ＩＳ ＆ Ｃ規格書

Image Save And Carry

―ディスクフォーマット規格書―

＜V1.0＞

平成３年12月

(財)医療情報システム開発センター・日本PACS研究会

ＩＳ ＆ Ｃ委員会

Ｉｍａｇｅ Ｓａｖｅ Ａｎｄ Ｃａｒｒｙ（ＩＳ&Ｃ）
ディスクフォーマット／ファイルマネージャ規格書（第一版）

１．序文

ＩＳ＆Ｃシステムの目的は、その頭文字が Image Save And Carry の略であるように、可搬型大容量記録媒体を用いたオフラインシステムの構築である。このシステムは、現在広く普及しているワードプロセッサと同じ感覚で、大容量の画像を含めた種々の大きさのデータを効率よくファイリングし、持ち運ぶことを可能とするもので、メディアを要としたオープンシステムを目指している。　したがってＩＳ＆Ｃシステムを実現するためには、以下の３項目に関する標準化を行うことで、メディアの完全な互換性を確立しなければならない。

①　メディアの物理フォーマット
②　ボリュームファイルフォーマット（ロジカルディスクフォーマット）
③　アプリケーションフォーマット　（データフォーマット）

ＩＳ＆Ｃ委員会の活動を開始する時点（１９８９年４月）では、①の項目以外は全く標準化作業が行われていなかったために、本委員会に２つのワーキンググループを設立し、②をＷＧ１が、③をＷＧ２が必要な作業を行うことにした。　ＷＧ１が担当したボリュームファイルフォーマットは、ＩＳ＆Ｃシステムの最も重要な応用である医療分野のみならず広く使われることを想定して、既存のＯＳに依存しない仕様を作成した。

作成時に考慮した点は以下の通りである。

(1) 記録されるファイルの大きさは、数バイトから数メガバイトまで様々である。
(2) 大容量データの高速入出力を可能とするために、連続セクタの使用を可能とする。
(3) 記録されるファイルの大きさに従って、連続セクタ数が最適になるように、セルフチューニングされる。
(4) 記録消去を繰り返しても、ガーベージコレクション無しで十分な性能が得られる。
(5) 階層構造等を構築するためのディレクトリを定義可能とする。
(6) 医療応用では、ファイル保護とセキュリティ管理が必要となるため、ＩＳ＆Ｃシステム内部においても保護機構を設ける。

等である。

現在までに実質作業時間は約２年であり、多くの委員の方々の努力により、本仕様を作成できたことは、誠に喜ばしいことである。特に本仕様書は、暫定仕様を用いた実験により得られた各種の経験を反映させており、本仕様書に携わった委員の自信作であると自負している。今後は、本仕様書がより広く普及するために、さらに発展することを強く希望する。

２．ＩＳ＆Ｃフォーマットの概念説明

２．１．ファイルマネージャの位置づけ

光磁気ディスクを交換媒体とし、複数メーカーの画像機器及び複数サイト間での画像を含む各種情報の交換を実現することを想定して、以下の機能レイヤーを設定した。（図2.1.参照）

(1) アプリケーションレイヤー

実際の医用画像を使用する場合の業務処理を規定するレイヤーであり、以下の機能が包含されている。

・会話制御

オペレータが見易いように医療情報を配置し、オペレータとの会話を司る。

・業務処理

実際の業務処理を定義する部分である。

・複数データファイル間の関係管理

オペレータが必要とする医療データの複数の関係および相互の関係を維持管理する部分であり、業務処理内に組み込まれることも多い。ＩＳ＆Ｃでは、ディレクトリ機能はアプリケーションにゆだねられている。

(2) データフォーマット管理レイヤー

アプリケーションレイヤーに対して、各種医療データのデータ形式を規定してアプリケーションのポータビリティを保証すると同時に、光磁気ディスク内の各データファイルの内容が複数メーカーの複数の医用画像システム間で互換となることを規定するレイヤーである。

(3) ディスクフォーマット管理レイヤー（ファイルマネージャ）

各データファイルへの必要スペースの割当や不要となったスペースの削除等の光磁気ディスク内の領域を管理するレイヤーであり、データファイルの構造が複数メーカーの複数の医用画像システム間で互換となることを規定するレイヤーである。このレイヤーをプログラムとして実現しているものがファイルマネージャであり、通常計算機システムにおけるファイル管理機能に相当する。

本仕様書は、これらのディスクフォーマットとファイルマネージャについて記述している。

(4) ドライバーソフト（デバイスドライバー）のレイヤー

上位のアプリケーションレイヤー、データフォーマット管理レイヤー、ディスクフォーマット管理レイヤーと異なって、このレイヤーはパソコンやワークステーションの種類、メーカ別のＯＳ、ハードに依存する。というのは、コンピュータ本体と外部装置とのデータ入出力、ＳＣＳＩコマンドの応答時間の監視、エラー処理などはＯＳに規定されるからである。

このレイヤーからドライブを駆動するＳＣＳＩコマンドが発行されるが、コマンド名称、アドレス、各種フラグ、データ長などがビット列のコマンドテーブルとして渡されなければならず、取扱いが複雑である。そのため、上位のファイルマネージャから関数ルーチンの形式でＳＣＳＩコマンドを使用できるようにしているのが、デバイスドライバーと呼ばれているドライバーソフトである。

(5) ＳＣＳＩインターフェイスのレイヤー

コンピュータ本体とドライブとの接続はＡＮＳＩ準拠のＳＣＳＩインターフェースであるが、ＩＳ＆Ｃではグループ１のコマンドを主体にコマンドを限定して互換性を確保する予定である。

(6) 光磁気ドライブとディスクのレイヤー

このレイヤーについても、ＩＳ＆Ｃ仕様を定め、互換性を確保する予定である。

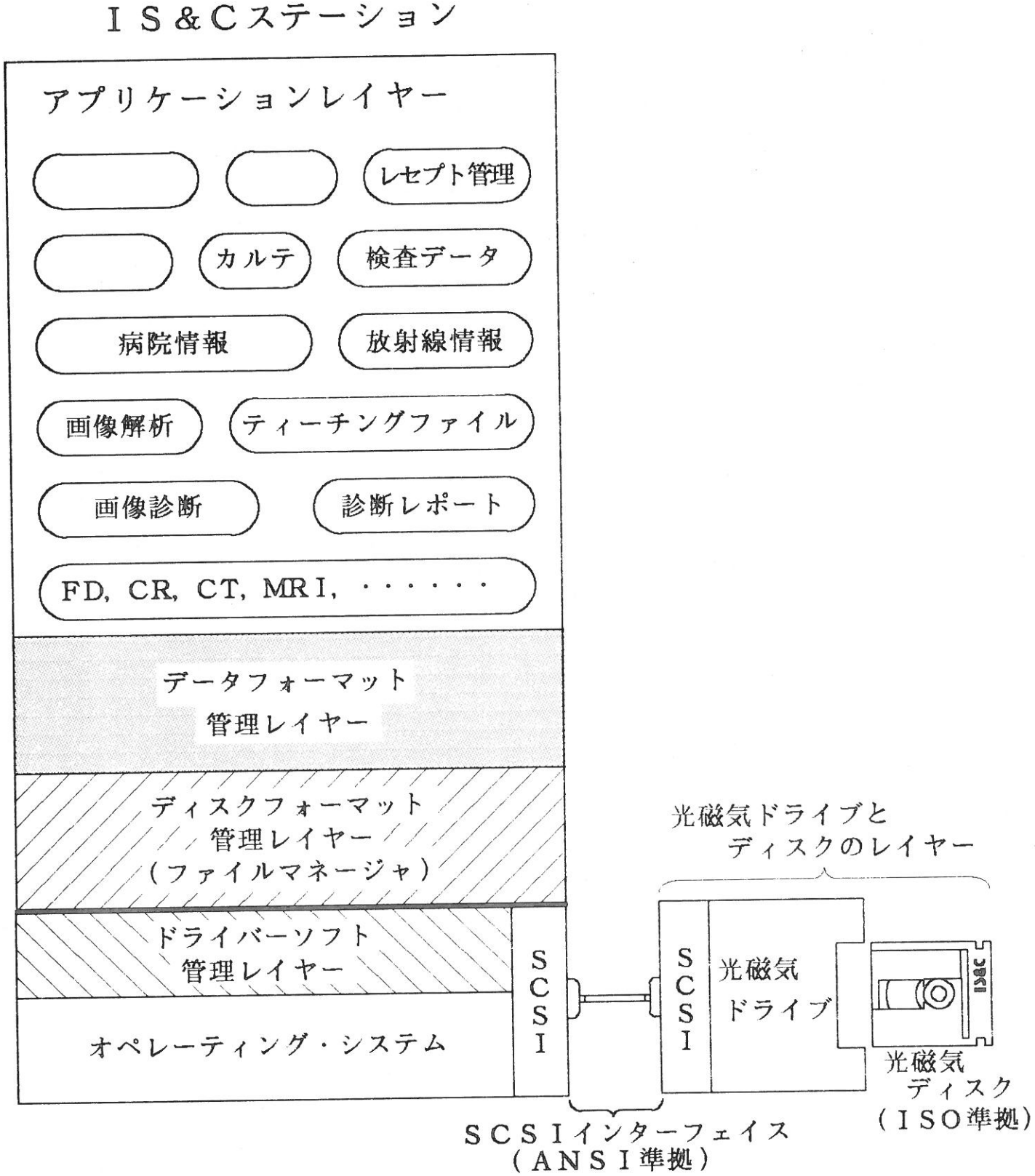


図２．１．ＩＳ＆Ｃの機能レイヤーの構成

2．2．フォーマット設計方針

大容量で可変長のデータをできるだけ効率良く記録し、高速に入出力でき、しかもＯＳに依存しないファイルマネジメントをすることを狙いとして、以下の方針の基で設計した。

(1) ヘッドの移動回数を減らす連続セクタの処理

光磁気ディスクは、螺旋（スパイラル）状のトラックに記録されるが、記録領域はセクタと呼ばれる単位に分割されている。図2.2.1.にＩＳＯで標準化されている１３０㎜光磁気ディスクのセクタ単位の物理フォーマットを示す。先頭にセクタマークと３個のＩＤ（安全のために、３重にトラック番号とセクタ番号がある）があり、データ領域がつづく。データ領域は、１０２４バイトの場合と５１２バイトの場合があるが、いずれもエラー訂正コードや同期信号が付加されるのでさらに長くなっている。ドライブではこの物理フォーマットに従ってセクタ単位の書込み／読出し処理を行っている。ところが光磁気ディスクはヘッドが大きいためハードディスクに比べるとアクセス速度が遅い。また、書込みの場合は消去、書き込み、確認と３回の走査が必要である。したがって、処理時間を短縮するためにはヘッドの移動時間をいかに少なくするかが重要であり、そのためにはできる限り連続セクタにデータを書込む必要がある。

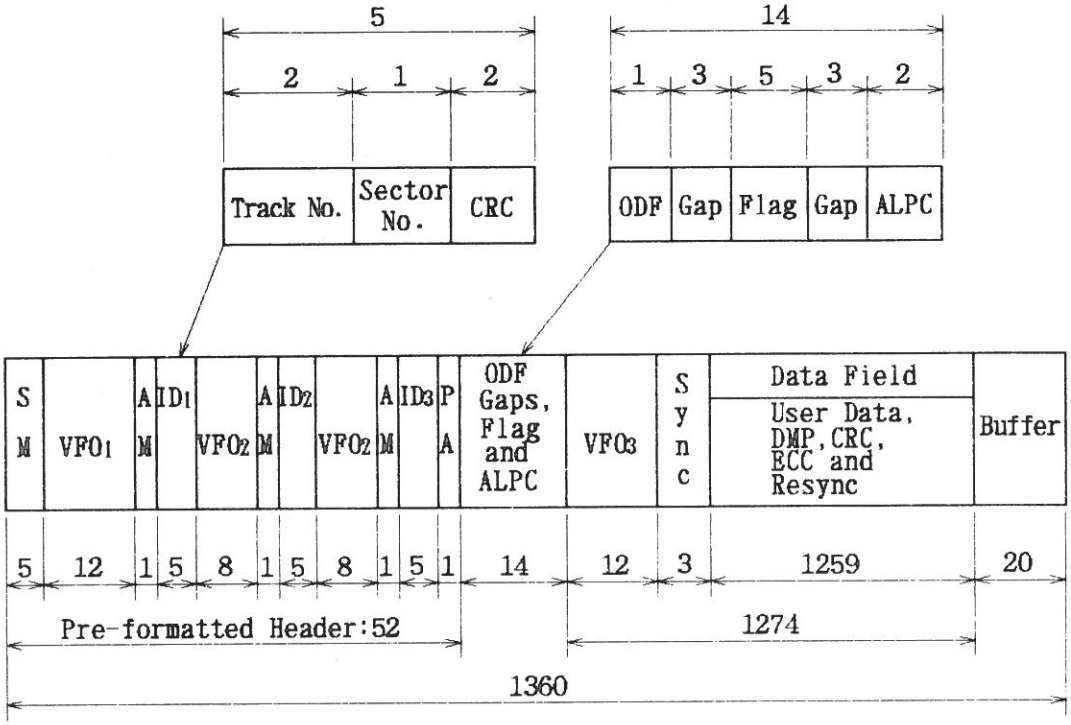
(2) 入出力回数を減らす長いセクタ長の採用

画像データは数メガバイト単位の大容量に及ぶことがあるので、セクタ長をできる限り長くした方が入出力回数が減って有利である。他方、画像データを説明する付属情報やファイルの管理情報はコードデータなので、あまり長くするとエリアが無駄になる。

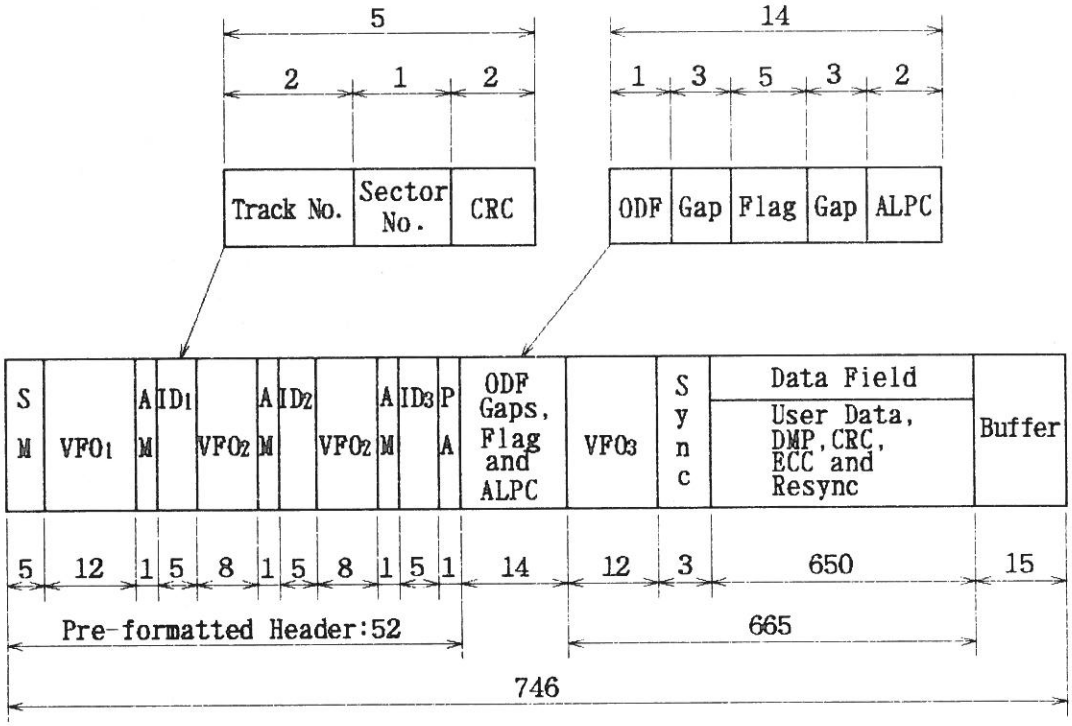
ＩＳ＆Ｃでは、画像データの処理を優先して１０２４バイト／セクタを採用している。

(3) ガーベジコレクションを不要とする定型長ブロックによるエリア確保

追記型の場合は単純に常に連続エリアに追加書きしていくだけで良いが、書換型の場合は削除／再書込み可能なので、その結果として虫食い状態になったエリアを有効に使う工夫が必要である。ＵＮＩＸのエリアの割当て単位は５１２、１０２４または４０９６バイトのブロックであり、またＭＳ－ＤＯＳの場合はセクタを言い替えたクラスターがエリアの割当て単位になっている。このように、ＵＮＩＸとＭＳ－ＤＯＳではいずれもコードデータ用の短い固定長ブロックを採用している。これに対してＩＳＡＣでは画像データが大容量可変長であることを考慮し、１Ｋバイト、４Ｋバイト、１６Ｋバイト、



