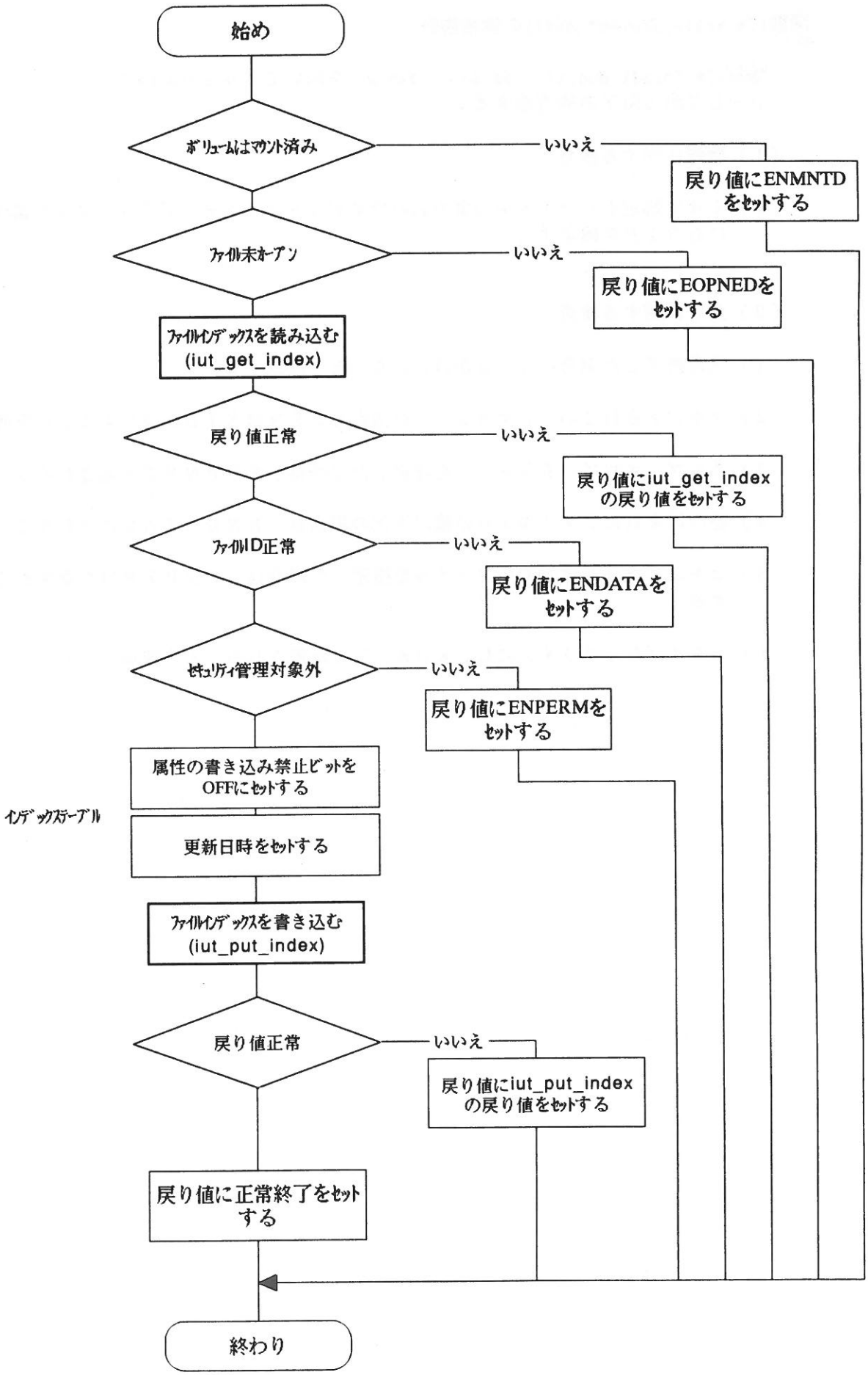
vd（入力）ボリュームディスクリプタ。

id（入力）書き込み可能にするファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_write_protect_off



関数ifm_write_protect_off()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルの書き込み禁止ビット（インデックステーブルの属性）がＯＦＦになることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

4．7．6　ファイルを読み込み禁止にする

```
long  ifm_read_protect_on(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルを読み込み禁止にする。

引数

vd（入力）ボリュームディスクリプタ。

id（入力）読み込み禁止にするファイルのファイルＩＤ。

戻り値

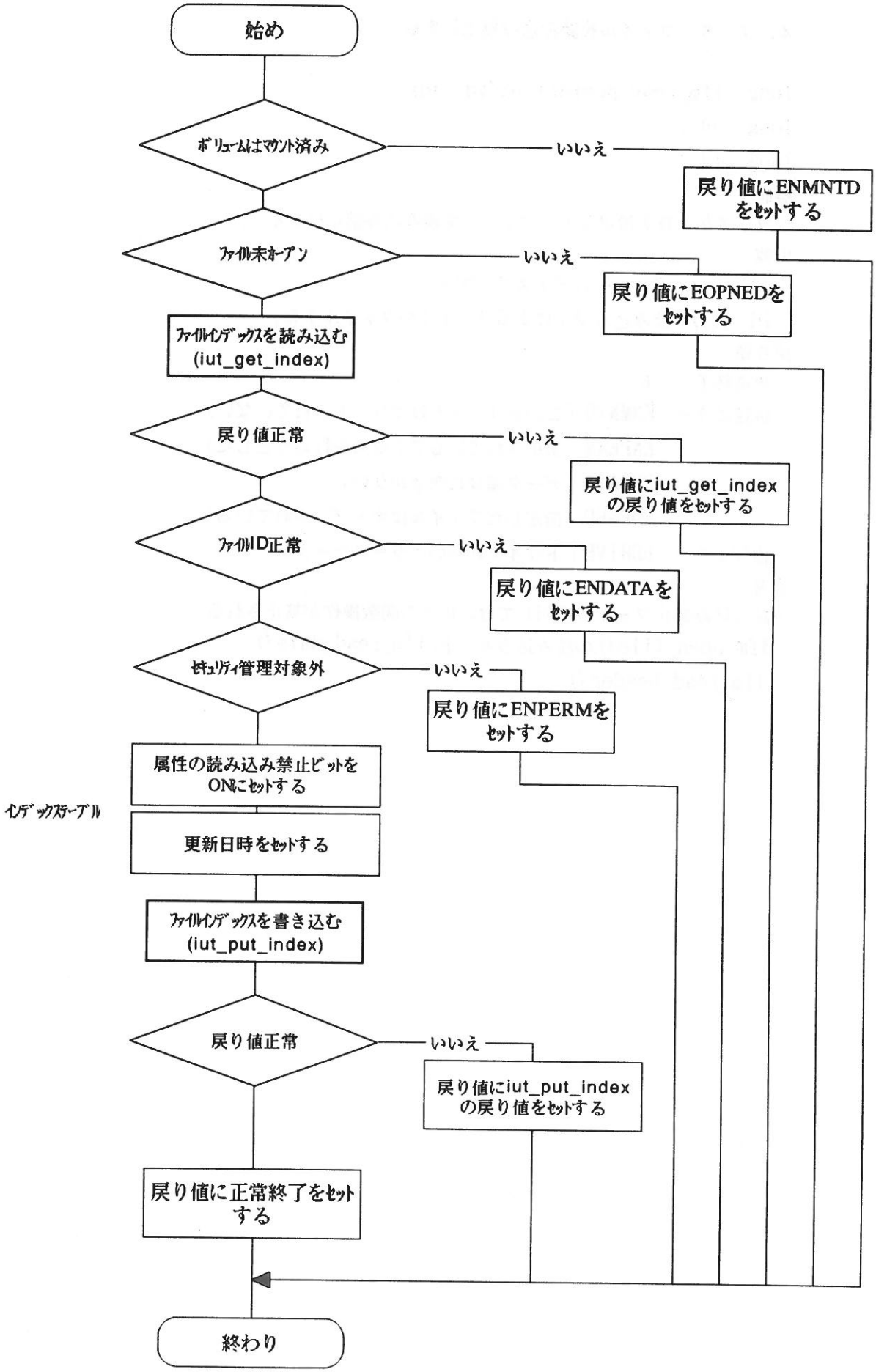
正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

備考

・読み込み禁止ファイルに対しては、以下の関数操作が禁止される。

ifm_open_file()の読み込みモード，ifm_read_data()，
ifm_read_header()

ifm_read_protect_on



関数ifm_read_protect_on()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルの読み込み禁止ビット（インデックステーブルの属性）がＯＮになることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．７　ファイルを読み込み可能にする

