96768040

CONTROL DATA CORPORATION

FILE MANAGER VERSION 2 REFERENCE MANUAL

CDC® OPERATING SYSTEM:
INTERACTIVE TERMINAL-ORIENTED SYSTEM

REVISION RECORD

REVISION	DESCRIPTION
01	Prereleased edition
(4/77)	
A	Manual released. This manual obsoletes all previous editions.
(6/77)	
B	Manual revised to incorporate ITOS 1.1 information. Section 2 also revised to incorporate format changes.
(10/77)	
C	Manual revised to incorporate ITOS 1.2 information. Pages ii, iii, v, 2-1, 2-5, 2-6, 2-12, 2-14, 2-18, 2-21, 2-26,
(1/78)	B-4, B-5, C-1, D-3, F-1, F-4, I-3, K-1, Index-1, and Index-2 are changed. Pages 2-6.1 and B-4.1 are added.
Publication No. 96768040	

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

Control Data Corporation
Publications and Graphics Division
4455 Eastgate Mall
La Jolla, California 92037
or use Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual, are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

PAGE	REV
Cover	--
ii thru vii	C
viii	B
1-1	B
1-2	A
1-3	B
1-4	A
1-5	B
1-6	A
1-7	A
1-8	B
2-1	C
2-2	B
2-3	B
2-4	B
2-5	C
2-6	C
2-6.1/2-6.2	C
2-7 thru 2-11	B
2-12	C
2-13	B
2-14	C
2-15 thru 2-17	B
2-18	C
2-19	B
2-20	B
2-21	C
2-22 thru 2-25	B
2-26	C
2-27	B
2-28	C
A-1	B
A-2	A
B-1	B
B-2	B
B-3	A
B-4	C
B-4.1/B-4.2	C
B-5	C
B-6 thru B-13	A
C-1	C
C-2	B
C-3 thru C-14	A
D-1	B
D-2	A
D-3	C
D-4	B
D-5	A
D-6	A
D-7	B
E-1	A
F-1	C
F-2	A
F-3	B
F-4	C
G-1	B
H-1	A
I-1	B
I-2	B
I-3	C
I-4	B
J-1	A
K-1	C
L-1	C
M-1	B
Index-1	C
Index-2	C
Comment sheet	C
Cover	--

PREFACE

The CDC® CYBER 18 File Manager 2 is a general purpose file management system that operates under the Interactive Terminal-Oriented System (ITOS), Version 1.2.

Most of the examples in this manual are written in CYBER 18 Mass Storage FORTRAN, Version 3. It is assumed that users of this manual are familiar with that language.

Familiarity with the Mass Storage Operating System (MSOS), Version 5, will be helpful in reading this manual, though not essential, since terms used are defined in a glossary (appendix A).

The following CYBER 18 manuals contain additional information useful to file manager users:

Publication	Publication Number
Mass-Storage Operating System (MSOS) Version 5 Reference Manual	96769400
MSOS Version 5 Ordering Bulletin	96769490
Mass Storage FORTRAN Version 3A/B Reference Manual	60362000
Macro Assembler Reference Manual	60361900
Interactive Terminal-Oriented System (ITOS) Version 1 Reference Manual	96768290

CDC manuals can be ordered from Control Data Literature and Distribution Services, 8001 East Bloomington Freeway, Minneapolis, MN 55420

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.

CONTENTS

APPENDIXES

INDEX

FIGURES

TABLES

SUMMARY

HARDWARE REQUIREMENTS

Files must be maintained on a direct access mass memory device, not on magnetic tapes.

SOFTWARE REQUIREMENTS

The file manager runs under ITOS 1.1. Partitioned main memory must be included in the system, but it is not necessary to include the partitioned main memory driver. Allocatable main memory is not used by the file manager. The file manager uses protected blank common. Therefore, protected blank common may not be used by non-file manager programs.

FILE MANAGER SUPPORT OF EXISTING SOFTWARE

RPG II, version 1.1, running under ITOS 1 and Sort/Merge 2, is supported by File Manager 2. Job processor files, pseudo tapes, Timeshare 3, and the background editor, which depend on File Manager 1, are not supported by File Manager 2.

FUNCTIONS

The file manager can be used by protected programs and by unprotected programs either in the background or in the ITOS user area. File manager users can create and maintain sequential and indexed files. Records within these files may be retrieved according to relative order within a file or according to an identifier or key value. A given file may be indexed by up to four keys. The file manager provides for deletion of a record, updating of a record, and deletion of an entire file. A file can be locked or particular records within a file can be locked by a user to prevent concurrent use by another file user.

USE OF MASS MEMORY SPACE

File manager space is predefined at system initialization. Space used by a deleted file is available to the file manager for a new file. Space used by deleted records can be used for new records if the file is compressed. Mass memory space defined for file storage cannot be used for other system purposes. In a given system, mass memory file space is usually not expanded or diminished. There are, however, procedures for expanding or diminishing file space. These procedures are described in appendix H.

ORGANIZATION OF MODULES

The file manager consists of a main-memory-resident file manager executive, a set of main-memory-resident request processors and support subroutines, a set of mass-resident request processors, and an interceptor module that must be in main memory whenever a program makes a file manager request. File manager parameters and tables are discussed in appendix D. Also see File Manager Interceptor Module and Request Processors later in this section.

FILE STORAGE

The files reside on mass memory together with information describing how to find information regarding a specified file (file definition directory) and how to find a record within a file (file control block table). These structures are described in appendix B.

KEY STORAGE

Indexed files require additional pointers. For a given key on a given file, these pointers are stored in conjunction with a pyramid of partitions. Each partition is a linearly ordered set of disjoint intervals of key values such that the union of these intervals is the range of the partition. For example, if records are stored according to the key, age in years, a typical pyramid of partitions with corresponding pointers can be represented schematically as shown in figure 1-1. In this figure, both end points of each interval are indicated.

In an actual file manager file index structure, only the right-hand endpoint of each interval is stored. In figure 1-1, the highest block of the pyramid contains the intervals (0 through 20), (21 through 30), (31 through 35), (36 through 43), and so forth. In the figure, the dashed lines show the path of a file manager search for all records for age 28.

This figure is only a schematic representation of the index structure used by the file manager. The actual details of the file index structure are described in appendix C. The purpose of this example is to show the user that file manager indexed files are set up to be particularly efficient when retrieving all file records for a given key in order. This structure is not particularly efficient for retrieval of a large set of particular records as specified by key values that are not in order.

The pyramid structure for each key is disjoint from all other pyramids for that file. Thus, if another key for the file represented in figure 1-1 is weight, there would be no information regarding weight in the age pyramid in figure 1-1, except in the file records in the bottom line of the figure.

FILE TYPES

SEQUENTIAL FILES

A sequential file is one in which each new record is added immediately following the last record stored in the file. Records within a sequential file may be retrieved consecutively in the order stored, or a particular record may be retrieved by specifying its position within the file; that is, the relative record number.

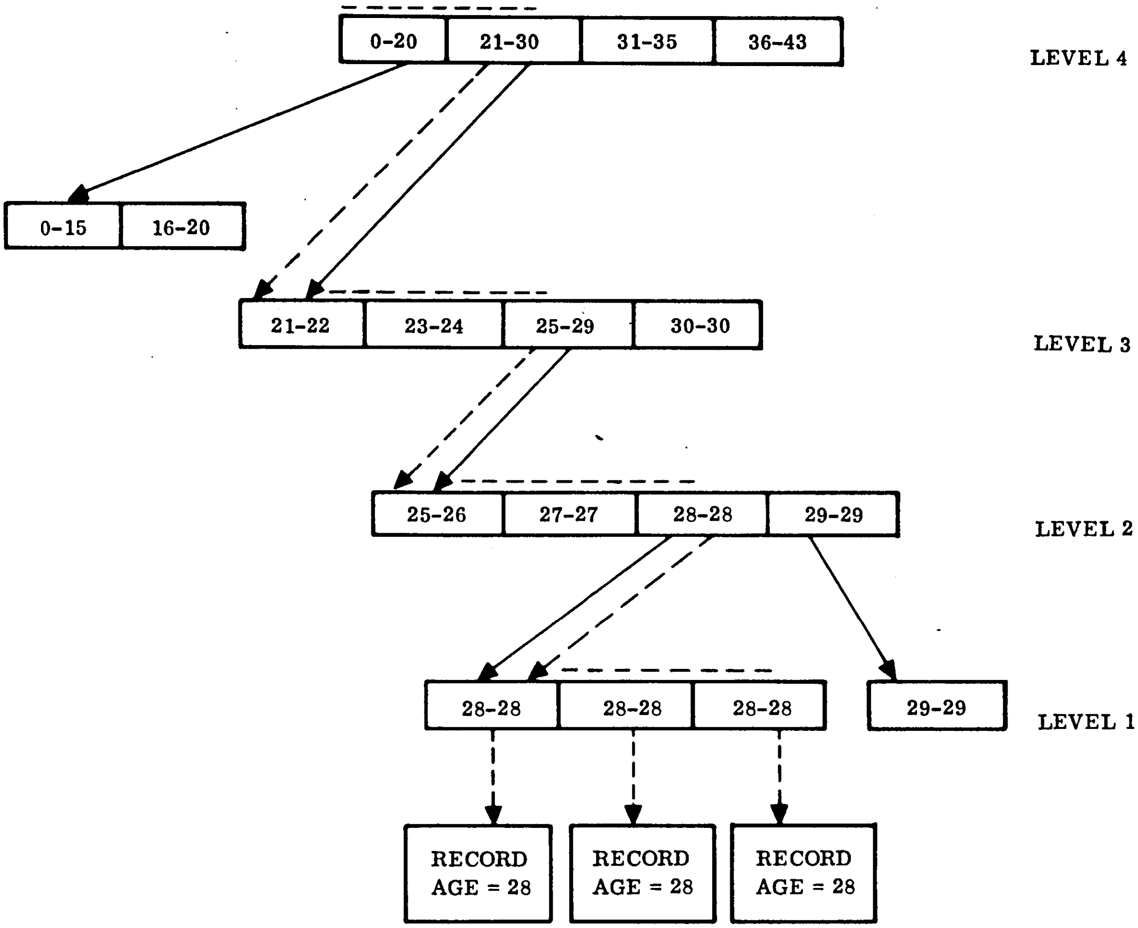


Figure 1-1. Example of Pyramid of Partitions

INDEXED FILES

An indexed file is one in which each record has at least one associated attribute or key (surname, social security number, age, sex, record number, etc.). In File Manager 2, a file may have up to four keys. Each key may range in length from one to twenty-nine bytes (eight to 232 bits). Each key value must be stored within its associated record. (The position of storage within the record is described in Create File (CREATE), section 2.) Multiple occurrences of any key value are permitted for each key except the primary key (the first key defined for that file), which must have unique values.

For each key for a given file, an index is set up so that records may be retrieved in numeric order according to the binary representation of the key values. For keys of more than two bytes, numeric order means that order achieved by considering each key value a binary number written as the concatenation of the bytes of the key value with the first bytes being most significant. This means that records with BCD keys may be retrieved in numeric order and records with EBCDIC or ASCII keys may be retrieved in alphabetic order of the key values.

For example, if state abbreviation is a key, the following key values might occur:

MN (Minnesota)

CA (California)

MO (Missouri)

CO (Colorado)

These have two-byte ASCII representations as follows:

MN = $4D4E_{16}$

CA = 4341_{16}

MO = $4D4F_{16}$

CO = $434F_{16}$

In numeric order, these key values are:

4341_{16} = (CA)

$434F_{16}$ = (CO)

$4D4E_{16}$ = (MN)

$4D4F_{16}$ = (MO)

Another example is the last name of a group of people:

SMITH

JONES

MEAD

JONE

If left-justified, these have ASCII representations as follows:

SMITH = $534D_{16}$, 4954_{16}, 4820_{16}

JONES = $4A4F_{16}$, $4E45_{16}$, 5320_{16}

MEAD = $4D54_{16}$, 4144_{16}, 2020_{16}

JONE = $4A4F_{16}$, $4E45_{16}$, 2020_{16}

In numeric order, these key values are:

JONE = $4A4F_{16}$, $4E45_{16}$, 2020_{16}

JONES = $4A4F_{16}$, $4E45_{16}$, 5320_{16}

MEAD = $4D54_{16}$, 4144_{16}, 2020_{16}

SMITH = $534D_{16}$, 4954_{16}, 4820_{16}

However, if right-justified, the keys have ASCII representations as follows:

SMITH = 2053_{16}, $4D49_{16}$, 5448_{16}

JONES = $204A_{16}$, $4F4E_{16}$, 4553_{16}

MEAD = 2020_{16}, $4D54_{16}$, 4144_{16}

JONE = 2020_{16}, $4A4F_{16}$, $4E45_{16}$

In numeric order these key values are:

JONE = 2020_{16}, $4A4F_{16}$, $4E45_{16}$

MEAD = 2020_{16}, $4D54_{16}$, 4144_{16}

JONES = $204A_{16}$, $4F4E_{16}$, 4553_{16}

SMITH = 2053_{16}, $4D49_{16}$, 5448_{16}

A schematic example of the index structures generated to enable retrieval by key value order is included in figure 1-1. Details of this structure are found in appendix C.

