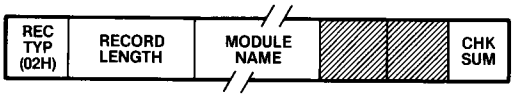


**Figure 8-7 Module Header Record**

The Content Record (see figure 8-8) provides contiguous data, from which a memory image may be constructed for a portion of memory.

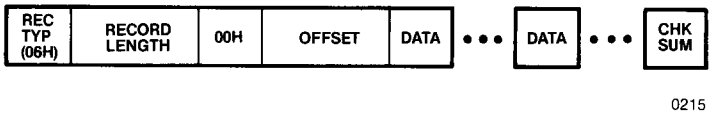


**Figure 8-8 Content Record**

The OFFSET field specifies the absolute location of the first data byte.

Following the OFFSET are one or more DATA bytes. Thus, this record provides N consecutive bytes of a memory image from OFFSET through OFFSET + N - 1, inclusive.

The Module End Record (see figure 8-9) has a MODULETYPE (MODTYPE) field with a value of 0 or 1. If the value is 1, the module is a main program. If the value is 0, the module is not a main program.

If the module is a main program, the OFFSET field specifies the module's execution start address. Otherwise, this field has no significance, but it must be present.

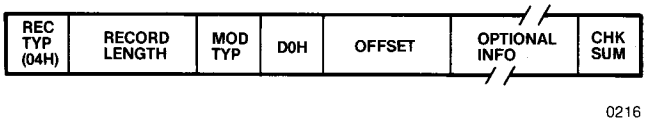


**Figure 8-9 Module End Record**

The OPTIONAL INFORMATION field may not be present depending on the language translator used. It contains debug information.

## Disk Structure

This section describes the structure of disks and disk files at the byte level. The information is not necessary to use the system calls described previously. Disk devices are discussed in Chapter 5. This section deals with the disk media and the structure of the files recorded on it.

Bubble memory multimodules are treated as virtual disk devices. Both flexible disks and bubble memory are organized into tracks and sectors. All flexible disks contain 80 tracks which are divided into 32 sectors of 256 bytes each. (Sixteen sectors are on side 0 and sixteen on side 1.) Bubble memory contains 16 tracks with 32 sectors of 256 bytes each. Disk capacities are given in the following chart.

| | Diskette | Bubble Memory |
|---|---|---|
| Tracks/Disk | 80 | 16 |
| Sectors/Track | 32 | 32 |
| Sectors/Disk | 2560 * | 512 |
| Bytes/Sector | 256 | 256 |
| Bytes/Disk | 655,360 | 131,072 |

* Only 2544 sectors available to the user.

### General Disk File Structure

Each disk contains a number of files. Each file is made up of 256-byte blocks. Each block corresponds to one disk sector, which is a hardware addressable unit. See figure 8-10. ISIS-PDS system files and commands occupy about 400 sectors on a system disk and 50 sectors on a non-system disk.