```
long   ifm_read_protect_off(vd, id)
long   vd ;
long   id ;
```

機能

ファイルＩＤを指定して、ファイルを読み込み可能にする。

引数

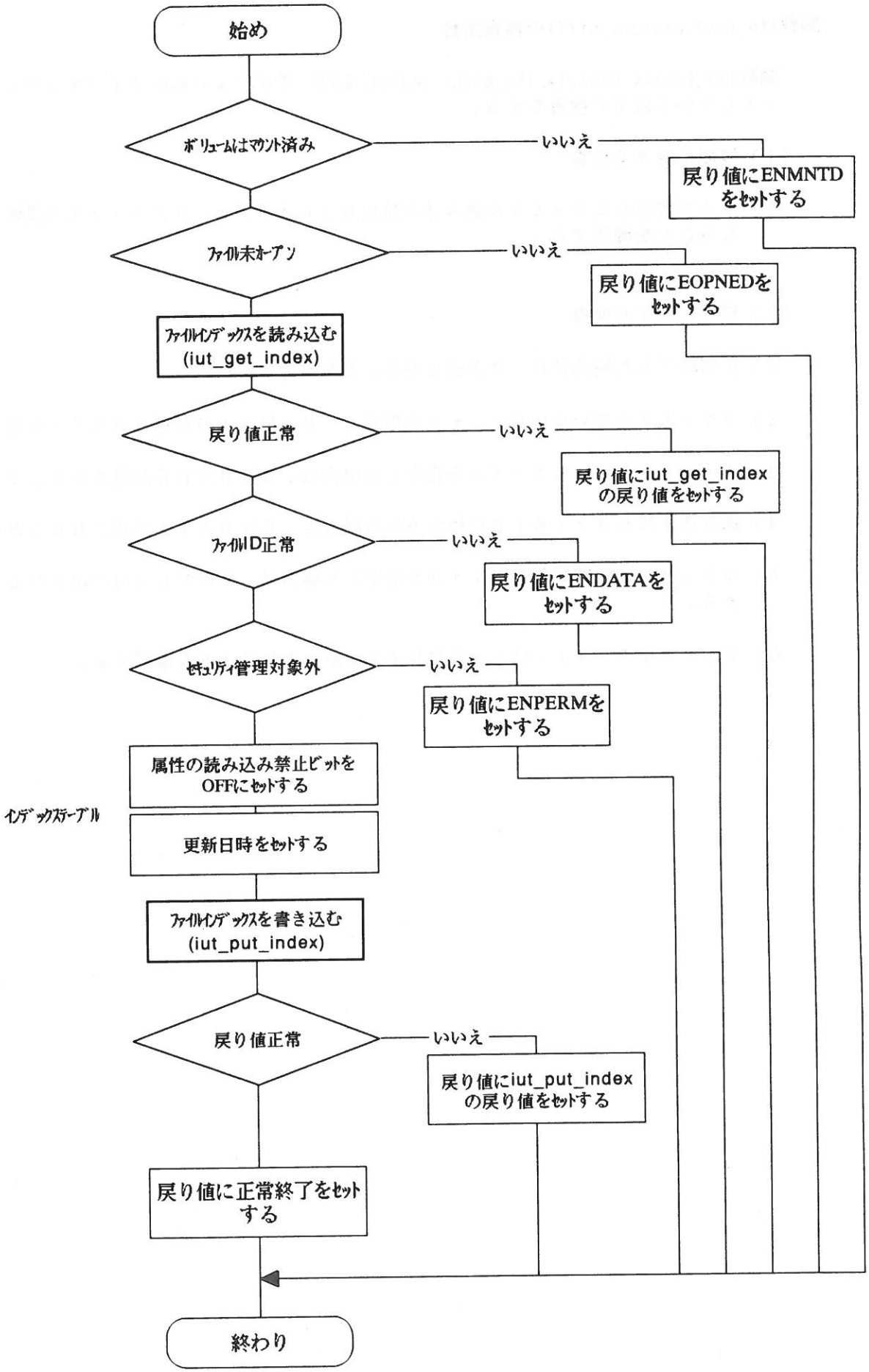
vd（入力）ボリュームディスクリプタ。

id（入力）読み込み可能にするファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	EOPNED（指定したファイルはオープンされている）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_read_protect_off



関数ifm_read_protect_off()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルの読み込み禁止ビット（インデックステーブルの属性）がＯＦＦになることを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．８　ファイルをシステム状態にする

```
long  ifm_system_flag_on(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルをシステム状態にする。