File Manager 2 indexed files are also sequential files in that each new record is added immediately following the last record stored in the file. The position of each record within an indexed file is known according to relative record number. Thus, File Manager 2 indexed files are indexed-sequential files.

Records may be retrieved from an indexed file in one of the following manners:

- All records corresponding to a given key value may be retrieved.
- A set of records may be retrieved in order according to numeric representation of key values.
- A specific record may be retrieved according to storage position within the file.
- A set of records may be retrieved in the order they were stored within the file.

FILE REQUESTS

REQUEST TYPES

There are two types of file requests – file specification requests and record accessing requests. These requests are described in section 2 under Specification Requests and Record Accessing Requests, respectively. File requests are summarized in figure I-1.

REQUEST BUFFER

Associated with each use of a particular file is a 24-word request buffer used to process the request. The same request buffer must be used for a sequence of requests referring to the same file. This buffer is used by the file manager to process requests, to save information between related requests, and to pass back information to the caller. The buffer may not be altered by the user between successive file manager calls. This buffer is normally within the user's program. For an unprotected user program, the buffer must be in unprotected main memory.

STATUS INDICATOR WORD

Also associated with each request is a status indicator word. Upon completion of the request, the status indicator word contains request execution status information. Each bit of the indicator word that is nonzero signifies an abnormal occurrence. If the entire word is zero, the request has been completed normally. If bit 15 is nonzero, the request has been rejected because of errors denoted in the other bits. If bit 15 is zero, but other bits are nonzero, the request has been completed with an irregular occurrence (for example, end-of-file is detected). All indicator bits are shown in figure F-1.

FILE IDENTIFICATION

Each file is identified at the time of its creation by a file name and a file owner. Both the file name and file owner are specified as ASCII strings of eight characters each.

A volume is a single physical unit of a peripheral storage device; for example, a removable disk cartridge, a disk pack, or a nonremovable disk cartridge. For each volume used in a system, a file identification (concatenated name/owner ASCII string) identifies a unique file. In order to access a file, a user must know the file's identification string. Two or more files on a volume may have the same name if they have different owners specified. A user may wish to define a file's owner name to be eight ASCII blanks (2020_{16},2020_{16},2020_{16},2020_{16}) if more than one user is to use the file.

RELATIVE RECORD NUMBER

A relative record number is assigned by the file manager to each new record stored in a file. Relative record number defines the position of a record within a file. This number is a 24-bit number stored as a right-adjusted three-byte field in a two-word array. The left byte of the first word is always zero. The first word contains a positive 8-bit number (n). The second word contains a positive 16-bit number (m). Then, if r is the relative record number:

$r = n \times 65536 + m$

where $0 \leq m \leq 65535$

The relative record number, r, may be thought of as the ordered pair (n,m).

Examples:

Relative record number 65,540 = $(0001_{16}, 0004_{16})$

Relative record number 35 = $(0000_{16}, 0023_{16})$

FILE OPEN AND CLOSE

A user must gain permission to access a file each time he uses it. This is done by a file manager open request. When accesses to the file have been completed by a user, he performs a file manager close request to relinquish permission to use the file at this time.

UPDATE PROTECTION

Locking and unlocking procedures are included in the file manager to prevent two users from simultaneously updating the same record and thus losing part of the updated data. The following situation can occur if locking is not used: user A retrieves record n of a given file. User B then retrieves record n of the same file. User A now modifies word 10 of the record and stores it back into the file. User B then modifies word 12 of record n and stores the record back into the file. Since the copy of record n retrieved by user B does not contain the modified word 10, the original value of word 10 is in record n after user A and user B have completed their updates. Locking and unlocking are used to prevent this situation.

A set of one or more records may be locked by a user or an entire file can be locked. Any record in a locked file or in a locked set of records cannot be retrieved, updated, or deleted by another user. A new record cannot be stored into a locked file by another user. A user may wish to lock an entire file when generating a report from the file, when dumping the file to magnetic tape, or when performing some other function that cannot allow any changes to the file during its operation.

Whenever a set of one or more records is to be updated, the user must retrieve and lock the records and subsequently store and unlock the records using an update record request. Only one set of records of a given file may be locked by a single user at any one time. This restriction is made to limit the main memory space required for file manager tables. After a user has locked a set of records in a file, his next retrieve or store request for that file automatically unlocks the locked records. A retrieve request that attempts to lock a locked record is rejected. The rejected request may be repeated until the retrieve is successful.

NOTE

Extreme care should be used when employing record locks in which records in two or more files must be used concurrently. Suppose user A retrieved and locked record m of file M, user B retrieved and locked record n of file N, user A repeatedly issues a retrieve request for record n , and user B repeatedly issues a retrieve request for record m . If both user A and user B repeatedly issue a retrieve request for their needed records without giving up after a certain number of rejects and unlocking their records, both users will wait indefinitely as there is no way to grant either of the requests.

SYSTEM EXECUTIVE CLOSE OF ALL FILES OPENED BY A PARTICULAR USER

A special file manager interface to the system executive allows the executive to request that a set of files opened by a particular user be closed. The system executive, while monitoring one or more user programs, may thus close all files left open by an aborted user. The set of files to be closed is identified by one of the following:

- The beginning and ending main memory addresses of the space occupied by the user program.
- A unique user identification code assigned to the user by the system executive at the time the user's open file requests were intercepted by the system executive.
- The volume on which the files reside (all open files on a given volume may be closed in this way).

FILE MANAGER INTERCEPTOR MODULE

A special request interceptor module is used to intercept all file manager request calls. This module redirects the file manager requests to the main-memory-resident request supervisor. There are two versions of this module – a reentrant version, FMCEPT, and a nonreentrant version, FMENTP.

If any main-memory-resident program uses the file manager, a copy of the reentrant interceptor module, FMCEPT, must be included in main memory.

A mass-resident program that runs in foreground-allocatable main memory must link to FMCEPT if it contains any reentrant code. Otherwise, it must have the nonreentrant interceptor module, FMENTP, included as a part of its absolutized load. If FMENTP is used, the program must declare all file manager request names as relative externals. (An example of such a program is shown in figure 2-6.)

A mass-resident program that runs in partitioned main memory must have the nonreentrant interceptor module, FMENTP, included as a part of its absolutized load. A partitioned main memory program need not declare file manager request names as relative externals.

A background program that uses the file manager is automatically linked to the interceptor module, FMENTP, in the program library at the time the program is loaded. Such a background program should not declare file manager request names as relative.

REQUEST PROCESSORS

There are two types of file manager request processors – reentrant processors and serial processors.

REENTRANT REQUEST PROCESSORS

The reentrant request processors are indicated in figure G-1. Each reentrant processor is main-memory-resident. The set of reentrant processors can concurrently process one request for each volume in the system. For a given volume, if a reentrantly executable request is made for the volume while another reentrant request is being processed for that volume, the new request is queued according to the priority of the request. Requests are then processed according to the queue. An example of the queuing is shown in figure 1-2.

SERIAL REQUEST PROCESSORS

An indication as to which request processors are serial is contained in figure G-1. Each serial processor is mass-memory-resident and executes in partitioned main memory. The manner of executing and queuing serially executable requests is the same as executing and queuing reentrantly executable requests on a single volume. An example of the queuing is shown in figure 1-3.

VOLUME LABELING, ENABLING, DISABLING

A volume is a removable disk cartridge, a disk pack, or a nonremovable disk cartridge. The label on a volume is a table of information written on the volume. The label format is described in appendix E. Labeling is a procedure for initializing the label on a volume so that the volume can be used by the file manager. Labeling software is external to the file manager. Labeling for the system volume (SYSVOL) is performed by the system initializer. Labeling for other volumes is performed using the ITOS UTIL command INIT. Labeling is described in the ITOS reference manual.

Enabling a volume is a procedure for notifying the system that a volume is mounted and ready for use by the file manager. Similarly, disabling is a procedure that disables use of a volume by the file manager so that the volume may be removed or shut down. The file manager provides the request processor, VOLUSE (see Enable/Disable Volume (VOLUSE), section 2), which performs the volume enabling and disabling functions. The VOLUSE request is to be used only by system utility programs.

RECORD RECOVERY FOLLOWING SYSTEM FAILURE

The file manager includes a procedure for record recovery in case of system failure. A similar scheme is used for key information structure recovery. A description of the file manager recovery procedures is contained in appendix K.

LIMITATIONS

The following limitations exist:

- If n = number files on a volume, then

  $1 \leq n \leq 2047.$

- If q = record length in bytes, then

  $1 \leq q \leq 32{,}766 = 2^{15} - 2.$

- If r = maximum number records in a file, then

  $1 \leq r \leq 16{,}777{,}215 = 2^{24} - 1.$

- If k = key value length in bytes, then

  $1 \leq k \leq 29$

  (see the Key Storage section).

SYSTEM-RESERVED WORDS; END-OF-FILE AND RECORD DELETION CODES

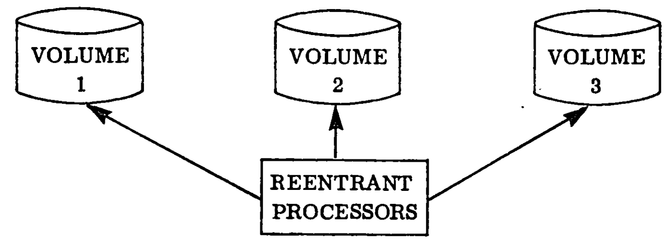
To signify that a file record has been deleted, the file manager stores a record-deleted code into the first word of the record. This code is determined at the time of system installation. It is usually an infrequently used ASCII code (see Main-Memory-Resident File Manager Operation Installation Parameters, appendix D).

To denote the last record in a file, the file manager stores an end-of-file code into the first two words of the next record space available after the last record in the file. This end-of-file code is a system installation parameter, usually an infrequently used ASCII code (see Main-Memory-Resident File Manager Operation Installation Procedure, appendix D).

To avoid false detection of deleted records and ends-of-file, the user must design his record format so that the above codes can never occur as data in the first two words of a record. Any random binary data may therefore never be stored in the first two words of a record.

AUTOMATIC VOLUME CHECKING; VOLUME DISABLING

The file manager provides the interface for periodic automatic checks of each volume in current use by the file manager. The file manager interface consists of periodic scheduling of the system ordinal, MNTCHK. If MNTCHK is in a system, it is scheduled at system startup and at timed



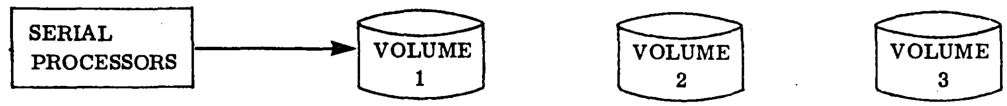
Heading Processor	Reentrant Request	Request Time	Pri-ority	Volume	Com-pletion Time
LOKFIL	U_0	t_0†	9	1	t_5
UNLFIL	U_1	t_1	13	2	t_7
PUTS	U_2	t_2	8	3	t_9
UPDATE	U_3	t_3	7	3	††
LOKFIL	U_4	t_4	0	3	
PUTS	U_5	t_6	8	3	
UNLFIL	U_6	t_8	6	1	
UPDATE	U_7	t_{10}	4	1	
PUTS	U_8	t_{11}	0	2	

†$t_0 < t_1 < t_2 \ldots < t_{11}$

††To simplify this example, not all completion times are included.

Time	Requests Currently Being Processed			Queues					
	Volume 1	Volume 2	Volume 3	Volume 1	Pri-ority	Volume 2	Pri-ority	Volume 3	Pri-ority
t_0	U_0	--	--	--		--		--	
t_1	U_0	U_1	--	--		--		--	
t_2	U_0	U_1	U_2	--		--		--	
t_3	U_0	U_1	U_2	--		--		U_3	7
t_4	U_0	U_1	U_2	--		--		U_3 U_4	7 0
t_5	--	U_1	U_2	--		--		U_3 U_4	7 0
t_6	--	U_1	U_2	--		--		U_5 U_3 U_4	8 7 0
t_7	--	--	U_2	--		--		U_5 U_3 U_4	8 7 0
t_8	U_6	--	U_2	--		--		U_5 U_3 U_4	8 7 0
t_9	U_6	--	U_5	--		--		U_3 U_4	7 0
t_{10}	U_6	--	U_5	U_7	4	--		U_3 U_4	7 0
t_{11}	U_6	U_8	U_5	U_7	4	--		U_3 U_4	7 0

Figure 1-2. Example of Reentrant Processor Queuing



Request Processor	Serial Request	Request Time	Priority	Volume	Completion Time
CREATE	U_0	t_0†	9	1	t_5
OPENFL	U_1	t_1	13	2	t_7
OPENFL	U_2	t_2	8	3	t_9
OPENFL	U_3	t_3	7	3	††
WRITER	U_4	t_4	0	3	
CLOSFL	U_5	t_6	8	3	
OPENFL	U_6	t_8	6	1	

†$t_0 < t_1 < t_2 \ldots < t_{11}$

††To simplify this example, not all completion times are included.

