IS&C Standard

Image Save And Carry

Volume and file structure

(Logical disk format Ver.1.0)

IS&C COMMITTEE / November,1992

Contents

1. Preface

The purpose of the IS&C system is, as the name 'Image Save and Carry' implies, to construct an off-line system using portable recording media with a high capacity. As are widely used word processors, this system would be an open system, which would allow us to effectively store and manually carry a huge amount of data and images. For this purpose, the three items listed below should be standardized in order to ensure a complete compatibility:

(1) Physical formats of recording media
(2) Volume and file structure (logical disk format)
(3) Application format (data format)

Immediately after the IS&C committee was organized in April 1989, it formed two working groups: the WG1 group to handle the volume and file structure and the WG2 group to handle the applica@tion format. At that time, ISO-SC23 was already working on standardi@ zation work for the physical format. The volume and file struc@ ture of the IS&C system was designed to be independent of conven@ tional operating systems so that it could be used in the medical field, as well as other fields. Items that have to be considered are listed as follows:

(1) The size of files stored in portable media varies from a few bytes to Megabytes.
(2) Sectors are used in a continuous mode for quick I/O of large-size data.
(3) The number of continuous sectors are self-adjustable depending on the size of file.
(4) It should be possible to maintain high performance without any garbage corrections even after a great deal of recording and erasing.
(5) Directory tables should be defined in a way that will permit several kinds of hierarchical structures.
(6) For medical application, the IS&C system itself should include a file protection mechanism.

It is our great pleasure to introduce the IS&C volume and file structure of vs. 1.0, which takes about 2 years to complete. This version has taken into account experience obtained through experiments using prototypes. We hope that IS&C will be widely used in many fields.

2. Concept of the IS&C format

2-1. Layer structure and file manager

The IS&C system defines functional layers to be explained as follows in order to establish full compatibility with the magneto-optical disks provided by multi-vendors (see Fig.2.1)

(1) Application layer

This layer represents the following functions:

* man-machine interface for command interpretation and informa@ tion display
* application software such as image processing
* management of the hierarchical structures of stored files

Since the IS&C system allows users to define several kinds of hierarchical structures, they are stored in disks as user files. Therefore, the use of these files will depend on the application software.

(2) Data format layer

This layer specifies both data and record structures, including the number of pixels, the hospital name and the patient's name. This layer also lightens the load of application software, since it allows access to files stored in the MOD by other stations.

(3) Disk format layer (file manager)

This layer specifies the volume and file structures. The volume structure has volume information including the volume name and the initialization date. The file structure specifies file IDs and their physical allocation. These structures are maintained by the file manager software that is provided as a sample. This draft of the IS&C standard describes this layer.

(4) Driver software layer

This layer controls the functions of the MOD drive via a certain interface such as SCSI, and is implemented by a device driver software, which depends on both CPU and operating system.

(5) SCSI interface layer

The MOD drive sends and receives data through the SCSI interface. Because SCSI commands have some options, including vendor unique commands, the sample software of file manager only uses group 1 commands, and the error codes are also commonly defined.

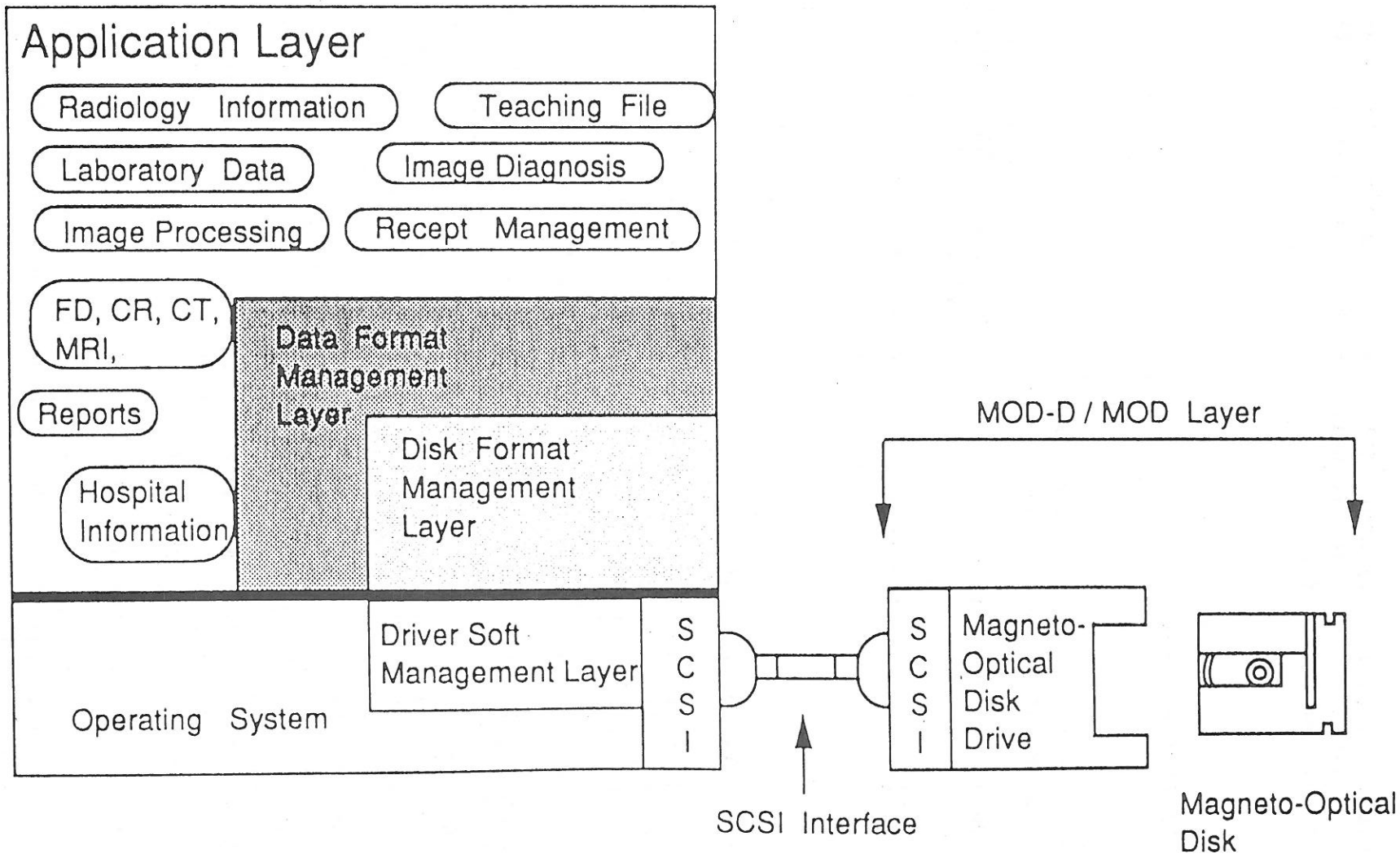


Fig. 2.1 Hierarchical structure of IS&C system

(6) MOD drive and disk layer

Because compatibility among MOD drives and media which are said to conform to the ISO standard is not guaranteed, IS&C specifies them more strongly.

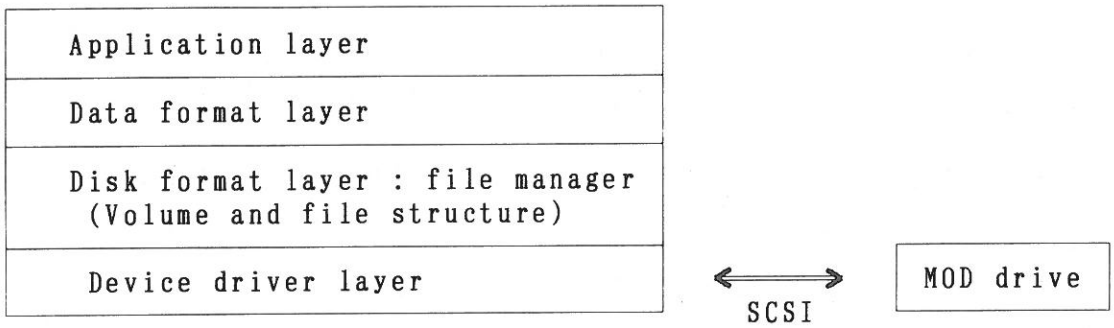


Fig. 2.1 Layer structure in the IS&C system

2-2. Disk format design policy

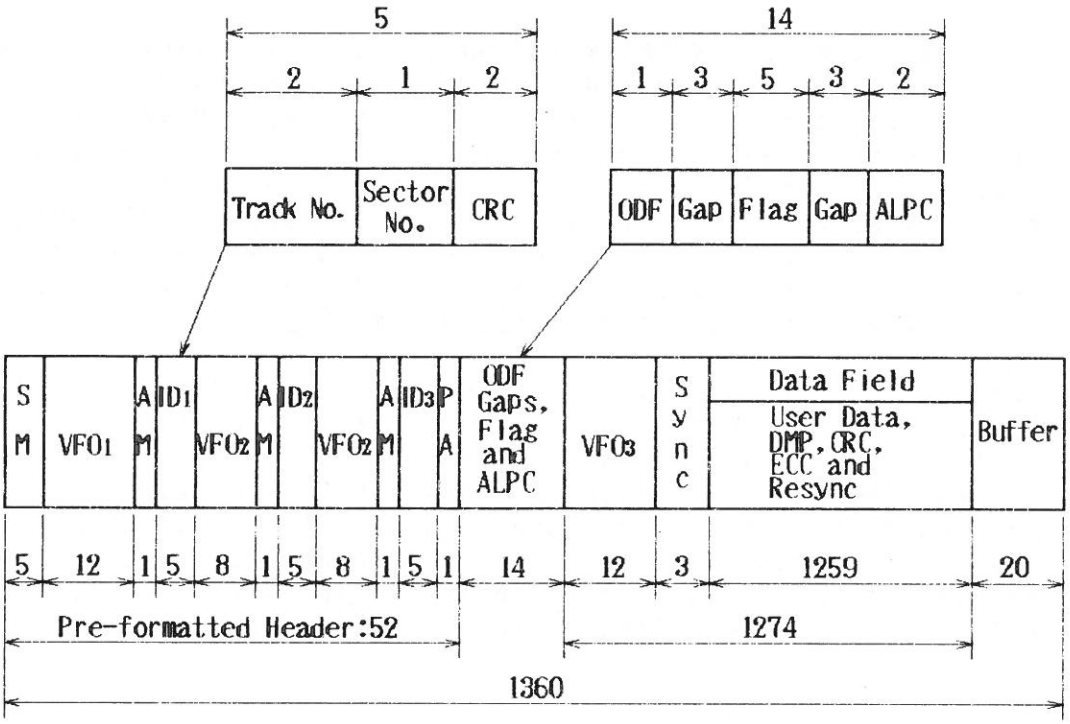
The IS&C disk format is designed to realize an effective file allocation of a large amount of data in varying quantities, ensure quick I/O, and be independent of the operating systems.