Sector format for 1024 user bytes



Sector format for 512 user bytes

図2.2.1.セクタの物理フォーマット

６４Ｋバイト、２５６Ｋバイト、さらに最大１メガバイトのブロックを設定できるようにしている。ブロック内はセクタが連続しており、またブロックを連続させることにより連続処理の効果が出る。さらにＩＳ＆Ｃでは、ゾーンという分割領域を設けて、その内部ではブロックサイズが等しくなるようにしている。

ファイル管理データから一群のブロックエリアを指示する場合には開始セクタ位置と連続セクタ数を指定するようにしている。

(4) ディスクの片面を１ボリュームとして管理する。ただし、１ボリュームはディスク内の連続した任意領域に定義できる。このため以後で定義されるセクタ番号は、相対番号である。

(注)１ファイルは１ボリューム内に完結するものとして、第１版ではマルチボリュームの処理はしない。

(5) システム領域

ボリューム管理情報、エリア管理、ファイルアロケーションのためのテーブル類を記述するシステム領域を設ける。(図2.2.2.参照)

(6) 画像データを説明するヘッダ

画像データの発生由来、内容を説明する付属データを標準として別に設け、専用領域にまとめて記述する。この付属データをヘッダと呼ぶ。(図2.2.2.参照)

(7) ディレクトリ

画像データ検索のためのディレクトリは、データファイルのひとつとして扱うことで、アプリケーションに自由度を持たせている。(図2.2.2.参照)

(8) 重要データの二重書き

システムの信頼性を確保するために、重要データの二重書きを可能にする。システム領域はとりわけ重要なデータであるので、ディスクの内周と外周に二重書きする。ただし、それらの間の対応関係は取れるようにしておく。(図2.2.2.参照)

(注) 第１版ではシステム領域の二重書きだけを行い、ヘッダ領域データの二重書きはサポートしない。

(9) 仮削除

データ保護のために、削除コマンドでは仮削除の状態にするだけにして、いつでも元のデータを復元できるようにする。仮削除状態にあるデータは、空きエリアがなくなったとき、またはユーザの確認を得た上で本削除とする。

(10)ボリューム使用中フラグ、書込み中フラグ

データの書込み時には最初に書込み中フラグを立てて、書込み時の異常を検出できるようにする。また、データファイルの属性に書込み禁止フラグを設けることにより、データの保護を図る。ディスクをマウントした最初にボリューム使用中フラグを設け、ディスクをマウントした状態での異常終了を検出できるようにする。

(11)ファイル属性の定義

書込み禁止、読出し禁止などのアクセス属性およびセキュリティに関連したファイル種別を識別できるように、ファイル属性フラグを設ける。

(12)データ表現形式

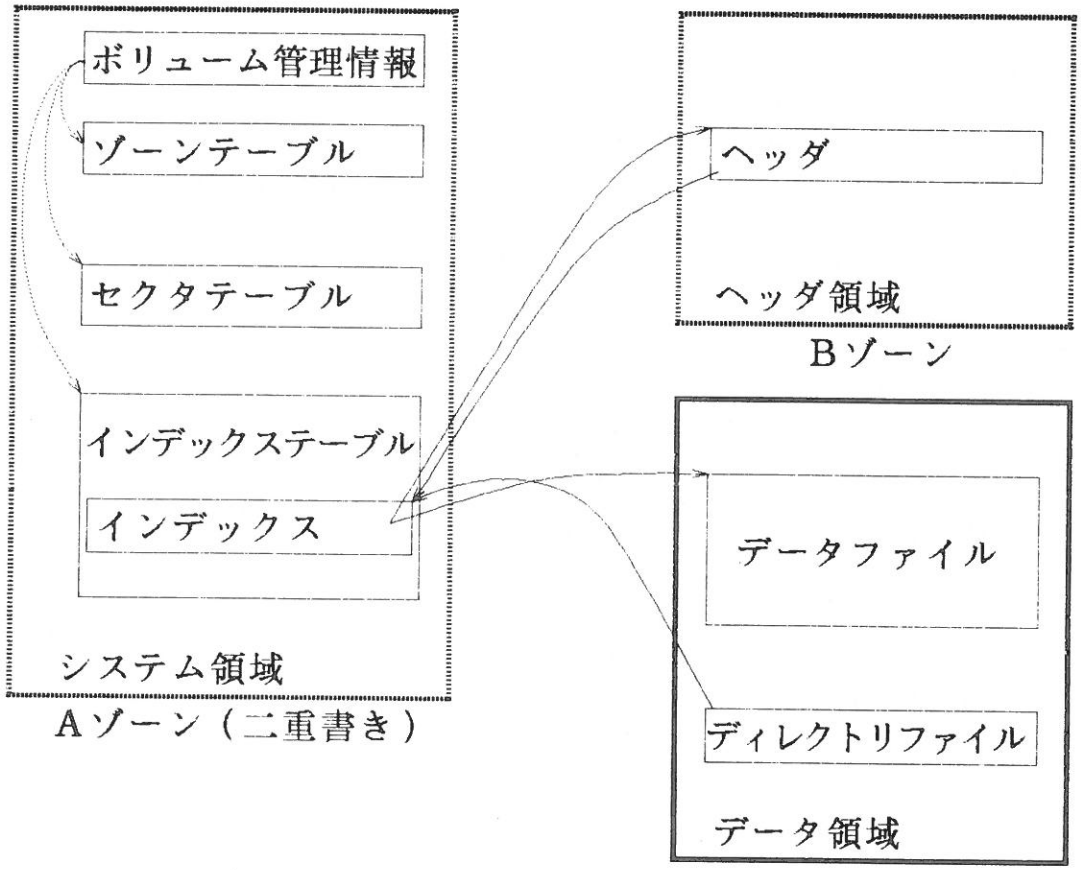
ＡゾーンとＢゾーンのデータ表現形式は、ＭＳＢ－ＬＳＢの順にアドレスが増加するバイト並びに表現を統一する。（詳細は３．１．参照）

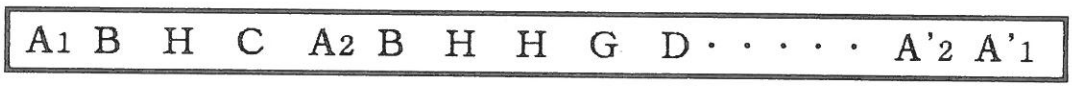
(13)書込み／読出し手順

書込みの場合は、ボリューム管理情報、ゾーンテーブル、セクタテーブルから空きエリアを知り、データとヘッダの割当て結果をインデックステーブルに記載する。読出しの場合は、ファイル名称をディレクトリから探し、該当インデックスに辿り着く。インデックスからは、ヘッダポインタでヘッダを検索し、データポインタで連続ブロック毎にデータを検索する。

医療画像を検索する際、ヘッダの内容から目的の画像を検索することができる。すなわち、ヘッダ領域を逐次検索し、ヘッダに記されているファイルＩＤからインデックスを辿り、データファイルを検索することができるようにしている。

（図2.2.2., 図2.2.3.参照）





ゾーンの配置例

データ領域における可変長連続領域処理の特徴

1．ブロック単位による書込み領域の指定
2．連続セクタによる読出し

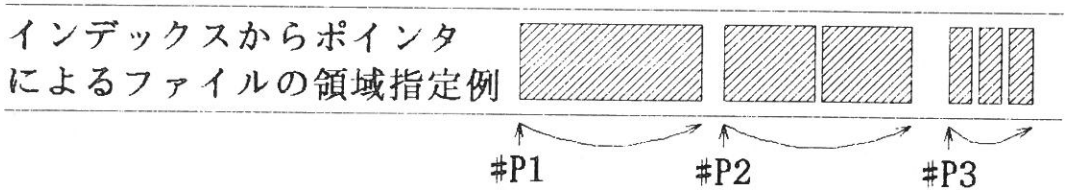


図2.2.2．ＩＳ＆Ｃのファイル管理方式

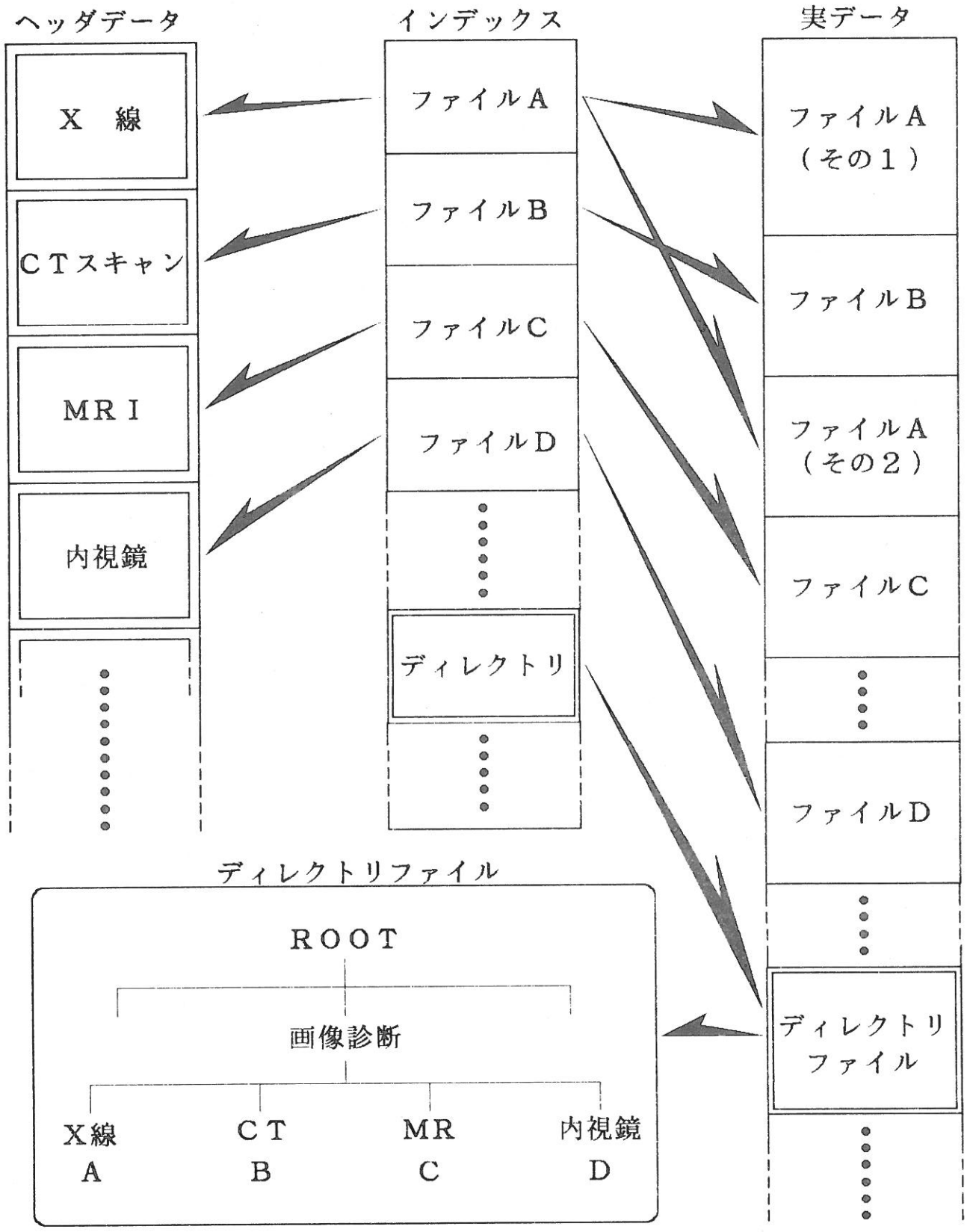


図2.2.3. ＩＳ＆Ｃのファイル管理の医療への適用例

2．3．ファイルの種類

2.3.1.生成方式による分類

ＩＳＡＣでは任意長のファイルを生成できる。また、生成時のパラメータの指定の仕方によりサイズ固定ファイルとサイズ可変ファイルの二通りのファイルを生成できる。

(1) サイズ固定ファイル

この型のファイルは、ファイル生成時には任意長のファイルを生成できるが、生成後はその長さを変えることができない。すなわち、ファイル生成後は固定長ファイルとして扱われ、生成時に指定した長さ以上のデータを書き込むことはできない。

ファイル生成時にサイズが既知の場合に用いる。

(2) サイズ可変ファイル

この型のファイルは、ファイル生成後もファイルの大きさをブロック単位で拡張できる。ただし、拡張するブロック単位の種別はファイル生成時に指定する。したがって、追加するブロック種別を毎回変更することはできない。

ファイル生成時にサイズが未知の場合に用いる。

２．４．仮削除機構

2.4.1.目的・概要

貴重な医療データは厳重に保護しなければならないが、光磁気ディスクは書換え可能なメディアであり、操作ミス等により誤ってファイルを消してしまう可能性がある。この様な事態はある程度アプリケーションレベルで避けることもできるが、それだけでなくファイルマネージャでも保護手段を用意し、より安全なシステムを構築できるようにする必要がある。

この手段として、ファイルの実体を削除しないで論理的にファイルの存在を見えなくした仮削除を定める。この仮削除状態は　ファイル管理情報の属性フラグで示される。通常の使用状況での削除は仮削除とし、仮削除したファイルは見えない。しかし、操作ミスなどで誤って消してしまった場合には論理的な削除状態を解除することでそのファイルを復活できる。

2.4.2.削除の考え方

削除状態として仮削除と実削除の２つがある。

・仮削除

ファイルの実体（ファイル管理情報、ヘッダ、(画像)データ）は、そのままディスク上に残し、ファイル管理情報のフラグ（仮削除フラグ）により、論理的にファイルの存在を見えなくする。この状態では仮削除フラグをクリアするだけでファイルを復活できる。