	Requests Currently Being Processed			Queue		
Time	Volume 1	Volume 2	Volume 3	Request	Volume	Priority
t_0	U_0	--	--	--		
t_1	U_0	--	--	U_1	2	13
t_2	U_0			U_1 U_2	2 3	13 8
t_3	U_0			U_1 U_2 U_3	2 3 3	13 8 7
t_4	U_0			U_1 U_2 U_3 U_4	2 3 3 3	13 8 7 0
t_5	--	U_1	--	U_2 U_3 U_4	3 3 3	8 7 0
t_6	--	U_1	--	U_2 U_5 U_3 U_4	3 3 3 3	8 8 7 0
t_7	--	--	U_2	U_5 U_3 U_4	3 3 3	8 7 0
t_8	--	--	U_2	U_5 U_3 U_6 U_4	3 3 1 3	8 7 6 0
t_9	--	--	U_5	U_3 U_6 U_4	3 1 3	7 6 0

Figure 1-3. Example of Serial Processor Queuing

intervals thereafter. The ordinal MNTCHK is part of ITOS. The functions of MNTCHK are to ascertain that each volume in use is mounted and ready and that the volume name on the label matches the volume name in the file manager main memory table for that volume. If a volume does not pass these tests, it is logically disabled by MNTCHK. (See appendix E for volume label description and appendix D for volume information table description.)

Similarly, the file manager provides the interface for automatic volume disabling in the event of a mass memory input/output error on the volume. When such an I/O error is detected in performing file manager input/output, the file manager checks for the existence of the system ordinal, DISMNT. The ordinal DISMNT is a part of ITOS. Functions of DISMNT are to request closure of all open files on the volume and to print a message on the comment device.

SPECIFICATION REQUESTS

CREATE FILE (CREATE)

A file must be created before it can be opened for storage and retrieval. A file cannot be created if a file with the identical name and owner is already defined on the same volume. A file may be recreated if it was previously deleted or renamed.

A file may be specified to have sector-aligned records. If this option is selected, each record starts at the first physical word of a sector. (In this case, if the record length is not an integral multiple of the sector length, the words between the end of a record and the start of the next sector are unused.) If records are not to be sector-aligned, mass memory space is assigned so that one record follows the next with no space in between. Sector alignment of records minimizes access time when randomly positioned records are being retrieved and updated. However, depending on record length, sector alignment of records may not make efficient use of mass storage space. See figures 2-1 and A-1.

A file may be specified to have essentially binary data. If this option is selected, the control information for the file is updated on mass memory often each set of one or more new records is stored into the file so that the control information always correctly reflects the number of records in the file. Further, a binary data file may not be compressed since binary data may be mistaken for the record deleted code used by the system (see Delete Record and Compress File in section 2 and the FURDEL parameter description in appendix D).

In FORTRAN, the create file request has the following form:

```
CALL CREATE (reqbuf,idata,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager in processing the request (see Request Buffer, section 1).

idata is a 24-word array containing the information needed to create the file. Words within the idata are as follows:

idata(1) - idata(4): File name, eight ASCII characters (name may contain blanks)

idata(5) - idata(8): File owner, eight ASCII characters (name may contain blanks)

idata(9) - idata(12): Name of the volume on which the file is to be defined; eight ASCII characters. The value of idata(9) must not be 2020_{16}.

idata(13): Record length in bytes:

$1 \leq \text{idata(13)} \leq 32{,}766$

If idata(13) is an odd integer, the actual record length includes an odd number of bytes for purposes of transfer to magnetic tape, etc. However, on mass memory the record length is an even number of bytes (the smallest even number of bytes that includes the whole record). The start of a record is always the left-hand byte of a word.

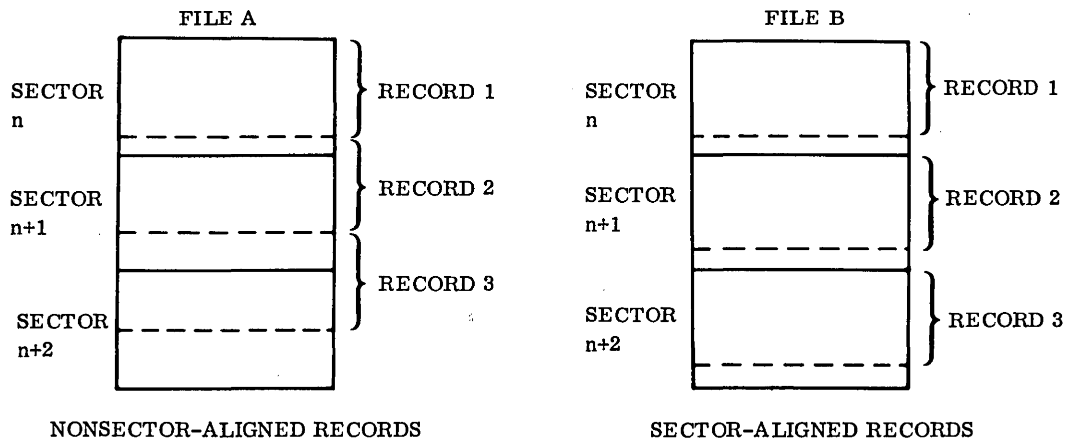
idata(14) - idata(15): Maximum number of records to be stored in file; the format of idata(14) and idata(15) is that of a relative record number as defined in Relative Record Number, section 1.

idata(16): Specifies options selected for file

- bit 0: File type
  - 0 Sequential file
  - 1 Indexed file
- bits 1-7: Unused
- bit 8: Binary data indicator
  - 0 Records do not contain essentially binary data
  - 1 Records contain essentially binary data
- bits 9-13: Unused
- bit 14: Meaningful only if the file is indexed
  - 0 Records are presented randomly with respect to primary key
  - 1 Records are presented in order with respect to primary key
- bit 15: Sector-aligned option
  - 1 All records are sector-aligned
  - 0 Otherwise

idata(17): Length of key 1 (primary key) in bytes:

$1 \leq \text{idata(17)} \leq 29$



NOTE: TO RETRIEVE RECORDS 1, 2, AND 3 SEQUENTIALLY AS A GROUP OF THREE FROM FILE A REQUIRES THE SAME TIME AS TO RETRIEVE RECORDS 1, 2, AND 3 SEQUENTIALLY AS A GROUP FROM FILE B. HOWEVER, TO RETRIEVE RECORD 2, THEN RECORD 3, THEN RECORD 1, EACH RECORD INDIVIDUALLY WOULD REQUIRE ADDITIONAL TIME WHEN ACCESSING RECORDS IN FILE A. TO RETRIEVE A RECORD THAT OVERLAPS TWO OR MORE SECTORS, EACH SECTOR IS RETRIEVED INDIVIDUALLY BY THE MASS MEMORY DRIVER. AFTER EACH RETRIEVAL, THE PART OF THE RETRIEVED SECTOR THAT INTERSECTS THE RECORD IS TRANSFERRED TO THE USER'S BUFFER. IF TWO SECTOR-ALIGNED RECORDS HAVE UNUSED WORDS BETWEEN THEM ON MASS MEMORY, THERE WILL BE THE SAME NUMBER OF UNUSED WORDS BETWEEN THEM IN THE USER'S BUFFER WHEN THESE RECORDS ARE RETRIEVED.

Figure 2-1. Sector-Aligned and Non-Sector-Aligned Records

idata(18) Byte position in key 1. For key starting in left byte:

idata(18) = 2 x n + 1

For key starting in right byte:

idata(18) = 2 x n + 2

Where: n is number words preceding key 1 in the record.

For example, for the primary key starting in the left byte of word 4:

idata(18) = 2 x 3 + 1 = 7

idata(19) Length of key 2 in bytes (same limits as for key 1 idata(17))

idata(20) Byte position of key 2 (computed as for key 1, idata(18))

idata(21) Length of key 3 in bytes (same limits as for key 1, idata(17))

idata(22) Byte position of key 3 (computed as for key 1, idata(18))

idata(23) Length of key 4 in bytes (same limits as for key 1, idata(17))

idata(24) Byte position of key 4 (computed as for key 1, idata(18))

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the create file request has one of the following forms:

```
EXT*   CREATE              EXT    CREATE
 :       :                  :       :
RTJ    CREATE     or       RTJ    CREATE
ADC    reqbuf              ADC    reqbuf
 :       :                  :       :
```

The use of CREATE as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to CREATE if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- The file name/owner string is not unique (bit 10 is also set).
- There is insufficient space in the mass memory file definition directory for this file (bit 11 is also set). If an unused file is present in the system, it may be deleted to provide additional directory space. The deleted file must have the same scatter code as the new file if the new file is to utilize the empty directory entry left by the deleted file (see appendix B).
- Insufficient mass memory file space exists for the file's records (bit 12 is also set). If an unused file exists in the system, it may be deleted to provide additional file space.
- Volume specified for the file is not mounted and ready (bit 13 is also set).
- File request is illegal (bit 14 is also set). This implies one or more of the following has occurred:

  -Record length is not in the required range.

  -Maximum number of records is not in the required range.

  -Length of one or more keys is not in the required range.

  -Position of one or more keys is not totally within the record.

  -Primary key is not specified but key 2, 3, or 4 is specified.

  -No keys are specified, but the indexed file is specified.

An example of a create file request is shown in figure 2-2.

CLEAR FILE (CLEAR)

A file may be cleared when there is no further use for the records currently in the file. Functionally, a clear file request is equivalent to a delete file request followed by a create file request. Executing the clear file request is faster, however, than executing a delete followed by a create request.

A file may not be open to any user when it is cleared. This means that a file cannot be locked when it is being cleared. If a user wishes to periodically dump a file's contents to another medium such as magnetic tape and then delete those dumped records from the file, he must ensure that no records are stored into the file after the dump has started and before the clearing has been completed. If the user does not do this, data may be lost. One way to prevent any data loss is for the user to provide his own protection system. This method is illustrated in the Lock File (LOKFIL) section.

In FORTRAN, the clear file request has the following form:

```
CALL CLEAR (reqbuf,idata,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager in processing the request (see Request Buffer, section 1).

idata is a 12-word array containing the information needed to define the file to be cleared. Words within idata are as follows:

idata(1) - idata(4)	File name (eight ASCII characters)
idata(5) - idata(8)	File owner name (eight ASCII characters)
idata(9) - idata(12)	Name of the volume on which the file is defined (eight ASCII characters); if idata(9) = 0 or idata(9) = 2020_{16} (two ASCII blanks), the file manager performs a search to locate the entry for the specified file and returns the corresponding ASCII volume name in idata(9) through idata(12).

istat is the file request status word as defined in Status Indicator Word, section 1 (also see error considerations below).

In assembly language, the clear file request has one of the following forms:

```
EXT*   CLEAR              EXT    CLEAR
 :       :                 :       :
RTJ    CLEAR      or      RTJ    CLEAR
ADC    reqbuf             ADC    reqbuf
 :       :                 :       :
```

The use of CLEAR as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to CLEAR if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is currently open to one or more users (bit 0 is also set).
- The file could not be located (bit 1 is also set).
- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bit 5 and bit 14 are also set).
- The volume specified for the file is not mounted and ready (bit 13 is also set).

```
C RESERVE SPACE FOR FILE REQUEST BUFFER AND FILE INFORMATION BUFFER
      INTEGER REQBUF(24)
      DIMENSION IDATA(24)
C SET FILE NAME = F-12
C SET FILE OWNER = ED SMITH, VOLUME NAME = V1
C SET RECORD LENGTH = 48 BYTES (=24 WORDS)
C     IDATA(13)=48
C SET MAXIMUM NUMBER RECORDS TO 500.
C     IDATA(14)=0
C     IDATA(15)=500
C SET FILE TYPE INDEXED WITH RECORDS TO BE PRESENTED AT RANDOM, RECORDS
C NOT SECTOR-ALIGNED.
C     IDATA(16)=$0001
C PRIMARY KEY TO BE STORED IN BYTES 5-7.
C     IDATA(17)=3
C     IDATA(18)=5
C SECONDARY KEY TO BE STORED IN BYTE 8
C     IDATA(19)=1
C     IDATA(20)=8
      DATA IDATA/"F-12    ED SMITH","V1      ",48,0,500,$0001,3,5,1,8,
     1     4*0/
       .            .
       .            .
       .            .
      CALL CREATE (REQBUF,IDATA,ISTAT)
C CHECK FOR ERRORS
      IF (ISTAT.NE.0) GO TO 9000
```

Figure 2-2. Create File Request Example (FORTRAN)

DELETE FILE (DELETE)

A file may be deleted when there is no further use for it. Deletion of a file permits reuse of its mass memory record storage space and reuse of its directory entry by the file manager. If the file is indexed, its index storage space on mass memory is also freed for use by the file manager.

A file may be deleted only if it is currently not open to any user.