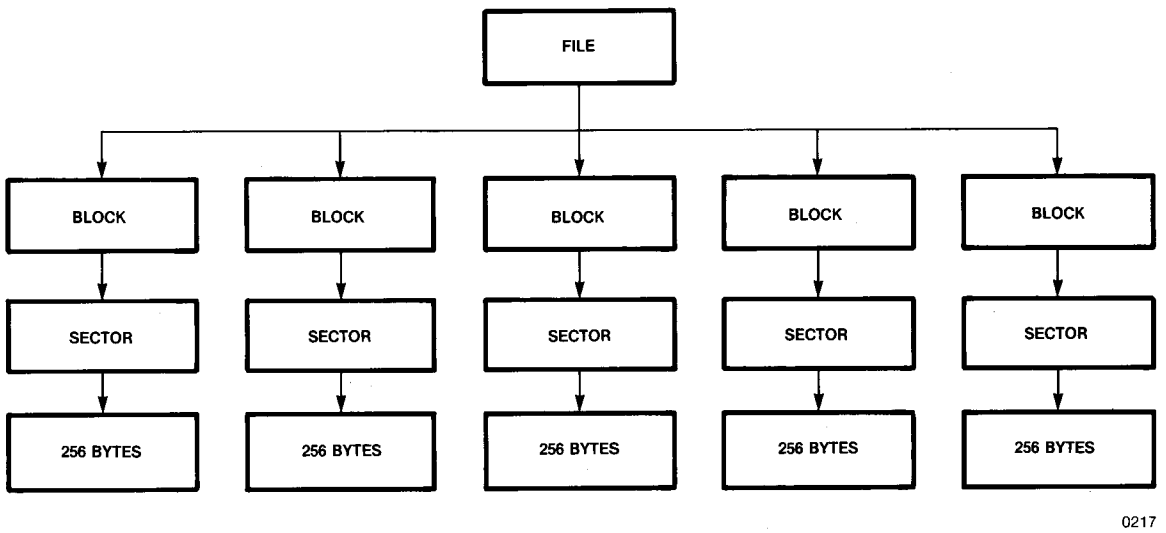


**Figure 8-10 Disk File Components**

Each sector on a disk has a unique address by which it can be accessed. The address consists of a one-byte track number and a one-byte sector (block) number. Tracks are numbered 0-79 on a diskette and 0-15 on bubble memory. The sectors on a track are numbered 1-32 on both diskettes and bubble memory. The address of a block is also referred to as a pointer to that block. Related blocks are linked together by pointers. That is, two of the bytes in a block may contain the address of a related block.

**Blocks.** A block is the data in one sector. There are two types of blocks in a file: pointer blocks and data blocks. Pointer blocks contain nothing but pointers to other blocks as shown in figure 8-11.

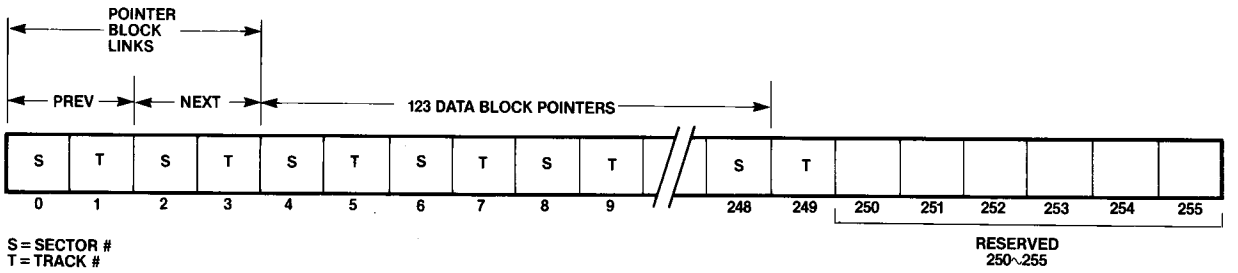


**Figure 8-11 Pointer Block**

All files begin with a pointer block that is called the header block. If the file contains fewer than 123 data blocks, the header block is the only pointer block in the file. If there are more than 123 data blocks, there is an additional pointer block for every 123 data blocks. For example, a file of 300 data blocks contains 3 pointer blocks, including the header block.

The first two pointers in a pointer block are links to other pointer blocks in the file. The first link contains the address of the previous pointer block. The header block always contains zeros in this field because there is no previous pointer block in the file. The second link contains the address of the next pointer block in the file. The last pointer block in the file has zeros in this field.

Following the links to other pointer blocks are 123 pointers to the data blocks in the file followed by six reserved bytes. If a pointer contains zeros, then no data block has been allocated for the pointer. A zero pointer does not necessarily mark the end of the file.

Data blocks, as shown in figure 8-12, have no particular format, since they contain user data.

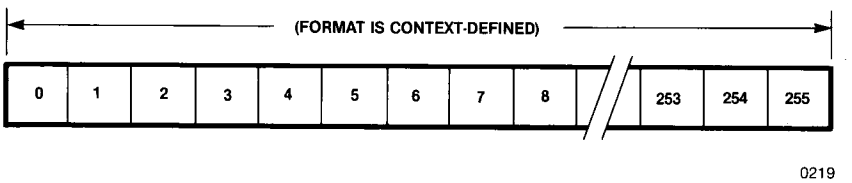


**Figure 8-12 Data Block**

Data blocks are fundamentally different from pointer blocks. Data blocks are visible to users; they contain the information that is transferred by read and write operations. Pointer blocks, on the other hand, are invisible to users; the data they contain is of interest only to the system. Data blocks can be thought of as destinations with pointer blocks as paths to those destinations. To access user data in a file, ISIS follows a path of pointers to a data block.