引数

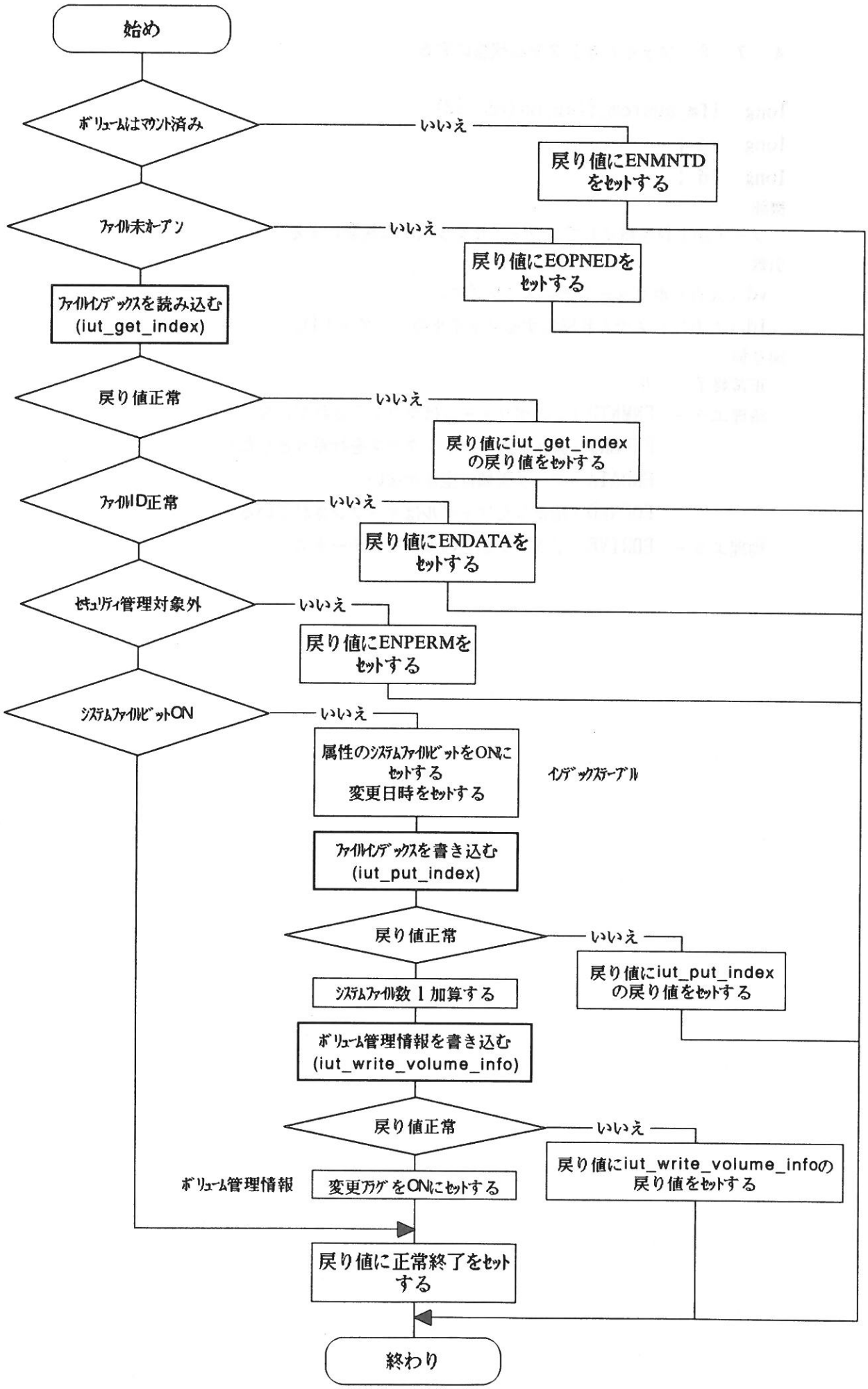
vd（入力）ボリュームディスクリプタ。

id（入力）システム状態にするファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_system_flag_on



関数ifm_system_flag_on()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルのシステムファイルビット（インデックステーブルの属性）がＯＮになることを確認する。

２）ボリューム管理情報部のシステムファイル数が１加算されることを確認する。

３）システムファイルビットがすでにＯＮの場合は、何もしないで正常終了することを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．９　ファイルのシステム状態を解除する

```
long  ifm_system_flag_off(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルのシステム状態を解除する。

引数

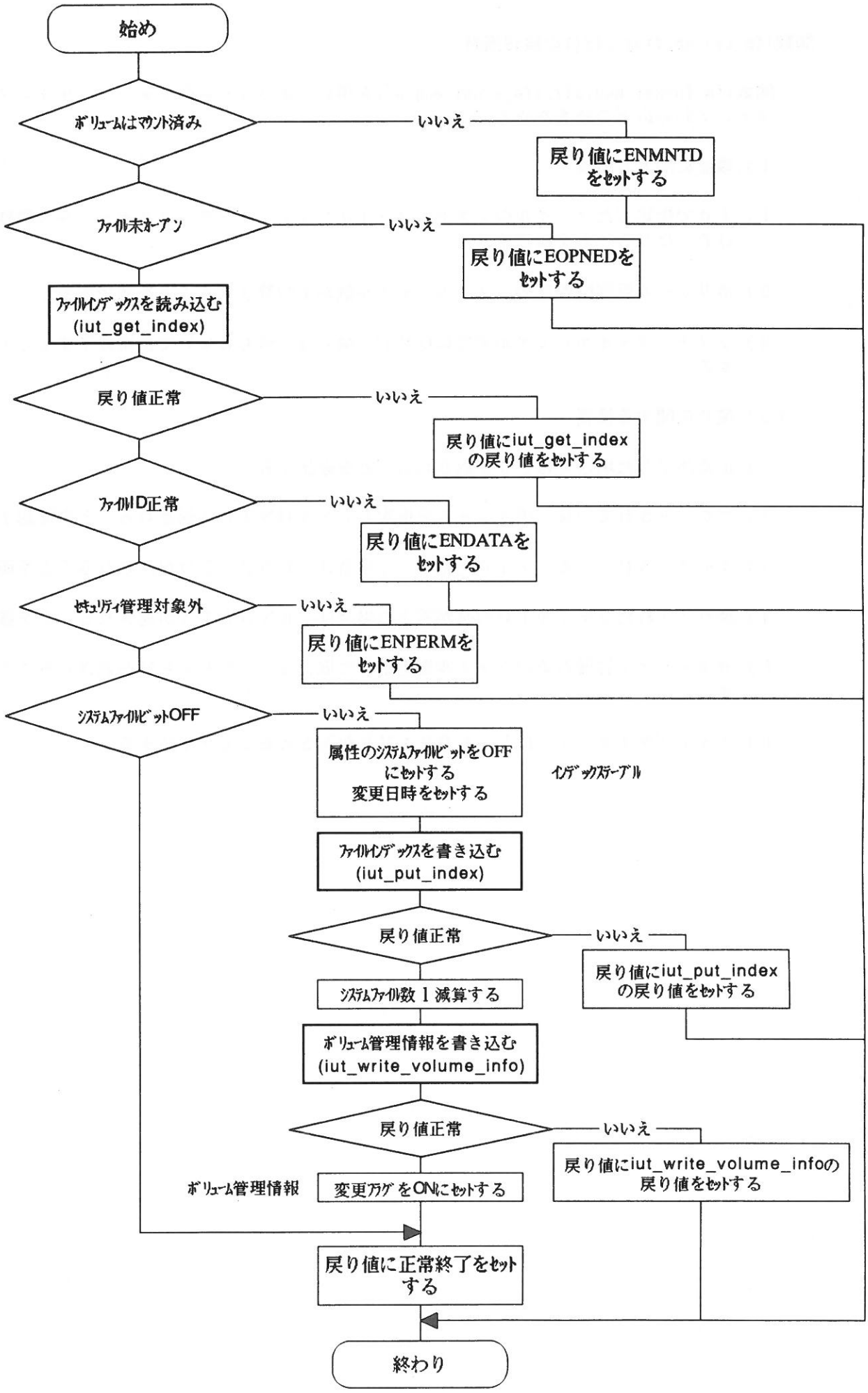
vd（入力）ボリュームディスクリプタ。

id（入力）システム状態を解除するファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	EOPNED（指定したファイルはオープンされている）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_system_flag_off



関数ifm_system_flag_off()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルのシステムファイルビット（インデックステーブルの属性）がＯＦＦになることを確認する。

２）ボリューム管理情報部のシステムファイル数が１減算されることを確認する。

３）システムファイルビットがすでにＯＦＦの場合は、何もしないで正常終了することを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．１０　ファイルをディレクトリ状態にする