(1) Use of continuous sectors to save on search time

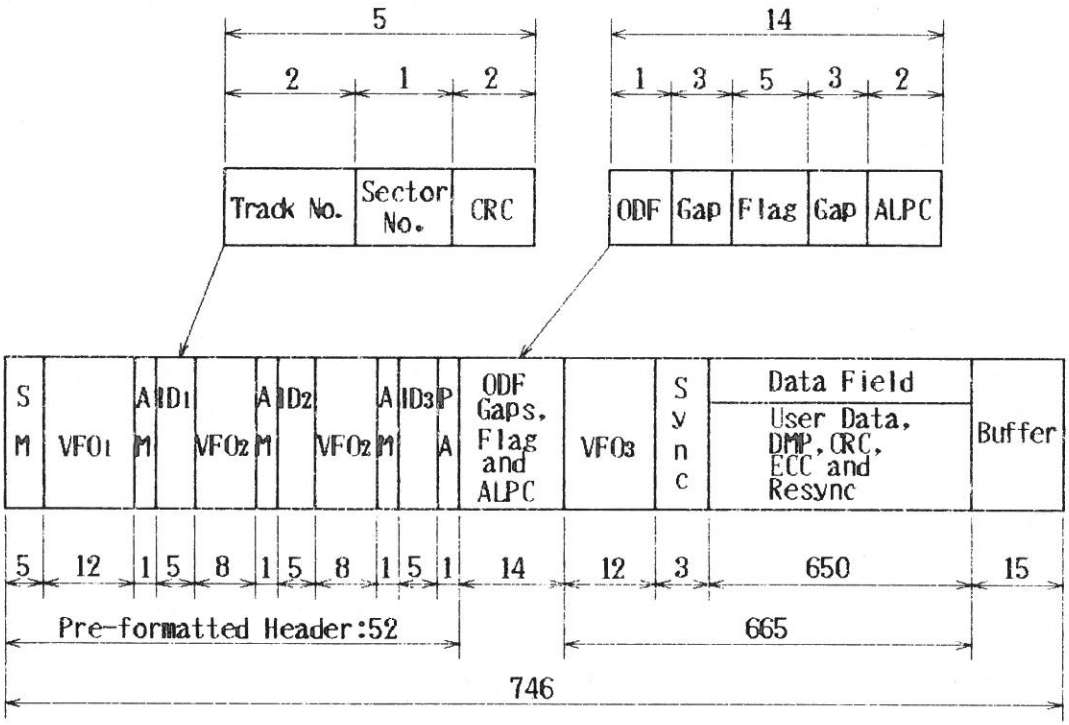
The MOD disk has a spiral structure of tracks, and each track is divided into several sectors. One sector is the minimum size of a recording unit. Fig.2.2.1 shows the physical format of one sector defined by ISO for a 130mm MOD disk. Each sector begins with a sector mark and its ID (for safety the track and sector numbers are recorded three times), which is followed by the data area. The data area size for the user's use is defined as either 1024 or 512 bytes, while the actual size is larger because of additional codes such as ECC and the sync. signal. The structure of the sector is basically similar to that of magnetic disks, but the optical head is rather heavier, which results in a longer search time. Therefore, the more continuous sectors the system uses, the shorter the I/O speed becomes.

(2) Length of the continuous sectors

For large-size data it is advantageous to use continuous sectors as much as possible. On the other hand, small-size data such as header or character data receives no preference in terms of space efficiency. This means that the cluster definition adopted in MS-DOS or some other systems is not suitable for purposes of IS&C.



Sector format for 1024 user bytes



Sector format for 512 user bytes

Fig.2.2.1 Physical format

(3) Introduction of the zone concept

In the case of a WORM (Write Once and Read Many) optical disk, we simply store data sequentially, while a MOD disk allows us to write, erase and read as many times as does a magnetic disk. Such sequences often cause the free area to be physically separated and therefore require a garbage correction of the sectors being used in order to gather the free area. UNIX systems use sectors in blocks of 512, 1024 or 4096 bytes, while MS-DOS introduces a cluster concept instead of a sector. Both of these systems, however, use blocks of a fixed size.

Because of the variety and large size of the data, the IS&C system introduces a zone concept, which enables several sizes of blocks to be defined. In vs. 1.0, zones are defined as having either 1k, 4k, 16k, 64k, 256k or 1024k bytes as a block, which uses continuous sectors. Each zone that is not for system use has the same size blocks, and blocks of different sizes are never defined as being in one zone. To allow the data to be read easily, the data is identified by the first sector number and its length.

(4) Volume definition

The IS&C vs. 1.0 system defines the volume on one side of the disk, and the capacity and location of each volume is logically defined. Consequently, the sectors within each volume are relatively numbered and ordered.

Note: The IS&C vs.1.0 system does not support multi-volume manage@ ment, although users can carry out such management by making use of the volume ID or name. One file should be completed within a given volume.

(5) System area

In this area tables indicating volume information, area usage and file allocation (see Fig.2.2.2) are stored. System areas are recorded in the system zones, which can be expanded when necessary.

(6) Header data

Header data refers to a set of information data or tables explaining the contents of a file. Such data is stored in the header zone. The merit of the usage of the header zone is that it allows for a quick survey of the contents of the files stored in a disk (see Fig.2.2.2).

(7) Directory file

The IS&C system itself does not support the hierarchical structure of the files, because of the various system operations. This file contains directory files which users can define as they like. The description of the directory files has been standar@ dized to ensure compatibility.

(8) Double recording

To improve system reliability, the IS&C system records data of importance in two areas, a primary area and a back-up area. In particular, the system area is recorded in both the outer and inner areas of the disk (see Fig.2.2.2).

Note: IS&C vs.1.0 supports double recording only for system data.

(9) Temporary delete

Because of its zone concept, the IS&C system usually uses temporary delete for safety reasons. Temporary delete simply hides a file from users or application software, but never actually deletes the data. Therefore, temporarily deleted files can be recovered anytime. If a disk is really full, users have to decide whether they will actually delete temporarily deleted files.

(10) Volume and write flags

During the data write, the IS&C system presents the write flag corresponding to the file being accessed. This flag indicates whether the file write procedure has been normally completed. While the volume is being mounted, the IS&C system presents the volume flag to indicate whether the volume has been normally dismounted.

(11) File attribute

The IS&C system supports file attributes for write protec@ tion, read protection, file access hierarchy and file classifica@ tion, etc.

(12) Order of bytes for the expression of an integer

Integer data stored in both system zone A and header zone B must be ordered in MSB-LSB. This is explained in greater detail in the section 3.1.

(13) Read/write procedure

In the case of writing data onto disks, volume information, the zone table and the sector table are sequentially referred to for access to the free area. Pointers for data and header data are then stored in the corresponding index table.

In the case of reading data from disks, there are two ways to identify the relevant file. The first way is in terms of the file ID, and the second way is through the file name. The file name, however, will not necessarily be unique. In this case header data must be referred to.

In the IS&C system, all header data is stored in the header zone symbolized by B. This allows for a quick search. From the header data, users can get the file ID (see Figs. 2.2.2 and 2.2.3).

Fig.2.2.2. File management in the IS&C system

Fig.2.2.3 Sample file structure

2.3 Variety of files

The IS&C system supports files with different sizes. When creating a file, users should choose either a fixed or a vaiable size.

(1) Fixed-size file

This type cannot change its size once it is specified at the time of creation. Therefore, data cannot be appended beyond the defined size. The advantage of this type is that data allocation can be optimized by consideration of the trade-off between space efficacy and speed.

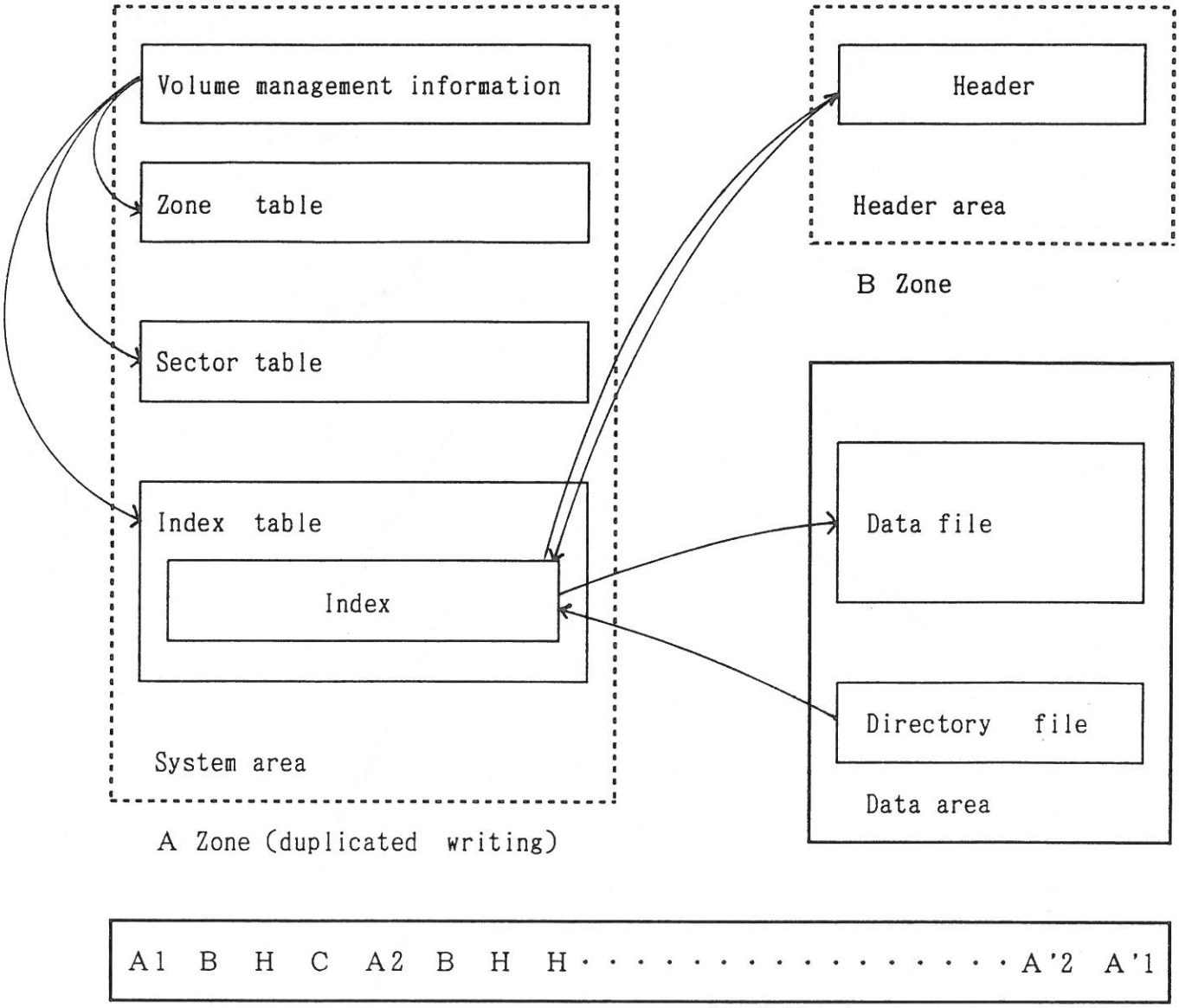
(2) Random-length file

This type can change its size in units of block size, which is specified at the time of file creation. The block size or the type of zone cannot changed once it has been created.

2.4 Temporary delete mechanism

2.4.1 Purpose and summary

Because MOD is rewritable, files recorded on these disks may be erased due to mis-operation. Therefore, the IS&C system also includes temporary delete mechanism for safety.