・実削除

ファイルの実体を削除する。この状態はファイルのファイル管理情報、ヘッダ、およびデータを未使用エリアとし、ゾーンテーブルとセクタテーブルの該当場所を未使用エリアとして解放した状態であり、削除後は元のファイルのエリアを他のファイル用に使用できる。

2.4.3.構造

仮削除機構を実現するために３つの情報を定める。

①仮削除フラグ

ファイルが仮削除されていることを示すフラグである。

仮削除フラグは、ファイル管理情報のファイル属性の１ビットが割当てられる。

（詳細は４．４．インデックステーブル参照）

②仮削除ファイル数

光磁気ディスク内に幾つの仮削除ファイルが有るかを示す。

この情報は、ボリューム管理情報内に定義する。

（４．１．ボリューム管理情報参照）

③ヘッダの負のファイルＩＤ

ヘッダがある場合、その先頭のファイルＩＤを負にすると、ヘッダは仮削除状態となる。

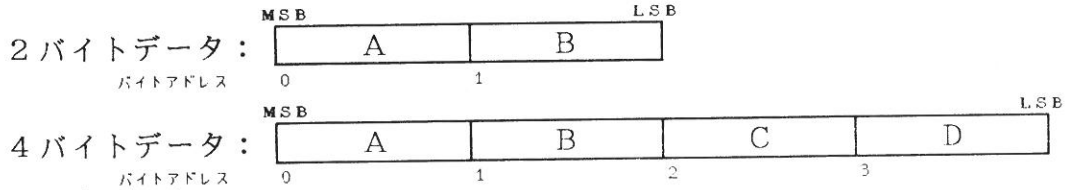
３．ボリュームとファイル構造の説明

３．１．データ表現形式

ＩＳ＆Ｃにおけるボリュームとファイル管理情報の内部の表現形式は以下の通りとする。

(1) データの並び

２バイトデータおよび４バイトデータからなるデータはＭＳＢ→ＬＳＢの順にアドレスが増加するバイト並びとする。



(2) 文字コード

ＡＳＣＩＩコードを基本とする。これ以外の文字コードの使用について、ＩＳＡＣでは特に規定しないが、「情報技術における日本語機能の標準化に関する調査研究報告書」((財)日本規格協会他、１９８８年３月発行）で推奨されている文字コードの使用を推奨する。

文字列は左詰めとする。また、データの実長がデータ領域の長さに満たない場合は、残りの桁の部分にはすべてｎｕｌｌ（００Ｈ）を埋める。

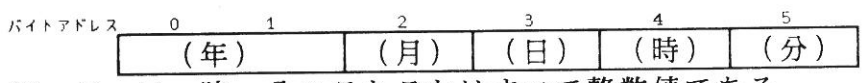
(3) 数値

数値データはすべて整数である。整数には符号付き整数と符号無し整数とがあるが、本仕様書では特に断りがなければ整数は符号付き整数とする。

(4) セクタ番号

４バイトの符号無し整数とする。本仕様書では、セクタ番号は論理セクタ番号を意味する。ただし、2.2.(4)で記されているように相対セクタ番号である。

(5) 日時の表現



年、月、日、時、分のそれぞれはすべて整数値である。

3．2．ゾーンとブロックの構造

3.2.1.ボリュームの構造

(1) ボリュームは複数のゾーンにより構成される。

(2) ゾーンは使用目的により以下の８つの種類に分けられる（図3.2.1.参照）。
このうちデータ領域として使用可能なのは、ＣからＨの６種類である。

Ａ：システム領域

Ｂ：ヘッダ領域

Ｃ：１(4^0)論理セクタのブロックよりなるデータ領域

Ｄ：４(4^1)連続論理セクタのブロックよりなるデータ領域

Ｅ：１６(4^2)連続論理セクタのブロックよりなるデータ領域

Ｆ：６４(4^3)連続論理セクタのブロックよりなるデータ領域

Ｇ：２５６(4^4)連続論理セクタのブロックよりなるデータ領域

Ｈ：１０２４(4^5)連続論理セクタのブロックよりなるデータ領域

(3) データは、ゾーンを分割するブロックを単位として管理される。

(4) １つのデータを、種類が異なる複数のゾーンにまたがって記録する事が可能である。

3.2.2.データ領域の管理

(1) データはインデックスによりファイルとして管理される。
（４．４．インデックステーブル参照）

(2) データ領域のゾーン管理（各ゾーンの種類、空き状況の管理等）はゾーンテーブルを用いて行われる。（４．２．ゾーンテーブル参照）

(3) ゾーンの種類はその使用目的により任意に定義できる。すなわちゾーンが未使用かつ未定義状態の時には、そのゾーンに対して任意のゾーン種類を定義する事が可能である。

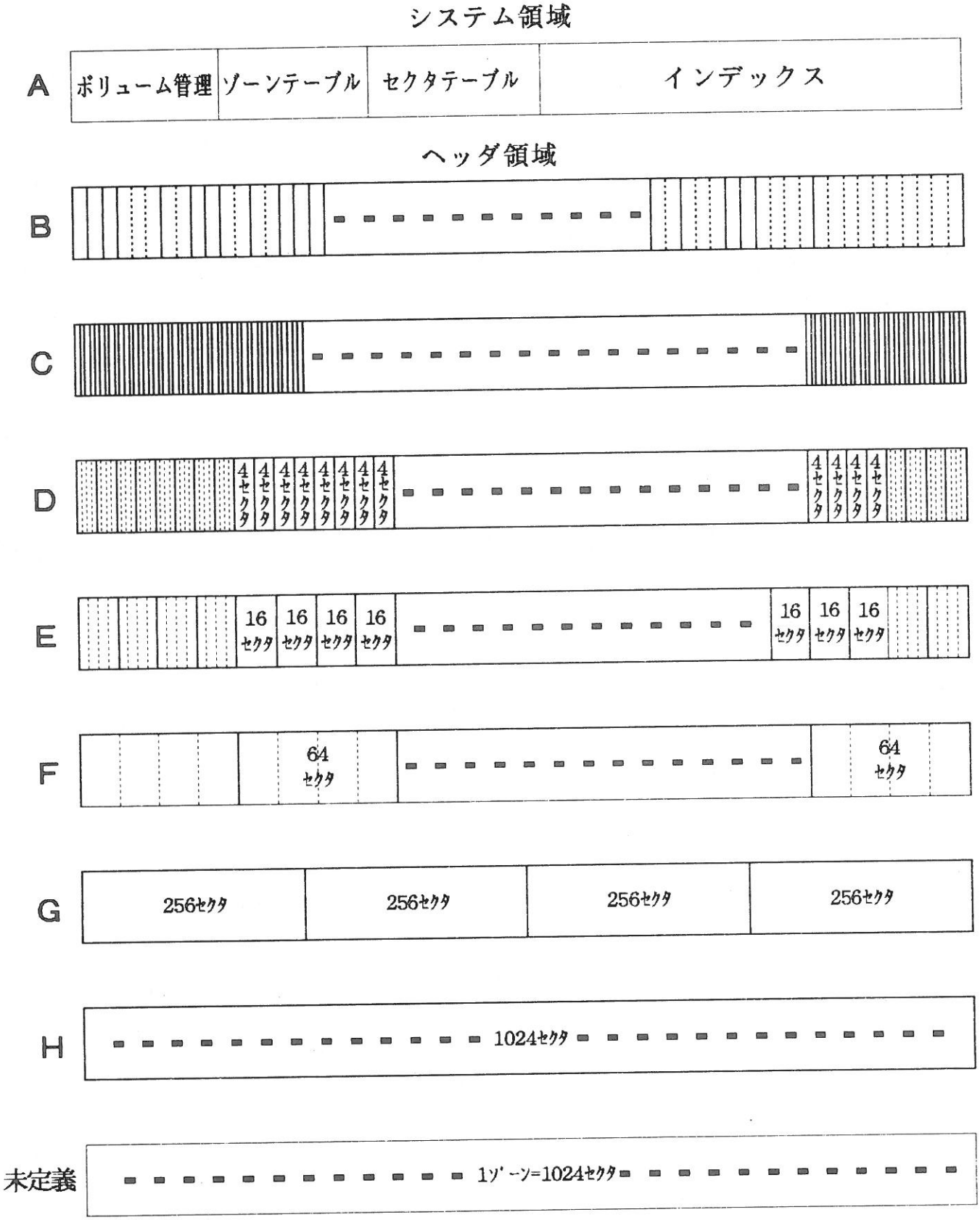


図3.2.1.ゾーンとブロック

3.2.3.データ領域の使用例

以下にデータ領域の使用例を示す。（図3.2.2参照）

(1) 画像データＤ１(2MB+512KB)を書き込む。（図3.2.2の①）

①第nゾーンの種類としてＨを選択する。

（第nゾーン内の総ブロック数1個）

第n+1ゾーンの種類としてＨを選択する。

（第n+1ゾーン内の総ブロック数1個）

第n+2ゾーンの種類としてＧを選択する。

（第n+2ゾーン内の総ブロック数4個）

②画像データＤ１を、第nゾーンの１ブロックと 第n+1ゾーンの１ブロックと第n+2ゾーンの２ブロックに書き込む。

（第nゾーン内の使用ブロック数1個、未使用ブロック数0）

（第n+1ゾーン内の使用ブロック数1個、未使用ブロック数0）

（第n+2ゾーン内の使用ブロック数2個、未使用ブロック数2個）

(2) 画像データＤ２(1MB+320KB)を書き込む。（図3.2.2の②）

①第n+3ゾーンの種類としてＨを選択する。

（第n+3ゾーン内の総ブロック数1個）

第n+4ゾーンの種類としてＦを選択する。

（第n+4ゾーン内の総ブロック数16個）

②画像データＤ２を、第n+3ゾーンの１ブロックと 第n+4ゾーン内の５ブロックに書込む。

（第n+3ゾーン内の使用ブロック数1個、未使用ブロック数0）

（第n+4ゾーン内の使用ブロック数5個、未使用ブロック数11個）

＊データ領域として、第n+2ゾーンの未使用ブロックを２個と、第n+3ゾーンのブロック（Ｈゾーンを選択）を１個使用しても良い。

(3) 画像データＤ３(512KB)を書き込む。（図3.2.2の③）

①画像データＤ３を、第n+2ゾーンの２ブロックに書き込む。

（第n+2ゾーン内の使用ブロック数4個、未使用ブロック数0個）

①画像データ D1(2MB+512KB)の書込み

	D1	D1	D1			
ゾーン番号	n	n＋1	n＋2	n＋3	n＋4	n＋5
ゾーン名	H	H	G			

②画像データ D2(1MB+320KB)の書込み

	D1	D1	D1	D2	D2	
ゾーン番号	n	n＋1	n＋2	n＋3	n＋4	n＋5
ゾーン名	H	H	G	H	F	

③画像データ D3(512KB)の書込み

	D1	D1	D1 D3	D2	D2	
ゾーン番号	n	n＋1	n＋2	n＋3	n＋4	n＋5
ゾーン名	H	H	G	H	F	

④テキストデータ D4(20KB)の書込み

						D4
	D1	D1	D1 D3	D2	D2	
ゾーン番号	n	n＋1	n＋2	n＋3	n＋4	n＋5
ゾーン名	H	H	G	H	F	ForD

図3.2.2.データ領域の使用例

(4) テキストデータＤ４(20KB)を書き込む。（図3.2.2の④）

＜テキストデータＤ４の高速読みだしを目的とした場合＞

①第n+5ゾーンの種類としてＦを選択する。

（第n+5ゾーン内の総ブロック数16個）

②テキストデータＤ４を第n+5ゾーンの１ブロックに書き込む。

（第n+5ゾーン内の使用ブロック数1個、未使用ブロック数15個）

（データ領域として、第n+4ゾーン内の１ブロックを使用しても良い）

この場合、１ブロック内に全データが書かれる為に 読みだし時の高速化が図れる。但し第n+5ゾーンの第１ブロック内の残り44ＫＢはデータの書かれていない無駄な空きスペースとなる。

＜スペースを効率良く使用する事を目的とした場合＞

①第n+5ゾーンの種類としてＤを選択する。

（第n+5ゾーン内の総ブロック数256個）

②テキストデータＤ４を第n+5ゾーンの５ブロックに書き込む。

（第n+5ゾーン内の使用ブロック数5個、未使用ブロック数251個）この場合、データが５ブロックに分かれて書かれる為に 読みだし時の速度は上記ほど速くはない。但し無駄な空きスペースは存在しない。

3．3．ヘッダ構造

3.3.1.概要

ヘッダはデータ領域に格納されているファイルの内容を説明するデータであり、ヘッダとファイルとの関連づけは、インデックステーブルに記述される。

3.3.2.ヘッダの管理

ヘッダは以下の方法で管理される。（図3.3.1.）

(1) ヘッダは１KB単位とする。

書込み／読出しが論理セクタ単位（＝１KB）である。ファイルマネージャのバッファリングを極力避けるために、512B単位のヘッダは認めていない。

(2) ヘッダデータの最大長は２バイト整数表現のため、（32K-6）ﾊﾞｲﾄである。

(3) ヘッダの開始セクタ番号と連続セクタ数は、システム領域（Ａゾーン）にあるインデックステーブル内のヘッダポインタで記述される。１KBを越える場合は連続セクタとする。また、１つのヘッダデータは、２個のＢゾーンにまたがらない。

(4) ヘッダはＢゾーン（ヘッダゾーン）に書かれる。

(5) ヘッダがないファイルも許容し、その場合はインデックステーブル内のヘッダのセクタ数を０とする。

(6) 図3.3.1.のごとく個々のヘッダの先頭に、ファイルＩＤ、ヘッダが占有している領域の長さを書込む。この目的は次のことを実現するためである。

①ヘッダから当該ファイルへのアクセスを速く容易にする。
②仮削除されているヘッダは、その旨が分かるようにする。
③個々のヘッダの先頭が容易に分かるようにする。

3.3.3.ヘッダの内容

ヘッダの内容は、ＩＳ＆Ｃ規格書「データフォーマット編」で説明されている。

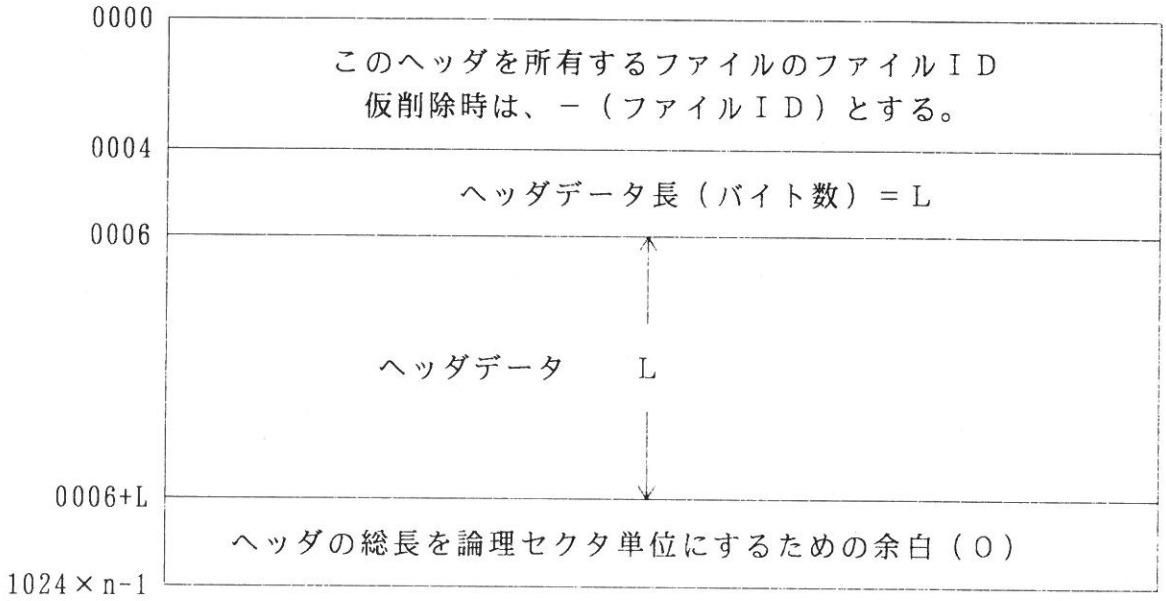


図3.3.1.ヘッダの構造

表3.3.1.ヘッダの構造

バイトアドレス	名　　称	長さ（バイト）	データタイプ／備考
０	ファイルＩＤ	４	３２ビット２進数 値＞０のとき 有効ファイルＩＤ 値＜０のとき 仮削除時ファイルＩＤ 値＝０のとき 実削除状態
４	ヘッダデータ長 （Ｌ）	２	１６ビット２進数
６	ヘッダデータ	Ｌ	ヘッダデータ形式による
６＋Ｌ ～1024×n-1 (n=1,2,･･,32)		1024×n-(L+6)	ｎｕｌｌ(００Ｈ)を埋める