In FORTRAN, the delete file request has the following form:

CALL DELETE (reqbuf,idata,istat)

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager to process the request (see Request Buffer, section 1).

idata is a 12-word array containing the information needed to define the file to be deleted. Words within idata are as follows:

idata(1) - idata(4)	File name (eight ASCII characters)
idata(5) - idata(8)	File owner name (eight ASCII characters)
idata(9) - idata(12)	Name of the volume on which the file is defined (eight ASCII characters); if idata(9) = 0 or idata(9) = 2020_{16} (two ASCII blanks), the file manager performs a search to locate the entry for the specified file and returns the corresponding ASCII volume name in idata(9) through idata(12).

istat is the file request status word as defined in Status Indicator Word, section 1 (also see error considerations below).

In assembly language, the delete file request has one of the following forms:

```
EXT*   DELETE              EXT    DELETE
 :       :                  :       :
RTJ    DELETE      or      RTJ    DELETE
ADC    reqbuf              ADC    reqbuf
 :       :                  :       :
```

The use of DELETE as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to DELETE if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is currently open to one or more users (bit 0 is also set).
- The file could not be located (bit 1 is also set).
- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- The volume is specified for the file that is not mounted and ready (bit 13 is also set).

OPEN FILE (OPENFL)

To gain access to a given file, the user must execute a file manager open file request, specifying the particular file. More than one user may access a given file at one time.

In order to open a file, the following conditions must hold:

- The file must have been created.
- The volume containing the file must be mounted and ready.
- The file must not be locked.
- The file must not be open to this user.

If records are to be retrieved from an indexed file by a given key, that key must be specified in the open request.

Access permission is obtained for either file compression, special file processing or record retrieval and storage. When the file manager grants permission for file compression or special processing, it automatically locks the file. This locking ensures that other users cannot retrieve or store records for this file during the compression permission for record retrieval and storage, the user has the following options with regard to locking:

- The entire file may be automatically locked at the time the access permit is granted. When a file is locked by a user, no other user can access the file.
- A record is automatically locked at the time the user retrieves it from the file. (In this case the record is automatically unlocked when the next file access request is made.)
- No automatic locking is to be performed. If the user selects this option, he may still perform file locking, but must do it by one or more requests distinct from the open file request.

No record locking can be performed if the third option is selected. When opening a file for record retrieval, the file request specifies the number of records to be retrieved in each retrieve record request made to the file. This number of records applies to all GETS requests and to all READR requests except for those READR requests that specify a particular key of an indexed file. If record locking is to be performed for an indexed file that is to be accessed by a key, the number of records per retrieval must be one. If a file is open to more than one user, the number of records per retrieval and the locking option need not be the same for the different users.

In compressing a file, a compress file request is repeatedly executed until the entire file has been compressed. By compressing only a portion of a file on each request, other file manager requests may be queued and executed between successive compression calls (see Reentrant Request Processors; Serial Request Processors, section 1). When opening a file for compression, the open file request specifies the number of file records to be processed in each compress file request execution. This number should be chosen according to the amount of buffer space available in the user's program (see the Compress File Request (COMFIL), section). If an indexed file is to be compressed, the number of records to be processed in each compression execution is one.

If a file is open for special processing, it may be accessed the same as if it had been opened for record retrieval and storage in which records are to be retrieved by relative record number (see idata (13) description below). While in this processing mode, the file's control information on mass memory is not periodically updated as new records are stored into the file. Further, the control information is not updated on mass memory if the file is closed by system executive close of all open files for a particular user; however, the control information is updated if the file is closed by the user that opened the file. This special processing mode is intended for use by system file manager utility programs.

In FORTRAN, the open file request has the following form:

```
CALL OPENFL (reqbuf,idata,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager to process the request; it must be set to all binary zeroes by the caller. An exception to this is described in appendix L.

After being initialized by the file manager, this buffer is used by the file manager for subsequent accesses for this file.

idata is a 15-word array containing the information needed to define the file to be opened. Words within idata are as follows:

idata(1) - idata(4)	File name (eight ASCII characters)
idata(5) - idata(8)	File owner name (eight ASCII characters)
idata(9) - idata(12)	Name of volume on which the file is defined (eight ASCII characters); if idata(9) = 0 or idata(9) = 2020_{16} (two ASCII blanks), the file manager performs a search to locate the entry for the specified file and returns the corresponding ASCII volume name in idata(9) through idata(12).
idata(13)	Access indicator

0	Access for record retrieval and storage requested when records are to be retrieved by relative record number
1, 2, 3, or 4	Access for record retrieval and storage requested where records are to be accessed by key 1, key 2, key 3, or key 4, respectively.
-1	Access for file compression.
-2	Access for special processing

idata(14) Number of records to be retrieved in each retrieve record request. (If records for a specific key value are to be retrieved from an indexed file and record locking is indicated, only one record may be retrieved per retrieval request; that is, idata(14) must equal 1.)

Or the number records to be processed during each execution of a compress file request (must be 1 if compressing an indexed file).

idata(15) Lock indicator

0 No automatic locking is to be performed

> 0 A record is to be automatically locked at the time it is retrieved

< 0 The entire file is to be automatically locked at the time the access permit is granted

NOTE

The value of idata(15) is ignored if the file is opened for compression or for special processing.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also status considerations below).

In assembly language, the open file request has one of the following forms:

```
EXT*   OPENFL              EXT    OPENFL
 :       :                  :       :
RTJ    OPENFL      or      RTJ    OPENFL
ADC    reqbuf              ADC    reqbuf
 :       :                  :       :
```

The use of OPENFL as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

Request status considerations (istat) are as follows:

- OPENFL returns bit 0 = 1 and bit 15 = 0 if the file is currently open to another user.
- OPENFL returns bit 2 = 1 and bit 15 = 0 if the file was locked as a part of the open file request.
- OPENFL sets bit 15 (rejecting the request) if:

  The file request was to open for compression and some other user currently has the file open (bit 0 is also set).†

  -The file request was to open for record access with file lock and some other user currently has the file open (bit 0 is also set).†

  -The file is already open to this user (bit-0 is also set).

  -The file could not be located (bit 1 is also set).

  -The file was locked at the time the request was made (bit 2 is also set).†

  -A mass memory error occurred (bit 5 is also set). This error may have occurred when the file was previously open and record recovery was not possible because of the timing of the failure. (See Record Recovery Following System Failure, section 1.) If the bit 5 error indication is not accompanied by a MASS MEMORY I/O message on the comment device, the error occurred when the file was previously open. In this case it may be possible to manually restore the file on mass memory by use of ODEBUG. Otherwise, it is necessary to delete and recreate the file.

  -File manager data structures on mass memory or in main memory contain one or more errors (bit 5 and bit 14 are set).

  -The request is for record access, not compression. When the file was last closed, a compression had been initiated but not completed. This compression must be completed before the file can be opened for record access (bit 9 is also set). The user should open the file for compression and finish up compression before proceeding.

  -The maximum number of concurrent open files permitted to a single user has already been granted to this user (bit 11 is also set).

  -The maximum number of open file permits that can be granted to all users in the system has been obtained (bit 12 is also set). In this case, if bit 11 is zero, the request may be retried after a delay.

  -The volume specified for the file is not mounted and ready (bit 13 is also set).

  -The request is illegal (bit 14 is also set). This implies one or more of the following has occurred:

    The value of idata(14) is invalid.

    The indexed file and idata(13) exceed the number of keys for the file.

    The definition of idata(13), idata(14), idata(15) is inconsistent.

    The number of records to be accessed (idata(14)) multiplied by the record length is greater than 32,767.

An example of an open file request is shown in figure 2-3.

†The request may be repeated after a delay. (See the note in Update Protection, section 1.)

CLOSE FILE (CLOSFL)

A close file request is the procedure used to relinquish a user's access permission for the file. A file should be closed when a user no longer needs to access it. It is important that this be done so that other users can open files as needed. (The maximum number of simultaneous open files is discussed in Main-Memory-Resident File Manager Operation Installation Parameters, appendix D.)

A file may be closed only if it is currently open to the requestor of the close. A file closed to one user may, at the same time, be open to other users.

If the file has been locked by the user, it is automatically unlocked at the time of the close. If a set of records within the file has been locked by this user, these records are automatically unlocked at the time of the close. (Records locked by another user are not unlocked.) An indication of any file or record unlocking performed during a close file request is relayed to the user via the status indicator word.

In FORTRAN, the close file request has the following form:

```
CALL CLOSFL(reqbuf,istat)
```

```
      DIMENSION NBUF(24),JBUF(24)
      DIMENSION NDATA(15),JDATA(15)
C PRESET REQUEST BUFFERS TO ALL ZEROES
      DATA NBUF,JBUF /48*0/
C PRESET REQUEST INFORMATION BUFFERS
C FOR FILE N.1 RECORD IS TO BE RETRIEVED AT A TIME ACCORDING TO THE
C PRIMARY KEY. EACH RECORD WILL BE LOCKED AS IT IS RETRIEVED.
      DATA NDATA /"FILE N  ZQX389  ","VOLUME 1",1,1,1/
C FOR FILE J.3 RECORDS ARE TO BE RETRIEVED PER RETRIEVAL REQUEST.
C RECORD LOCKING IS TO BE PERFORMED.  RETRIEVAL IS TO BE BY RELATIVE
C RECORD NUMBER.
      DATA JDATA /"FILE J  ZQX389  ","VOLUME 1",0,3,1/
C REQUEST PERMISSION TO ACCESS FILE N
      CALL OPENFL (NBUF,NDATA,NSTAT)
C TEST FOR REJECT
      IF (NSTAT.LT.0) GO TO 8000
C REQUEST PERMISSION TO ACCESS FILE J.
      CALL OPENFL (JBUF,JDATA,JSTAT)
C TEST FOR REJECT
      IF (JSTAT.LT.0) GO TO 8010
C NOTE- SUBSEQUENT REQUESTS WHICH ACCESS FILE N MUST USE NBUF AS THE
C       REQUEST BUFFER.  REQUESTS WHICH ACCESS FILE J MUST USE JBUF AS
C       THE REQUEST BUFFER.
      .                          .
      .                          .
      .                          .
```

Figure 2-3. Open File Request Example (FORTRAN)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also status considerations below).

In assembly language, the close file request has one of the following forms:

```
EXT*   CLOSFL            EXT    CLOSFL
 :      :                 :      :
RTJ    CLOSFL     or     RTJ    CLOSFL
ADC    reqbuf            ADC    reqbuf
 :      :                 :      :
```

The use of CLOSFL as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

Request status considerations (istat) are as follows:

- Bit 2 is set if the file was unlocked by the close file request.
- Bit 3 is set if a set of locked records was unlocked by the close file request. (These records were initially locked by requestor of the close.)

CLOSFL sets bit 15 (rejecting the request) if:

- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- The file request buffer (reqbuf) was altered by the user before the close file request (bit 13 is also set).

LOCK FILE (LOKFIL)

In some situations, the user may wish to lock an entire file to temporarily prevent access to the file by other users. Some situations that warrant file locking are listed in Update Protection, section 1, in the discussion of file locks.

A file may be locked under the following conditions:

- The user has opened the file.
- The file is not open to any other user.
- The file was not opened for record locking.

In FORTRAN, the lock file request has the following form:

CALL LOKFIL (reqbuf,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the lock file request has one of the following forms:

```
EXT*  LOKFIL          EXT   LOKFIL
 :      :              :      :
RTJ   LOKFIL    or    RTJ   LOKFIL
ADC   reqbuf          ADC   reqbuf
 :      :              :      :
```

The use of LOKFIL as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to LOKFIL if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is currently open to another user (bit 0 is also set).
- Record locking was indicated when the file was opened (bit 3 is also set).
- The file request buffer (reqbuf) was altered by the user before the lock file request (bit 13 is also set).
- The file was closed by an executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

A lock file request is shown in figure 2-4.

UNLOCK FILE (UNLFIL)

A file may be unlocked when the user no longer needs to have it locked. This enables other users to regain access to the file. A file may be unlocked by the same user who locked the file.

In FORTRAN, the unlock file request has the following form:

CALL UNLFIL (reqbuf,istat)

Where:

reqbuf is the file request buffer and is a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the unlock file request has one of the following forms:

```
EXT*  UNLFIL          EXT   UNLFIL
 :      :              :      :
RTJ   UNLFIL    or    RTJ   UNLFIL
ADC   reqbuf          ADC   reqbuf
 :      :              :      :
```

The use of UNLFIL as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to UNLFIL if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is not currently locked by the user (bit 2 is also set).
- The file request buffer (reqbuf) was altered by the user before the unlock file request (bit 13 is also set).
- The file was closed by an executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

An unlock file request is included in the example in figure 2-4.

GET FILE CONTROL BLOCK (GETFCB)

A file control block (FCB) is an array associated with a file. It is stored on mass memory. The array contains information such as file name, file owner, file type, location of data records, and so forth. The format of the FCB is shown in File Control Block Table, appendix B.

The get file control block request is intended for use only by system level utility programs. There are two ways to specify the file for which the FCB is to be retrieved:

- An FCB may be retrieved for a particular open file. In this case, the file is defined by the contents of the file request buffer, reqbuf, corresponding to the open file request.
- An FCB may be retrieved according to its relative position in the set of FCBs on a particular volume. This relative position is the FCB index number in the file definition directory as defined in File Control Block Table, appendix B. For this method of FCB retrieval, the file need not be open.

Using the second method of FCB specification, all the FCBs on a given volume may be retrieved by successive repetitions of a get FCB request. The first request specifies that the FCB index equals 1. Before each repetition of the request, the FCB index is incremented by one, until the file manager rejects the request due to an out-of-range FCB index for that volume.

In FORTRAN, the retrieve file control block request has the following form:

CALL GETFCB (reqbuf,volnam,index,fcbbfr,istat)

Where:

reqbuf is the file request buffer, a 24-word array used to process the request. If retrieval is requested for a particular open file, reqbuf must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

volnam is an array of four words containing volume name (eight ASCII characters). If the first word of the array volnam is zero, the FCB to be retrieved is for the open file defined by the file request buffer, reqbuf.