An example of zone arrangement

Advantages of sequential processing of variable size data

1. Assignment of write area by block unit handling
2. Reading by means of sequential sectors

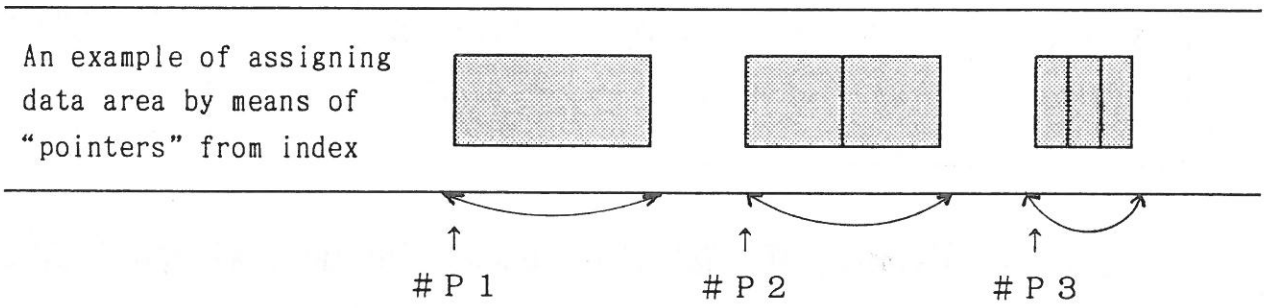


Fig. 2. 2. 2 File management of IS&C system

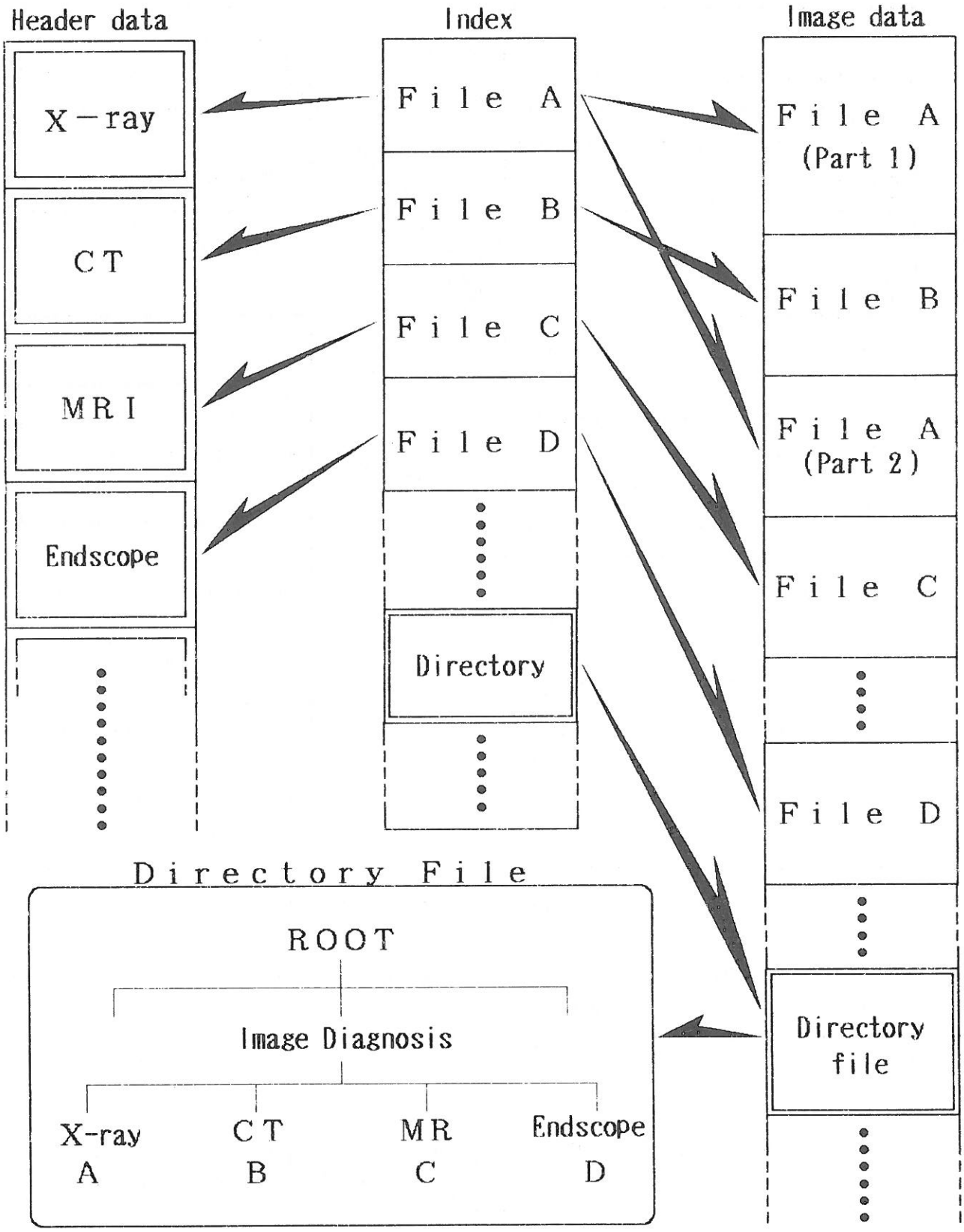


Fig.2.2.3 Example of IS&C file system for medical application

Temporary delete allows files to be logically but not actually deleted by presenting a flag in one of the file attri@ butes. The IS&C file manager does not show the temporarily deleted files to users. Such a temporarily deleted file can always be recovered.

2.4.2 States of deleted files

(1) Temporary delete

As previously noted, a temporarily deleted file is indicated by a flag. All data except for the flag remains the same.

(2) Actual delete

All data including the index table are actually deleted; the index table, header data area and data area are freed and made ready to be used by other files. The zone and sector tables are updated.

2.4.3 Structure

The IS&C system keeps temporary delete files by changing status as follows:

(1) A flag in the attribute

This flag is used to indicate the file status if it has been temporarily deleted. This flag is assigned to one bit of file attribute and is explained in greater detail in section 4.4.

(2) The number of temporarily deleted files

The number of temporarily deleted files in a volume is listed in the volume information table (refer to the section 4.1).

(3) Negative file ID in the header data area

The temporarily deleted file has a negative file ID added to the header data if attached (see Table 3.3.1).

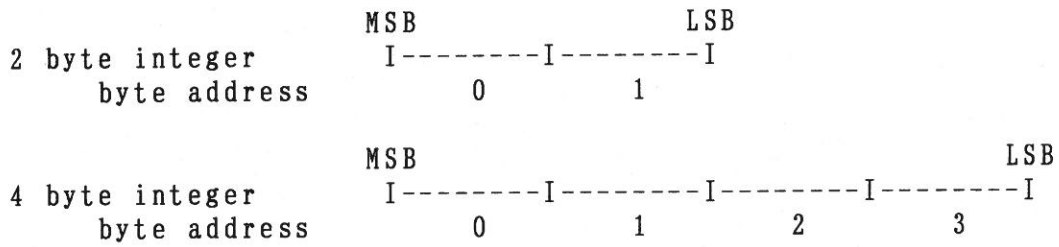
3. Volume and File Structure

3.1 Data expression

The IS&C system specifies the data expression used in the volume and file information.

(1) Byte order

Two-byte and 4-byte integer data are expressed in the order of MSB to LSB as the logical address increases.



(2) Character data

Character data is expressed in terms of ASCII codes and is lined up to the left. If the character stream is shorter than the area, the rest must be filled up by null(00H).

(3) Binary data

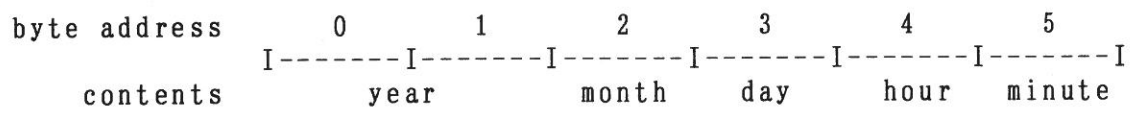
All binary data must have an integer with 1, 2 or 4 bytes. Integer data can be signed or unsigned, but they must be signed-integers unless otherwise stated.

(4) Sector number

The sector number is given by an unsigned integer with 4 bytes. In this draft the sector number is logical and relative, as explained in 2.2.(4).

(5) Date information

Dates are expressed by 6 bytes as follows:



3.2 Zone and block

3.2.1 Volume structure

(1) The volume is divided into zones.

(2) Zones are classified into 8 types depending on their use (see Fig.3.2.1). Among these, the 6 zones assigned letters from C to H are for data storage as shown below.

zone name	block size	purpose
A	1 LS	system area
B	1 LS	header data
C	1 LS	data
D	4 LS	data
E	16 LS	data
F	64 LS	data
G	256 LS	data
H	1024 LS	data

Note: LS stands for Logical Sectors.

(3) All data is stored in a disk on the basis of the block as a unit.

(4) Data is separately stored in the several blocks of different zones when necessary.

3.2.2 Management of the data area

(1) Data is stored as a file and its location in a disk is defined in the index table (refer to 4.4 Index Table).

(2) The status of the zone is defined in the zone table showing its type, the number of blocks available, and its back-up zone number if it has been defined (refer to 4.2 Zone Table).

(3) Zones are assigned to one of the 8 types upon request. If all data stored in a given zone is actually deleted, that zone becomes free and may be assigned to a different zone type.

3.2.3 Examples of block use (Fig.3.2.2)

(1) Write a D1 file with a size of 2.5MB

In this case, the IS&C file manager tries to find 2 blocks in the H zone (2MB), and 2 blocks in the G zone (512KB) as shown in Fig.3.2.2-1. Consequently, zones with a number of n and n+1 are assigned to H, and the n+2 zone is assigned to G. Because the G zone has 4 blocks, two of them remain free.

(2) Write a D2 file with a size of 1MB+320KB

D2 requires 1 block in the H zone (1MB) and 5 blocks in the F zone. Then, the file manager assigns the n+3 zone to H and the n+4 zone to F. After writing the n+4 zone, 11 free blocks remain (see Fig.3.2.2-2).