```
long  ifm_directory_flag_on(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルをディレクトリ状態にする。

引数

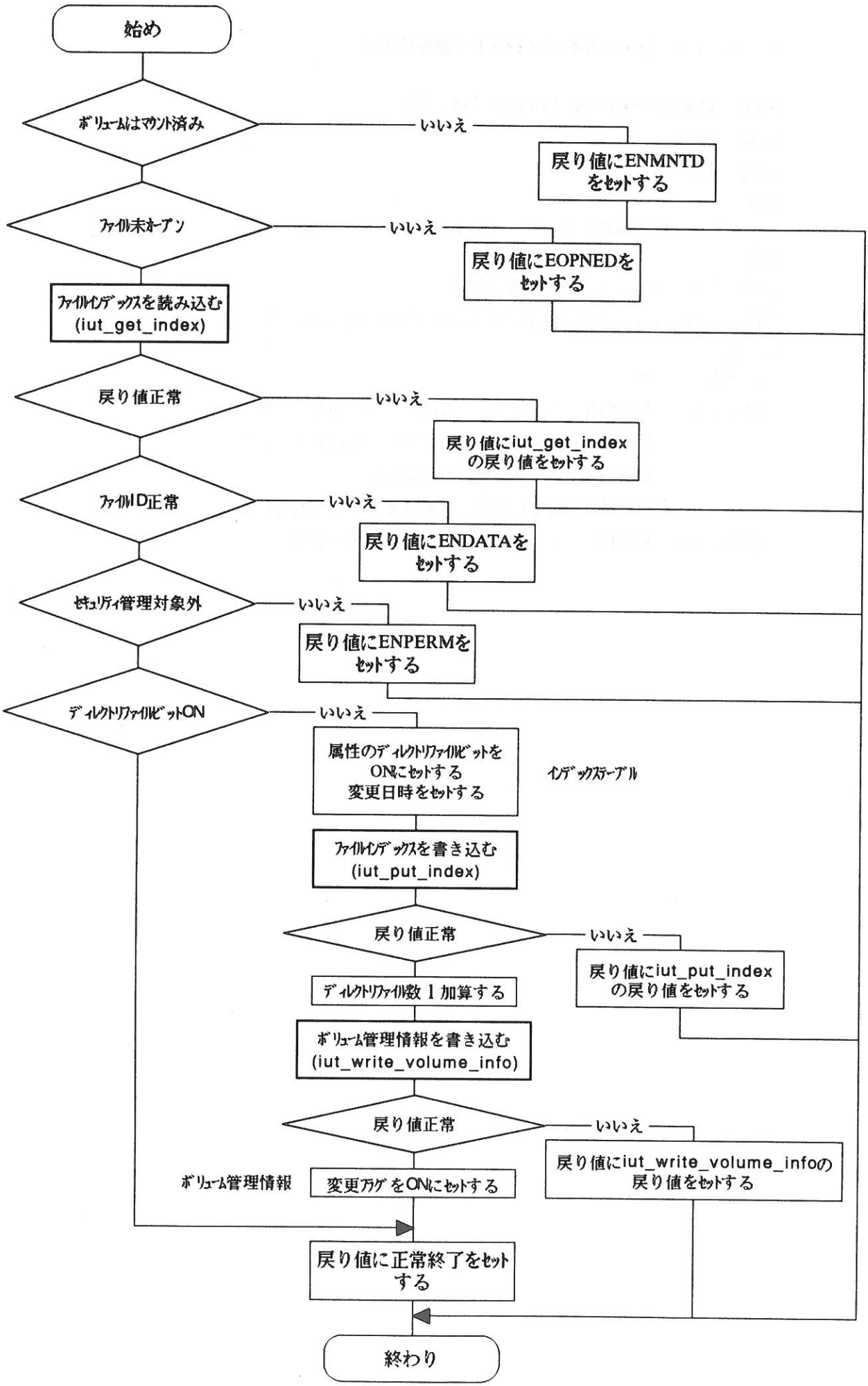
vd（入力）ボリュームディスクリプタ。

id（入力）ディレクトリ状態にするファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_directory_flag_on



関数ifm_directory_flag_on()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（１）機能に関する検査

１）ｉｄで指定したファイルのディレクトリファイルビット（インデックステーブルの属性）がＯＮになることを確認する。

２）ボリューム管理情報部のディレクトリファイル数が１加算されることを確認する。

３）ディレクトリファイルビットがすでにＯＮの場合は、何もしないで正常終了することを確認する。

（２）戻りに関する検査

１）正常終了した場合には、０が返されることを確認する。

２）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

３）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

４）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

５）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

６）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

４．７．１１　ファイルのディレクトリ状態を解除する

```
long  ifm_directory_flag_off(vd, id)
long  vd ;
long  id ;
```