The relationship of pointer and data blocks is shown in figure 8-13.

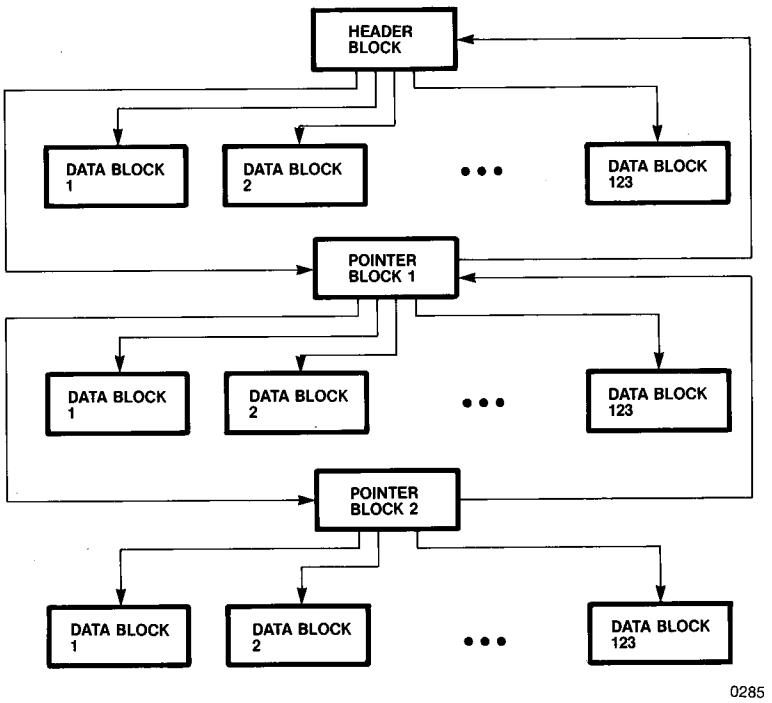


**Figure 8-13 Relation of Data and Pointer Blocks**

Figure 8-14 shows an example file which consists of data in five blocks: DATA 1 through DATA 5. The diagram in figure 8-14 is simplified in that it shows only four sectors per track instead of 32 and only two data pointers per pointer block instead of 123. The file begins at the header block which contains pointers to the first two data blocks, DATA1 and DATA2.

The header block is linked to a second pointer block at sector 3 of track 8. The second pointer block contains pointers to DATA3 and DATA4. It is linked back to the header block and forward to the last pointer block at sector 1 of track 9.

The third pointer block contains the last data pointer in the file. Because it is the last pointer block, it contains a backward link to the second pointer block but no forward link.

Notice that it is the data pointers which order the data blocks for sequential access. The physical locations of the data blocks and the pointer blocks are irrelevant. Because of this ability to scatter files on the disk, the system can make efficient use of available space. Note also the data capacity is reduced by the number of pointer blocks on a disk.