3.3.4.解説

(1) ヘッダとそれに対応するファイル（画像データなど）を媒体へ格納するには、いくつかの方法がある。ＩＳ＆Ｃでは、ヘッダとファイルを連続して格納するのではなく、すべてのヘッダをヘッダ専用のＢゾーンにまとめて格納する方式を採用している。これにより、複数のファイルのヘッダを高速で読取ることが可能になり、検索の高速化に結びつく。ヘッダとファイルとの対応関係はシステム領域のインデックステーブルに記述されている。

(2) ＩＳ＆Ｃでは、研究用や医療応用以外の分野でも使われることが予想されるため、ヘッダ無しのファイルも許容する。

(3) ヘッダ無しのファイルは、ファイルを管理するファイルＩＤまたはファイル名によって検索することができる。ファイルＩＤは、ユニーク性が保証されなければならない。

４．システム管理情報

４．１．ボリューム管理情報

ボリューム管理情報はディスクの初期化時に書込むもの（No.0ｾｸﾀ に書込む）と、データ書換え時に書換えるもの（No.1ｾｸﾀ に書込む）とがある。

4.1.1.ボリューム管理情報　その１　－－－　No.0ｾｸﾀ

表4.1.1.ボリューム管理情報　その１

ﾊﾞｲﾄｱﾄﾞﾚｽ	項　目　名	バイト数	データタイプ
0～3	初期化識別子	4	文字型
4～7	バージョン番号	4	文字型
8～23	適用分野	16	文字型
24～55	ボリューム名称	32	文字型
56～59	ボリュームＩＤ	4	整数型
60～91	所有者名	32	文字型
92～123	所有者コード	32	文字型
124～129	初期化日時	6	整数型
130～133	ゾーン数	4	整数型
134～135	ゾーン内セクタ数	2	整数型
136～137	セクタ容量	2	整数型
138～141	ｿﾞｰﾝﾃｰﾌﾞﾙ開始セクタ番号	4	整数型
142～145	ｾｸﾀﾃｰﾌﾞﾙ開始セクタ番号	4	整数型
146～149	ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ開始ｾｸﾀ番号	4	整数型
150～151	インデックスサイズ	2	整数型
152～1023	予約領域	872	

(1) 初期化識別子

ＩＳ＆Ｃフォーマットで初期化されていることを示す。

初期化時に、”ＩＳ＆Ｃ”（ASCIIｺｰﾄﾞで 49 53 41 43H）と書き込む。

(2) バージョン番号

初期化時にＩＳ＆Ｃのバージョン番号を文字型で書き込む。

バージョン番号は、”０１．０”（ASCIIｺｰﾄﾞで 30 31 2E 30H）から開始する。

(3) 適用分野

保健医療で使用する場合は”ＭＥＤＩＣＡＬ”と書込む。

(4) ボリューム名称

初期化時にボリュームの名称を文字型で書込む。

(5) ボリュームＩＤ

初期化時にボリュームのＩＤを文字型で書込む。

(6) 所有者名

初期化時にボリュームの所有者名を文字型で書込む。

(7) 所有者コード

初期化時にボリュームの所有者ＩＤを文字型で書込む。

(8) 初期化日時

ＩＳ＆Ｃフォーマットで初期化した日時を、3.1.(5)の形式で書込む。

初期化した西暦年を２バイトで

初期化した月を１バイトで

初期化した日を１バイトで

初期化した時（24時間表記）を１バイトで

初期化した分を１バイトで

各々整数型で書込む。

(9) ゾーン数

ゾーン数を記入する。

(10)ゾーン内セクタ数

ゾーン内のセクタ数を記入する。

(11)セクタ容量

１セクタ（論理セクタ）の容量をバイト単位で記入する。

現在は1024を記入する。

(12)ゾーンテーブル開始セクタ番号

ゾーンテーブルの先頭のセクタ番号を記入する。

(13)セクタテーブル開始セクタ番号

セクタテーブルの先頭のセクタ番号を記入する。

(14)インデックステーブル開始セクタ番号

インデックステーブルの先頭のセクタ番号を記入する。

(15)インデックスサイズ

インデックステーブルの１要素の大きさをバイト単位で記入する。

(16)予約領域

将来の標準化のために確保されている領域である。

ｎｕｌｌ（００Ｈ）を埋めるものとする。

4.1.2.ボリューム管理情報　その２　－－－　No.1ｾｸﾀ

表4.1.2.ボリューム管理情報　その２

ﾊﾞｲﾄｱﾄﾞﾚｽ	項　目　名	バイト数	データタイプ
0～3	インデックス総数	4	整数型
4～7	登録ファイル数	4	整数型
8～11	仮削除ファイル数	4	整数型
12～15	空インデックス数	4	整数型
16～17	システムファイル数	2	整数型
18～19	ディレクトリファイル数	2	整数型
20～25	更新日時	6	整数型
26～29	空インデックス開始番号	4	整数型
30～31	ボリューム使用中フラグ	2	整数型
32～1007	予約領域	976	
1008～1023	システム予約領域	16	

(1) インデックス総数

現在確保されているインデックスの総数を整数型で書込む。

(2) 登録ファイル数

現在登録されているファイル数を整数型で書込む。（登録されているファイル数には仮削除ファイル数を含まない。）

(3) 仮削除ファイル数

現在仮削除されているファイル数を整数型で書込む。

(4) 空インデックス数

現在未使用のインデックス数を整数型で書込む。

(5) システムファイル数

現在ボリューム内に登録されているシステムファイル数を整数型で書込む。

(6) ディレクトリファイル数

現在ボリューム内に登録されているディレクトリファイル数を整数型で書込む。

(7) 更新日時

ボリューム管理情報を更新した日時を4.1.1.(7)の初期化日時と同じ書式で書込む。

(8) 空インデックス開始番号

インデックステーブルの空インデックスをチェイン管理するため、その先頭のインデックステーブル番号を整数型で書込む。

(9) ボリューム使用中フラグ

光磁気ディスクのマウント時には・・・・・・1

ディスマウント時には・・・0

を整数型で書込む。

(10)予約領域

将来の標準化のために確保されている領域である。

ｎｕｌｌ（００Ｈ）を埋めるものとする。

(11)システム予約領域

セキュリティ管理用にシステムであらかじめ予約されている。

4．2．ゾーンテーブル

(1) ボリューム上の各ゾーンの種別、空きブロック数、バックアップゾーン番号および同一種別ゾーン毎の連番を管理するテーブルである。ゾーン番号は１から始まる。

(2) ゾーンテーブルの各エントリの内容を表4.2.1.に示す。

表4.2.1.ゾーンテーブルエントリ情報

ﾊﾞｲﾄｱﾄﾞﾚｽ	項　目　名	ﾊﾞｲﾄ数	ﾃﾞｰﾀﾀｲﾌﾟ	内　　容
0	ゾーン種別	2	整数型	１～８の値でゾーン名Ａ～Ｈを示す。　－１～－８でＡ～Ｈのバックアップゾーンであることを示す。 未定義ゾーンは０とする。
2	空きブロック数	2	整数型	ＡおよびＡ’ゾーンの場合：次のＡゾーン番号。最終ゾーンは－１。 Ｂゾーンの場合：空セクタ数。 Ｃ～Ｈゾーンの場合：空ブロック数。 未定義ゾーンの場合：０。
4	バックアップゾーン ・ 同一種別内連番	2	整数型	Ａ、Ａ’ゾーンの場合：各々のバックアップゾーン番号。 Ｂ、Ｃ～Ｈおよび未定義ゾーンの場合：０。

４．３．セクタテーブル

(1) ボリューム上の全論理セクタの使用状況を各１ビットに対応させて管理するテーブルである。

(2) Ａ～Ｈの全てのゾーンが対象である。

(3) ゾーンテーブル、セクタテーブルおよびインデックステーブルの各テーブルは、セクタの先頭からアロケーションされる。

そのため、ゾーンテーブル、セクタテーブルの最終セクタの後部の空領域は、ｎｕｌｌ（００Ｈ）が埋められる。

4．4．インデックステーブル

表4.4.1.インデックステーブル（親）

ﾊﾞｲﾄ ｱﾄﾞﾚｽ	ﾊﾞｲﾄ数	項　目　名	ﾃﾞｰﾀ ﾀｲﾌﾟ	備　　考
0	4	ファイルＩＤ	整数型	
4	24	ファイル名	文字型	
28	4	作成年月日	整数型	年(2b)、月(1b)、日(1b)の順
32	2	作成時刻	整数型	時(1b)、分(1b)の順
34	4	最終変更年月日	整数型	年(2b)、月(1b)、日(1b)の順
38	2	最終変更時刻	整数型	時(1b)、分(1b)の順
40	4	ファイルバイト長	整数型	
44	2	属性	整数型	フラグビット形式
46	12	予備	整数型	（零をダミーに詰める）
58	6	ヘッダポインタ	整数型	開始セクタ番号(4b) + 連続セクタ数(2b)
64	6	# 1 ポインタ	整数型	〃
70	6	# 2 ポインタ	整数型	〃
		：		
118	6	#10 ポインタ	整数型	〃
124	4	子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号	整数型	

表4.4.2.インデックステーブル（子）
〔拡張されたインデックステーブル〕

ﾊﾞｲﾄ ｱﾄﾞﾚｽ	ﾊﾞｲﾄ数	項　目　名	ﾃﾞｰﾀ ﾀｲﾌﾟ	備　　考
0	4	ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号	整数型	自分のインデックス番号の２の補数
4	4	親のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号	整数型	
8	14	予備	整数型	（零をダミーに詰める）
22	6	#1 ポインタ	整数型	開始セクタ番号(4b) + 連続セクタ数(2b)
28	6	#2 ポインタ	整数型	〃
		：		
118	6	#17ポインタ	整数型	〃
124	4	次の子のｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ番号	整数型	

(1) ファイルＩＤ

インデックステーブル番号は１番から始まり、該当インデックスのテーブル内の順序位置である。ファイルＩＤは、このインデックステーブル番号と一致する。

ファイルＩＤの内容については次の規則がある。

内容＝０　このインデックステーブルは空きである。

内容＞０　このインデックスは親として使用中である。
内容は自分のインデックステーブル番号であり、ファイルＩＤ番号でもある。

内容＜０　このインデックスは子として使用中である。
内容の二の補数が自分のインデックステーブル番号である。

(2) 最終変更年月日および時刻

ファイル属性フラグが変更された場合にも、その変更日時を記録する。

(3) ヘッダポインタ

ヘッダ領域の取り方には次の規則がある。

ヘッダ領域は1024バイト（１セクタ）単位でなおかつ連続したセクタで取られなければならない。

(4) ポインタのマージ

あるポインタが示す領域とそれに続くポインタが示す領域が連続したセクタの領域である場合、子のインデックステーブルも含め二つ以上のポインタを一つのポインタとしてマージ（併合）することが許される。また、異なるゾーンにまたがっていても連続セクタ領域であれば可能である。

なお、当然の事であるがマージするポインタが置かれた順序と連続するセクタの順序は同じでなければならない。

２バイトで表現できる最大の整数は（2^{15}-1）であるので、１個のポインタで指定できる最大の連続セクタ数の領域は、約32MBである。

(5) 属性

属性はフラグビット＝１によって表現されるもので次の様に割り当てられている。

ﾊﾞｲﾄ ｱﾄﾞﾚｽ	ﾌﾗｸﾞﾋﾞｯﾄ(xになっている所が該当のビット位置) MSB← →LSB	意　味
44	x	サイズ可変ファイル*1
	.... ..x.	書込み禁止ファイル
	.... .x..	読出し禁止ファイル
	.... x...	システムファイル
	...x	ディレクトリファイル
	..x.	予備（零にセットしておくこと）
	.x..	書込み中（ファイルに書込み中は１を立てる）*2
	x...	仮削除
45	 xxxx	セキュリティ用ファイル種別*3
	xxxx xx..	予備（全て零にセットしておくこと）

*1　ファイル生成の指定方法には、サイズ可変ファイルとサイズ固定ファイルの二通りの方法がある。

・サイズ可変ファイルはデータを書込むまでエリアは確保されない。
確保する単位はゾーン種別によって指定されたブロックである。
このタイプのファイルはディスク上に空きがある限り追加拡張することが可能である。

・サイズ固定ファイルはファイル作成時（クリエート又はオープン時）に指定されたバイト数を満たすブロック数のエリアが確保される。
このタイプのファイルは一旦作成されると追加拡張はできない。

*2　このビットは書込みモードでファイルオープン時に１にセットし、ファイルクローズ時に０に戻される。

*3　セキュリティ用ファイル種別フラグとして、４ビットが予約確保される。

(6) 子の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

子のインデックステーブル番号を利用して、使用占有する親インデックス→子インデックスのチェインと、ボリューム管理情報その２の空インデックス開始番号から始まる空きエリアチェインの管理を行う。

1)該当インデックスが使用中の場合：

子のインデックスが継続する場合、子のインデックステーブル番号が入れられてインデックステーブルをチェインする。子のインデックステーブルが継続しない場合は、－１が入れられる。

2)該当インデックスが空（未使用）の場合：

ファイルＩＤにはゼロが入れられるが、子のインデックステーブル番号には次の空インデックス番号が入れられ、空インデックスのチェインを作る。ただし最後の空インデックスの子のインデックステーブル番号には、－１が入れられて終端する。また、最初の空インデックスの番号は、ボリューム管理情報その２の空インデックス開始番号で示される。