機能

ファイルＩＤを指定して、ファイルのディレクトリ状態を解除する。

引数

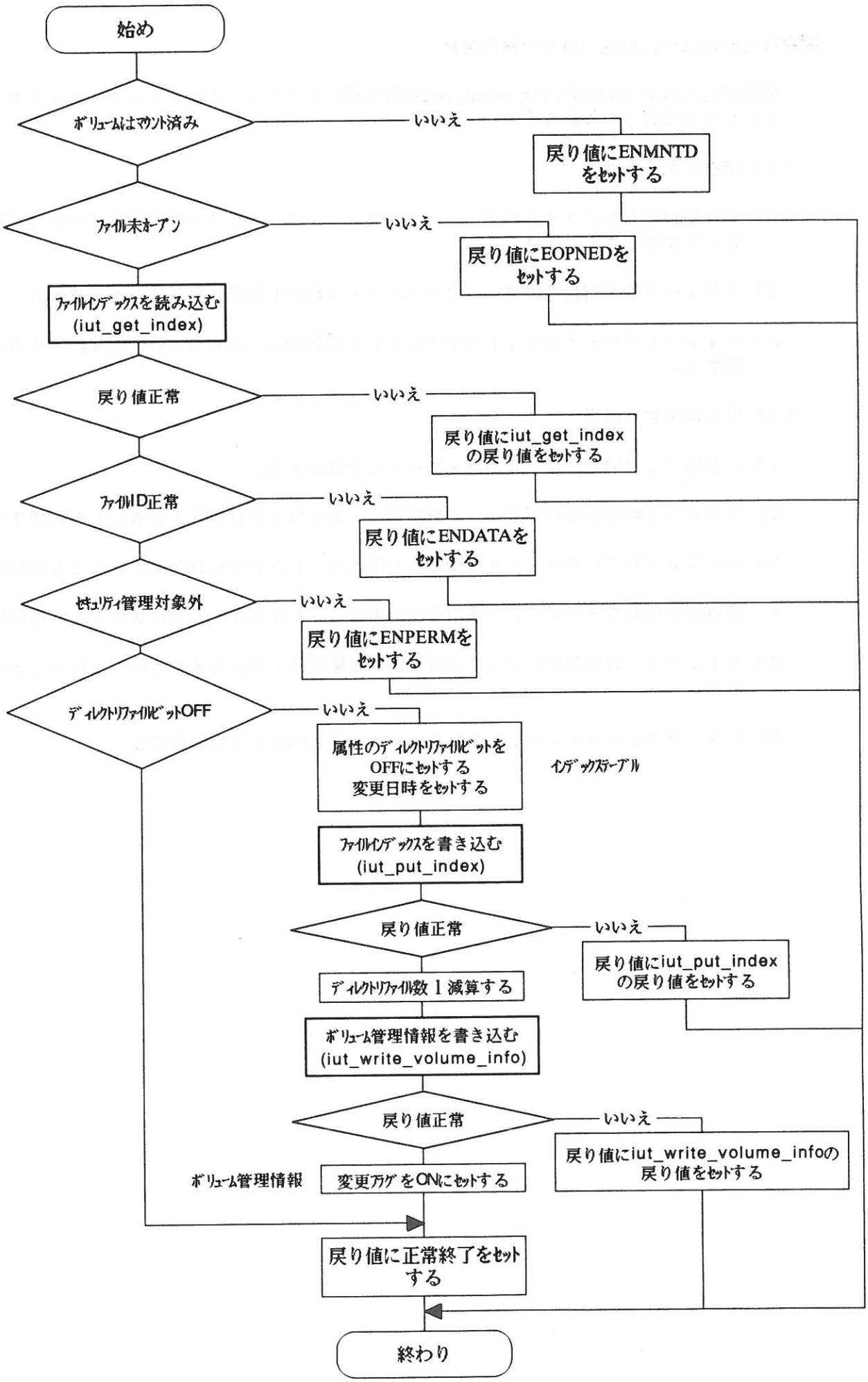
vd（入力）ボリュームディスクリプタ。

id（入力）ディレクトリ状態を解除するファイルのファイルＩＤ。

戻り値

正常終了	0
論理エラー	ENMNTD（このボリュームはマウントされていない）
	ENPERM（禁止されているアクセスを行おうとした）
	ENDATA（データ領域に空きがない）
	EOPNED（指定したファイルはオープンされている）
物理エラー	EDRIVE｜ドライブからのエラーステータス

ifm_directory_flag_off



関数ifm_directory_flag_off()の検査指針

関数ifm_format_media(),ifm_mount_media()を用いてボリュームのフォーマッティング、マウントしてから以下の検査をする。

（1）機能に関する検査

1）ｉｄで指定したファイルのディレクトリファイルビット（インデックステーブルの属性）がＯＦＦになることを確認する。

2）ボリューム管理情報部のディレクトリファイル数が1減算されることを確認する。

3）ディレクトリファイルビットがすでにＯＦＦの場合は、何もしないで正常終了することを確認する。

（2）戻りに関する検査

1）正常終了した場合には、0が返されることを確認する。

2）マウントされていないボリュームを指定し、ＥＮＭＮＴＤが返されることを確認する。

3）オープンされているファイルを指定した場合は、ＥＯＰＮＥＤが返されることを確認する。

4）読み込まれたファイルＩＤの値が不正の場合は、ＥＮＤＡＴＡが返されることを確認する。

5）セキュリティ管理対象のファイルを指定した場合は、ＥＮＰＥＲＭが返されることを確認する。

6）ドライブをオフラインにし、ＥＤＲＩＶＥが返されることを確認する。

5．サービス関数および内部関数のモジュール構造

本関数群の参照関係について説明する。

①ボリューム管理

②ファイル管理

③データ入出力

④ヘッダのシーケンシャル読み込み、システム管理

⑤内部関数

サービス関数	呼び出し関数（サービス、内部関数）
ifm_initial()	
ifm_format_media()	iut_get_time() *1iut_open_devices() *1*2iut_close_devices() *1iut_test_unit_ready() *1iut_read_data() *1iut_lock_media() *2iut_unlock_media() iut_read_mod() *1iut_read_capacity() *1iut_rezero_unit() iut_write_volume_info_0() iut_write_mod() iut_unlock_mod()
ifm_mount_media()	*1iut_open_devices() *1*2iut_close_devices() *1iut_test_unit_ready() iut_read_mod() iut_copy_word() iut_copy_dword() iut_write_volume_info() iut_lock_mod() *2iut_unlock_mod()
ifm_dismaount_media()	*3ifm_close_file() iut_unlock_mod() *1iut_close_devices() iut_write_mod() iut_write_volume_info() iut_get_time() iut_copy_word() iut_copy_dword() iut_read_mod()
ifm_get_space()	

*1　モジュール構造の最下位に位置するデバイスドライバーとのインタフェースルーチン

*2　iut_open_devices()が呼び出されている場合

*3　オープンされているファイルがある場合

②ファイル管理

サービス関数	呼び出し関数（サービス、内部関数）
ifm_get_list()	iut_get_index()
ifm_create_file()	iut_define_zone() iut_get_index() iut_write_volume_info() iut_get_time() iut_get_pointer() iut_put_index()
ifm_create_file_variable()	iut_define_zone() iut_get_index() iut_write_volume_info() iut_get_time() iut_get_pointer() iut_put_index()
ifm_get_file_id()	iut_get_index()
ifm_delete_file()	iut_get_index() iut_read_mod() iut_write_mod() iut_put_index() iut_write_volume_info()