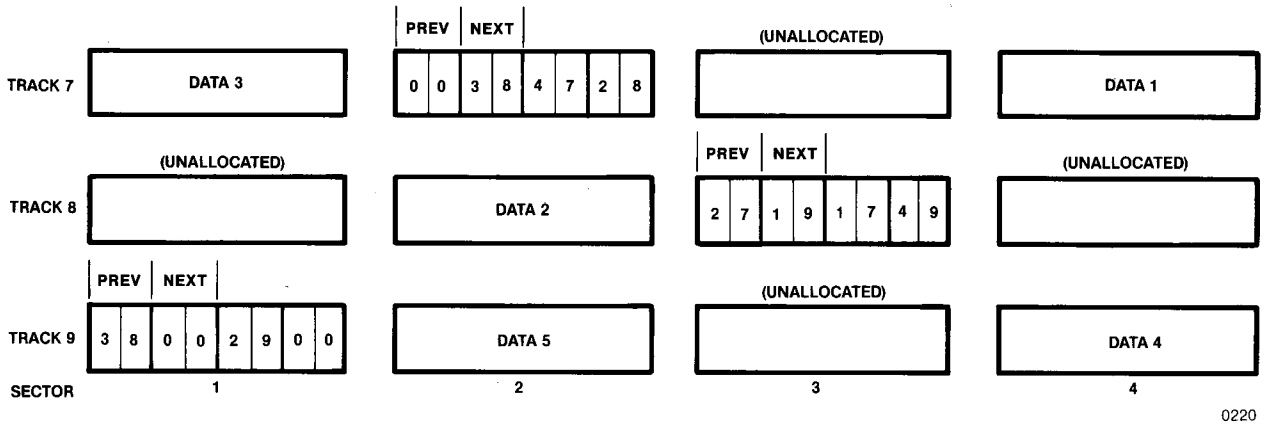


**Figure 8-14 Pointer and Data Blocks in a File**

**Interleaving Factors.** Interleaving factors are used to speed up the sequential access of blocks on the same track. Often a program reads a block, processes that block, reads the next block, and so on. If the blocks were stored in physically adjacent sectors, the disk drive read/write head would pass by the second sector while the program was processing the first sector. The program would have to wait one full disk revolution for the read/write head to seek the second sector again. The effect of an interleaving factor of 3 is shown in figure 8-15.

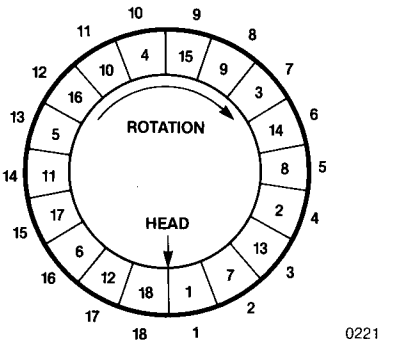


**Figure 8-15 Sector Interleaving**

Physical sector addresses are shown outside the track (which is simplified for the drawing to show only 18 sectors). Logical sector addresses, which are the basis for accessing blocks stored in the sectors, are shown on the track.

With interleaving, a program which reads and processes every block in logical sequence has a processing window equivalent to the time it takes for two sectors to pass by the head.

Assuming that processing each block takes slightly less time than is available in this window, all 18 blocks can be processed in three revolutions of the disk. Without interleaving, 18 revolutions would be required.

The ISIS-PDS operating system uses an interleaving factor of 4 except for Track 0, Sectors 1-16 which has an interleaving factor of 1. Track 0 contains the file ISIS.T0 which is formatted at 128 bytes/sector and contains information needed to initialize the system. The interleaving information is used by the IDISK command when disks are formatted.

## System Disk Files

All ISIS non-system disks contain four system files: ISIS.T0, ISIS.LAB, ISIS.DIR, and ISIS.FRE. These are created automatically when the disk is initialized.

The location of these files is fixed as shown in table 8-4. The FROM and THRU values are given in the form T,S where T is the track number and S is the sector number. The values are given in hexadecimal.

**Table 8-4 System File Locations**

| File Name | | Double Density Mini-Diskette | | Bubble Memory | |
|---|---|---|---|---|---|
| | | FROM | THRU | FROM | THRU |
| ISIS.T0 | (Header) | 00H,11H | 00H,11H | 00H,11H | 00H,11H |
| | (Data) | 00H,12H | 00H,20H | 00H,12H | 00H,20H |
| ISIS.LAB | (Header) | 01H,01H | 01H,01H | 01H,01H | 01H,01H |
| | (Data) | 01H,02H | 01H,04H | 01H,02H | 01H,04H |
| ISIS.DIR | (Header) | 27H,01H | 27H,01H | 00H,01H | 00H,01H |
| | (Data) | 27H,02H | 27H,10H | 00H,02H | 00H,04H |
| ISIS.FRE | (Header) | 27H,11H | 27H,11H | 00H,05H | 00H,05H |
| | (Data) | 27H,12H | 27H,14H | 00H,06H | 00H,08H |

On a system disk, 6 files are reserved for the operating system. These are: ISIS.PDS, ISIS.CLI, ISIS.TO, ISIS.LAB, ISIS.DIR, and ISIS.FRE. Note that four of these appear on a non-system disk as well. In addition to these six files, there are a number of command files containing programs and a library file named SYSPDS.LIB.