```
C THIS CODE INCLUDES A USER-DEVISED DATA PROTECTION SYSTEM
C TO PREVENT LOST DATA BETWEEN FILE DUMP TO TAPE AND FILE CLEARING.
C THIS EXAMPLE INCLUDES LOKFIL, UNLFIL, AND CLEAR REQUESTS.
      DIMENSION IBUF(24),IDATA(24)
       DATA IBUF/24*0/
       .                      .
       .                      .
       .                      .
      DIMENSION ITEMP(4)
      DATA IFLAG /$0021/, ITIME/1/
C DETERMINE LOCATION OF STATEMENT 200. STORE INTO ICOMP FOR LATER USE.
      ASSIGN 200 TO ICOMP
      CALL OPENFL (IBUF,IDATA,ISTAT)
      IF (ISTAT.LT.0) GO TO 9700
C TEST TO SEE IF TIME FOR TAPE DUMP
        .                     .
        .                     .
        .                     .
C LOCK FILE IN PREPARATION FOR DUMPING TO MAG TAPE
      CALL LOKFIL (IBUF,ISTAT)
C TEST FOR REJECT
      IF (ISTAT.LT.0) GO TO 9800
C DUMP RECORDS TO MAG TAPE
       .                       .
       .                       .
       .                       .
C AFTER DUMPING TO MAG TAPE, RECORDS NOW ON MAG TAPE MAY BE CLEARED
C FROM FILE.  FILE IS NOT OPEN TO ANY OTHER USER AT THIS TIME
C BECAUSE FILE IS LOCKED.
C SET WORD $47 OF MAIN MEMORY TO PREVENT STORAGE OF RECORDS DURING THE
C DUMP AND CLEAR OPERATION.  ANY PROGRAM  STORING RECORDS INTO THE
C FILE SHOULD CHECK WORD $47. A RECORD SHOULD BE STORED ONLY WHEN
C WORD $47 IS ZERO.  THIS AVOIDS LOSING A RECORD STORED BETWEEN THE
C MAG TAPE DUMP AND THE CLEAR.
      ASSEM $0A01,$B047
C UNLOCK FILE TO ALLOW CLEAR FILE REQUEST TO EXECUTE.
      CALL UNLFIL (IBUF,ISTAT)
      IF (ISTAT.LT.0) GO TO 9810
C CLOSE FILE TO PERMIT CLEAR REQUEST TO EXECUTE
      CALL CLOSFL (IBUF,ISTAT)
      IF ISTAT.LT.0)  GO TO 9820
  200 CALL CLEAR (IBUF,IDATA,ISTAT)
C TEST FOR REJECT
      IF (ISTAT.LT.0) GO TO 9850
C CLEAR WORD $47 TO ALLOW RECORD STORAGE INTO FILE.
      ASSEM $0A00,$B047
       .                       .
       .                       .
       .                       .
C WAS REQUEST REJECTED BECAUSE FILE IS OPEN TO ANOTHER USER
 9850 IF (AND(ISTAT,1).EQ.0) GO TO 9910
C YES, FILE WAS OPEN TO ANOTHER USER.
C DELAY 1 SECOND AND TRY AGAIN
C STATEMENT 200 IS COMPLETION LOCATION FOR TIMER REQUEST.
      CALL TIMER (ICOMP,IFLAG,ITIME,ITEMP)
      CALL DISPAT
       .                       .
       .                       .
       .                       .
```

Figure 2-4. Lock, Unlock File Requests Example (FORTRAN)

index — is used only if the first word of the array volnam is nonzero. In this case, index is the relative position of the FCB on the volume specified, and the value of index must be such that 1 < index < 2047. (The position of the FCB index in the file definition directory is given in File Definition Directory, appendix B.)

fcbbfr — is the file control block buffer to receive the block to be retrieved. This buffer must be 96 words in length.

istat — is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the get file control block request has one of the following forms:

```
EXT*   GETFCB           EXT    GETFCB
 :       :               :       :
RTJ    GETFCB     or    RTJ    GETFCB
ADC    reqbuf           ADC    reqbuf
 :       :               :       :
```

The use of GETFCB as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return of a call to GETFCB if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are set).
- The FCB index is out of range for the specified volume (bit 12 is also set).
- The first word of array volnam is nonzero and the volume specified is not mounted and ready (bit 13 is also set).
- The first word of array volnam is zero and the file request buffer (reqbuf) was altered by the user before the retrieve FCB request (bit 13 is also set).
- The first word of array volnam is zero and the file was closed by an executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).
- The FCB index is not a positive integer (bit 14 is also set).

Examples of get file control block requests are shown in figure 2-5 and figure 2-6.

UPDATE FILE CONTROL BLOCK (UPDFCB)

This request may be used in conjunction with the get file control block request as described in the Get File Control Block (GETFCB) section; however, it is not necessary to retrieve a file control block (FCB) before updating it. This request is intended for use only by system level utility programs. The portion of an FCB not used by the file manager may be updated by the use of this request. The portion that may be updated consists of words 38 through 96 of the FCB. (The format of the FCB is contained in File Control Block Table, appendix B.) In this portion of the FCB a utility program may store file creation date, retention period, file description, and so forth.

The FCB to be updated is specified in the same manner that the FCB to be retrieved by a GETFCB request is specified (see the Get File Control Block (GETFCB) section).

In FORTRAN, the update file control block request has the following form:

```
CALL UPDFCB (reqbuf,volnam,index,fcbbfr,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array. If the FCB to be updated corresponds to a particular open file, this must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

volnam is an array of four words containing the volume name (eight ASCII characters). If the first word of the array volnam is zero, the FCB to be updated is for the open file defined by the file request buffer, reqbuf.

index is ignored if the first word of the array volnam is all binary zeroes. If the first word of the array volnam is nonzero, index is the relative position of the FCB on the volume specified, and the value of index must be such that $1 \leq \text{index} \leq 2047$. (The position of the FCB index in the file definition directory is given in File Definition Directory, appendix B.)

fcbbfr is the file control block buffer. This buffer must be 96 words in length. Words 38 through 96 are written to the FCB on mass memory.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the update file control block request has one of the following forms:

```
EXT*   UPDFCB           EXT    UPDFCB
 :       :               :       :
RTJ    UPDFCB     or    RTJ    UPDFCB
ADC    reqbuf           ADC    reqbuf
 :       :               :       :
```

The use of UPDFCB as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return of a call to UPDFCB if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- The FCB index is out of range for the volume specified (bit 12 is also set).
- The first word of array volnam is zero and the file request buffer (reqbuf) was altered by the user before the UPDFCB request (bit 13 is also set).
- The first word of array volnam is nonzero and the volume specified is not mounted and ready (bit 13 is also set).
- The first word of array volnam is zero and the file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

```
        .     .
        .     .
        .     .
      INTEGER DAYBUF(24),DAYDAT(15),DSTAT,VOL(4),DAYFCB(96)
      DATA DAYBUF /24*0/, DAYDAT/"DAYFILE UTILUSER","SYSVOL  ",0,6,-1/
      DATA VOL(1)/0/
C REQUEST ACCESS TO DAY FILE.  REQUEST SPECIFIES RECORD ACCESS DESIRED
C AND FILE LOCK DESIRED AT TIME OF GAINING ACCESS PERMIT
      CALL OPENFL(DAYBUF,DAYDAT,DSTAT)
      IF (DSTAT.LT.0) GO TO 8000
        .                     .
        .                     .
        .                     .
C RETRIEVE FCB FOR DAY FILE.  VOL(1)=0.  FCB WILL BE STORED IN DAYFCB
C ARRAY.  IDUM WILL BE IGNORED BY FILE MANAGER SINCE VOL(1)=0
      CALL GETFCB (DAYBUF,VOL,IDUM,DAYFCB,IGSTAT)
      IF (IGSTAT.LT.0) GO TO 8020
        .                     .
        .                     .
        .                     .
```

Figure 2-5. Example of FCB Retrieval for a Particular Open File (FORTRAN)

```
C PURPOSE OF THIS CODE IS TO RETRIEVE EACH FCB ON A VOLUME,
C PROCESS IT, AND UPDATE THE FCB IF NECESSARY.
C THIS PROGRAM RUNS IN ALLOCATABLE MAIN MEMORY.
C THIS PROGRAM LINKS TO THE NON-REENTRANT INTERCEPTOR FMENTP.
C A BINARY COPY OF FMENTP MUST BE INCLUDED AT THE TIME THIS PROGRAM IS
C LOADED.  THE FOLLOWING RELATIVE EXTERNALS ARE DECLARED TO ALLOW
C PROPER LINKING TO FMENTP.
      RELATIVE GETFCB, UPDFCB
      INTEGER ABUF(24),VOL(4),AFCB(96)
      DATA   VOL /"PACK ONE"/
C INITIALIZE  FCB INDEX
      INDFCB=1
C RETRIEVE FCB FOR FILE WITH FCB INDEX = INDFCB
   15 CALL GETFCB (ABUF,VOL,INDFCB,AFCB,ISTAT)
C WAS FCB INDEX OUT OF RANGE
      IF (AND(ISTAT,$1000).NE.0) GO TO 9800
C TEST FOR REJECT DUE TO ERROR OTHER THAN FCB INDEX RANGE
      IF (ISTAT.LT.0) GO TO 9600
C FCB INDEX CORRESPONDS TO A CREATED FILE
C HAS THIS FILE BEEN DELETED
C IF FIRST WORD OF NAME-OWNER STRING IS ZERO, FILE HAS BEEN DELETED.
      IF (AFCB(25).EQ.0) GO TO 250
C TEST AFCB(50) TO SEE IF FILE CONTAINS ANY MESSAGE TO BE PRINTED ON
C COMMENT DEVICE
  150 IF (AFCB(50).EQ.0) GO TO 200
C GO PRINT MESSAGES
        .                     .
        .                     .
        .                     .
C RE-SET MESSAGE COUNTER FOR COMMENT DEVICE
      AFCB(50)=0
      CALL UPDFCB (ABUF,VOL,INDFCB,AFCB,ISTAT)
      IF (ISTAT.LT.0) GO TO 9700
  200 CONTINUE
C INCREMENT INDFCB
  250 INDFCB=INDFCB+1
C RETRIEVE NEXT FCB
      GO TO 15
C PROCESS COMPLETE
 9800 CONTINUE
        .                     .
        .                     .
        .                     .
```

Figure 2-6. Example of GETFCB,UPDFCB Requests with FCB Specified by FCB Index (FORTRAN)

- The FCB index is not a positive integer (bit 14 is also set).

An example of the UPDFCB request is included in figure 2-6.

REDUCE FILE SPACE (REDUCE)

A previously created file can be modified to contain fewer records with the reduce file request. The file must meet the following four requirements:

- The file must be closed.
- The file must be sequential.
- The new number of records must be less than the number the file was orginally defined to contain when the file was created.
- The new number of records must be greater than or equal to the number of currently-existing records in the file.

In FORTRAN, the reduce file request has the following form:

CALL REDUCE (reqbuf,idata,istat)

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager to process the request

idata is a 14-word array containing the information needed to define the file to be modified. Words within idata are as follows:

idata(1) - idata(4)	File name (eight ASCII characters)
idata(5) - idata(8)	File owner name (eight ASCII characters)
idata (9) - idata(12)	Name of the volume on which the file is defined (eight ASCII characters); if idata(9) = 0 or idata(9) = 2020_{16} (two ASCII blanks), the file manager performs a search to locate the entry for the specified file and returns the corresponding ASCII volume name in idata(9) through idata(12).

istat is the file request status work as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the reduce file request has one of the following forms:

```
EXT*  REDUCE          EXT   REDUCE
 :      :       or     :      :
RTJ   REDUCE          EXT   REDUCE
ADC   reqbuf          ADC   reqbuf
```

The use of REDUCE as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to REDUCE if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is currently open to one or more users (bit 0 is also set).
- The file could not be located (bit 1 is also set).
- The file is not a sequential file (bit 2 is also set).
- The new number of records is greater than specified for the file, is zero, is negative, or is less than the number of records currently existing in the file (bit 3 is also set).
- A mass memory error occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bit 5 and bit 14 are also set).
- The volume specified for the file is not mounted and ready (bit 13 is also set).