3)ファイルを実削除する場合（図４．４参照）：

実削除されるファイルが使用しているインデックスの空エリアチェインへの接続を、次の手順で行う。

①該当ファイルが使用している先頭のインデックス番号を、ボリューム管理情報その２の空インデックス開始番号に入れる。

②該当インデックスのチェインをたどりながら、ファイルＩＤをゼロにし、その個数を求める。

③インデックスチェインの最終のインデックスの子のインデックステーブル番号に、処理開始前のボリューム管理情報その２の空インデックス開始番号をセットする。

④最後に、ボリューム管理情報その２の空インデックス数に、②で求めた個数を加える。

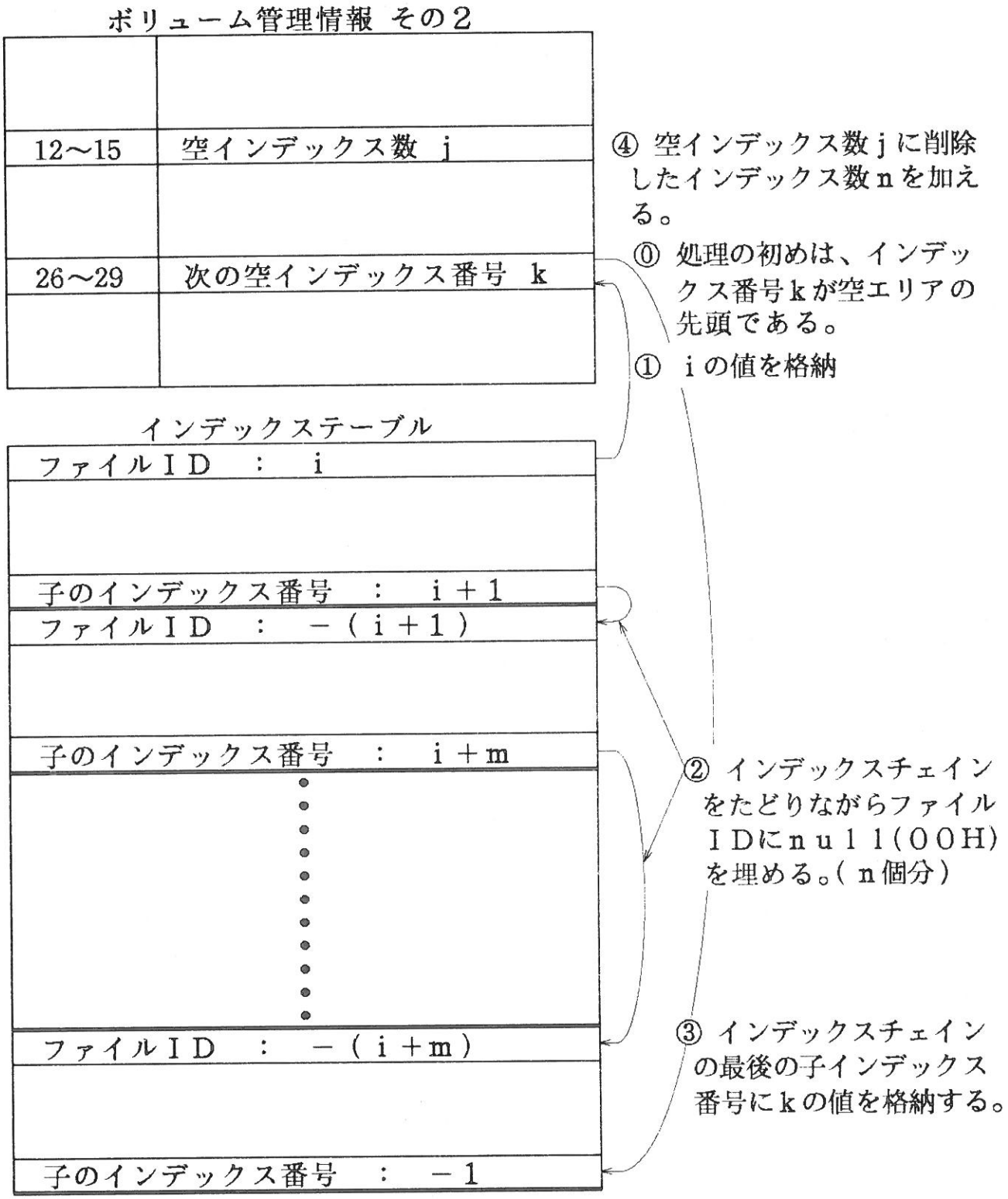


図4.4.ファイルＩＤ：iのファイルが削除された場合の空エリアチェインの処理手順

５．アルゴリズムの説明

５．１．Ａゾーンデータの二重書き

システム領域であるＡゾーンデータの記録は、二箇所のゾーンに同一のデータを記録することにより行う。二箇所のゾーンのうち、外周側のゾーン番号の大きい方のゾーンが、Ａゾーンのバックアップゾーンとして割当てられる。Ａゾーンは、メモリ上のデータが更新される都度、更新される。ただし、バックアップゾーンの書込みはディスマウント時にのみ実行される。

5.1.1.Ａゾーンの割当て（図5.1.1.参照）

(1) 最初のＡゾーンの割当て

最初のＡゾーンは最内周（つまり、ゾーンテーブルエントリ番号１）に割当てられる。このゾーンのゾーン種別は１、バックアップゾーンの番号はＮ（最大ゾーン番号）である。

(2) 追加Ａゾーンの割当て

追加Ａゾーンは、ゾーン番号の小さい方から未定義ゾーンを探し出して順次割当てる。（このゾーン番号を仮にＡ１、Ａ２、・・・とする）このゾーンのゾーン種別は１、バックアップゾーンの番号は次項で割当てられるバックアップゾーンの番号（つまり、Ｎ－１、Ｎ－２、・・・）となる。

(3) バックアップＡゾーンの割当て

バックアップＡゾーンは最外周から割当てられる。最外周のゾーン番号をＮとすれば、Ｎ、Ｎ－１、Ｎ－２、・・・のように順次割当てられる。

バックアップＡゾーンのゾーン種別は－１であり、バックアップゾーンの番号は、１、Ａ１、Ａ２、・・・のようになる。

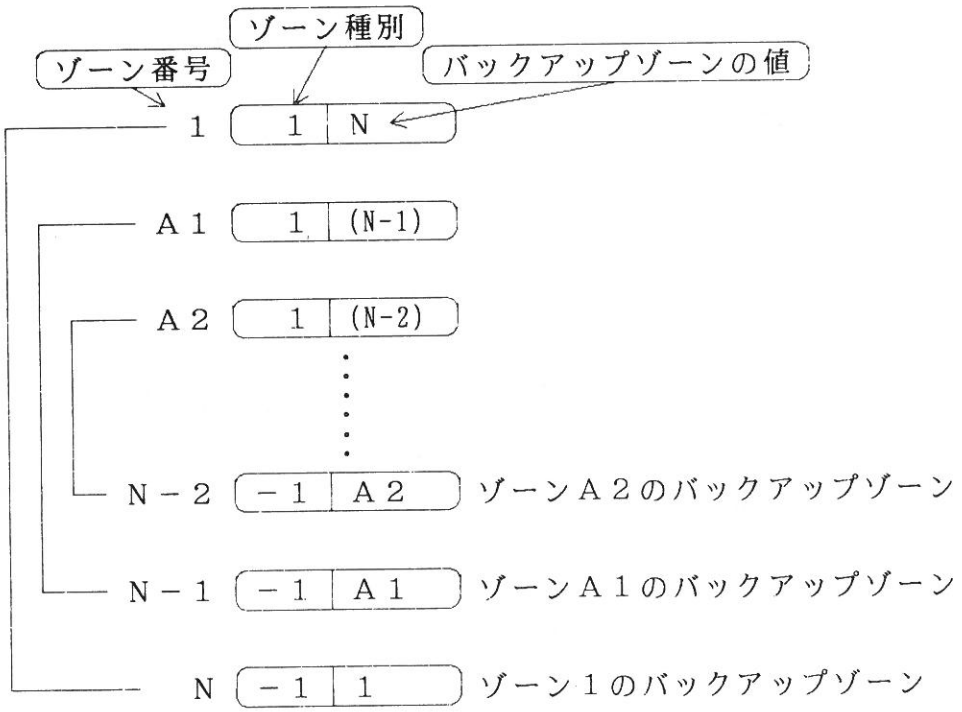


図5.1.1.Aゾーンの割当て

5.1.2.Aゾーンデータの書込みと読出し

システム領域に記録するデータａについて、Aゾーンに書込むデータをａ１、Aゾーンのバックアップゾーンに書込むデータをａ２とすると、ａの書込みと読出しは次のような順で行う。

リターンコードについては、付録Bを参照のこと。

(1) 元のAゾーンの書込み

```
はじめ：
    ａをａ１として書込む.
    ＩＦ　失敗
        ＴＨＥＮ　ＤＯ
            『エラー：Aゾーンへの書込みにおいてａ１の書込みに失敗した』を
            リターンコードとして戻す.
        ＥＬＳＥ　ＤＯ
            『正常終了』をリターンコードとして戻す.
    ＥＮＤ　ＩＦ
おわり：
```

(2) バックアップAゾーンの書込み

ディスマウントの時、Aゾーン中のボリューム使用中フラグに『使用していない』を記録した直後、バックアップゾーンではない全てのAゾーンの内容を対応するバックアップゾーンにコピーする。

(3) Aゾーンの読出し

```
はじめ：
    ａ１を読込む.
    ＩＦ　失敗
        ＴＨＥＮ　ＤＯ
            ａ２を読込む.
            ＩＦ　失敗
                ＴＨＥＮ　ＤＯ
                    『エラー：Aゾーンに対する読出しに完全に失敗した』
                    をリターンコードとして戻す.
                ＥＬＳＥ　ＤＯ
                    ａ２をａとする.
                    ａをａ１として書き込む.
                    『警告：Aゾーンに対する読出しでａ１の読出しに失敗し、
                    ａ２をａとした』をリターンコードとして戻す.
            ＥＮＤ　ＩＦ
    ＥＮＤ　ＩＦ
    ａ１をａとする.
    『正常終了』をリターンコードとして戻す.
おわり：
```

5．2．ファイルのアロケーション

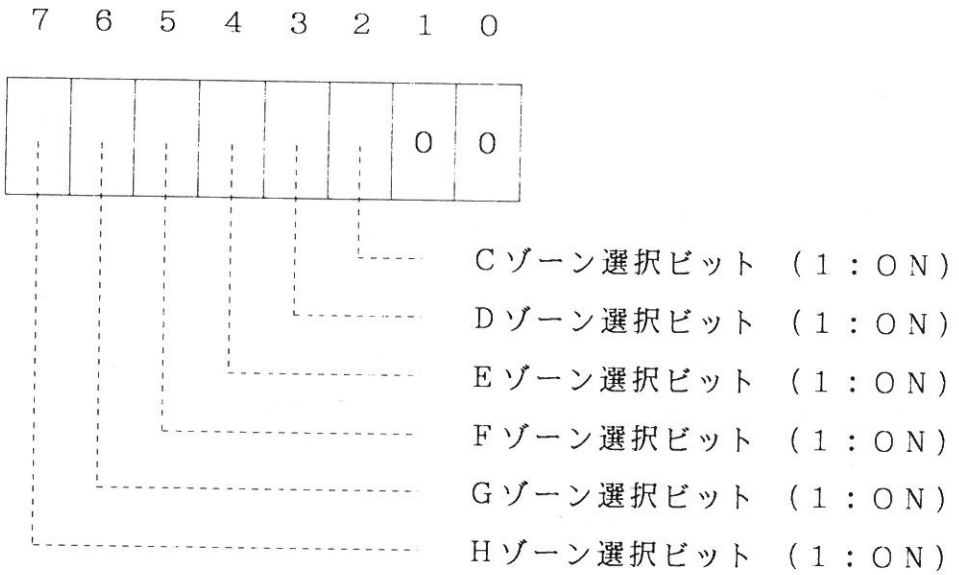
(1) 概要

ファイル生成時の指定が、サイズ可変ファイルかサイズ固定ファイルかでゾーンの割当て方法が異なる。サイズ可変ファイルの場合は、アプリケーションから指定されるゾーン種別によりアロケーションのブロックの大きさが一意に定まる。　しかし、サイズ固定ファイルの場合は、アプリケーションから指定されるゾーン選択条件によってゾーンの割当て方が変化する。

ヘッダデータは、Ｂゾーンの連続セクタ領域に記録される。書込み領域は書込み時に確保され、以後の領域変更は新規領域に確保される。

(2) サイズ固定ファイルのアロケーション規則

アプリケーション・プログラムより指定される使用ゾーンフラグに対応するゾーン種別が割当てられる。使用ゾーンの指定形式は、以下の通りである。



上記の選択ビットが、ＯＮになっている部分のゾーン種別のみが割当てられる。全てＯＦＦの場合は、内部アルゴリズムにより自動的に割当てられる。

(3) サイズ固定ファイルのアロケーション例

1)アプリケーションから指定されたボリューム・ディスクリプタの正当性をチェックする。

```
ＩＦ　エラー
　　ＴＨＥＮ　ＤＯ
　　　エラー・コードの設定
　　　リターン
ＥＮＤ　ＩＦ
```

2)アプリケーションから使用ゾーンが指定されているかどうかをチェックし、指定されたゾーンを割当てる。

```
ＩＦ　使用ゾーンの指定有り
　　ＴＨＥＮ　ＤＯ
```

(3-1) 使用ゾーンが指定された時のアルゴリズム

```
　　ＥＬＳＥ　ＤＯ
```

(3-2) 使用ゾーンが指定されなかった時のアルゴリズム

```
　　ＥＮＤ　ＩＦ
```

3)該当インデックスのポインタ部を更新する。

```
ＩＦ　2)項で求められた各領域のポインタの数が１０個以下
　　ＴＨＥＮ　ＤＯ
　　　該当インデックスのポインタ部に各領域へのポインタを設定する.
　　ＥＬＳＥ　ＤＯ
　　　先ず、該当インデックスのポインタ部に最初の１０個分のポインタを設定
　　　する。次に子インデックスを新たに生成して、残りのポインタを設定する。
```

4)インデックステーブルのその他の対応部を更新する。

5)ゾーンテーブルとセクタテーブルを更新する。

6)ボリューム管理情報その２を更新する。

7)メモリ上のボリューム管理情報（その２）、ゾーンテーブル、セクタテーブル、インデックステーブルをディスク上に書込む。

(3-1) 使用ゾーンが指定された時のアルゴリズム

1)アプリケーションから指定される使用ゾーン引き数より、使用するゾーン種別を取り出す。

2)アプリケーションから指定されるアロケーション・ファイルのデータ長をもとに、使用ゾーンのブロック数を求める。各ゾーンのブロック数の算出方法は以下の通りとする。

＜各ゾーンのブロック数算出方法＞

①最初に指定されたゾーンの内から一番大きなゾーン種を選択する。

②データ長を選択ゾーンのブロック長で除算し、商をＡとし、余りをＢとする。

```
ＩＦ　（Ｂ＝０）
　　ＴＨＥＮ　ＤＯ
　　　選択ゾーン・ブロック数（ｉ）＝Ａ
　　　リターン
　　ＥＮＤ　ＩＦ
ＩＦ　（もう使用ゾーンが存在しない）
　　ＴＨＥＮ　ＤＯ
　　　選択ゾーン・ブロック数（ｉ）＝Ａ＋１
　　　リターン
　　ＥＬＳＥ　ＤＯ
```