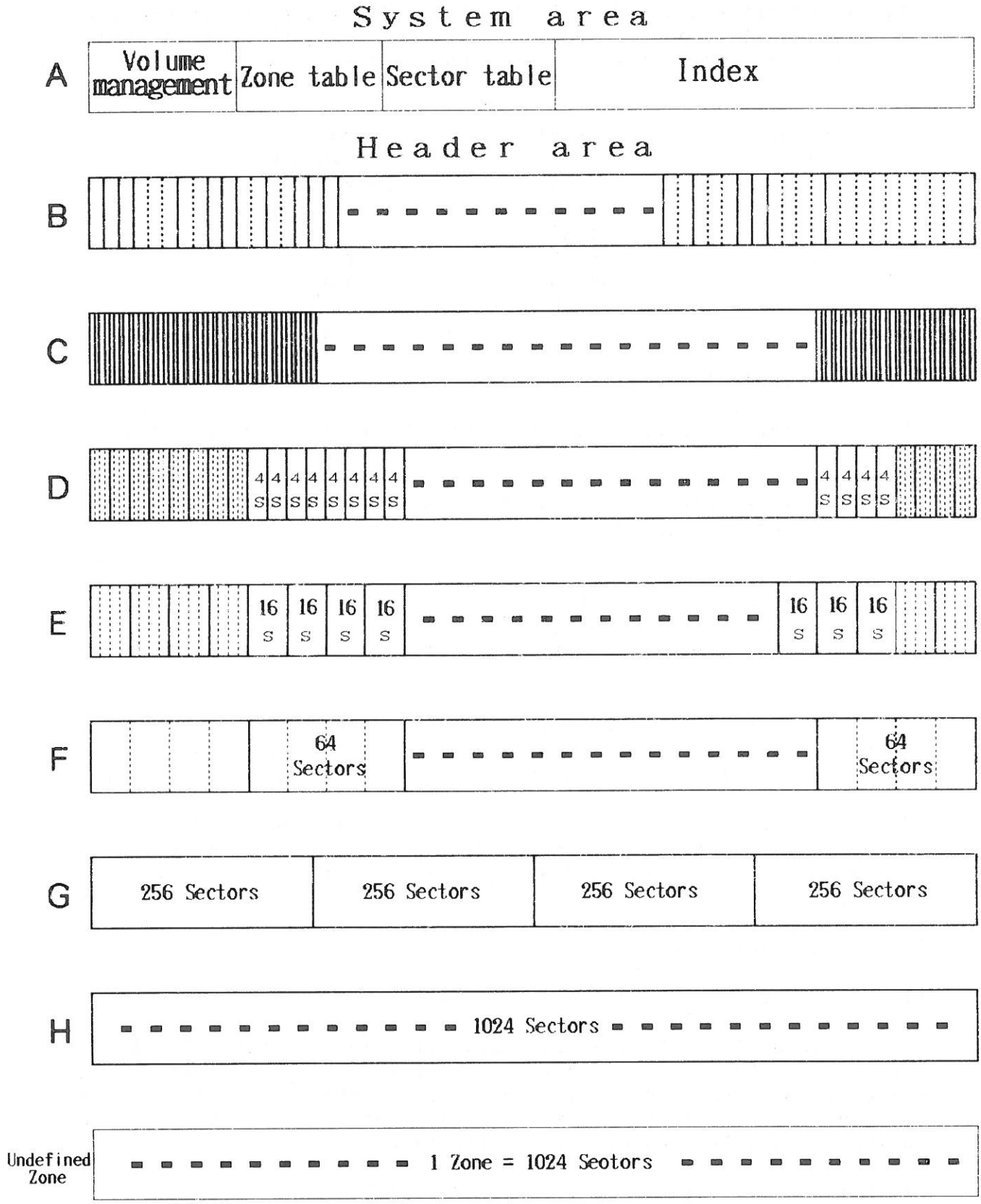


Fig. 3. 2. 1 Zone and block

Note: The D2 file can use 2 blocks in the n+2 zone, which still has 2 free blocks instead of the 5 blocks in the F zone. Because 5 blocks may be chosen to locate in a different zone number, users can choose a larger block for a faster I/O.

(3) Write a D3 file with a size of 512KB.

Two blocks of the n+2 zone, which was assigned to G and still has 2 free blocks, are used for the D3 file. Thereafter, the n+2 zone becomes full and does not have any free blocks (see Fig.3.2.2-3).

(4) Write a text D4 file with a size of 20KB

<Fast I/O case>

One block in the F zone is used. In this case 44KB are wasted. Simply in Fig.3.2.2-4 the n+5 zone is shown as being assigned to F, although in practice one of the free blocks in the n+4 zone are preferably used.

<Good space efficiency>

The n+5 zone is assigned to D, and 5 of the 256 blocks defined in the D zone are used. In this case, 5 blocks may be separated in a different zone. As a result the I/O speed is not as fast as as in the previous case, although no space is wasted.

3.3 Header data

3.3.1 Outline

Header data is defined to explain the contents or data iden@ tification of the file. Moreover, its location is also recorded in the index table.

3.3.2 Specification of the header data (refer to Fig.3.3.1)

(1) The header data area, the B zone, uses a 1KB block as the minimum size of the I/O unit.

(2) The maximum length of the header data is 32K-6 bytes, and the length is expressed by a signed 2-bytes integer.

(3) The location of the header data is indicated by the header pointer, which is stored in the index table in the system zone A. Header data larger than 1K-6 bytes is stored in a continuous sector, and is never separated.

(4) All header data is stored in the B zone.

① Writing of image data D1(2MB+512KB)

	D 1	D 1	D 1			
Zone number	n	n + 1	n + 2	n + 3	n + 4	n + 5
Zone name	H	H	G			

Fig.3.2.2-1

② Writing of image data D2(1MB+320KB)

	D 1	D 1	D 1	D 2	D 2	
Zone number	n	n + 1	n + 2	n + 3	n + 4	n + 5
Zone name	H	H	G	H	F	

Fig.3.2.2-2

③ Writing of image data D3(512KB)

	D 1	D 1	D 1 D 3	D 2	D 2	
Zone number	n	n + 1	n + 2	n + 3	n + 4	n + 5
Zone name	H	H	G	H	F	

Fig.3.2.2-3

④ Writing of image data D4(20KB)

						D 4
	D 1	D 1	D 1 D 3	D 2	D 2	
Zone number	n	n + 1	n + 2	n + 3	n + 4	n + 5
Zone name	H	H	G	H	F	ForD

Fig.3.2.2-4

Fig.3.2.2 Example of data area allocation

(5) Files may not always have header data. In this case the size of the header data is simply assigned as zero.

(6) In addition to the header data, the file ID and its length are also stored in the corresponding header area as shown in Fig.3.3.1. The addition of this data ensures the quick acquisi@tion of a file ID, of the file status (if it has been temporarily deleted), and of all the data.

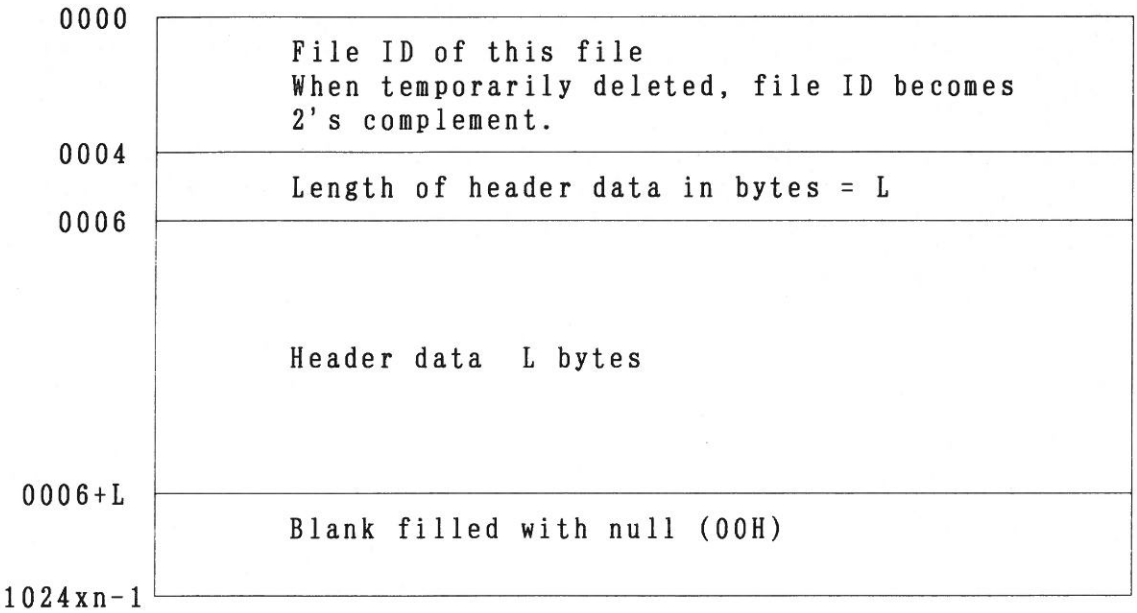


Fig. 3.3.1 Structure of header data area

Table 3.3.1 Structure of header data

byte address	item	byte length	data type
0	File ID	4	long integer >0 Active file <0 Temporarily deleted =0 free area
4	Length of header data	2	short integer
6	header data	L	see the IS&C data format
6+L - 1024xn-1 (n=1,2,..32)	blank	1024xn -(6+L)	null (00H)

3.3.3 Details of the header data

The details of the header data are explained in the IS&C standard draft on data formats.

3.3.4 Notes

(1) Because all header data is continuously stored in the B zone, the IS&C system provides for the quick search of a relevant file. It also becomes very easy to obtain information on the contents.

(2) The IS&C system can be used in the medical field, as well as other fields including research fields. Therefore, header data is not mandatory.

(3) Files without header data can be identified by either the file ID or the file name. However, the file name will not necessarily be unique as the file ID is.

4. System data

4.1 Volume information

Volume data is stored in the #0 and #1 sectors. The data in the #0 sector is completed at the time of initialization, and is never updated. The data in the #1 sector is updated when the volume contents are changed.

4.1.1 Contents of #0 sector

This sector contains the information listed in Table 4.1.1.

Table 4.1.1. Volume information (#0 sector)

address	item	length(B)	data type
0-3	format name	4	character
4-7	version #	4	character
8-23	application field	16	character
24-55	volume name	32	character
56-59	volume ID	4	binary
60-91	owner's name	32	character
92-123	owner's code	32	character
124-129	initialization date	6	binary
130-133	# of zones	4	binary
134-135	# of sectors/zone	2	binary
136-137	# of bytes/sector	2	binary
138-141	zone table address	4	binary
142-145	sect. table address	4	binary
146-149	first index table address	4	binary
150-151	index table size	2	binary
152-1024	reserved area	872	

(1) System identification

The IS&C volume is identified by this entity written as "IS&C".(49 53 26 43H)

(2) Version number

The IS&C format subject to this standard draft is denoted by "01.0".

(3) Application field

For a medical application, "MEDICAL" should be written here.

(4) Volume name

The volume name is written by character data.

(5) Volume ID

The volume ID is given by a 4-byte integer.

(6) Owner's name

The owner's name is written by character data.

(7) Owner's ID

The owner's ID is written by character data.

(8) Initialization date

The initialization date of the volume is written in 6 bytes, according to the format explained in 3.1.(5).

(9) The number of zones

The number of zones defined in the volume is written by a 4-byte integer.

(10) The number of sectors in one zone

The number of sectors in one zone is expressed by a 2-byte integer. The IS&C vs. 1.0 system uses 1024.

(11) The number of bytes in a sector

The number of bytes in one logical sector is expressed by a 2-byte integer. The IS&C vs. 1.0 system uses 1024 bytes for one logical sector.

(12) Address of the zone table

The sector address of the zone table is expressed by a 4-byte integer.

(13) Address of the sector table

The address of the sector table is expressed by a 4-byte integer.

(14) Address of the index table

The address of the first index table is expressed by a 4-byte integer. Other indices are stored sequentially.

(15) The size of one index table

The size of one index table is expressed by a 2-byte integer.

(16) Reserved area

The rest of the #0 sector is reserved for future use, and is filled with null (00H).

4.1.2 Contents of the #1 sector

This sector contains the information listed in Table 4.1.2.

Table 4.1.2. Volume information (#1 sector)

address	item	length(B)	data type
0-3	# of index tables	4	binary
4-7	# of active files	4	binary
8-11	# of temporarily deleted files	4	binary
12-15	# of free index tables	4	binary
16-17	# of system files	2	binary
18-19	# of directory files	2	binary
20-25	latest update	6	binary
26-29	next free index number	4	binary
30-31	volume flag	2	binary
32-1007	reserved area	976	
1008-1023	system reserved area	16	

(1) Total number of indices

The total number of indices defined in a volume is expressed by a 4-byte integer. If the number of the A zone changes, this value also changes.