RENAME FILE (RENAME)

The rename file request permits the concatenated file name/file owner string for an existing file to be changed. A file must be closed to all users at the time it is being renamed. The new name/owner string must be unique for the volume on which the file resides.

In FORTRAN, the rename file request has the following form:

CALL RENAME (reqbuf,idata,newnam,istat)

Where:

reqbuf is the file request buffer, a 24-word array used by the file manager to process the request.

idata is a 12-word array containing the information needed to define the file to be renamed. Words within idata are as follows:

idata(1) - idata(4)	Current file name (eight ASCII characters)
idata(5) - idata(8)	Current file owner name (eight ASCII characters)
idata(9) - idata(12)	Name of volume on which file is defined (eight ASCII characters). If idata(9) = 0 or idata(9) = 2020_{16} (two ASCII blanks), the file manager searches a directory to locate the entry for the specified file and returns the corresponding ASCII volume name in idata(9) through idata(12).

newnam is an eight-word array specifying the new file name/owner string. Words within newnam are as follows:

newnam(1) through newnam(4)	New file name (eight ASCII characters)
newnam(5) through newnam(8)	New file owner name (eight ASCII characters)

istat is a file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the rename file request has one of the following forms:

```
EXT*   RENAME            EXT   RENAME
 :       :                :      :
RTJ    RENAME     or     RTJ   RENAME
ADC    reqbuf            ADC   reqbuf
 :       :                :      :
```

The use of RENAME as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to RENAME if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The file is currently open to one or more users (bit 0 is also set).
- The file could not be located (bit 1 is also set).
- A mass memory error occurred (bit 5 is also set).
- File manager data structures in main memory or on mass memory contain one or more errors (bits 5 and 14 are also set).
- The new file name/owner string is not unique (bit 10 is also set).
- Insufficient file definition directory (FDD) space exists (bit 11 is also set). (Renaming the file does make available the directory entry space previously used for this file, but the new name/owner string may not hash into that space. Refer to File Definition Directory, appendix B, for further information on the directory.)
- The volume for this file is not mounted and ready (bit 13 is also set).

ENABLE/DISABLE VOLUME (VOLUSE)

This request is intended for use only by system utility programs. Its functions are to enable or disable use of a volume as described in Volume Labeling, Enabling, Disabling, section 1.

In FORTRAN, the enable/disable volume request has the following form:

```
CALL VOLUSE (reqbuf,volnam,vlunit,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array.

volnam is a four-word array containing either of the following:

For enabling:	volume name (eight ASCII characters).
For disabling:	binary zeroes in volnam (1); other words of the array, arbitrary values.

vlunit is the file manager unit number of the drive containing the volume; this is not the same as the system logical unit number.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the enable/disable volume request has one of the following forms:

```
EXT*   VOLUSE            EXT   VOLUSE
 :       :                :      :
RTJ    VOLUSE     or     RTJ   VOLUSE
ADC    reqbuf            ADC   reqbuf
 :       :                :      :
```

The use of VOLUSE as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to VOLUSE if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- The first word of the array volnam is nonzero and the volnam array does not match the name on the volume label (bit 1 is also set).
- A mass memory error has occurred (bit 5 is also set).
- The first word of the array volnam is nonzero and the drive specified by vlunit already has a volume enabled (bit 13 is also set).

RECORD ACCESSING REQUESTS

A brief summary of the request calls and parameter lists is to be found in appendix I. A chart of status bits and their meanings is contained in appendix F.

The number of records accessed by the individual requests varies. For some it is a constant. For others it is a user-supplied number, sometimes supplied with the record accessing request itself, and sometimes supplied by the user when the file is opened. A summary of the number of records accessed by the various requests is shown in figure I-4.

STORE NEW RECORDS SEQUENTIALLY (PUTS)

The PUTS request stores one or more records into a file immediately following any existing file records. The PUTS request may be used for an existing sequential file that is open to the user. PUTS may not be used for an indexed file. The number of records to be stored is specified in the request. (This number is independent of the number of records per retrieval as specified in the OPENFL request.) For each group of records stored, the relative record number of the first record of the group is passed back to the caller. This relative record number may be subsequently used in a READR request for record retrieval.

In FORTRAN, the store new record sequentially request has the following form:

```
CALL PUTS (reqbuf,recbuf,numrec,istat)
```

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user. On completion of the request, words 15, 16, and 17 of reqbuf are defined as follows:

reqbuf(15)	Number of records actually stored by the file manager.
reqbuf(16), reqbuf(17)	Relative record number of the first record stored (as defined in Relative Record Number, section 1).

recbuf is the record buffer containing records to be stored by this request plus two words following the last record to be stored. These final two words are used by the file manager.

The length of recbuf is determined as follows. Let

b = Length of recbuf

n = Number of records to be stored

r = Record length in words

s = Number of words per section for volume containing this file

Then, if records in the file are not sector-aligned:

$$b = n \times r + 2$$

If records are sector-aligned:

$$b = n \times \left\lceil \frac{r}{s} \right\rceil \times s + 2$$

Where: $\lceil y \rceil$ = the least integer greater than or equal to y.

For example, if n = 3, r = 189, and s = 96, and the records are sector-aligned:

$$b = 3 \times \left\lceil \frac{189}{96} \right\rceil \times 96 + 2$$

$$= 3 \times 2 \times 96 + 2$$

$$= 578$$

NOTE

If the records are sector-aligned and the record length is not an integral multiple of sector length, the record buffer must contain unused words between records corresponding to unused words on mass memory. For example, suppose a file has sector-aligned record of 93 words and the sector length is 96 words. If 3 records are to be stored, the first record is transferred from words 1 through 93 of the record buffer; record 2 is transferred from words 97 through 189; record 3 is transferred from 193 through 285. Words 94 through 96, 190 through 192, and 286 through 288 of the record buffer are unused.

numrec is the number of records to be stored by this request.

istat is the file request status word as defined in Status Indicator Word, section 1 (also see status considerations below).

In assembly language, the store new records sequentially request has one of the following forms:

```
EXT*   PUTS              EXT    PUTS
 :      :                 :      :
RTJ    PUTS       or     RTJ    PUTS
ADC    reqbuf            ADC    reqbuf
 :      :                 :      :
```

The use of PUTS as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

Status considerations (istat) are as follows:

- The file is currently locked by the user (bit 2 is set).
- Insufficient room exists in file to store all numrec records (bit 12 is set). (See reqbuf(15).)

PUTS sets bit 15 (rejecting the request) if:

- A mass memory error occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- Insufficient room exists in the file to store any records (bit 12 is set).
- The file request buffer, reqbuf, was altered by the user before the PUTS request (bit 13 is also set).
- The file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

An example of the PUTS request is shown in figure 2-7.

```
C PURPOSE OF THIS CODE IS TO STORE A NEW RECORD INTO A SEQUENTIAL FILE.
      DIMENSION IBUF(24),IDAT(15),INP(34)
      DATA IBUF/24*0/
C USER RECORDS ARE 30 WORDS LONG, NOT SECTOR-ALIGNED.  RESERVE 32
C WORDS FOR RECORD BUFFER.
      INTEGER USEREC(32)
      COMMON INP
       .                           .
       .                           .
       .                           .
      CALL OPENFL(IBUF,IDAT,ISTAT)
      IF (ISTAT.LT.0) GO TO 9500
       .                           .
       .                           .
       .                           .
C TRANSFER NEW USER"S IDENTIFICATION, STATUS CODE, AND BILLING CODE
C FROM INPUT BUFFER TO FILE RECORD BUFFER.
      DO 100 I=1,30
  100 USEREC(I)= INP(I+4)
C STORE NEW USER RECORD INTO FILE
      CALL PUTS(IBUF,USEREC,1,ISTAT)
C TEST FOR REJECT
      IF (ISTAT.LT.0) GO TO 9000
       .                           .
       .                           .
       .                           .
```

Figure 2-7. Store New Records Sequentially Example (FORTRAN)

STORE NEW INDEXED RECORD (WRITER)

This request is similar to the PUTS request described in the Store New Records Sequentially (PUTS) section with the following important differences:

- WRITER stores a new record into an indexed file.
- In a WRITER request, in addition to record storage, the file manager updates the file key structure with the key value(s) associated with the new record.
- Only one new record may be added to a file with a given WRITER request.

To execute a WRITER request, the file to be accessed must be open to the user. Each new record added to an indexed file is stored sequentially; that is, the new record is stored immediately following any existing file records.

In specifying the value of the new record's primary key, the user must specify a value distinct from all other primary key values previously specified for the file. A nonunique primary key value causes rejection of the request.

On completion of the request, the relative record number of the new record is passed back to the caller.

In FORTRAN, the write new indexed record request has the following form:

CALL WRITER (reqbuf,recbuf,keyval,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user.

On completion of the request, reqbuf(16),reqbuf(17) is the relative record number of the record stored (as defined in Relative Record Number, section 1).

recbuf is the record buffer containing the record to be stored by this request plus two words at the end of the buffer that are used by the file manager. The length of recbuf is determined as in the Store New Records Sequentially (PUTS) section, with the number of records, n , equal to 1.

keyval is an array containing the left-justified key value of the primary key. The first word of keyval contains the first two bytes of key value. For example, if the key value was contained in byte 2 of the record, key value 35_{16} would appear as $xy35_{16}$ in the record but as $35wz_{16}$ in keyval, where x and y are unknown digits in the record and w and z are any random digits.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the store new indexed record request has one of the following forms:

```
EXT*   WRITER              EXT    WRITER
 :       :                  :       :
RTJ    WRITER      or      RTJ    WRITER
ADC    reqbuf              ADC    reqbuf
 :       :                  :       :
```

The use of WRITER as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

The status consideration (istat) is the following: the file is currently locked by this user (bit 2 is set).

WRITER sets bit 15 (rejecting the request) if:

- The primary key value is not unique (bit 4 is also set).
- A mass memory error has occurred (bit 5 is also set).
- File manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- The primary key value contained in the record is not the same as that in the keyval array (bit 9 is also set).
- There is insufficient room in the key index structure to store the keys. The record was stored, but it cannot be retrieved by key value (bit 11 is also set).
- Insufficient room exists to store the record (bit 12 is also set).
- The file request buffer, reqbuf, was altered by the user before the WRITER request (bit 13 is also set).
- The file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

An example of a WRITER request is shown in figure 2-8.

READ SPECIFIC RECORD (READR)

The user may specify a particular set of records to be retrieved with the READR request. Execution of the READR request requires that the file be open to the user.

If a sequential or indexed file was opened for record access by relative record number, the number of records specified in the open request may be retrieved by a READR request by specifying the relative record number of the first record of the set. This relative record number may have been obtained when the record was initially stored (PUTS or WRITER request) or when the record was previously retrieved (READR or GETS request). On completion of the request, the number of records actually retrieved is passed back to the caller.

If an indexed file was opened for record access by key value, one record may be retrieved according to the key value specified. Specifically, the record retrieved has a key value defined as follows:

Let k_s = Key value specified

K = Set of all key values for which there is at least one record in the file

k_r = Key value of record retrieved