選択ゾーン・ブロック数（ｉ）＝Ａ

ｉ＝ｉ＋１

次の使用ゾーンをセットアップ

ＥＮＤ　ＩＦ

③データ長をＢとし、②の処理から繰り返す。

＜算出方法の例＞

アプリケーションより、固定長ファイル・データ＝１２４０ＫＢ、使用ゾーン＝Ｈ，Ｆ，Ｅの指定をされた時、ゾーンの割当て状態は以下のようになる。

Ｈゾーンのブロック数＝（１２４０÷１０２４）の商＝１

（１２４０÷１０２４）の余りは２１６ＫＢ

Ｆゾーンのブロック数＝（２１６÷６４）の商＝３

（２１６÷６４）の余りは２４ＫＢ

Ｅゾーンのブロック数＝（２４÷１６）の商＝１

（２４÷１６）の余りは８ＫＢ

余りが、８ＫＢ存在するため、Ｅゾーンのブロック数が更に一個増加する。

3)使用ゾーンの必要なブロック数を確保するため、ゾーンテーブルをサーチし、ゾーンの割当てをする。

①ゾーンテーブルの先頭から、選択した使用ゾーン種別と一致したゾーン、または未定義ゾーン（定義はしてあるが全て空きも含む）をさがす。

ＩＦ　（一致ゾーン無し、または未定義ゾーン無し）

ＴＨＥＮ　ＤＯ

エラーコードの設定（対象のゾーン種無し）

リターン

ＥＮＤ　ＩＦ

②見つかったゾーン以降の空きブロック数を調べ、使用ゾーンを確保する。

③ブロックが続く連続エリアを確保できる場合は、連続エリアの開始セクタ番号と連続セクタ数を記憶し、インデックスのポインタを更新する。
次の使用ゾーンの割当てのセットアップをする。使用ゾーン全ての割当てが終了した場合は、ゾーンテーブルとセクタテーブルを更新しリターンする。未終了の場合は、①から繰り返す。

④連続エリアを確保できない場合は、ブロック毎に割当てていき、その都度、開始セクタ番号と連続セクタ数を記憶する。現在対象としている使用ゾーンの割当てが終了したら、次の使用ゾーンの割当てのセットアップをする。
使用ゾーン全ての割当てが終了した場合は、ゾーンテーブルとセクタテーブルを更新しリターンする。
未終了の場合は、①から繰り返す。

(3-2) 使用ゾーンが指定されなかった時のアルゴリズム

基本的に (3-1)のアルゴリズムと同じである。但し、ゾーンの割り当て候補として無条件にH，G，F，E，D，Cを順番に採用する。

5．3．ファイルの書込み

5.3.1.概要

アプリケーション・プログラムがファイル書込みを行う場合、事前にファイルのアロケーションおよびオープンをしておく必要がある。また、ファイルの書込みを終了した後は、ファイルをクローズして、メモリ上に存在していた管理テーブルをディスクに書込む必要がある。

5.3.2.書込みアルゴリズム

(1) ディスク上の該当親インデックスの属性情報のうち、書込み中フラグをＯＮにする。

(2) ファイル・ディスクリプタの正当性をチェックする。

(3) 該当の親インデックスをメモリ上に読出す。

(4) 該当インデックスの属性をチェックし、書込み可能ファイルかどうかを調べる。

(5) 該当インデックスの属性をチェックし、サイズ固定ファイルか、またはサイズ可変ファイルかどうかを調べ、サイズ可変ファイルの場合は追加拡張エリアを確保する。

<u>サイズ可変ファイルの拡張エリアの確保</u>

1)メモリ上のデータから書込みデータエリアを求め、その長さを書込み指定バイト長とする。

2)該当インデックスのポインタを調べる。

①第一ポインタの開始セクタ番号からそのゾーン番号を知り、指定済みのブロックサイズを知る。

②各ポインタの連続セクタ数を合計して、既に確保されているファイルの現バイト長を求める。

3)ファイルの現バイト長が書込み指定バイト長より小さい場合は、拡張エリアを追加確保する。

①追加必要となるエリアのバイト長を求める。
②追加必要なブロック数を求める。
③インデックスチェインの最後のインデックスのポインタに、追加ブロックを割当てる。ブロックが連続する場合はポインタのマージをする。
④子インデックスを追加した場合は、メモリ上およびディスク上のボリューム管理情報その２の空インデックス数を更新する。
⑤メモリ上およびディスク上のインデックステーブル(空エリアチェインとポインタ)、ゾーンテーブル、セクタテーブルを更新する。

サイズ固定ファイルの場合

あらかじめ書込みエリアが確保されているので、以下共通にデータ書込み処理に続く。

(6) 書込みデータをディスク上に書込む。

(7) ヘッダデータをディスク上に書込む。

(8) ディスク上の親インデックスの書込み中フラグをＯＦＦにする。

5．4．ファイルの読出し

5.4.1.ヘッダデータと画像データの読出し

(1) 読出し条件

・アプリケーションレイヤーから、必要とされる画像データのファイル名またはファイルＩＤにより、読出し処理を行うものとする。

・必要とされるファイルのオープン・クローズ処理は、別途行われることとする。

・Ａゾーンのシステム領域は二重書きされており、読出し時にエラーが発生した場合には、バックアップ領域を読みにいく。

・ヘッダ情報の読出しと、画像ファイルの読出しは、アプリケーションレイヤーから、サービス関数による２段階の呼び出しにより、読出されるものとする。

(2) 画像ファイルの読出し

①ボリューム管理情報をリード

ゾーン種別　：　Ａゾーン

アドレス指定　：　セクタＮｏ．０～１

処理項目　：　・ＩＳ＆Ｃフォーマット及び Version No.の確認。

・各テーブル開始アドレスを得る。

↓

②インデックステーブルをリード

ゾーン種別　：　Ａゾーン

アドレス指定　：　ボリューム管理情報のインデックステーブル開始アドレス

処理項目　：　・インデックステーブルをサーチし、ファイル名またはファイルＩＤの一致するインデックスを得る。

・インデックス内のヘッダポインタを得る。

・インデックス内のデータポインタを得る。

・子のインデックスが有る場合は子インデックスも得る。

③ヘッダデータと画像データのリード

ゾーン種別　：　Ｃ～Ｈゾーン

アドレス指定　：　インデックスのヘッダポインタと画像ファイルポインタ

処理項目　：　・ヘッダポインタによりヘッダデータを読出す。

・第１ポインタより順次リードし、ポインタの内容がゼロになるまで画像データを読出す。

(3) ヘッダ情報の読みだし

①ボリューム管理情報をリード　（処理内容は上記と同じ）

②インデックステーブルをリード

ゾーン種別　：　Ａゾーン

アドレス指定　：　ボリューム管理情報のインデックステーブル開始アドレス

処理項目　：　・インデックス情報をサーチし、ファイル名またはファイルＩＤの一致するインデックスを得る。

・ヘッダポインタを得る。

③ヘッダをリード

ゾーン種別　：　Ｂゾーン

アドレス指定　：　インデックスのヘッダポインタ

処理項目　：　・ヘッダ情報として上位へ渡す

・連続した複数ヘッダが有る場合には、全ヘッダ情報をリード

5.4.2.アプリケーションレイヤーからの画像検索方法

アプリケーションレイヤーにおける画像データの検索方法の例と、その際の留意点について以下に述べる。

(1) ファイル名またはファイルＩＤが既に分かっている場合：

・上記 5.4.1. に示す読出し手順により、ファイルリードを行う。

(2) ヘッダ情報から画像データを選択する場合：

・Ｂゾーンのヘッダ情報をすべて逐次読出し、その情報を元に木構造等の検索リストを作成し、画像を選択する。

・ヘッダデータの先頭にインデックスのファイルＩＤが格納されているので、選択された画像のファイルＩＤからインデックステーブルを検索し、該当のインデックスを得ることができる。そのインデックスのデータポイントから画像データを得ることができる。

5．5．仮削除から実削除へ

5.5.1.実削除できるファイル

実削除は仮削除されているファイルに対してのみ可能である。

5.5.2.仮削除の状態

仮削除されているファイルの内部状態は、登録されているファイルの内部状態と次の点が異なる。

すなわち、仮削除されているファイルは、①そのインデックス内の属性情報に仮削除フラッグが立ち、②ヘッダのファイルＩＤが負になっている。

従って、ゾーンテーブル、セクタテーブル、インデックス、ヘッダ、データのすべては媒体を占有する形で存在している。

5.5.3.実削除とは

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) ゾーンテーブルとセクタテーブルの該当エリアを更新する。

(6) 「ボリューム管理情報その２」の「仮削除ファイル数」、「空インデックス数」および「空インデックス開始番号」を更新する。

個々のアルゴリズムの例について次項で述べる。

5.5.4.アルゴリズムの例

(1) ヘッダの解放

1)セクタテーブルを非占有にする。

・該当インデックス内の「ヘッダポインタ」にもとずき、対応するセクタテーブルを未使用状態にする。

未使用状態（＝０）にするビットは、
先頭ビット番号　＝「ヘッダポインタ」の「開始セクタ番号」から
継続ビット数　　＝「ヘッダポインタ」の「連続セクタ数」である。

2)ヘッダ実体をクリアする。

・ヘッダデータ先頭４バイトのファイルＩＤをゼロクリアする。

(2) データ領域の解放と、必要な場合にゾーンの解放

1)親インデックスに基ずく解放

残バイト長 ＝ 画像データの「バイト長」
ｐ＝１　（親インデックスの＃１ポインタ）

ＷＨＩＬＥ（残バイト長＞０　かつ　ｐ＜＝１０）

①＃ｐポインタ内の「開始セクタ番号」と「連続セクタ数」に基づきセクタテーブルを未使用状態にする。
②解放領域先頭のゾーンについて、ゾーンテーブルの空ブロック数を更新する。全ブロックが使用可能であれば、そのゾーンを解放する。

当該先頭ゾーン番号 ＝ 「開始セクタ番号」／「ゾーン内セクタ数」の商

③解放領域が隣接ゾーンにまたがる場合は、そのゾーンの空ブロック数を更新する。全ブロックが使用可能であれば、そのゾーンを解放する。

残バイト長 ＝ 残バイト長 － 「セクタ数」＊「論理セクタ容量」
ｐ＝ｐ＋１

ＥＮＤ　ＷＥＩＬＥ

2)子インデックスに基ずく解放

ＷＨＩＬＥ（残バイト長＞０　かつ　次に子インデックスあり）

ｐ＝１

ＷＨＩＬＥ（残バイト長＞０　かつ　ｐ＜＝１７）

①＃ｐポインタ内の「開始セクタ番号」と「連続セクタ数」に基づきセクタテーブルを未使用状態にする。
②解放領域先頭のゾーンについて、ゾーンテーブルの空ブロック数を更新する。全ブロックが使用可能であれば、そのゾーンを解放する。

当該先頭ゾーン番号 ＝ 「開始セクタ番号」／「ゾーン内セクタ数」の商

③解放領域が隣接ゾーンにまたがる場合は、そのゾーンの空ブロック数を更新する。全ブロックが使用可能であれば、そのゾーンを解放する。
＃ｐポインタ内の「開始セクタ番号」と「セクタ数」に基づきセクタテーブルを未使用状態にする。

残バイト長 ＝ 残バイト長 － 「セクタ数」＊「論理セクタ容量」
ｐ＝ｐ＋１

ＥＮＤ　ＷＨＩＬＥ

ＥＮＤ　ＷＨＩＬＥ

(3) インデックステーブルの解放

親インデックスの「ファイルＩＤ」を０にする。
空エリアチェイン処理をする（4.4.(6)を参照）。

ＷＨＩＬＥ（「子のインデックステーブル番号」＜０）

子のインデックスについてその中の「インデックステーブル番号」を０にする。

空エリアチェイン処理をする（4.4.(6)を参照）。

ＥＮＤ　ＷＨＩＬＥ

(4) ボリューム管理情報その２の更新

（仮削除ファイル数）←（仮削除ファイル数）－１

（空インデックス数）←（空インデックス数）＋（実削除したファイルが所有していた親インデックスと子インデックスの数）

（空インデックス開始番号）←（実削除したファイルのファイルＩＤ）

（セキュリティ種別ファイル数）←（セキュリティ種別ファイル数）－１

６．ディレクトリテーブルの取り扱い

パーソナルコンピュータやワークステーションでは、ファイルを階層的に取り扱うことができ、データをその種類、属性などで分類、保存することができる。（ディレクトリ機能）ＩＳ＆Ｃにおいてもこのような機能をサポートする予定である。

６．１．ディレクトリの扱い方法

ＩＳ＆Ｃ論理フォーマットでは、ディレクトリの記述の方法を示し、それを１つのファイル（ディレクトリファイル）として管理することにより、階層構造をサポートするものとする。そして、その応用方法（階層構造の作成方法）は、各ユーザあるいは、各システムメーカにより異なるのが普通である。また、１つのボリューム内に複数のディレクトリファイルも存在可能である。