(2) The number of active files

The number of active files is expressed by a 4-byte integer. This number excludes the number of temporarily deleted files.

(3) The number of temporarily deleted files

The number of temporarily deleted files is expressed by a 4-byte integer.

(4) The number of free indices

The number of free indices is expressed by a 2-byte integer.

(5) The number of system files

The number of system files registered in a given volume is expressed by a 2-byte integer.

(6) The number of directory files

The number of directory files registered in a given volume is expressed by a 2-byte integer.

(7) Updated date

The updated date of the volume information is written by 6 bytes according to the format shown in 4.1.1.(7).

(8) The number of the first free index

Free indices are chained to allow the next available free index to be found easily.

(9) Volume flag

This flag is used to show whether the volume is mounted or dismounted. When mounted, it is assigned to 1. When dismounted, it is assigned to 0.

(10) Reserved area

An area reserved for future use is filled with null (00H).

(11) System reserved area

This area is used for security purposes, and access to it should not be permitted.

4.2 Zone table

(1) The zone table is defined to indicate the types of zones, the number of free blocks within each zone, the back-up zone number, and the number ordered within each zone type. Zones are numbered from 1.

(2) The contents of the zone table are shown in Table 4.2.1.

Table 4.2.1 Contents of the zone table

byte address	item	data type	description
0	zone type	2-byte integer	1-8 apply to zones A-H, respectively. A negative value indicates a back-up zone of the same kind. A free zone is indicated by 0.
2	# of free blocks	2-byte integer	For zone A : next A zone's number. -1 indicates the last. For zone B : the number of free sectors For zone C-H : the number of free blocks For a free zone : 0
4	Back-up zone # or	2-byte integer	For zone A : Back-up zone number. For B to H and free zone : 0

4.3 Sector table

(1) This table indicates each logical sector being used or not by 1 bit. Therefore, the number of bits defined in this table is equivalent to that of the logical sectors in a given volume.

(2) All the sectors in zones A-H are involved.

Note: All tables are followed by null (00H) in order to fill in the last sector of each table.

4.4 Index table

Table 4.4.1 Primary index table

address	length(B)	item	data type	description
0	4	file ID	integer	
4	24	file name	character	
28	4	creation date	integer	year, month, day
32	2	creation time	integer	hour, minute
34	4	update date	integer	year, month, day
38	2	update time	integer	hour, minute
40	4	file size	integer	byte length of date
44	2	attribute	integer	flag bits
46	12	reserved		filled with null (00H)
58	6	header pointer	integer	first sector (4B) + length (2B)
64	6	#1 data pointer	integer	same as above
70	6	#2 data pointer	integer	same as above
:	:	:	:	:
:	:	:	:	:
118	6	#10 data pointer	integer	same as above
124	4	next index number	integer	

Table 4.4.2 Extended index table

address	length(B)	item	data type	note
0	4	index number	integer	2's complement of own index
4	4	father index #	character	
8	14	reserved		filled with null (00H)
22	6	#1 data pointer	integer	first sector (4B) + length (2B)
28	6	#2 data pointer	integer	same as above
70	6	#2 data pointer	integer	same as above
:	:	:	:	:
:	:	:	:	:
118	6	#17 data pointer	integer	same as above
124	4	next index number	integer	

(1) File ID

The index table is numbered from 1, and the file ID is given by the index table number for primary use. The file ID is classi@ fied as follows:

ID=0 : This index is not used.
ID>0 : This index is being used as a primary index.
ID<0 : This index is being used as an extended index.
Two's complement of the ID shows its index number.

(2) Date of the latest change

This entity is updated by any change in the file data, header data or attributes.

(3) Header pointer

Header data is transferred to and from a disk by a unit of 1024 bytes. If the header data is larger than (1K-6) bytes, continuous sectors are utilized.

(4) Merge of pointers

When the areas indicated by the consecutive pointers form continuous sectors, those pointers can be merged even if the areas are in different zones. Because the length of the continuous sectors is given by a 2-byte integer, its maximum length is about 32 MB.

(5) Attribute

Attributes are expressed by a bit stream. Each bit is indicated as follows :

byte address	flag bit MSB<- ->LSB	meaning
44	x	variable file size
	.... ..x.	write protection
	.... .x..	read protection
	.... x...	system file
	...x	directory file
	..x.	reserved (set 0)
	.x..	set 1 during the write operation*
	x...	temporarily deleted
45	 xxxx	(set 0)
	xxxx	reserved (set 0)

Note : when a bit is on or set to 1, the corresponding meaning becomes true.

* This bit is set to 1 when the file is opened in the write

mode, and it is set to 0 when closed.

(6) Index table chain

When the number of pointers exceeds 10, the maximum number allowed in the primary index, the index table can be extended by defining extended table as shown in Table 4.4.2.

1) Next index table number

Case 1 : Next index table number is given at the end of the index table. When an index table is extended, this entity shows the extended index table number. If it does not, -1 is written in this box as a terminator.

Case 2 : When an index table is not being used, its file ID and next index table number are respectively given by 0 and the next free index table number. For the last one, -1 is used as a terminator.

2) Actual delete (see Fig.4.4)

When a file is actually deleted, the index of the file should be chained as follows:

Step 1 : Check the total number of index tables that the file uses until the terminator is found. Then, keep the file ID and clear all index IDs by 0.

Step 2 : Write the file ID into the box for the free index table number into the volume information of the #1 sector.

Step 3 : Write the next free index number, which has been written into the volume information of #1 sector, into the extended index of the last relevant index table.

Step 4 : Add the total number of index tables obtained in step 1 to the number of free index tables listed in the volume information of the #1 sector.

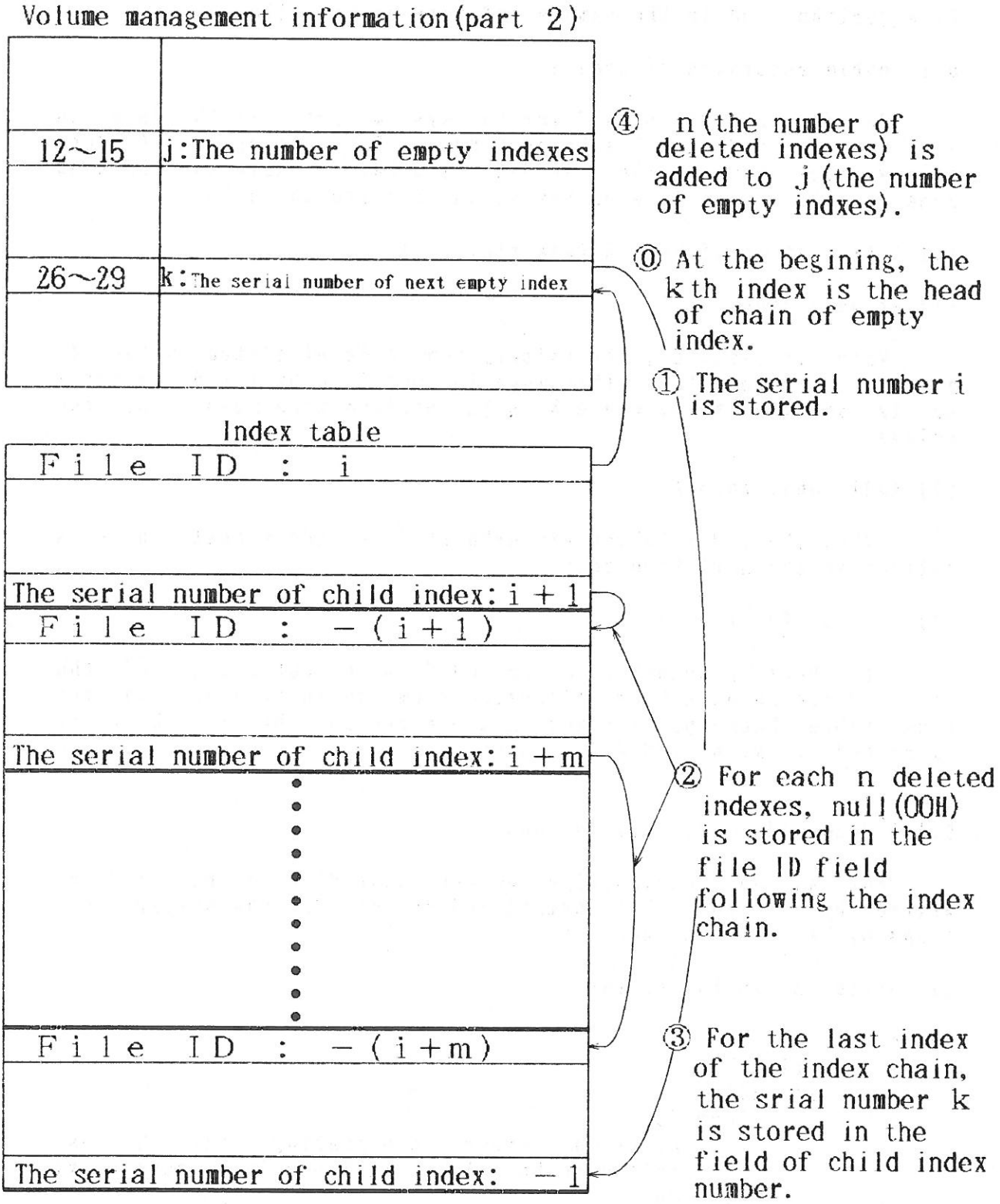


Fig.4.4 Handling process of chain of empty indexes in the case of deleting file i

5. Algorithm used in the sample software of the file manager

5.1 Double recording of zone A

Because zone A is used for the system area, all the data in zone A is copied into its back-up zone. The primary zone A data is rewritten whenever the disk data is updated, while the back-up zone is rewritten only when the volume is dismounted.

5.1.1 Allocation of zone A (see Fig.5.1.1)

(1) Primary zone

When initialized, the primary zone A is allocated in the #1 zone. In the zone table this zone is specified by 1 and its back-up is assigned to N, where N is the maximum zone number of the volume.

(2) Additional zone A

When the index tables are exhausted, an additional zone A is defined in the next free zone.

(3) Back-up for zone A

All back-up zones are allocated from the outer side of the disk, while zones A-H are allocated from the inner side. In the zone table back-ups for zone A are expressed by -1, and are allocated in N, N-1, N-2.... zones.

5.1.2 Read and write data in zone A

For an easy explanation, we represent data to be read or written by 'a', and 'a' is symbolized by 'a1' for the primary zone A and by 'a2' for its back-up.

(1) Write 'a' in the primary

```
Begin :
          write 'a1'
          if fail
               then do
                    gives an error code meaning that 'a' has
                    failed to be written in the primary zone A. *1
               else do
                    normal return
          end if
end
```

Note : The MOD drive automatically provides an alternative sector. Therefore, if this error occurs, the error will be so serious that the disk will hardly be used anymore.