③データ入出力

サービス関数	呼び出し関数（サービス、内部関数）
ifm_open_file()	iut_get_index() iut_put_index()
ifm_close_file()	iut_put_index()
ifm_read_data()	iut_get_index() iut_read_mod()
ifm_write_data()	iut_get_index() iut_get_pointer() iut_put_index() iut_write_volume_info() iut_write_mod()
ifm_read_header()	iut_read_mod()
ifm_write_header()	iut_release_pointer() iut_get_header_pointer() iut_put_index() iut_write_mod()
ifm_get_file_size()	iut_read_mod()
ifm_get_filename()	
ifm_put_filename()	iut_put_index()

④ヘッダのシーケンシャル読み込み、システム管理

サービス関数	呼び出し関数（サービス、内部関数）
ifm_read_header_sequential()	iut_read_mod() iut_copy_word()
ifm_reset_header_counter()	
ifm_get_list_all()	iut_get_index()
ifm_delete_actually()	iut_get_index() iut_release_pointer() iut_write_volume_info() iut_put_index()
ifm_recover_temporaly_deleted()	iut_get_index() iut_read_mod() iut_write_mod() iut_put_index() iut_write_volume_info()
ifm_write_protect_on() ifm_write_protect_off() ifm_read_protect_on() ifm_read_protect_off()	iut_get_index() iut_get_time() iut_put_index()
ifm_system_flag_on() ifm_system_flag_off() ifm_directory_flag_on() ifm_directory_flag_off()	iut_get_index() iut_get_time() iut_put_index() iut_write_volume_info()

⑤内部関数

内部関数	呼び出し関数（内部関数）
iut_write_mod()	iut_byte_swap() *1iut_write_data()
iut_read_mod()	*1iut_read_data() iut_byte_swap()
iut_get_index()	iut_read_mod()
iut_put_index()	iut_read_mod() iut_write_mod()
iut_lock_mod()	*1iut_lock_media()
iut_unlock_mod()	*1iut_unlock_media()
iut_define_zone()	iut_write_mod() iut_write_volume_info()
iut_get_pointer()	iut_define_zone() iut_get_start_sector() iut_write_mod()
iut_get_start_sector()	iut_write_mod()
iut_release_pointer()	iut_write_mod()
iut_get_header_pointer()	iut_write_mod()
iut_copy_word() iut_copy_dword()	
iut_byte_swap()	
iut_write_volume_info()	iut_write_mod()
iut_write_volume_info_0()	iut_write_mod() iut_write_volume_info()

*1　モジュール構造の最下位に位置するデバイスドライバーとのインタフェースルーチン

6．ＩＳ＆Ｃファイルマネージャ使用の手引き

ＩＳ＆Ｃファイルマネージャは、異なるコンピュータにおいても移植性あるコーディングがされています。ここでは、ファイルマネージャを使用するにあたってのイントロダクションを行います。

1．使用可能なコンピュータ

本プログラムは、以下のコンピュータで開発されましたので、以下のコンピュータでは、ソースの変更なく使用可能です。

ＳＵＮ　ｉｐｃ　（ＲＡＭ８Ｍ）
ＳＵＮ　ｓｐ－２　（ＲＡＭ８Ｍ）
Ｄａｔａ－Ｇｅｎｅｒａｌ　ＡＶ４００　（ＲＡＭ２４Ｍ）

また、移植可能なコンピュータは、最低以下のようなスペックが必要です。

＊Little-Endian のＣＰＵ
＊SCSI接続可能
＊char=byte=8bit, short=16bit, long=32bit
＊Ｋ＆Ｒコンパイラを持つ
＊２４文字以上の変数、関数、define定数を扱えるコンパイラ

現状では、以上の点が必要なことがわかっています。今回は、コンピュータ依存で問題となる境界問題はおきないようなコーディングとなっています。

2．インストール

インストールは以下の手順を追っておこないます。

１）フロッピーディスクからハードディスクへのファイルのコピー
２）ＳＣＳＩドライバの作成
３）コンピュータ依存部のソース変更
４）コンピュータの属性変更
５）コンパイル

１）フロッピーディスクからハードディスクへのファイルのコピー

フロッピーディスクの内容をコンピュータ内の適当なディレクトリにコピーしてください。

２）ＳＣＳＩドライバの作成

ＳＣＳＩドライバのないコンピュータは、ＳＣＳＩドライバを作成する必要があります。最低限、ＩＳ＆ＣファイルマネージャがコールするスＣＳＩコマンドはサポートする必要があります。

Ｓｕｎに関しては、ＩＳ＆Ｃ委員会から別途ＳＣＳＩドライバを提供することができます。

３）コンピュータ依存部のソース変更

ＳＣＳＩインターフェース部とのやり取りをする部分の変更が必要です。
詳細は別項の「ＩＳ＆Ｃファイルマネージャの移植手順」に書かれています。

４）コンピュータの属性変更

isac pr.h 内の

Ｎ－ＶＯＬ　同時に扱えるＭＯの数

Ｎ－ＦＩＬ　同時にオープンできるファイル数

の大きさを、主記憶の大きさに合わせて変更しなければならないコンピュータが在ります。主記憶の量が問題となるコンピュータにおいては、この数値を少なくする必要があります。

また、スタックサイズの使用量が実行するプログラム毎に限られている様なコンピュータでは、スタックサイズを大きくしなくてはなりません。Unixでは、limit コマンドを用います。

５）コンパイル

ｍａｋｅを実行することにより、コンパイルが実行されます。

ＩＳ＆Ｃを正式にサポートするためには、

isac　pr.h内において

#define　ＢＡＣＫＵＰ　ＷＨＥＮ　ＤＩＳＭＯＵＮＴ

とするか、コンパイラオプションのシンボル定義

－ＤＢＡＣＫＵＰ　ＷＨＥＮ　ＤＩＳＭＯＵＮＴ

を利用してシンボルを定義する必要があります。

これにより、規格通りのdismount時にＡゾーンがバックアップゾーンにバックアップされます。しかしながら、現状の光磁気ディスクドライブではこのバックアップにかなりの時間がかかるため次のような対策を講じたプログラムも作成できるようになっています。

即ち、このシンボルを定義しないようにしてコンパイルを行うと、必要に応じてバックアップゾーンにも整合性良くデータを書込むようになり、dismountをほとんど時間をかけずに行うことができます。しかしながら、この方法では、バックアップゾーンにもmount 後にデータを書いてしまうため、プログラムの暴走などによりディスクの内容に整合性が保たれなくなった場合、バックアップゾーンからの起動ができなくなる恐れがあります。