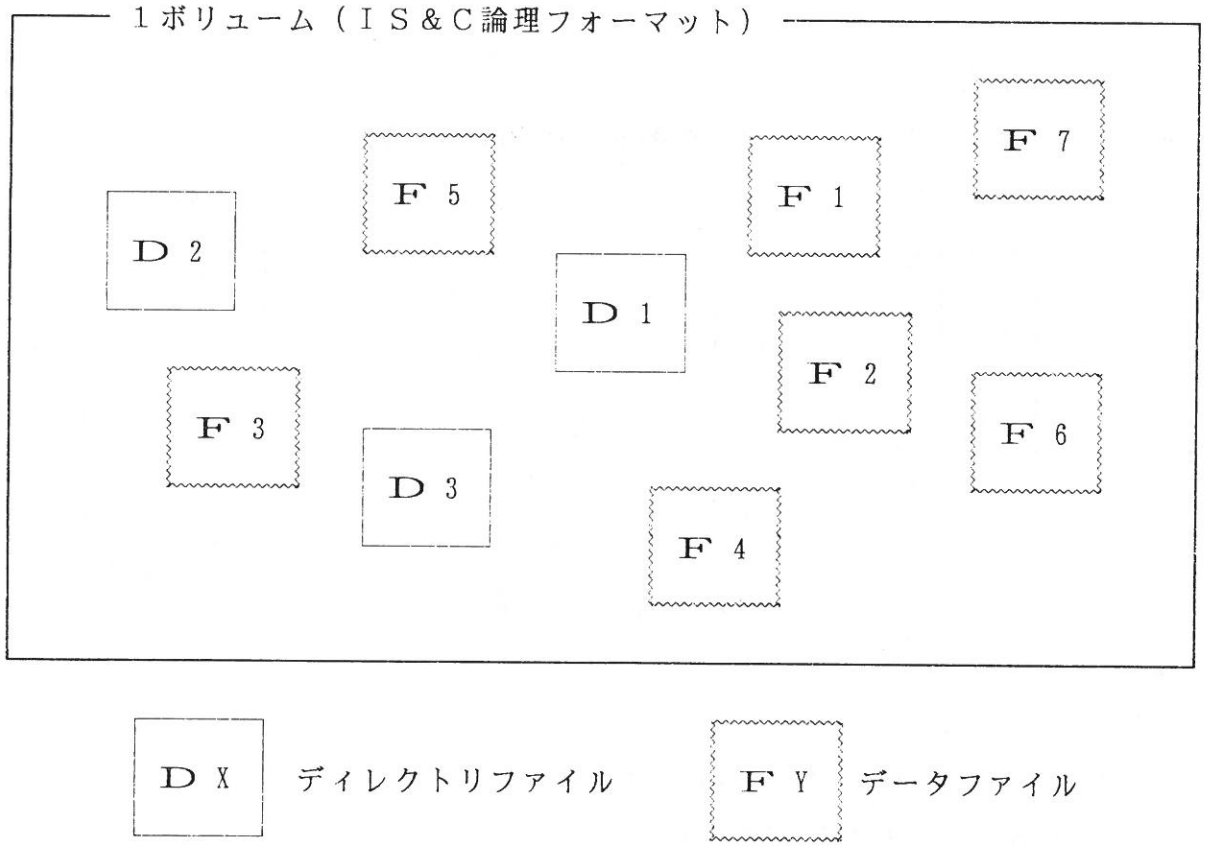


図6.1.1.ディレクトリファイル

6．2．ディレクトリファイルの構造例

6.2.1.ファイル構成

ディレクトリファイルは、固定長レコード(32ﾊﾞｲﾄ)ファイルである。

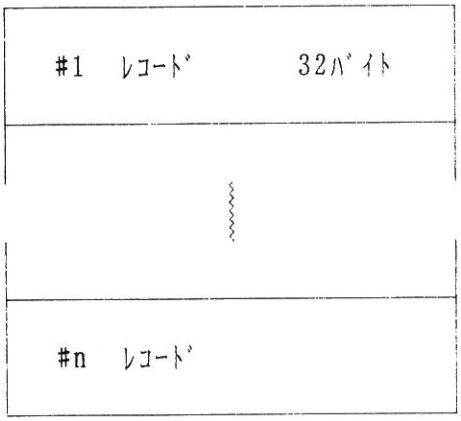


図6.2.1.ファイル構成

6.2.2.レコード構成

各レコードは、以下のような３つの部分より構成される。

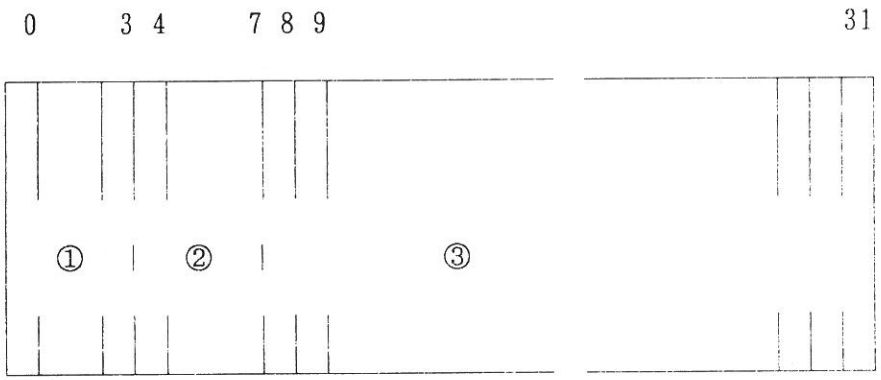


図6.2.2レコード構成

①IDﾅﾝﾊﾞｰ(4ﾊﾞｲﾄ)

<0:ﾃﾞｨﾚｸﾄﾘIDﾅﾝﾊﾞｰ

ﾃﾞｨﾚｸﾄﾘﾌｧｲﾙ内で便宜的につけられるID。

ﾃﾞｨﾚｸﾄﾘは名称のみで実態は無い。

>0:ﾌｧｲﾙIDﾅﾝﾊﾞｰ

ｲﾝﾃﾞｯｸｽﾃｰﾌﾞﾙ内のﾌｧｲﾙIDと同意。

②親ﾃﾞｨﾚｸﾄﾘIDﾅﾝﾊﾞｰ(4ﾊﾞｲﾄ)

自分の親にあたるﾃﾞｨﾚｸﾄﾘの①IDﾅﾝﾊﾞｰである。

すべて負数である。

③名称ｴﾘｱ(24ﾊﾞｲﾄ)

①のIDﾅﾝﾊﾞｰが負数の時、そのﾃﾞｨﾚｸﾄﾘ名を格納する。

正数の時、ﾌｧｲﾙ名を格納する。

6.2.3.その他のレコード

上記のような一般レコードの他に、ディレクトリファイルの終端を示す終端レコードと、空レコードが存在する。

・終端レコード

①IDﾅﾝﾊﾞｰ = 0

②親ﾃﾞｨﾚｸﾄﾘIDﾅﾝﾊﾞｰ = 0

・空レコード

①IDﾅﾝﾊﾞｰ = 0

②親ﾃﾞｨﾚｸﾄﾘIDﾅﾝﾊﾞｰ ≠ 0

6．3．ディレクトリファイルの作成例

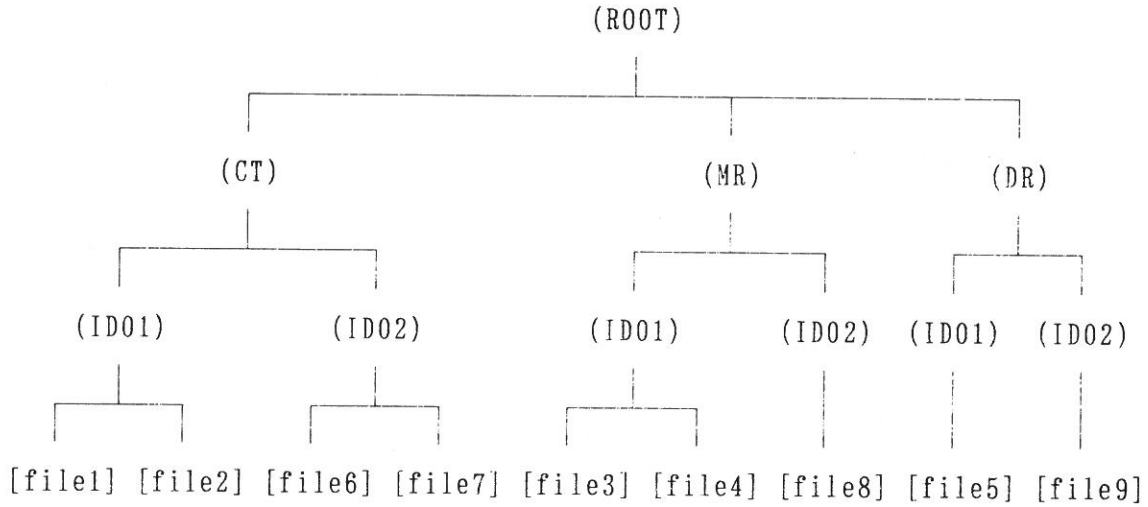


図6.3.1.ディレクトリの例

図6.3.1.に示すような階層構造をＩＳ＆Ｃ仕様で作成すると、表6.3.1.ディレクトリファイル作成例の通りである。

①ID ﾅﾝﾊﾞｰ	②親ﾃﾞｨﾚｸﾄﾘ IDﾅﾝﾊﾞｰ	③名称ｴﾘｱ	備　　考
-1	-1	ROOT	名称のみ
-2	-1	CT	名称のみ
-3	-2	ID01	名称のみ
1	-3	file1	
2	-3	file2	
-4	-1	MR	名称のみ
-5	-4	ID01	名称のみ
3	-5	file3	
4	-5	file4	
-6	-1	DR	名称のみ
-7	-6	ID01	名称のみ
5	-7	file5	
-8	-2	ID02	名称のみ
6	-8	file6	
7	-8	file7	
-9	-4	ID02	名称のみ
8	-9	file8	
-10	-6	ID02	名称のみ
9	-10	file9	
0	0		終端ﾚｺｰﾄﾞ

表6.3.1.ディレクトリファイル作成例

７．結言

本規格書で述べられてきたディスクフォーマットは医療分野、光磁気ディスク装置およびディスク、ワークステーションおよび医療情報システムなどの多方面の専門家が共同作業を行い、約２年の歳月と情熱をかけて完成させたものである。深夜におよぶ熱心な議論を何度も繰り返された委員の方々には感謝と敬意を表したい。

本フォーマットは主に１３０ｍｍ光磁気ディスクを規定しているが、これに限定するものではなく、容量が増加した次世代のものでも、９０ｍｍの光磁気ディスクでもさらにハードディスクの場合にでも適用可能である。すなわち１０２４バイト／論理セクタが定義できるランダムアクセス可能な記録媒体であれば、いずれにも適用可能である。

本フォーマットは汎用性を考慮して規格化したので、医療分野以外のフォーマットとしても利用可能である。大容量の情報交換で互換性が必要な分野には特に有用である。その為、医療固有部分は別冊とし本規格書は共通部分のみを規定した。

本規格書に基づきファイルマネージャを自由に作成することは妨げないが、作成の労を省き、互換性を促進するために、有償でファイルマネージャのサンプルプログラムを提供している。ご利用願いたい。

また互換性を進めるためには国際標準とする事が必須であり、今後こうした活動が重要であるので、ＩＳＯおよびＡＣＲ－ＮＥＭＡと連絡をとり準備を進めている。

本フォーマットは実用レベルということでＶ１．０とした。活用上の問題点は今後、ご指導いただき、バージョンアップに反映させて行きたい。問題点の収集方法およびバージョンアップ方法についても今後ルール化して行く予定である。

本フォーマットが広く普及し、光磁気ディスクの互換性が計られると、色々なワークステーションで画像情報などの大容量データを互いに利用できるようになり、情報システムの質的変化が期待できる。

新しい情報交換手段の出現は新しい社会システムを可能にする。また新しい情報の蓄積は新たな価値を創造する。そのためにも本規格書を十分活用願い役立てて頂きたい。

付録Ａ．ファイルマネージャ・サービス関数仕様

本サービス関数はＣ言語で利用可能なものであり、戻り値は long 型のエラーコードとなっている。

1．ディスクの管理

1.1.ディスクのマウント

```
long    ifm_mount_media( un, vd );
    long      un,           /* Unit number */
              *vd;          /* Volume discripter */
```

サービスの開始時に使用。ディスクの管理情報を読込む。

1.2.ディスクのディスマウント

```
long    ifm_dismount_media( vd );
    long    vd;             /* Volume discripter */
```

サービス終了時に使用。オープンされているファイルをクローズし、管理情報をディスクに書込む。

1.3.ディスクのサイズを得る

```
long    ifm_get_space( vd, msize, usize );
    long        vd,         /* Volume Discripter */
              *msize,       /* メディアのサイズ(KB)  */
              *usize;       /* メディアの使用量  */
```

ボリュームのトータルサイズ及び使用量を得る。 使用量には仮削除されているファイルを含んでいる。

1.4.ディスクのＩＳ＆Ｃ用初期化

```
long    ifm_format_media ( un, voldata );
    long        un;         /* Unit nuber */
    struct volumedatatype  *voldata;
                            /* ０＆１セクターのボリューム管理データ */
```

物理フォーマットされたディスクをＩＳ＆Ｃで使用するためにボリューム管理データの初期化をする。

2. ファイル管理

2.1.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,             /* Volume discripter */
            id,             /* 若番のファイルＩＤ */
            req_size,       /* 要求ファイル数 */
            *act_size;      /* 獲得ファイル数 */
    char    *array;         /* 管理情報のはいるデータアレイとしてreq_size *
                               64 bytes の領域が必要 */
```

id（＞０）で指定したファイルからインデックス中のポインター以外の情報（各々64バイト）を得る。仮削除したデータの情報を得ることはできない。

2.2.指定した大きさのサイズ固定ファイルを作成する

```
long    ifm_create_file( vd, name, size, flag, id );
    long    vd,             /* Volume discripter */
            size,           /* ファイルの大きさ */
            flag,           /* 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 */
            *id;            /* ファイルのＩＤ */
    char    *name;          /* ファイル名 */
```

ファイル名の文字列が２４文字未満の場合には ｎｕｌｌ（００Ｈ）で終了してる必要がある。また、この関数はインデックスと領域の確保だけでオープンは行わない。

2.3.指定した大きさのブロックを持つサイズ可変ファイルを作成する

```
long    ifm_create_file_variable( vd, name, cell, id );
    long    vd,             /* Volume discripter */
            cell,           /* ブロックの大きさ */
            *id;            /* ファイルＩＤ */
    char    *name;          /* ファイル名 */
```

ファイルの大きさを指定せずにファイルを作成することができる。
書込み時に、指定したブロック・サイズの単位で領域が確保されていく。

2.4.指定したファイル名に合致するファイルＩＤを得る

```
long    ifm_get_file_id( vd, name, id, req_size, act_size );
    long    vd ,            /* Volume Descripter */
            id[ ],          /* ファイルＩＤ、req_size以上の配列であること */
            req_size,       /* 要求ファイル数 */
            *act_size;      /* 獲得ファイル数 */
    char    *name;          /* ファイル名 */
```

ファイル名にはワイルドカード＊と？が使用可能である。
？は１文字を、＊は何れかの文字列があることを意味する。

2.5.ファイルの仮削除を行う

```
long    ifm_delete_file ( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

3．ファイル単位のデータの入出力等

3.1.ファイルＩＤによるファイルのオープン

```
long    ifm_open_file( vd, id, mode, fd );
    long    vd,             /* Volume discripter */
            id,             /* ファイルＩＤ */
            mode,           /* オープンのモード */
            *fd;            /* File discripter */
```

ファイルＩＤを指定してファイルをオープンする。
オープンのモードには読出し専用(Read Only)、書込み専用(Write Only)、
読み書き(Read and Write)の３種がある。

3.2.ファイルのクローズ

```
long    ifm_close_file( fd );
    long    fd;             /* File discripter */
```

3.3.ファイル単位の相対セクタ指定によるデータの読出し

```
long    ifm_read_data( fd, buf, st, req_len, act_len, swp );
    long    fd,             /* File discripter */
            st,             /* ファイルの先頭からの位置
                               (1024の整数倍のバイト単位) */
            req_len,        /* 要求する大きさ（1024の整数倍のバイト単位）*/
            *act_len,       /* 実際に読み込まれた大きさ */
            swp;            /* バイトスワップを行うか否かのフラグ */
    char    *buf;           /* データアレイ */
```

swp は０以外の時にデータのバイトスワップを行う。
st　は1024の整数倍でないとエラーとなる。
また、req_len は1024の整数倍に切り上げられる。

3.4.ファイル単位の相対セクタ指定によるデータの書込み

```
long    ifm_write_data( fd, buf, st, req_len, act_len, swp );
    long    fd,             /* File discripter */
            st,             /* ファイルの先頭からの位置
                               (1024の整数倍のバイト単位) */
            req_len,        /* 要求する大きさ (1024の整数倍のバイト単位) */
            *act_len,       /* 実際に書込まれた大きさ */
            swp;            /* バイトスワップを行うか、否かのフラグ */
    char    *buf;           /* データアレイ */
```

swp は０以外の時にデータのバイトスワップを行う。
st は1024の整数倍でないとエラーとなる。
また、req_len は1024の整数倍に切り上げられる。

3.5.ファイルのヘッダの読出し

```
long    ifm_read_header( fd, buf, req_len, act_len );
    long    fd,             /* File discripter */
            req_len,        /* バッファの大きさ */
            *act_len;       /* 獲得した大きさ */
    char    *buf;                   /* データアレイ */
```

ヘッダデータ長の最大は３２Ｋ－６バイトである。

3.6.ファイルのヘッダの書込み

```
long    ifm_write_header( fd, buf, req_len, act_len );
    long    fd,             /* File discripter */
            req_len,        /* 要求する大きさ */
            *act_len;       /* 書込んだ大きさ */
    char    *buf;           /* データアレイ */
```