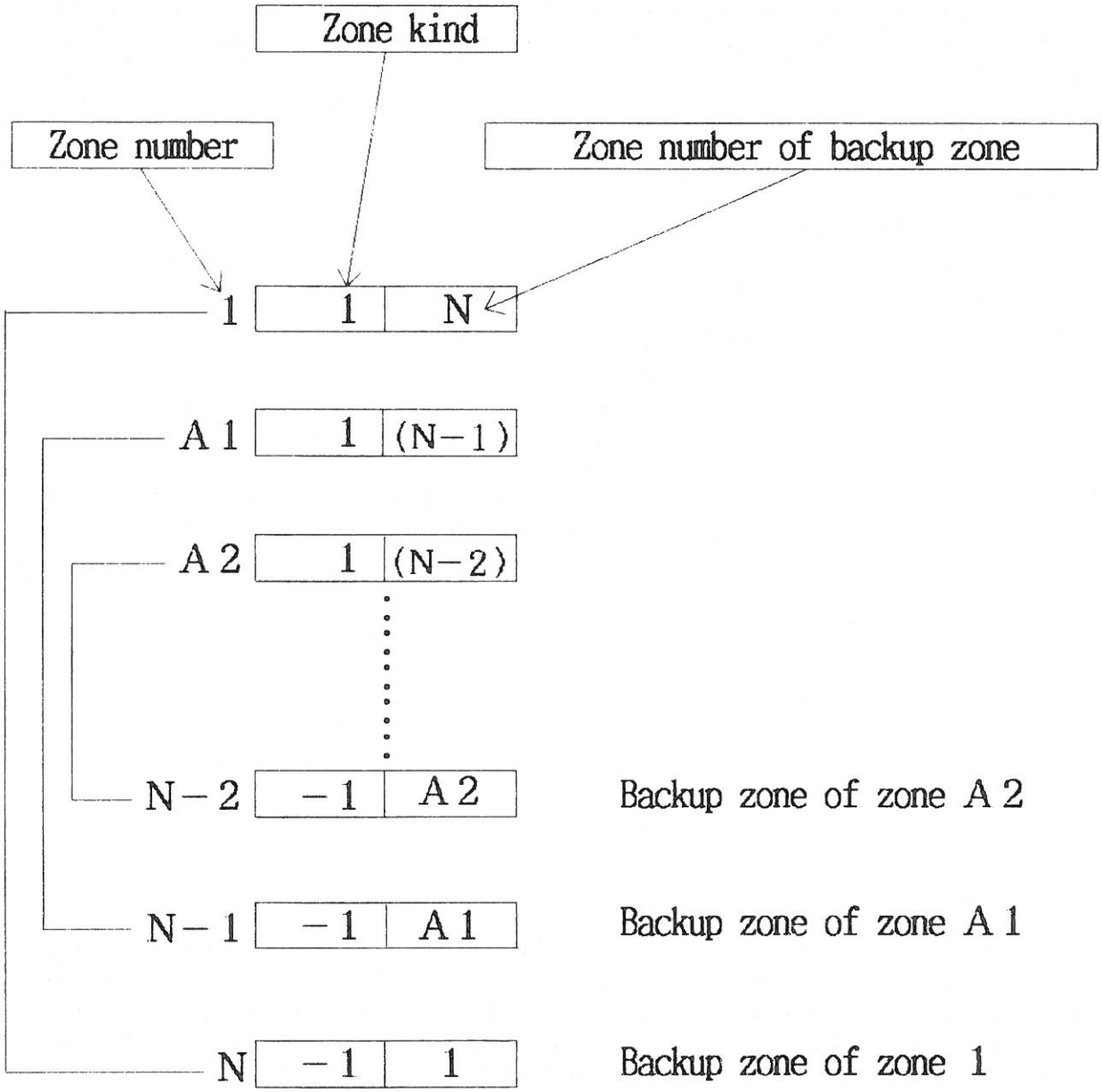


Fig. 5. 1. 1 Allocation of A zone

(2) Write a back-up zone

When the volume is dismounted, all primary zones are copied onto their back-up zones after the volume mount flag has been cleared.

(3) Read 'a'

```
Begin :
          read 'a1'
          if fail
               then do
                    read 'a2'
                    if fail
                         then do
                              gives an error code meaning that 'a'
                              has failed to be read.
                         else do
                              'a2' is written onto 'a1' and give a
                              warning that 'a1' has failed.
                    end if
          end if
          normal return
end
```

5.2 File creation and data allocation

(1) File creation

For a variable-length file, the choice of zone or block size is specified by the user when it is created, and the number of continuous sectors is fixed because this file uses only one type of zone for data recording. When created, this file defines at least one block that is being used and has one pointer.

For a fixed-size file, choices for zone are specified either by the user or by the file manager in a default mode.

(2) Header data allocation

Header data is continuously allocated in zone B when stored, and may be used in a different area when updated.

(3) Data allocation of the variable-length file

Data areas are allocated on the basis of the block size specified when it is created.

(4) Data allocation in the fixed-size file

The zone is specified by a flag as shown below:

```
flag bit      MSB <-          7 6 5 4 3 2 1 0   ->LSB
                      .........x x x x x x 0 0
zone                          H G F E D C B A
```

Flag bits correspond to zones A-H, and specify the zones by setting corresponding bits to 1.

Note : The number 0 and 1 bits are not used. If all the bits are off or 0, zones are selected in the way which ensures the best space efficiency.

(5) Examples of data allocation in the fixed-size file

step 1 : check the volume descriptor if the volume is mounted.

```
if error
     then do
          give the error code
          error return
end if
```

step 2 : check the flag bit if the zones have been specified.

```
if the zone is being specified
     then do
          case 1    ---> refer to (5-1)
     else do
          case 2    ---> refer to (5-2)
end if
```

step 3 : obtain the area and define pointers. ---> refer to (5-3)

```
if the total number of pointers is within 10
     then do
          define pointers in the primary index table
     else do
          define pointers in the primary, and extend
          index tables until all the pointers are
          stored.
end if
```

step 4 : update relevant index tables.

step 5 : update the zone and sector tables.

step 6 : update the volume information in #1 sector.

step 7 : write all the updated data in the volume.

(5-1) Case 1

Determine the number of blocks in the zones specified by the flag in order to provide enough space to store the file data.

<Example>

If the file to be stored and specified blocks, respectively, have N, n1, n2 and n3 bytes, where n1>n2>n3, the file manager gets m1, m2 and m3, the necessary number of blocks for each zone, as follows :

1) m1=N/n1+s1, where m1 and s1 are the quotient and the surplus, respectively.

2) m2=s1/n2+s2, where m2 and s2 are the quotient and the surplus, respectively.

3) m3=s2/n3+1 unless s2=0. When s2=0, m3=0.

Note : when the number of zones specified is more than 3, simply repeat the calculation as explained above.

(5-2) Case 2

In the default mode, the file manager automatically assigns zones C-H to the file.

<Example>

Assuming that the file has N bytes, the file manager will calculate the necessary sectors in a 1-Kbyte unit by M=(N-1)/1024+1, and will use blocks for each zone as follows:

```
          MSB <-- .. 11 10  9  8  7  6  5  4  3  2  1  0 --> LSB
bit stream :          0  1  1  0  1  1  1  0  1  1  0  1
zone       :   <------ H -> <-G-> <-F-> <-E-> <-D-> <-C->
```

Note : the bit stream represents M Kbytes.

(5-3) Area and pointer's definition

1) In order to obtain the addresses of the blocks to be used, the zone table should be searched first to find the necessary number of free blocks.

2) If the necessary number of blocks are not found, free zones will be defined in a way that will provide the required blocks.

3) If the areas being used are located next to each other, all such pointers can be merged into one.

5.3 Write data in a file

5.3.1 Summary

If a user wants to write a file on the IS&C disk, a file should be created and opened by the file manager. It should then be closed before dismounting.

5.3.2 Algorithm

(1) set the write flag in the attribute.

(2) check the file descriptor, if it has been assigned to the file.

(3) load the primary index table onto the work area in the file manager.

(4) check the write protection flag in the attribute if it is not protected.

(5) obtain the pointers

```
if the pointer is not defined
     then do
          if the file is a variable.
               then do
                    define additional pointers --> see note
               else
                    give the error
          end if
end if
```

Note : definition of additional pointers for a variable-size file

1) Determine the size of the block by checking in which zone the first pointer is defined.

2) If the data requires an area for storage beyond the file size currently defined, a sufficient number of areas are actually allocated for the data.

(6) write data into the area specified by the pointers.

(7) write header data into zone B.

(8) set off the write flag in the primary index table.

5.4 Read the data in a file

5.4.1 Read data

step 1 : read the volume information, then load both the zone and sector tables in the buffer.

step 2 : obtain the index tables, both the primary and extended, specified by file ID.

step 3 : read data from the area indicated by the pointers. The IS&C file manager provides random access to the file data.

5.4.2 read header data

Steps 1 and 2 should be carried out as previously explained in 5.4.1.

step 3 : read header data stored in zone B from the area indicated by the header pointers.

5.4.3 Read all the header data stored in a given volume

step 1 : as previously explained in 5.4.1.

step 2 : read header data from zone B sequentially. The loca@tion of zone B is written in the zone table.

5.4.4 File search

The IS&C file manager provides three methods to search a relevant file : by file ID, by file name or by header data

```
if file ID is known
     then do
          open the file identified by the ID
     else do
          if the file name is known
               then do
                    obtain the file ID by name and specify
                    the relevant file. --> see (1)
               else do
                    read all the header data sequentially,
                    and specify the relevant file.
                                        --> see (2)
          end if
end if
```

(1) Because the file name will not necessarily be unique within a given volume, there could be several files which have the same file name. In such a case, the header data should be examined to specify the relevant file.

(2) Header data is stored in zone B so that it can be read con@ tinuously. Header data is added by file ID, and users can obtain the file ID easily.

5.5 Actual delete of temporarily deleted files

5.5.1 A file to be actually deleted

Only temporarily deleted files can be actually deleted. Active files cannot be actually deleted in one step.

5.5.2 Temporarily deleted file

Temporarily deleted files differ from active files on the following points:

1) the temporarily deleted flag has been set on, which is one of its attributes.

2) in the header file the file ID has become negative.

All other information such as zone table, sector table etc., is not in any way different.

5.5.3 Actual delete

The actual delete procedure is as follows:

1) all the index tables including extended tables, if used, should be freed, and the file ID should be cleared by 0.

2) header data should be cleared or filled with null (00H). The area used by the header data should be free.

3) data area should be freed so that it can be used by other file.

4) when a certain zone becomes totally free, that zone should be freed for its next use.

5) both the zone and sector table should be updated.

6) update the volume information in the #1 sector. The items to be changed are the number of temporarily deleted files, the number of free index tables, and the number of next free index table.

5.5.4 Algorithm

(1) Release header area

1) release sectors

Bits in the sector table indicating the location of the header data should be cleared and set to 0.

2) clear the file ID in the header data

The file ID stored in the header area should be cleared by filling it with null (00H).

(2) Release data area and zone

1) Primary index table

data_length = file size stored in the primary index table

p=1 (#1 pointer in the primary index table)

while (data_length>0 and p<=10)

step 1 : clear the bits in the sector table indicated by the p-th pointer.

step 2 : add the number of released blocks to the number of free blocks.

step 3 : if the zone is completely unused, it should be assigned to an unused zone.

Note : if a pointer indicates a data area beyond the zone border, blocks in the next zones should be released.

data_length = data_length - # of sectors x bytes in the sector

p = p + 1

end while

2) Extended index tables

while (data_length > 0 and extended index table defined)

p=1

while (data_length>0 and p<=17)

step 1 : clear the bits in the sector table indicated by the p-th pointer.

step 2 : add the number of released blocks to the number of free blocks.

step 3 : if the zone is completely unused, it should be assigned to an unused zone.