```
C PURPOSE OF THIS CODE IS TO STORE NEW TEST RESULTS RECORD INTO
C INDEXED FILE
C RECORD LENGTH IS 31 WORDS.
C RECORD FORMAT IS AS FOLLOWS-
C   WORDS       CONTENTS
C   -----       --------
C    1-15       PATIENT NAME
C   16-20       SOCIAL SECURITY NUMBER (PRIMARY KEY)
C   21-31       TEST RESULTS
      INTEGER PRMKEY(5),SS(5),PATNAM(15)
      DIMENSION JBUF(24),JREC(33),JDATA(15),NTRAN(11),ITOT(111)
      COMMON SS,PATNAM,NRES,NTRAN
      DATA JDATA /"FILE J  ZQX389  ","VOLUME 1",0,1,0/
      DATA JBUF /24*0/
C OBTAIN ACCESS TO FILE J
      CALL OPENFL (JBUF,JDATA,JSTAT)
      IF (JSTAT.LT.0) GO TO 8010
C STORE SOCIAL SECURITY NUMBER (10 ASCII CHARACTERS) IN RECORD BUFFER
C AND IN PRMKEY ARRAY.
      DO 100 ISS=1,5
      ISTOR=SS(ISS)
      PRMKEY(ISS)=ISTOR
      JREC(ISS+15)=ISTOR
  100 CONTINUE
C STORE PATIENT NAME IN RECORD
      DO 200 J=1,15
  200 JREC(J)=PATNAM(J)
C STORE LAB TEST CODES AND TEST RESULTS IN RECORD BUFFER.
C NRES = NUMBER OF RESULTS
      NF=NRES
      DO 300 N=1,NF
  300 JREC(N+20)=NTRAN(N)
C STORE NEW RECORD INTO FILE J, INDEXED BY SOCIAL SECURITY NUMBER.
      CALL WRITER (JBUF,JREC,PRMKEY,JRSTAT)
      IF (JRSTAT.LT.0) GO TO 8090
        .                        .
        .                        .
        .                        .
```

Figure 2-8. Store New Indexed Record (WRITER) Example (FORTRAN)

Then k_r = The least key value, k, such that there is a record in the file corresponding to k and $k \geq k_s$. In set notation,

$k_r = \min (k{:}k \in K \text{ and } k \geq k_s)$

Order within a set of key values is defined in Indexed Files, section 1.

For a primary key, the definition of k_r uniquely defines the record retrieved. For a key other than a primary key, k_r may correspond to a set of more than one record. In this case, the first record stored in the file with key value k_r is the one retrieved. An example of record retrieval by a nonprimary key is shown in figure 2-9.

Records in File	
Relative Record Number	Key Value
1	50
2	74
3	74
4	18
5	21
6	0
7	7
8	2
9	2
10	37
11	35

Key Value Specified in Request = k_s	Record Retrieved	
	Relative Record No.	Key Value = k_r
2	8	2
34	11	35
0	6	0
36	10	37
38	1	50
39	1	50
50	1	50
73	2	74

Figure 2-9. Example of READR Record Retrieval by Key Value (Non-Primary Key)

The READR request may be used in conjunction with the retrieve next records (GETS) request, described in the Retrieve Next Records (GETS) section. In this case, the user may think of the READR request as a request that positions the file to a record with a particular relative record number or to a particular record according to key value. For example, the file in figure 29 could be positioned to record 8, or to the record with the least key value greater than or equal to two or to any other record specified by the user. Once positioned, subsequent records may be read from the file in order according to key value or in order according to relative record number by a GETS request.

A file may be positioned to the record with the lowest key value by specifying a key value of all binary zeroes in the READR request. A file may be positioned to the first record stored in the file by specifying a relative record number of one in the READR request.

In FORTRAN, the retrieve specific record request has the following form:

CALL READR (reqbuf,recbuf,recspc,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. Contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user. On completion of the request, words 15, 16 and 17 of reqbuf are defined as follows:

reqbuf(15) Number of records actually retrieved by the file manager.

reqbuf(16), reqbuf(17) Relative record number of the first record retrieved (defined in Relative Record Number, section 1).

recbuf is the record buffer to receive any records retrieved. The length of recbuf is the number of words required to contain the records retrieved plus any unused words between retrieved records if the records are sector-aligned.

The length of reqbuf is determined as follows. Let

n = Number records to be retrieved

r = Record length

s = Number of words per sector

Then if u is the length of recbuf:

$$u = (n-1) \times \left\lceil \frac{r}{s} \right\rceil \times s + r \text{ for sector-aligned records}$$

u = n x r for non-sector-aligned records.

For example, suppose n = 3, r = 189, and s = 96, and records are sector aligned.

$$\text{Then } u = 2 \times \left\lceil \frac{189}{96} \right\rceil \times 96 + 189 = 573$$

Where $\lceil y \rceil$ is the least integer greater than or equal to y.

recspc is the record specifier, an array containing either a relative record number or a left-justified key value.

If the file was opened for retrieval by relative record number, recspc is a two-word array containing the relative record number of the first record of the set of records to be retrieved. (Relative record number format is defined in Relative Record Number, section 1.)

If the file was opened for retrieval by key value, recspc is an array initially containing the key value specified k_s as defined above. This key value must be left-justified in the array recspc. (An example of a left-justified key value is included in the description of the keyval array, in the Store New Indexed Record (WRITER) section.)

On completion of a READR request specifying a key value, the array recspc contains the key value, k_r, of the record actually retrieved. This key value is also left-justified in the array.

NOTE

This array recspc must not be equivalenced with the key value in the record, since the file manager may alter the value of recspc.

It is necessary to shift bytes within a key value stored in the array recspc before comparing with a key value stored in the record buffer if the key length is an odd number of bytes and the key is right-justified in the record buffer.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the read specific record request has one of the following forms:

```
EXT*  READR            EXT   READR
 :      :               :      :
RTJ   READR     or     RTJ   READR
ADC   reqbuf           ADC   reqbuf
 :      :               :      :
```

The use of READR as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

Status considerations (istat) are as follows:

- The file is currently locked by this user (bit 2 is set).
- Retrieval is by relative record number and one or more of the records are marked as deleted (bit 4 is set). The contents of the deleted records have been stored in the buffer, recbuf. By testing the first word of each record, the user may determine which records are deleted records. The first word of a deleted record has the value of the external FMRDEL. (See File Identification, section 1; Main-Memory-Resident File Description Parameters, appendix B; and figure 2-10.)
- End-of-file is reached before the number of records specified could be retrieved (bit 8 is set). At least one record is retrieved if bit 15 is zero. An end-of-file indication implies an insufficient number of records in the file to satisfy the conditions specified in the reqbuf and recspc arrays. If retrieval is by key value, no record in the file has a key value greater than or equal to the key value specified by the user. If retrieval is by relative record number, there are not enough records in the file starting at the record number in recspc to retrieve the number records specified in the OPENFL request (see also request rejections below).
- The specified key value, k_s, does not equal k_r, the key value retrieved (bit 9 is set).

NOTE

To test whether or not a record for key value k_s is in the file, it is necessary to test for the simultaneous setting of bits 8 and 15 as well as testing for the setting of bit 9. (See end-of-file status discussed previously in this section.)

READR sets bit 15 (rejecting the request) if:

- A mass memory error has occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- Record locking was requested, but the maximum number record locks in the system are currently in use (bit 6 is also set; the request may be delayed and retried).
- The record is locked by another user (bit 7 is also set; the request may be delayed and retried if care is taken to avoid the situation described in the note in Update Protection, section 1).
- End-of-file was reached before any records are retrieved (bit 8 is also set).
- The file request buffer, reqbuf, was altered by the user before the READR request (bit 13 is also set).
- The file was closed by an executive forced file close due to hardware failure or operator shutdown of the volume (bit 9 is also set).
- The relative record number was specified in recspc as 0,0 (bit 14 is also set).

Examples using the READR request are shown in figures 2-10 and 2-11. An example of READR used in conjunction with a GETS request is shown in figure 2-13.

Differences between the READR request and the GETS request are listed in the Retrieve Next Records (GETS) section.

RETRIEVE NEXT RECORD (GETS)

The GETS request retrieves a set of records from a file open to the user. The number of records retrieved is as specified in the OPENFL request. Record locking on retrieval is performed if it was specified in the OPENFL request. Records retrieved are in order by relative record number if the file was opened for retrieval by relative record number. The retrieved records are in order by key value if the file was opened for retrieval by key value.

For retrieval by relative record number, the file may be positioned to a particular record by a previous READR call that specifies a relative record number. For retrieval by relative record number, if no READR call is executed between the OPENFL request and the first GETS request, the first GETS request retrieves a set of records starting with the first record stored in the file. Subsequent GETS requests using the same request buffer retrieve records immediately following the last record retrieved by a GETS request.

For retrieval by key value, the file may be positioned to a particular key value by a previous READR request. If no READR request is executed between the OPENFL request and the GETS request, the first GETS request retrieves a set of records starting with the first record stored of the records with the lowest key value for the key specified in the OPENFL request. Subsequent GETS requests using the same request buffer retrieve records in order by key value, and within a key value by order of storage.

```
C PURPOSE OF THIS ROUTINE IS TO RETRIEVE THE FIRST 3 FILE RECORDS STORED
C AFTER RECORD 99,999 WAS STORED.
      DIMENSION JBUF(24),JDATA(15),IDEL(3)
      INTEGER RLRCNM(2)
C SET RELATIVE RECORD NUMBER = 100,000 = 1*65536 + 34464
      DATA RLRCNM /1,$86A0/
C FILE IS TO BE OPEN FOR RETRIEVAL OF 3 RECORDS PER REQUEST
      DATA JDATA /"FILE J  ZQX389  ","VOLUME 1",0,3,1/
      DATA JBUF /24*0/
C RECORD LENGTH IS 31.  RESERVE SPACE FOR 3 RECORDS.
      DIMENSION JREC(93)
      INTEGER FMRDEL
C DECLARE DELETE CODE AN EXTERNAL
      EXTERNAL FMRDEL
C INITIALIZE DELETE FLAGS
      DATA IDEL /3*0/
C OBTAIN ACCESS TO FILE J
      CALL OPENFL(JBUF,JDATA,JSTAT)
C TEST FOR REJECT
      IF (JSTAT.LT.0) GO TO 9000
C RETRIEVE 3 RECORDS, STARTING WITH RECORD NUMBER 100,000.
      CALL READR (JBUF,JREC,RLRCNM,JSTAT)
C TEST FOR REJECT
      IF (JSTAT.LT.0) GO TO 9050
C AT LEAST ONE RECORD WAS RETRIEVED
C WAS END-OF-FILE REACHED BEFORE ALL 3 RECORDS WERE READ
      IF (AND(JSTAT,$0100).NE.0) GO TO 450
C ALL 3 RECORDS WERE RETRIEVED.  ARE ALL OF THESE NON-DELETED RECORDS
      IF (AND(JSTAT,$0010).EQ.0) GO TO 500
C NO, AT LEAST ONE OF THESE RECORDS WAS PREVIOUSLY DELETED.
C TEST TO SEE WHICH RECORD(S) WERE DELETED, SET FLAGS ACCORDINGLY
      DO 400 ID=0,2
      IDP1=ID+1
      IF (JREC(ID*31+1).EQ.FMRDEL) IDEL(IDP1)=1
  400 CONTINUE
      GO TO 500
C END-OF-FILE DETECTED.DETERMINE ITS POSITION.
C THIRD RECORD IS BEYOND END OF FILE
  450 IDEL(3)=1
C WAS SECOND RECORD BEYOND END OF FILE
      IF (JBUF(15).EQ.1) IDEL(2) =1
  500 CONTINUE
C PROCESS THIS SET OF RECORDS
      .                         .
      .                         .
      .                         .
```

Figure 2-10. Example of READR Request with Access by Relative Record Number (FORTRAN)

When there are not enough records to satisfy a GETS request, an end-of-file indication is returned to the caller. The number of records actually retrieved is also passed to the caller. If a GETS request results in an end-of-file indication and this same GETS request is repeated, the first execution of the GETS after the end-of-file retrieves a set of records at the beginning of the file; that is, the set of records that would be retrieved by this GETS if no READR preceded it. After this cycle back to the beginning of the file, subsequent GETS requests are processed as usual. The end-of-file indication occurs only on the request for which there are not enough records. For that request no records at the beginning of the file are retrieved. The GETS request is similar to the READR request as described in the Read Specific Record (READR) section with the following differences:

- When retrieval is by key value, only one record may be retrieved for each READR request executed, but more than one record may be retrieved for each GETS request executed. For a nonprimary key, any record that is not the first record stored for a given key value cannot be retrieved by a READR request. Such a record can be retrieved by a GETS request.
- When retrieval is by relative record number, a READR request can retrieve records starting at any relative record number. A GETS request by relative record number must start at record 1 of the file or at the record set immediately following the record set read by either a previous GETS request or a previous READR request.
- When retrieval is by key value, a READR request may specify any key value, but a GETS request must start with records of lowest key value or with the records following those retrieved by a previous GETS or READR request.