## ISIS.PDS

This file contains the ISIS kernel, that is, the resident system routines.

## ISIS.CLI

This file contains the command line interpreter which occupies part of the user program area of memory.

## ISIS.T0

This file contains a program called T0BOOT. When the RESET button is pressed, this file is read in from the disk. Once it is loaded into memory, T0BOOT begins executing. This program reads the contents of ISIS.PDS and displays the ISIS sign on message. If an attempt is made to initialize the system from a non-system disk, T0BOOT displays the message:

NON-SYSTEM DISKETTE

on the screen. When running from a hardware reset, T0BOOT then returns control to the initialization PROM. If running from a FUNCT-R (a software reset), T0BOOT stops after attempting to initialize from the system diskette.

## ISIS.LAB

The first nine bytes of this file contain the disk label stored as nine ASCII characters with a six-character name and a three-character extension.

The rest of the bytes are undefined except for the last 256 bytes (corresponding to Track 1, Sector 4) which are filled with repetitions of the ASCII characters:

DIAGNOSTICSECTOR

These bytes are used by diagnostic programs described in Appendix A.

## ISIS.DIR

Each disk contains one directory. This file contains 15 data blocks (3 for bubble memory); each block has room for 16 directory entries. One entry is used for each file on the disk, so there is room in the directory for 240 files (48 for bubble memory). Each directory entry is 16 bytes long and is formatted as shown in figure 8-16.

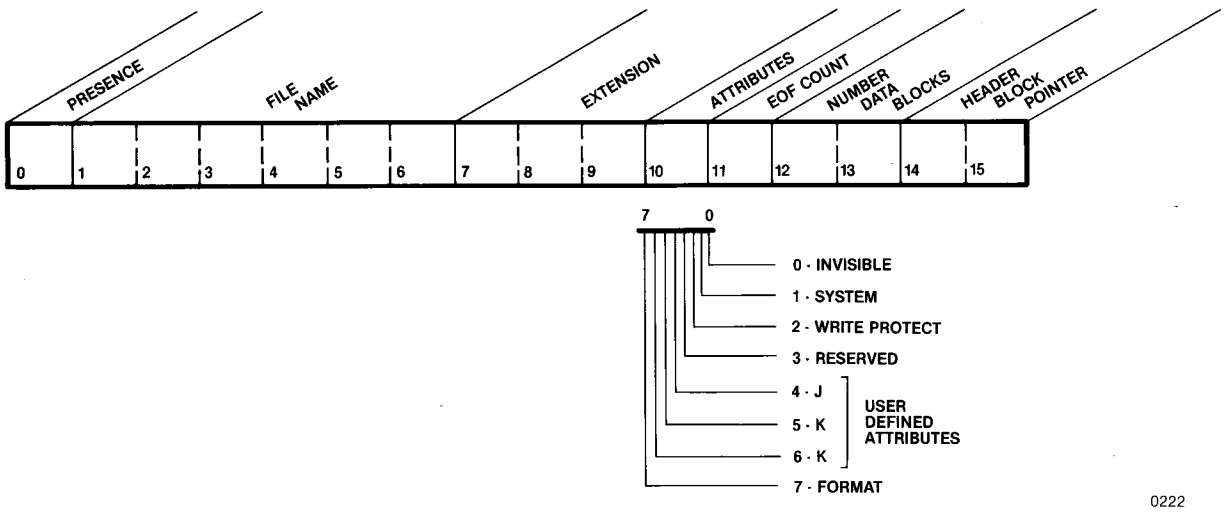


**Figure 8-16 Directory Entry**

The following chart explains the field names used in figure 8-16.

**PRESENCE** is a flag which can contain one of three values:

00H: The file associated with this entry is on the disk.

7FH No file is associated with this entry; the content of the rest of the entry is undefined. The first entry with its flag set to 7FH marks the current logical end of the directory; directory searches stop at this entry.