Note : if a pointer indicates a data area beyond the zone border, blocks in the next zones should be released.

```
    data_length = data_length - # of sectors x bytes in the sector

        p = p + 1

          end while

      end while
```

(3) Release index table

Set 0 in the file ID of the primary index table, and make up the free index table chain (see 4.4.(6)).

```
      while ( extended index table number < 0 )

         step 1 : the index table number should be set to 0.

         step 2 : make up the free index table chain (see 4.4.(6))

      end while
```

(4) Update the volume information of #1 sector

The following entities in the volume information of #1 sector should be modified as

* The number of temporarily deleted files should be decreased by 1.

* The number of free index tables should be added by the number of index tables used by the file that has actually been deleted.

* The number of next free index table is indicated by the index table number of the actually deleted file.

6. Directory

In the IS&C system, relationships between files in a given volume can be defined as a file, which is used to construct a tree structure or classify files.

6.1 Definition of a directory in the IS&C system

Because the directory structure may vary depending on the users' requirements, it is defined as a file in the IS&C system. Therefore, only the description of the structure is clearly stan@ dardized to enable common use among users. Consequently, there may be several directory files in a given volume.

6.2 Structure of the directory file

6.2.1 File elements of the directory file

The directory file consists of fixed-size records. One record has 32 bytes.

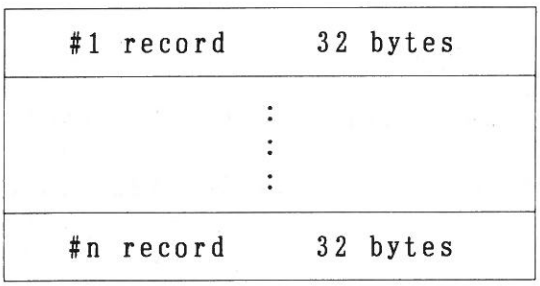


Fig. 6.2.1 File elements of the directory file

6.2.2 Components of a record

Each record has three components as shown in Fig.6.2.2.

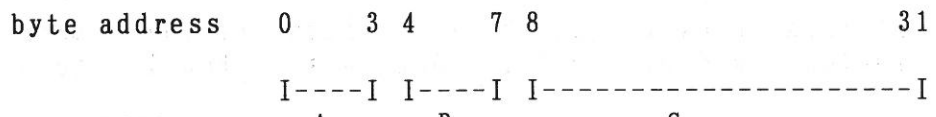


Fig. 6.2.2 Components of a record

<A> ID number (4 bytes)

ID<0 directory ID number

each directory is numbered according to its description.

ID>0 File ID number

This number is equivalent to the one used in the index table.

<B> Upper directory's ID number (4 bytes)

All values are negative.

<C> Directory and file name area

ID of A < 0 : directory name
ID of A > 0 : file name

6.2.3 Other records

In addition to the records discussed in 6.2.2, the directory file contains terminal and free records.

1) terminal record

ID number of A = 0
Upper directory's ID of B = 0

2) free record

ID number of A = 0
Upper directory's ID of B =/ 0

6.3 Sample directory file

A directory file in the IS&C system has the structure illust@rated in Table 6.3.1, which indicates the relations shown in Fig.6.3.1.

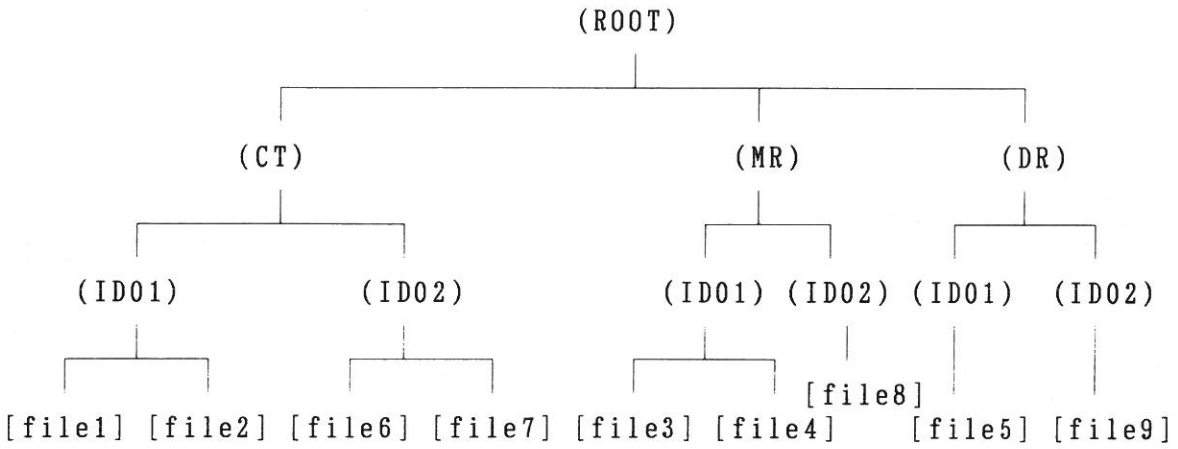


Fig.6.3.1 Sample directory structure

Table 6.3.1 Example of a directory file

ID number	Upper directory ID number	name
-1	-1	ROOT
-2	-1	CT
-3	-2	ID01
1	-3	file1
2	-3	file2
-4	-1	MR
-5	-4	ID01
3	-5	file3
4	-5	file4
-6	-1	DR
-7	-6	ID01
5	-7	file5
-8	-2	ID02
6	-8	file6
7	-8	file7
-9	-4	ID02
8	-9	file8
-10	-6	ID02
9	-10	file9
0	0	Note: terminator

7. Conclusion

This draft assumes the use of 130 mm magneto-optical disks; moreover, all formats could be directly applied to the next generation disk with a higher capacity, 90 mm small disk, or even the magnetic disk, since the format is totally defined as logi@ cally independent of the physical media specifications.

The IS&C format is also applicable to general fields other than the medical field and is very useful for users, who want to exchange large-size data. Some specifications (such as file protection), which are indispensable for the Japanese medical system, are defined and explained in another draft.

Users may create a file management software to support the IS&C format, and sample software is provided at some charge by the IS&C committee. This software was created to clarify specifica@ tions and to avoid any confusion among users. This sample soft@ ware is written in C language and consists of many modules. Users should consider memory size, driver software, etc. when modifying it to make it more suitable for their systems.

This format is version 01.0, and it may be updated whenever necessary. To make the IS&C format more widely used, and to construct new application fields, the IS&C committee invites all comments or suggestions.

Apendix A. Service functions of the file manager

Sample software of the IS&C file manager is written in C language, and error codes are returned as long-type integers.

1. Disk Management

1.1 Disk mount

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

This function enables users to access the volume. System data such as volume information and tables are loaded into the buffer area in the file manager.

1.2 Disk dismount

```
long ifm_dismount_media( vd )
     long      vd,       /* Volume discriptor */
```

This function terminates access to the volume. All files are closed and system data is stored in the volume if necessary.

1.3 Obtain the disk capacity

```
long ifm_get_space( vd, msize, usize )
     long      vd,       /* Volume discriptor */
          *msize,       /* Media size (KBytes) */
          *usize,       /* Area size under use (KB) */
```

This function gives the sizes of the volume and of the used area. Temporarily deleted files are regareded as being under use.

1.4 Initialization of an IS&C disk

```
long ifm_format_medaia( un, voldata )
     long      un,       /* Unit number */
     struct volumedatatype *voldata, /* # 0&1 sector data */
```

This function initializes a disk subject to the IS&C format.

2. File Management

2.1 Obtain the lists of files

```
long ifm_get_list( vd, id, array, req_size, act_size )
     long      vd,        /* Volume discriptor */
               id,        /* Youngest file ID under request */
          req_size,       /* The number of files under request */
         *act_size,
               /* The number of files given through the service */
     char  *array,
          /* Primary index tables without pointers are transferred
          to  this buffer.  The array size should be equal  to  or
          larger than req_size*64 bytes */
```

This function gives index tables in an array, whose file ID is equal to or larger than the number of id. Each index table has 64 bytes and the pointers are suppressed. Temporarily deleted files are not shown.

2.2 Make a fixed size file

```
long ifm_create_file( vd, name, size, flag, id )
     long      vd,        /* Volume discriptor */
              size,       /* File size in bytes */
              flag,
                    /* Zone specification (-1 shows default) */
               *id,       /* File ID */
     char    *name,       /* File name */
```

The file name should have 24 bytes. If it has less than 24 bytes, the array should be terminated by null (00H). This func@ tion makes primary and extended index tables if necessary, and fully secures the area.

2.3 Create a variable size file

```
long ifm_create_file_variable( vd, name, cell, id )
     long      vd,        /* Volume discriptor */
              cell,       /* Block size */
               *id,       /* File ID */
     char    *name,       /* File name */
```

The file name should have 24 bytes. If it has less than 24 bytes, the array should be terminated by null (00H). This func@ tion creates a primary index table and assigns one block to the file.

2.4 Obtain file IDs that match the specified file names

```
long ifm_get_file_id( vd, name, id, req_size, act_size )
     long       vd,      /* Volume discriptor */
                id,      /* An array of file IDs */
         req_size,      /* The number of file IDs under request */
        *act_size,
                        /* The  number of file IDs  actually  given
                           through the service */
     char  *name,       /* File name */
```

Users can use * and ? wildcards. The asterisk stands for 1 character and the question mark represents a character string.

2.5 Delete a file temporarily

```
long ifm_delete_file( vd, id )
     long       vd,      /* Volume discriptor */
                id,      /* File ID */
```

3. File data I/O

3.1 Open file

```
long  ifm_open_file( vd, id, mode, fd )
     long       vd,      /* Volume discriptor */
                id,      /* File ID */
              mode,      /* Open mode */
                           mode= 1   ; Read only
                           mode= 2   ; Write only
                           mode= 3   ; Read and Write
               *fd,      /* File descriptor */
```

3.2 Close file

```
long ifm_close_file( fd )
     long       fd,      /* File descriptor */
```

3.3 Read data

```
long ifm_read_data( fd, buf, st, req_len, act_len, swp )
     long       fd,    /* File discriptor */
                st,    /* The first byte address to be transferred.
                          st=1024*n, where n is an integer. */
          req_len,    /* The number of bytes under request.
                          req_len=1024*n, where n is an integer. */
         *act_len, /* The number of bytes actually transferred */
              swp,    /* Flag indicating byte swap */
                swp=0  ; original data to be transferred
                swp=/0 : 2-byte integer swapped, then transferred
     char     *buf,    /* Data array */
```

Note: when st=/1024*n, the file manager gives an error code.
If req_len=/1024*n, it is reassiged to (req_len/1024)*1024.

```
3.4 Write data
long ifm_write_data( fd, buf, st, req_len, act_len, swp )
     long      fd,    /* File discriptor */
               st,    /* The first byte address to be transferred.
                         st=1024*n, where n is an integer. */
          req_len,    /* The number of bytes being requested.
                         req_len=1024*n, where n is an integer. */
         *act_len, /* The number of bytes actually transferred */
              swp,    /* Flag indicating byte swap */
                swp=0 ; original data to be tansferred
                swp=/0 : 2-byte integer swapped, then transferred
     char     *buf,    /* Data array */

Note: when st=/1024*n, the file manager gives an error code.
      If req_len=/1024*n, it is reassiged to (req_len/1024)*1024.

3.5 Read header data

long ifm_write_header( fd, buf, req_len, act_len )
     long      fd,    /* File discriptor */
          req_len,    /* The number of bytes being requested */
         *act_len, /* The number of bytes actually transferred */
     char     *buf,    /* Data array */

Note: The maximun length of the header data is (32*1024-6) bytes.


3.6 Write header data

long ifm_write_header( fd, buf, req_len, act_len )
     long      fd,    /* File discriptor */
          req_len,    /* The number of bytes being requested */
         *act_len, /* The number of bytes actually transferred */
     char     *buf,    /* Data array */

3.7 Obtain file size

long ifm_get_file_size( fd, fsize, hsize )
     long      fd,    /* File discriptor */
           *fsize,    /* File data size */
           *hsize,    /* Header data size */

3.8 Obtain file name

long ifm_get_filename( fd, name )
     long      fd,    /* File discriptor */
     char    *name,    /* File name */

3.9 Enter the file name

long ifm_put_filename( fd, name )
     long      fd,    /* File discriptor */
            *name,    /* New file name to be stored */
```