３．サンプルプログラム

ファイルマネージャサービス関数は、「ＩＳ＆Ｃディスクフォーマット規格書」にその使用法が、また「ファイルマネージャ仕様説明書」に、サービス関数の詳細な仕様が記載されています。ここでは、ファイルマネージャに追加して提供されているサンプルプログラムにより、移植が完全に行われているかの確認を行うことができます。また、サービス関数の使用法についても、これらのプログラムを参考にできます。

isac_fmt
光磁気ディスクを初期化する。

fm_test1
ハードディスクからファイルを読み光磁気ディスクに書き込む。

fm_test3
光磁気ディスク上にあるファイルにヘッダを書き込み、それを読み出して、ヘッダデータの読み書きの整合性をチェックする。

fm_test4
ハードディスクからファイルを読み光磁気ディスクに書き込み、その後光磁気ディスクからデータを読み出して、ハードディスク上のデータとデータの整合性をチェックする。

fm_test5
ファイルＩＤの取得、ボリューム管理情報の取得、データ、ヘッダサイズの取得、ファイル名からＩＤを取得、ファイル名の変更、仮削除、仮削除からの復活、ファイル属性の変更。（gt_fid: ファイルＩＤの取得では、＊と？のテンプレートを用いることができる）

fm_test6
fm_test3で書かれたヘッダをsequential Read し、ヘッダデータの整合性をチェックする。

fm_test9
同時にオープンした複数のファイルの読み書きのチェックを行う。ひとつのファイルをオープンしダミーデータを書き、もうひとつのファイルをオープンしてコピーする。このファイルを読みだし、データの整合性をチェックする。

fm_test10
可変サイズファイルを二つ作成し、両者にダミーデータを交互に書き込み、この二つのファイルをそれぞれ読みだし、データが整合性良く記録されているかをチェックする。

fm_test の後に続く番号が不連続なのは、サンプルプログラムの内容がほぼＶ０．９の時のものを継承しているからです。

７．ＩＳ＆Ｃファイルマネージャの移植手順について

1.概　要

移植性を高めるためＩＳ＆Ｃファイルマネージャー内では、ＳＣＳＩインタフェースドライバーとのやり取りは"spctrl.c"という一つのモジュールで行っています。

移植を行うユーザーは、この"spctrl.c"内の各関数を自己のシステムに合わせて書き換える必要があります。

修正の必要なファイルは"spctrl.c"と"isac_pr.h"の二つです。

ここでは、移植のための概略的な説明をしています。

2."isac_pr.h"の変更

(1) 提供プログラムでは"#define N_VOL　4"と定義されていますが、この値を自己のシステムで可能な最大数に合わせて下さい。
この値は、接続可能な最大数で良く実際にドライブが接続されていなくてもかまいません。
（但し、ＳＣＳＩのドライバーの都合で接続されていないドライブにアクセスを許されない場合は接続されている値になります。通常は大丈夫なはずです）

(2) 異なるコンピュータ上でファイルマネージャを稼働する為、アライメントが起きない様に以下の部分を変更する必要があります。
isac_pr.h において、struct fatheridxtype内のson_noの、構造体の先頭からのアドレスとstruct　sonidxtypeのnxt_noの、構造体の先頭からのアドレスを一致させる必要があります。
以下にその１手法を示します。適当なサンプルプログラムで、
sizeof(struct fatheridxtype)とsizeof(struct　sonidxtype)をprintfにて表示し、構造体の大きさを見ます。両者が同じ場合は問題ありません。
両者に差がある場合は、小さいほうの構造体にchar型のdummy[ ]を入れます。
標準ファイルマネージャを開発したsunにおいては、fatheridxtypeのほうがsonidxtypeよりも８バイト小さいので、struct fatheridxtypeの

```
long　son_no;
```

の前に

```
char　dummy[8];　　/* For adusting the structure size */
```

を入れています。sonidxtypeの方が小さい時は

```
long  nxt_no;
```
の前に同様にして入れます。
この方法は、sizeof()が一致すれば、struct fatheridxtype内のson_noとstruct sonidxtypeのnxt_noの位置が一致するコンピュータにおいて可能です。両者の構造体の大きさが一致しても、son_noとnxt_noのアドレス位置が違う様なコンピュータにおいては、利用出来ません。何らかの方法でアドレス位置を一致されて下さい。いずれにせよ、一致させる為には、dummyのchar 文字列を入れる方法を取ります。

```
/*  インデックステーブル（親）  */
struct fatheridxtype {
        long  id ;              /*  ファイルＩＤ                */
        char  fn[24] ;          /*  ファイル名                  */
        struct datetype
              crtd,             /*  作成日時                    */
              updd ;            /*  最終変更日時                */
        long  fsz ;             /*  ファイルバイト長            */
        char  attr,             /*  属性                        */
              attr1 ;
        struct idxptrtype
              head,             /*  ヘッダポインタ              */
              ptr[10] ;         /*  ポインタ                    */
        char  dummy[8] ;        /*  ダミー                      */
        long  son_no ;          /*  子のインデックステーブル番号  */
} ;
/*  インデックステーブル（子）  */
struct sonidxtype {
        long  tblno,            /*  インデックステーブル番号      */
              fatno ;           /*  親のインデックステーブル番号  */
struct idxptrtype
              ptr[17] ;         /*  ポインタ                    */
        long  nxt_no ;          /*次の子のインデックステーブル番号*/
} ;
```

3."spctrl.c"の変更

(1) スタティック変数"fd_mo[]"の初期値の変更

```
static long  fd_mo[] = {-1, -1, -1, -1};
```

と４ドライブ分初期値が定義されていますが、これを前述の"N_VOL"で定義した値と同じ数だけ"-1"で初期化して下さい。

(2) "iut_open_devices"の変更

```
static char  *devname[] = {  "/dev/isac0",
                             "/dev/isac1",
                             "/dev/isac10",
                             "/dev/isac11" };
```

を変更して下さい。unix系ではドライバーの書き方によってこの名前が変わりますので、その名前に合わせて下さい。

また、unix系とは異なりSCSI上のドライブがファイルシステムとしてアクセスできない、直接制御しているようなシステムではこの変数は不要となります。（以後、この様なシステムを直接制御タイプと表します）
さらに直接制御タイプのシステムでは、

```
fd_mo[ iut ] = open( devname[ iut ], O_RDWR );
```

となっている部分の、"open"関数をドライブの接続を確認し接続されていたら"0L"、接続されていなかった場合は"ERDRIVE"となるリターン値を返すような関数と置き換えて下さい。

(3) "iut_close_devices"の変更

直接制御タイプの場合のみ変更が必要です。
通常は、"close"関数を、ドライバーのドライブリリース処理を行う関数と置き換えて下さい。この場合、その関数のリターン値は正常終了時は"0L"、異常終了時は"ERDRIVE"を返すようにして下さい。

(4) "iut_read_data"及び"iut_write_data"の変更

unix系では、システムのドライバー中のioctl部に機能追加が必要で、ここでのioctlは、システムドライバーにこの後のread/writeアクセスするセクターアドレスを通知するためのもので実際の制御は不要です。
従って、直接制御タイプの場合にも同様の機能の関数と置き換えれば良いでしょう。但し、read/writeを実行する関数で直接IO開始セクターアドレスを指定できるならば、ioctlをコールしている所から6行は不要となります。
さらに、直接制御タイプでは"read"及び"write"関数に相当する関数と置き換えて下さい。IOの実行は"stb"セクターアドレスから"m"バイト分データを入出力するものです。正常終了時は"0L"、異常発生時は非零の値をリターン値として返すようにして下さい。