FFH The file named in this entry once existed on the disk, but is currently deleted. The next file added to the directory will be placed in the first entry marked FFH. This flag cannot, therefore, be used to find a file that has been deleted, unless no other files have been created or written since the deletion. A value of FFH should be thought of as marking a free directory entry.

**FILENAME** is a string of up to 6 non-blank ASCII characters specifying the name of the file associated with the directory entry. If the filename is shorter than six characters, the remaining bytes contain 00H. For example, the name ALPHA would be stored as 41H 4CH 50H 48H 41H 00H.

**EXTENSION** is a string of up to three non-blank ASCII characters that specify an extension to the filename. Extensions often identify the type of data in the file such as OBJ for object module or PLM for PL/M source module. As with filename, unused positions in the extension field are filled with zeroes.

**ATTRIBUTES** are bits that identify certain characteristics of the file. A 1 bit indicates that the file has the attribute, while a 0 bit means that the file does not have the attribute. The bit positions and their corresponding attributes are listed below (bit 0 is the low order or rightmost bit, bit 7 is the leftmost bit):

0: Invisible. Files with this attribute are not listed by the DIR command unless the I option is used. All system files are invisible.

1: System. Files with this attribute are copied to any disk being initialized as a system disk.

2: Write Protect. Files with this attribute cannot be opened for output or for update, nor can they be deleted or renamed.

3: Reserved.

4-6: J, K, and L. User defined attributes.

7: Format. Files with this attribute are treated as though they are write protected. Some of the system files have this attribute. It should not be given to other files.

Attributes can be written with the ATTRIB command or the ATTRIB system call.

EOF COUNT contains the number of the last byte in the last data block of the file minus 1. If the value of this field is 80H, for example, the last byte in the file is byte number 129 in the last data block.

NUMBER OF DATA BLOCKS is a two-byte variable indicating the number of data blocks currently used by the file. To calculate the current number of bytes in the file, use the following formula:

(NUMBER OF DATA BLOCKS) * 256 + EOF COUNT + 1

HEADER BLOCK POINTER is the address of the file's header block. The low order byte in this field is the sector number, and the high order byte is the track number. The system finds a disk file by searching the directory for the name, then using the header block pointer to seek the beginning of the file.

## ISIS.FRE

This file contains a bit map of the disk, with each bit position representing one cluster of the disk. A cluster is 4 blocks or 4 sectors of the disk. Since there are 32 sectors/track, each byte of the bit map represents one track on the disk. For diskettes, the bit map is 80 bytes long and for bubble memory the bit map is 16 bytes long.

If a bit in the bit map is 1, the corresponding cluster is allocated, that is, in use as a pointer block and/or as data blocks. If a bit in the bit map is 0, the corresponding cluster is free space on the disk. When a file is deleted, the bits that correspond to the clusters it previously occupied are reset to 0.

Table 8-5 shows the values of the bit maps for the system files located at tracks 00H, 01H, and 27H on the mini-diskette and tracks 0 and 1 on the bubble memory.

**Table 8-5 Values of System File Bit Maps**

| | Track 00H | Track 01H | Trace 27H |
|---|---|---|---|
| Mini-diskette | 0FFH<br>ISIS.T0 | 01H<br>ISIS.LAB | 01FH<br>ISIS.FRE<br>ISIS.DIR |
| Bubble Memory | 0F3H<br>ISIS.T0<br>ISIS.DIR<br>ISIS.FRE | 01H<br>ISIS.LAB | |

ISIS uses a pre-allocation scheme to allocate disk space to requesting programs. A pre-allocation table is maintained in memory containing a list of available clusters. These clusters have already been set on in the bit map, so they will not be allocated by any other program.

If a program requests disk space and there are no clusters available in the pre-allocation table, ISIS gets 5 clusters and sets the bit map in ISIS.FRE for these 5 clusters. One of these clusters is made available to the requesting program. The other four are saved in memory in a pre-allocation table for future requests.

This technique saves time since ISIS only has to access the disk one time to allocate 5 clusters. Thus, if the system is reset before clusters 2-5 are used, they will remain allocated but will never be usable again.