4. Header Zone I/O

4.1 Read header data sequentially

```
long ifm_read_header_sequential( vd, id, buf, req_size, act_size )
     long      vd,        /* Volume discriptor */
              *id,        /* File ID */
         req_size,        /* The size of buffer : max. 32k */
        *act_size,
             /* The number of header data actually transferred */
     char  *buf,          /* Data buffer */
```

4.2 Reset the header counter

```
long ifm_reset_header_counter( vd )
     long      vd,        /* Volume discriptor */
```

5. System Management

5.1 Obatin all index tables including temporarily deleted files

```
long ifm_get_list_all( vd, id, array, req_size, act_size )
     long      vd,        /* Volume discriptor */
               id,        /* Smallest file ID being requested */
         req_size,        /* The number of files being requested */
        *act_size,  /* The number of files actually transferred */
     char  *array,
                /* Data array, whose size is req_size*64 bytes */
```

This function gives index tables in an array, whose file ID is equal to or larger than the number of id. Each index table has 64 bytes and pointers are suppressed. Temporarily deleted files are also shown.

5.2 delete file actually

```
long ifm_delete_actually( vd, id )
     long      vd,        /* Volume discriptor */
               id,        /* File ID */
```

5.3 Recover a temporarily deleted file

```
long ifm_recover_temporarily_deleted( vd, id )
     long      vd,        /* Volume discriptor */
               id,        /* File ID */
```

5.4 Set the write protection flag on

```
long  ifm_write_protect_on ( vd, id )
long       vd,           /* Volume discriptor */
           id,           /* File ID */
```

5.5 Set the write protection flag off

```
long  ifm_write_protect_off ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.6 Set the read protection flag on

```
long ifm_read_protect_on ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.7 Set the read protection flag off

```
long ifm_read_protect_off ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.8 Set the system file flag on

```
long ifm_system_flag_on ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.9 Set the system file flag off

```
long ifm_system_flag_off ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.10 Set the directory file flag on

```
long ifm_directory_flag_on ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

5.11 Set the directory file flag off

```
long ifm_directory_flag_off ( vd, id )
     long       vd,         /* Volume discriptor */
                id,         /* File ID */
```

6. Utility functions used by the file manager

6.1 Write data to MOD

```
long iut_write_mod( un, dt, st_sct, len, swp, stt )
     long      un,        /* Unit number */
           st_sct,        /* The first sector number */
              len,        /* Length of data in bytes */
              swp,        /* Flag-indicating byte swap */
                swp=0 ; original data to be transferred
                swp=/0 : 2-byte integer swapped, then transferred
             *stt,        /* Status given by the MOD drive */
     char     *dt,        /* Data array */
```

6.2 Read data from the MOD

```
long iut_read_mod( un, dt, st_sct, len, swp, stt )
     long      un,        /* Unit number */
           st_sct,        /* The first sector number */
              len,        /* Length of data in byte */
              swp,        /* Flag-indicating byte swap */
                swp=0 ; original data to be transferred
                swp=/0 : 2 byte integer swapped, then transferred
             *stt,        /* Status given by MOD drive */
     char     *dt,        /* Data array */
```

6.3 Obtain an index table from the MOD

```
long iut_get_index( un, idx_no, idx_dt, len, flag )
     long      un,        /* Unit number */
           idx_no,        /* Index table number */
              len,        /* Length of the index table
                                      to be read <=128 */
            *flag,        /* Flag to show the sort of index */
     union idxtype *idx_dt,   /* Index table buffer */
```

6.4 Write an index table onto the MOD

```
long iut_put_index( un, idx_no, idx_dt )
     long      un,        /* Unit number */
           idx_no,        /* Index table number */
     union idxtype *idx_dt,   /* Index table buffer */
```

6.5 Prohibit manual release of the MOD

```
long iut_lock_mod( un )
     long      un,        /* Unit number */
```

6.6 Allow manual release of the MOD

```
long iut_unlock_mod( un )
     long      un,        /* Unit number */
```

6.7 Define the zone

```
long iut_define_zone( un, ztype )
     long      un,       /* Unit number */
             ztype,      /* Zone sort (A-H) */
```

6.8 Obtain a pointer of the specified block

```
long iut_get_pointer( un, ztype, pnt )
     long      un,       /* Unit number */
             ztype,      /* Zone sort (A-H) */
     struct idxptrtype *pnt,  /* Pointer */
```

6.9 Release the area denoted by the pointer

```
long iut_release_pointer( un, pnt )
     long      un,       /* Unit number */
     struct idxptrtype *pnt,  /* Pointer */
```

6.10 Obtain a pointer for the header data

```
long iut_get_header_pointer( un, nsect, pnt )
     long      un,       /* Unit number */
            nsect,       /* The number of continuous sectors */
     struct idxptrtype *pnt,  /* Pointer */
```

6.11 Copy 2-byte data array

```
long iut_copy_word( sc, dst, len )
     short     *sc,      /* Source data array */
              *dst,      /* Destination data array */
     long       len,      /* The amount of 2-byte data */
```

6.13 Copy 4-byte data array

```
long iut_copy_dword( sc, dst, len )
     long      *sc,      /* Source data array */
              *dst,      /* Destination data array */
               len,      /* The amount of 4-byte data */
```

6.14 Swap bytes

```
long iut_byte_swap( ar,len )
     long       len,      /* The amount of data in bytes (even) */
     char      *ar,       /* Data array */
```

Note: function arguments with an astarisk mean that a pointer should be given to the function.

Appendix B. Error codes

In the case of an error, the file manager makes an error return with following error codes. The step or the sequence of the file manager's service is kept in a global variable to show where the error takes place.

code	nuemonic	description
0001H	ENUNIT	Specified unit does not exist.
0002H	EMNTED	Specified volume has already been mounted.
0005H	ENBUFF	No more volume could be mounted or no more files could be open.
0006H	ENMNTD	Specified volume is not mounted.
0007H	EPARAM	Parameters in the arguments are ir@regularly given.
0008H	ENINDX	No free space to define another index table.
0009H	ENDATA	Data area is full.
000AH	EFLNEX	Specified file does not exist.
000BH	EOPNED	Specified file is already open.
000CH	ENPERM	The file prohibits access to be requested.
000DH	ENOPND	Specified file is already open.
000EH	ESZOVR	Specified data position is beyond the file size.
000FH	ENDELD	Specified file is not temporarily deleted.
0010H	ESYSRD	Read error of the primary system data and the back-up data is used.
0011H	ESYSWR	Write error of the primary system data.
0020H	ENHEAD	No space for the header data.
5000H	EMWRPT	Media is write-protected.
5001H	EUNRDY	Unit is not ready.
6000H	ENVINF	Volume information is not correct.
6001H	ENIS&C	Media does not meet the IS&C format require@ment.
7000H	ETMOUT	Time out.
8000H	EDRIVE	Physical error in the drive. Lower byte may have an error sense code from the drive.
C000H	ESYSIO	Physical error in the system area.

Appendix C. Initialization

Picking up an example of 130mm media with 1024 bytes/sector, the volume information areas and tables are initialized as follows:

1. Volume information (#0 sector)

The capacity of media could become known by using the "read capacity" command. Table 1 assumes that the media has 306 zones.

Table 1. Volume information (#0 sector)

address	item	length(B)	data type	description
0-3	format name	4	character	IS&C
4-7	version #	4	character	01.0
8-23	application field	16	character	MEDICAL
24-55	volume name	32	character	user defined
56-59	volume ID	4	binary	user defined
60-91	owner's name	32	character	user defined
92-123	owner's code	32	character	user defined
124-129	initilization date	6	binary	system defined
130-133	# of zones	4	binary	306
134-135	# of sectors/zone	2	binary	1024
136-137	# of bytes/sector	2	binary	1024
138-141	zone table address	4	binary	2
142-145	sector table address	4	binary	4
146-149	first index table address	4	binary	43
150-151	index table size	2	binary	128
152-1024	reserved area	872		0

2. Volume information (#1 sector)

Table 2. Volume information (#1 sector)

address	item	length(B)	data type	descritpion
0-3	# of index tables	4	binary	7848
4-7	# of active files	4	binary	0
8-11	# of temporarily deleted files	4	binary	0
12-15	# of free index tables	4	binary	7848
16-17	# of system files	2	binary	0
18-19	# of directory files	2	binary	0
20-25	latest update	6	binary	system defined
26-29	next free index number	4	binary	1
30-31	volume flag	2	binary	0
32-1007	reserved area	976		0
1008-1023	system reserved area	16		0

3. Zone table

address	zone number	zone class	# of free blocks	back-up zone
0-5	1	1	-1	306
6-11	2	0	0	0
12-17	3	0	0	0
18-23	4	0	0	0
:	:	:	:	:
:	:	:	:	:
1824-1829	305	0	0	0
1830-1835	306	-1	-1	1
1836-2047	This area should be filled with null (00H).			

4. Sector table

All bits corresponding to the system zones (#1 and #306 zones) should be on (1:under use), while the rest should be off (0:not used). This table should be followed by null area to fit the sector size.

5. Index tables

All index tables have to be cleared or filled with null (00H) except for the last 4 bytes. These 4 bytes form a long integer and are used for the index chain.

Table 3. Index chain

index number	data type	contents
1	integer	2
2	integer	3
3	integer	4
:	:	:
:	:	:
7847	integer	7848
7848	integer	-1

6. Header zone

When a header zone is defined, all areas should be cleared or filled with null (00H).

Appendix D.

Table D. Procedure flow and related tables

Table / Procedure	Volume Information NO.1	Volume Information NO.2	Zone table	Sector table	Index table	A zone Backup area	B zone	Data area
Initialize	I	I	I	I	I	I		
Mount	R	R & U/W Volume occupied flag	R	R	R			
create-file		U/W	U/W	U/W	U/W			
open-file					U/W Writing flag			
Write data: Size-fixed file								W
Write data: Size-free file		U/W	U/W	U/W	U/W			W
write-header			U/W	U/W	U/W		W	
change of file attribute		U/W			U/W			
close-file					U/W Writing flag			
Dismount		U/W Volume occupied flag				copy of A zone		

I:Initialization of IS&C file
U:Update data on the main memory
R:Read data from MO disk
W:Write data to MO disk

Questions about IS&C Specification should be addressed to Headquarters of IS&C committee shown below.

HEADQUARTERS of IS&C OFFICE

MEDIS-DC
(The Medical Information System Development Center)

TEL : 03-3586-6321
FAX : 03-3505-1996

ADDRESS : 10F Landic Akasaka Bldg.,
2-3-4 Akasaka Minatoku
Tokyo 107, Japan