(5) "iut_test_unit_ready"、"iut_rezero_unit"、"iut_read_capacity"、"iut_unlock_media"、"iut_lock_media"、"iut_error_sense"

これらの関数は、unix系ではioctlを通じてドライブの情報を得たり、状

態を制御するためのものです。

iut_test_unit_ready	ドライブに"test unit ready"コマンドを送りドライブユニットがready状態か否かをioctlのリターン値として得ます。
iut_rezero_unit	"rezero"コマンドを送りヘッド位置を零ポジションに戻します。
iut_read_capacity	"read capacity"コマンドを送りメディアの容量情報を内部のストラクチャー変数read_capに得ます。
iut_unlock_media	"media lock/unlock"コマンドを送りメディアの排出禁止を解除します。
iut_lock_media	"media lock/unlock"コマンドを送りメディアの排出を禁止します。
iut_error_sense	"read extended sense data"コマンドを送り拡張センスデータ情報を内部のストラクチャー変数sense_dataに得て、その中のセンスキーコードからＩＳ＆Ｃエラーコードに直しています。

unix系では、上記の機能に対応したファンクション番号に合わせてシステムのドライバーのioctl部に各機能を付加する必要があります。
ioctlのリターン値は正常終了時は"0L"、異常発生時は非零の値を返します。
直接制御タイプでは、上述の機能に対応した関数に置き換える必要があります。ストラクチャー変数の内容と意味は、各SCSIコマンドに対するSCSIの説明書を参照して下さい。

(6) "iut_get_time"の変更

この関数はシステムのカレンダー情報から、現在の西暦年・月・日・時刻・分の情報を得るものです。
unix系では変更は不要でしょう。
直接制御タイプの場合は、システムカレンダ情報を得るような関数がたぶんあると思われますので、それと置き換えて下さい。

4．ボリューム使用中フラグ－ライトプロテクト－排他制御

（1）異状監視用のボリューム使用中フラグは、ディスクのマウント時にセットされるため、カートリッジのライトプロテクトスイッチが入ると、読み出し専用処理もできません。

（2）これはマルチタスクの場合に同時にアクセスされないように、排他制御フラグとしても使用しているためです。

（3）ファイルマネジャ側の排他制御が不要でライトプロテクトスイッチを生かす場合は、最初の書込み発生時にボリューム使用中フラグを立てるように変更してください：書込みは全て、iut_write_mod() を介して行われています（モジュール構造参照）。

８．提供フロッピーのプログラムの説明

（★ＳＵＮ用のテキストファイル・ｆｏｒ　ｔａｒ　ｆｏｒｍａｔ）

１．　Ｍａｋｅｆｉｌｅ　……コンパイル用ファイル

２．　ｉｓａｃ＿ｐｒ．ｈ　……変数・定数・テーブルの定義

３．　ｄａｔａ＿ｉｏ１．ｃ……ファイルの open/close/read/write
①ｉｆｍ＿ｏｐｅｎ＿ｆｉｌｅ
②ｉｆｍ＿ｃｌｏｓｅ＿ｆｉｌｅ
③ｉｆｍ＿ｒｅａｄ＿ｆｉｌｅ
④ｉｆｍ＿ｗｒｉｔｅ＿ｆｉｌｅ

４．　ｄａｔａ＿ｉｏ２．ｃ……ヘッダのread/write、ファイルのシステム管理
①ｉｆｍ＿ｒｅａｄ＿ｈｅａｄｅｒ
②ｉｆｍ＿ｗｒｉｔｅ＿ｈｅａｄｅｒ
③ｉｆｍ＿ｇｅｔ＿ｆｉｌｅ＿ｓｉｚｅ
④ｉｆｍ＿ｇｅｔ＿ｆｉｌｅｎａｍｅ
⑤ｉｆｍ＿ｐｕｔ＿ｆｉｌｅｎａｍｅ

５．ｆｍ＿ｔｅｓｔ１．ｃ

６．ｆｍ＿ｔｅｓｔ２．ｃ

７．ｆｍ＿ｔｅｓｔ３．ｃ

８．ｆｍ＿ｔｅｓｔ４．ｃ

９．ｆｍ＿ｔｅｓｔ５．ｃ

１０．ｆｍ＿ｔｅｓｔ６．ｃ

１１．ｆｍ＿ｔｅｓｔ７．ｃ

12．fm_md．c……ディスクの初期化

①ifm_format_media

②iut_write_volume_info_0

13．head_io．c……ヘッダデータの読み出し

①ifm_read_header_sequential

②ifm_reset_header_counter

14．isac_fm．c……ファイル管理

①ifm_get_list

②ifm_create_file

③ifm_create_file_variable

④ifm_get_file_id

⑤ifm_delete_file

15．isac_fmt．c……メディアの初期化

16．isac_sys．c……ファイルのシステム管理中・属性操作

①ifm_get_list_all

②ifm_delete_actually

③ifm_recover_temporaly_deleted

④ifm_write_protect_on

⑤ifm_write_protect_off

⑥ifm_read_protect_on

⑦ifm_write_protect_off

⑧ifm_system_flag_on

⑨ifm_system_flag_off

⑩ifm_directory_flag_on

⑪ifm_directory_flag_off

17．isac_util．c……内部関数

①iut_write_mod

②iut_read_mod

③iut_get_index

④iut_put_index

⑤iut_lock_mod

⑥iut_unlock_mod

⑦iut_define_zone
⑧iut_get_pointer
⑨iut_get_start_sector
⑩iut_release_pointer
⑪iut_get_header_pointer
⑫block size of zone
⑬iut_copy_str
⑭iut_copy_word
⑮iut_copy_dword
⑯iut_byte_swap
⑰iut_write_volume_info

18. isac_util.c……ディスクの管理
①ifm_mount_media
②ifm_dismount_media
③ifm_get_space
④ifm_initial

19. spctrl.c……デバイスドライバーとのインターフェース

（問い合わせ先）

本説明書についての問い合わせは、下記にお願い致します。

ＩＳ＆Ｃ委員会事務局

㈶医療情報システム開発センター　　研究開発部内

〒107　東京都港区赤坂２－３－４　ランディック赤坂ビル10Ｆ

TEL　０３（３５８６）６３２１　　FAX　０３（３５０５）１９９６

ＩＳ＆Ｃ標準ファイルマネージャ仕様説明書

平成４年８月発行

発　行　　財団法人 医療情報システム開発センター

港区赤坂２－３－４　ランディック赤坂ビル10Ｆ

TEL　０３（３５８６）６３２１