## Disk File Structure Summary

Figure 8-17 provides an overall view of the most important elements in the file structure. Some simplifications have been made for clarity. There are only four directory blocks, pointer blocks contain only four data block addresses, and so on. However, the key relationships of file elements are shown.

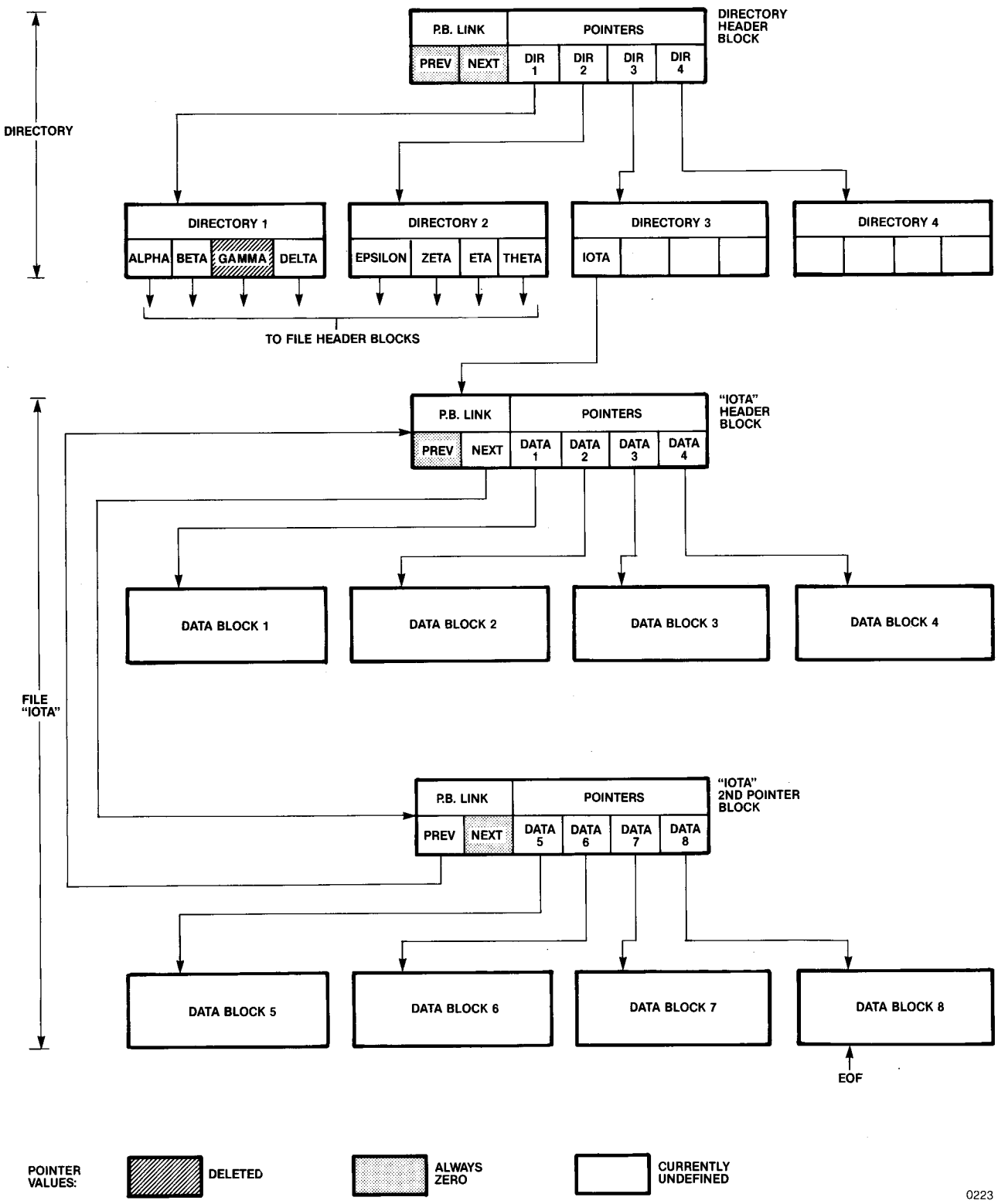


**Figure 8-17 Disk File Structure Summary**