3.7.ファイルのサイズを得る

```
long    ifm_get_file_size( fd, fsize, hsize );
    long    fd,             /* File discripter */
            *fsize,         /* ファイルの大きさ */
            *hsize;         /* ヘッダの大きさ */
```

3.8.ファイル名を得る

```
long    ifm_get_filename( fd, name );
    long    fd;             /* File discripter */
    char    *name;          /* ファイル名 */
```

3.9.ファイル名を記録する

```
long    ifm_put_filename( fd, name );
    long    fd;             /* File discripter */
    char    *name;          /* 新しいファイル名 */
```

4. ヘッダデータの入出力

4.1.順番にヘッダを読む

```
long    ifm_read_header_sequential(vd, id, buf, req_size, act_size );
    long    vd,             /* Volume discripter */
            *id,            /* ファイルＩＤ */
            req_size,       /* バッファの大きさ */
            *act_size;      /* 実際に読込んだ大きさ */
    char    *buf;           /* データバッファ */
```

4.2.ヘッダの読出し位置カウンターを先頭に戻す

```
long    ifm_reset_header_counter( vd );
    long    vd;             /* Volume discripter */
```

5. システム管理

5.1.仮削除を含む全てのファイルの管理情報を得る

```
long    ifm_get_list_all( vd, id, array, req_size, act_size );
    long    vd,             /* Volume discripter */
            id,             /* ファイルＩＤ */
            req_size,       /* 要求ファイル数 */
            *act_size;      /* 獲得ファイル数 */
    char    *array;         /* データアレイ req_size * 64 バイトの領域が
                               必要 */
```

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（各々６４バイト，ただし、４６～６４バイトはｎｕｌｌである）を得る。

5.2.ファイルの実削除を行う

```
long    ifm_delete_actually( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.3.仮削除されていたファイルを再登録する

```
long    ifm_recover_temporaly_deleted( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.4.書込み禁止フラグの設定

```
long    ifm_write_protect_on( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.5.書込み禁止フラグの解除

```
long    ifm_write_protect_off( vd , id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.6.読出し禁止フラグの設定

```
long    ifm_read_protect_on( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.7.読出し禁止フラグの解除

```
long    ifm_read_protect_off( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.8.システムファイルフラグの設定

```
long    ifm_system_flag_on( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.9.システムファイルフラグの解除

```
long    ifm_system_flag_off( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.10.ディレクトリファイルフラグの設定

```
long    ifm_directory_flag_on( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

5.11.ディレクトリファイルフラグの解除

```
long    ifm_directory_flag_off( vd, id );
    long    vd,             /* Volume discripter */
            id;             /* ファイルＩＤ */
```

☆☆　以下はファイルマネージャが内部で使用する関数である　☆☆

6. ユーティリティー

6.1.ＭＯＤへの書込み

```
long    iut_write_mod( un, dt, st_sct, len, swp, stt );
    long    un,             /* ユニットナンバー */
            st_sct,         /* 先頭セクタ番号 */
            len,            /* 大きさ（バイト単位） */
            swp,            /* バイトスワップのフラグ */
            *stt;           /* ドライブからのステータス */
    char    *dt;            /* データアレイ */
```

6.2.ＭＯＤからの読出し

```
long    iut_read_mod( un, dt, st_sct, len, swp, stt );
    long    un,             /* ユニットナンバー */
            st_sct,         /* 先頭セクタ番号 */
            len,            /* 大きさ（バイト単位） */
            swp,            /* バイトスワップのフラグ */
            *stt;           /* ドライブからのステータス */
    char    *dt;            /* データアレイ */
```

6.3.ファイルインデックスを読出す

```
long    iut_get_index( un, idx_no, idx_dt, len, flag );
    long    un,             /* ユニットナンバー */
            idx_no,         /* インデックスナンバー */
            len,            /* 読出す長さ（≦１２８） */
            *flag;          /* インデックス識別フラグ */
    union idxtype *idx_dt;  /* インデックスデータ */
```

6.4.ファイルインデックスを書込む

```
long    iut_put_index( un, idx_no, idx_dt );
    long    un,             /* ユニットナンバー */
            idx_no;         /* インデックスナンバー */
    union idxtype *idx_dt;  /* インデックスデータ */
```

6.5.ディスクの排出を禁止する

```
long    iut_lock_mod( un );
    long    un;             /* ユニットナンバー */
```

6.6.ディスクの排出を許す

```
long        iut_unlock_mod( un );
    long    un;             /* ユニットナンバー */
```

6.7.新たなゾーンを得る

```
long        iut_define_zone( un, ztype );
    long    un,             /* ユニットナンバー */
            ztype;          /* ゾーンの種別 */
```

6.8.指定した種別のゾーン中の予約領域のポインタを得る

```
long        iut_get_pointer( un, ztype, pnt );
    long     un,            /* ユニットナンバー */
             ztype;         /* ゾーンの種別 */
    struct idxptrtype *pnt; /* ポインタ */
```

6.9.指定したゾーン中の予約領域の先頭のポインタを得る

```
long    iut_get_start_sector( un, zone_no, pos );
    long     un,            /* ユニットナンバー */
             zone_no,       /* ゾーン番号 */
             *pos;          /* 先頭のセクタ号 */
```

6.10.指定したポインタの領域を解放する

```
long    iut_release_pointer( un, pnt );
    long    un;             /* ユニットナンバー */
    struct idxptrtype *pnt; /* ポインタ */
```

6.11.Bゾーン中にヘッダのポインタを得る

```
long    iut_get_header_pointer( un, nsect, pnt );
    long    un,             /* ユニットナンバー */
            nsect;          /* 連続セクタ数 */
    struct idxptrtype *pnt; /* ポインタ*/
```

6.12.ワードデータ列のコピー

```
long    iut_copy_word( sc, dst, len );
    short   *sc,            /* ソースデータアレイ */
            *dst;           /* デスティネーションデータアレイ */
    long    len;            /* コピーするワード数 */
```

6.13.４バイトデータ列のコピー

```
long    iut_copy_dword( sc, dst, len );
    long    *sc,            /* ソースデータアレイ */
            *dst,           /* デスティネーションデータアレイ */
            len;            /* コピーする４バイトデータ列の数 */
```

6.14.バイトスワップを行う

```
long        iut_byte_swap( ar, len );
    long    len;            /* スワップする長さ（偶数） */
    char    *ar;            /* データアレイ */
```

参考資料：

○ ファイルマネージャサービス一覧　東工大　大山永昭　１９９０年９月
○ ＩＳ＆Ｃ光磁気ディスクマネージャ・ソフトウェア設計仕様書　ＪＢＡ
１９９０年１１月
○ ＵＮＩＸ版暫定仕様ファイルマネージャ（サンプルソフト）

付録Ｂ．エラーコード

ここではＩＳ＆Ｃファイルマネージャをコールした場合にユーザに返されるエラーコードをまとめている。尚、ファイルマネージャ内のエラー検出状況をより詳細に調べる必要がある場合のため、ファイルマネージャ内のエラー検出部位を知るためのシークエンス番号をグローバル変数に格納する様になっている。

ユーザに返されるエラーコードとその内容

コード	ニーモニック	内容
(1) 論理的なエラー		
0x0001	ENUNIT	存在しないユニット番号を指定した。
0x0002	EMNTED	指定したユニットはマウントされている。
0x0005	ENBUFF	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の数が多すぎる。
0x0006	ENMNTD	指定のボリュームはマウントされていない。
0x0007	EPARAM	関数のパラメーターが不正である。
0x0008	ENINDX	インデックス領域に空きが無い。
0x0009	ENDATA	データ領域に空きが無い。
0x000A	ENFILE	指定したファイルは存在しない。
0x000B	EOPNED	指定したファイルはオープンされている。
0x000C	ENPERM	禁止されているアクセスを行おうとした。
0x000D	ENOPND	指定したファイルはオープンされていない。
0x000E	ESZOVR	開始位置がファイルのデータサイズ以上である。
0x000F	ENDELD	指定したファイルは削除されていない。
0x0010	ESYSRD	Ａゾーンの READに失敗し、A’ゾーンのデータを使用した。
0x0020	ENHEAD	ヘッダ領域に空きが無い。
(2) オペレータの介入でリカバリー出来るエラー		
0x5000	EMWPRT	ライトプロテクトメディアに書込もうとした。
0x5001	EUNRDY	ユニットノットレディー

コード　ニーモニック　内　　容

(3) メディアの診断を必要とするエラー

0x6000　ENVINF　Vinfの内容が正しいボリューム管理情報ではない。

0x6001　ENISAC　メディアがＩＳ＆Ｃ対象フォーマットではない。

0x6002　EMEDUM　ディスクの物理エラーが発生した。

(4) ドライブの診断を必要とするエラー

0x7000　ETMOUT　タイムアウト

0x8000　EDRIVE　物理エラーが発生した。このエラーは上記のいずれにも当てはまらないエラーの場合にのみ発行される。　また、このエラーコードの下位バイトにはエラー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。

0xC000　ESYSIO　システム領域アクセス時に物理エラーが発生した。
0x8000 EDRIVE と同様の注意が必要。

付録Ｃ．ボリュームの初期化

130mmの光磁気ディスク(1024ﾊﾞｲﾄ/ｾｸﾀ)を例にしている。
物理的な初期化は、工場出荷時に実施済みとする。
Ａゾーンのバックアップゾーンをそのコピーとして作成する。
特別な指定がなければＢゾーンを作成しない。
文字型データは左詰めとし、空白部分はｎｕｌｌ（ＯＯＨ）を埋める。

１．ボリューム管理情報（第０セクタ）

ゾーン数は、デバイスドライバーコマンドのＲＥＡＤ　ＣＡＰＡＣＩＴＹコマンドで求めた最大有効論理セクタ数、あるいは、上位システムで指定されたボリューム上の任意領域から設定する。

以下では、ゾーン数＝３０６と仮定している。

表1.ボリューム管理情報（第０セクタ）

address	名　　前	長さ(B)	ﾃﾞｰﾀﾀｲﾌﾟ	初期値
0～3	初期化識別子	4	文字型	IS&C
4～7	バージョン番号	4	文字型	01.0
8～23	適用分野	16	文字型	MEDICAL
24～55	ボリューム名称	32	文字型	ﾕｰｻﾞ指定
56～59	ボリュームＩＤ	4	整数型	ﾕｰｻﾞ指定
60～91	所有者名	32	文字型	ﾕｰｻﾞ指定
92～123	所有者コード	32	文字型	ﾕｰｻﾞ指定
124～129	初期化日時	6	文字型	ｼｽﾃﾑ日付
130～133	ゾーン数	4	整数型	306
134～135	ゾーン内セクタ数	2	整数型	1024
136～137	セクタ容量	2	整数型	1024
138～141	ｿﾞｰﾝﾃｰﾌﾞﾙ開始ｾｸﾀ番号	4	整数型	2
142～145	ｾｸﾀﾃｰﾌﾞﾙ開始ｾｸﾀ番号	4	整数型	4
146～149	ｲﾝﾃﾞｯｸｽ・ﾃｰﾌﾞﾙ開始ｾｸﾀ番号	4	整数型	43
150～151	インデックス・サイズ	2	整数数	128
152～1023	予約領域	872		0

2. ボリューム管理情報（第１セクタ）

表2.ボリューム管理情報（第１セクタ）

address	名　　前	長さ(B)	ﾃﾞｰﾀﾀｲﾌﾟ	初期値
0～3	インデックス総数	4	整数型	7848
4～7	登録ファイル数	4	整数型	0
8～11	仮削除ファイル数	4	整数型	0
12～15	空きインデックス数	4	整数型	7848
16～17	システムファイル数	2	整数型	0
18～19	ディレクトリファイル数	2	整数型	0
20～25	更新日時	6	整数型	初期化日時
26～29	空きインデックス開始番号	4	整数型	1
30～31	ボリューム使用中フラグ	2	整数型	0
32～1007	予約領域	976		0
1008～1023	システム予約領域	16		0

3. ゾーンテーブル

表3.ゾーンテーブルの初期値

項　　目	長さ（B）	初　期　値		
		ｿﾞｰﾝ種別(2B)	空きﾌﾞﾛｯｸ数(2B)	ﾊﾞｯｸｱｯﾌﾟｿﾞｰﾝ番号(2B)
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ1	6	1	-1	N
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ2	6	0	0	0
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ3	6	0	0	0
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘ4	6	0	0	0
:				:
:				:
ｿﾞｰﾝﾃｰﾌﾞﾙｴﾝﾄﾘN	6	-1	-1	1

※　最終セクタ後部の空領域は、ｎｕｌｌ（００Ｈ）が埋められる。

4．セクタテーブル

セクタテーブルは、ボリューム上の全論理セクタの使用状況を各１ビットに対応させて管理するテーブルである。セクタテーブルの初期値は、Ａゾーン（第１ゾーン）とそのバックアップゾーン（第３０６ゾーン）に対応するセクタは全てＯＮ（１：使用済）とし、残りは全てＯＦＦ（０：未使用）である。

※　最終セクタ後部の空領域には、ｎｕｌｌ（００Ｈ）を埋める。

5．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、子のテーブル番号（１２４～１２７バイトアドレス）に、２，３，４，～，７８４７，-１の継続番号を入れる。

6．ヘッダゾーン

各々のセクタの先頭４バイトのファイルＩＤに、ｎｕｌｌ（００Ｈ）を埋める。

付録Ｄ．

表Ｄ．処理手順と使用テーブル

処理手順＼使用ﾃｰﾌﾞﾙ	ﾎﾞﾘｭｰﾑ管理情報 その１	ﾎﾞﾘｭｰﾑ管理情報 その２	ｿﾞｰﾝ ﾃｰﾌﾞﾙ	ｾｸﾀ ﾃｰﾌﾞﾙ	ｲﾝﾃﾞｯｸｽ ﾃｰﾌﾞﾙ	Aｿﾞｰﾝ ﾊﾞｯｸｱｯﾌﾟ 領域	Bｿﾞｰﾝ	ﾃﾞｰﾀ 領域
初期化	I	I	I	I	I	I		
マウント	R	R&U／W ﾎﾞﾘｭｰﾑ使用中Ｆ	R	R	R			
create-file		U／W	U／W	U／W	U／W			
open-file					U／W 書込中Ｆ			
Writeldata ｻｲｽﾞ固定		U／W	U／W	U／W	U／W			W
Writeldata ｻｲｽﾞ可変		U／W	U／W	U／W	U／W			W
write-header			U／W	U／W	U／W		W	
fileの属性変更		U／W			U／W			
close-file					U／W 書込中Ｆ			
ディスマウント		U／W ﾎﾞﾘｭｰﾑ使用中Ｆ				Aｿﾞｰﾝ のｺﾋﾟｰ		

Ｉ：IS&C fileの初期化　　Ｒ：ﾃﾞｨｽｸから主記憶への読込み
Ｕ：主記憶上のﾃﾞｰﾀの更新　　Ｗ：ﾃﾞｨｽｸへの書込み

（問い合わせ先）

ＩＳ＆Ｃ規格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 10 Ｆ

TEL　０３（３５８６）６３２１　　FAX　０３（３５０５）１９９６

ＩＳ&Ｃ規格書

平成３年12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