```
C PURPOSE OF THIS CODE IS TO RETRIEVE RECORD WITH LEAST PRIMARY KEY
C VALUE GREATER THAN OR EQUAL TO 12.
      DIMENSION NBUF(24),NDATA(15),NREC(96)
      DATA NBUF /24*0/
C SPECIFY ACCESS BY KEY 1, NO LOCKING
      DATA NDATA /"FILE N  GETOOR  ","SYSVOL20",1,1,0/
C SPECIFY KEY VALUE 12.
      DATA KEY1/12/
      DATA MAXTRY /300/
      DIMENSION ITEMP(4)
      DATA IFLAG /$0021/, ITIME/1/
C DETERMINE LOCATION OF STATEMENT 200. STORE INTO ICOMP FOR LATER USE.
      ASSIGN 200 TO ICOMP
C OBTAIN ACCESS TO FILE N
      CALL OPENFL (NBUF,NDATA,NSTAT)
      IF (NSTAT.LT.0) GO TO 8000
C RETRIEVE RECORD.  NOTE THAT THE NAME KEY1 MUST APPEAR IN THE PARAMETER
C LIST OF THE CALL, NOT THE CONSTANT 12, SINCE THE FILE MANAGER MAY
C CHANGE THE VALUE OF KEY1.
C INITIALIZE NUMBER RETRIALS.  (NTRY=NUMBER RETRIALS DUE TO RECORD LOCK
C BY ANOTHER USER.)
      NTRY=0
  200 CALL READR (NBUF,NREC,KEY1,NSTAT)
      IF (NSTAT.GE.0) GO TO 500
C TEST FOR LOCK BY ANOTHER USER
      IF (AND(NSTAT,$0080).EQ.0) GO TO 250
C RECORD LOCKED BY ANOTHER USER.  DELAY 1 SECOND AND RETRY.
C HAVE MAXIMUM NUMBER RETRIALS BEEN MADE ALREADY
      IF (NTRY.GE.MAXTRY) GO TO 9092
C INCREMENT NUMBER RETRIALS
      NTRY=NTRY+1
C DELAY ONE SECOND
C INITIATE TIMER CALL.
C STATEMENT 200 IS COMPLETION LOCATION FOR TIMER REQUEST.
      CALL TIMER (ICOMP,IFLAG,ITIME,ITEMP)
      CALL DISPAT
C BIT 15 OF ISTAT IS SET.  TEST FOR END OF FILE.
  250 IF (AND(NSTAT,$0100).NE.0) GO TO 350
C OTHER ERROR DETECTED
      GO TO 9100
C NO RECORD IN FILE WITH PRIMARY KEY VALUE GREATER THAN OR EQUAL TO 12.
C PRINT MESSAGE
  350 CONTINUE
       .   .
       .   .
       .   .
      GO TO 900
C RECORD RETRIEVED.  PROCESS RECORD.
  500 CONTINUE
       .   .
       .   .
       .   .
C GO ON TO NEXT PROCEDURE
  900 CONTINUE
```

Figure 2-11. Example of READR Request with Access by Key Value (FORTRAN)

- When retrieval is by relative record number and all the records in a file are to be retrieved, successive executions of a READR request require the user to increment the relative record number between READR calls. This incrementing is done by the file manager, not the user, when accomplishing the same purpose with successive executions of a GETS request. (Execution time is approximately the same when using READR as when using GETS for this purpose.)

- When retrieval is by key value and a large number of records in the file are to be retrieved in order, the use of a READR request to position the file followed by repeated execution of a GETS request is faster than repeated execution of a READR request. This is because for each READR request, the file manager makes a search starting at the beginning of the key structure. Such a search involves mass memory transfers. For a GETS, this search is not made.

In FORTRAN, the retrieve next records request has the following form:

CALL GETS (reqbuf,recbuf,keyval,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file. The contents of reqbuf may have been altered by the file manager in performing file access requests, but contents of reqbuf must not have been altered by the user. On completion of the request, words 15, 16 and 17 of reqbuf are defined as follows:

reqbuf(15)	Number records actually retrieved by file manager.
reqbuf(16), reqbuf(17)	Relative record number of the first record retrieved (defined in Relative Record Number, section 1).

recbuf is the record buffer to receive retrieved records. The required length of recbuf is determined as for the length of recbuf in the READR request (see the Read Specific Record (READR) section).

keyval is an array that is ignored if retrieval is not by key value. If retrieval is by key value, the array keyval must initially contain zero or the left-justified key value stored by the user in the array, recspc, in the READR call preceding the GETS request. On completion of the GETS request, the array keyval contains the key value, left-justified, of the last record retrieved. If a GETS request is repeatedly executed, the user must not alter the contents of the array keyval between successive executions.

NOTE

It is necessary to shift bytes within a key value stored in the array keyval before comparing with a key value stored in the record buffer if the key is right-adjusted in the record buffer.

istat is the file request status word as defined in Status Indicator Word, section 1 (see also error considerations below).

In assembly language, the retrieve next records file request has one of the following forms

```
EXT*   GETS                EXT    GETS
 :      :                   :      :
RTJ    GETS        or      RTJ    GETS
ADC    reqbuf              ADC    reqbuf
 :      :                   :      :
```

The use of GETS as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

Status considerations (istat) are as follows:

- The file is currently locked by this user (bit 2 is set).
- Retrieval is by relative record number and at least one retrieved record is marked as deleted (bit 4 is set). Contents of deleted records are included in the buffer recbuf. By testing the first word of each record, the user may determine which records are deleted records. The first word of a deleted record has the value of the external, FMRDEL. (See File Identification, section 1; Main-Memory-Resident Volume Description Parameters, appendix B; and figure 2-10.) This indication cannot occur when access is by key value; that is, a deleted record is never retrieved by key value since its pointer was deleted from the key index.
- End-of-file is reached before the number of records specified could be retrieved (bit 8 is set). At least one record was retrieved if bit 15 is zero. (See also request rejections below.)

GETS sets bit 15 (rejecting the request) if:

- A mass memory error has occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- Record locking was requested, but the maximum number record locks in the system are currently in use (bit 6 is also set; the request may be delayed and retried a finite number of times).
- The record is locked by another user (bit 7 is also set; the request may be delayed and retried if care is taken to avoid the situation described in the note in Update Protection, section 1).
- End-of-file was reached before any records were retrieved (bit 8 is also set).
- The file request buffer, reqbuf, was altered by the user before the GETS request (bit 13 is also set).
- The file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

Examples using the GETS request are shown in figures 2-12, 2-13, and 2-14.

STORE UPDATED RECORD (UPDREC)

To store an updated record into a file, the record must have been first retrieved by a READR request or a GETS request. Before the UPDREC request, either the file must be locked or the record to be updated must be locked as retrieved.

A key value for a record in an indexed file may not be changed by an UPDREC request. If a record's key value must be changed, this may be done by a delete record request (see the Delete Record Request (DELREC) section) followed by a WRITER request (see the Store New Indexed Record (WRITER) section).

The number of records stored by the execution of an UPDREC request is one if the preceding GETS or READR request accessed the file by key value. In this case, the preceding READR or GETS request must retrieve only the record to be updated.

If the preceding GETS or READR request accessed the file by relative record number, the number of records stored by each UPDREC execution is the number of records specified for retrieval in the OPENFL request.

If an end-of-file indication occurred on the retrieval preceding the UPDREC, the file manager updates only those records that precede the end-of-file. The file manager ignores the number of records specified in the OPENFL request in this case. (The number of records actually

```
C PURPOSE OF THIS ROUTINE IS TO RETRIEVE SEQUENTIALLY EACH RECORD OF
C FILE J. PRINTING THE CONTENTS OF EACH RECORD AFTER RETRIEVAL.
      DIMENSION JBUF(24),JDATA(15),JREC(96)
      DATA JBUF /24*0/
      DATA JDATA /"FILE J  RXT436  ","SYSVOL20",0,1,0/
      DATA MAXTRY /300/
      DIMENSION ITEMP(4)
      DATA IFLAG /$0021/, ITIME/1/
C DETERMINE LOCATION OF STATEMENT 200. STORE INTO ICOMP FOR LATER USE.
      ASSIGN 200 TO ICOMP
C OBTAIN ACCESS TO FILE J
      CALL OPENFL (JBUF,JDATA,JSTAT)
      IF (JSTAT.LT.0) GO TO 9000
C INITIALIZE NUMBER OF RETRIALS. (NTRY=NUMBER RETRIALS DUE TO RECORD
C LOCK BY ANOTHER USER)
  190 NTRY=0
  200 CALL GETS (JBUF,JREC,JDUM,JSTAT)
      IF (JSTAT.GE.0) GO TO 500
C TEST FOR LOCK BY ANOTHER USER
      IF (AND(JSTAT,$0080).EQ.0) GO TO 250
C HAVE MAXIMUM NUMBER RETRIALS BEEN MADE ALREADY
      IF (NTRY.GE.MAXTRY) GO TO 9092
C INCREMENT NUMBER RETRIALS
      NTRY=NTRY+1
C INITIATE TIMER CALL.
C RECORD LOCKED BY ANOTHER USER.  DELAY 1 SECOND AND RETRY.
C STATEMENT 200 IS COMPLETION LOCATION FOR TIMER REQUEST.
      CALL TIMER (ICOMP,IFLAG,ITIME,ITEMP)
      CALL DISPAT
C BIT 15 OF ISTAT IS SET.  TEST FOR END OF FILE.
  250 IF (AND(JSTAT,$0100).NE.0) GO TO 5000
C OTHER ERROR DETECTED
      GO TO 9100
C PRINT RECORD CONTENTS
  500 CONTINUE
C TEST FOR RECORD DELETION
      IF (AND($0010,JSTAT).NE.0) GO TO 190
C RECORD NOT DELETED.  CONTINUE WITH PRINTING
      .          .
      .          .
      .          .
      GO TO 190
C END-OF-FILE REACHED.  PROCESSING COMPLETE
 5000 CONTINUE
      .   .
      .   .
      .   .
```

Figure 2-12. GETS Request Example, Access by Relative Record Number (FORTRAN)

retrieved is stored in the request buffer on the preceding retrieval.)

If some of the records retrieved in the preceding retrieval are marked as deleted records, these records are written back to the file by the UPDREC request in the same manner as nondeleted records.

In FORTRAN, the store updated record request has the following form:

CALL UPDREC (reqbuf,recbuf,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file and in retrieving the record(s) to be updated. No file manager call may reference this buffer between the retrieval of the record(s) to be updated and the UPDREC call. The contents of reqbuf must not have been altered by the user. In executing the preceding retrieval, the file manager stores the number records retrieved in word 15 of reqbuf and a relative record number in words 16 and 17 of reqbuf. This is the relative record number of the first record of the set of records to be updated. Other words of reqbuf are dependent on this relative record number. Therefore, the user may not modify words 16 and 17 of reqbuf even if he only wishes to modify a proper subset of the records retrieved.

recbuf is the record buffer containing records to be stored by this request. The length of recbuf is determined as for the PUTS request (see the Store New Records Sequentially (PUTS) section), except that the two extra words are not needed at the end of the buffer.

```
C PURPOSE OF THIS CODE IS TO PRINT A REPORT OF THOSE RECORDS IN FILE J
C WHICH CORRESPOND TO AGES 2 THROUGH 4 YEARS.  AGE IS STORED IN
C YEARS IN KEY VALUE 2.
C  (AGE IS STORED IN BYTE 10 OF EACH RECORD.)
C FOUR RECORDS ARE TO BE RETRIEVED FOR EACH GETS REQUEST.  RECORD
C LENGTH IS 31 WORDS.
C AN EXAMPLE SHOWING WHICH RECORDS WOULD BE RETRIEVED FROM A
C HYPOTHETICAL FILE BY THE USE OF THIS CODE IS SHOWN IN FIGURE 2-14.
      DIMENSION JBUF(24),JREC(124),JDATA(15)
C SET UP JDATA TO ACCESS FILE BY KEY 2 WITH NO RECORD LOCKING
      DATA JDATA /"FILE J  ZQX389  ","VOLUME 1".2.4.0/
C TOTAL NUMBER RECORDS FOR EACH AGE IS COMPUTED.
      INTEGER TOTREC(4).  AGERET .RETREV
C INITIALIZE TOTAL RECORDS FOR ALL AGES
      DATA TOTREC /4*0/
C SET MAXIMUM RETRIALS WHILE WAITING FOR UNLOCK BY OTHER USER = 60
C RETRIALS
      DATA MAXTRY/60/. NTRY/0/
      DIMENSION ITEMP(4)
      DATA IFLAG /$0021/. ITIME/1/
      CALL OPENFL (JBUF.JDATA.JSTAT)
      IF (JSTAT.LT.0) GO TO 9900
C INITIALIZE IAGE. (KEY VALUE= AGE IN YEARS) TO TWO
C LEFT ADJUST AGE IN IAGE
      IAGE=$200
C REQUEST RETRIEVAL OF ONE RECORD.  NOTE THAT READR IGNORES NUMBER
C RECORDS SPECIFIED IN OPEN REQUEST SINCE ACCESS IS BY KEY VALUE.
C THIS READR CALL POSITIONS THE FILE FOR THE FIRST GETS CALL.
      ASSIGN 200 TO RETREV
  200 CALL READR ( JBUF.JREC.IAGE.ISTAT )
C TEST FOR REJECT
  250 IF (ISTAT.LT.0) GO TO 850
C RETRIEVAL COMPLETED.
C RE-INITIALIZE NUMBER RETRIALS
      NTRY=0
C INITIALIZE NUMBER RECORDS PROCESSED FOR THIS RETRIEVAL
      NPROC=0
C HAVE ALL RECORDS FROM THIS RETRIEVAL BEEN PROCESSED
  275 IF (NPROC.GE.JBUF(15)) GO TO 500
C AT LEAST ONE MORE RECORD MUST BE PROCESSED
C NO NEED TO TEST FOR DELETED RECORD SINCE RETRIEVAL WAS BY KEY VALUE
C AND REQUEST WAS COMPLETED WITHOUT ERROR
C COMPUTE POSITION OF RECORD WORD 5 IN BUFFER
      I5= NPROC*31+5
C OBTAIN RIGHT-ADJUSTED AGE FROM RECORD.
C NOTE THAT AGE RETRIEVED (AGERET) MAY NOT BE THE SAME AS KEYAGE
      AGERET = AND(JREC(I5).$FF)
C WAS KEY VALUE OF THIS RECORD FIVE OR LARGER.  IF SO. ALL REQUIRED
C RECORDS HAVE BEEN RETRIEVED AND PROCESSED.
      IF (AGERET.GE.5) GO TO 800
C INCREMENT APPROPRIATE TOTAL
      TOTREC(AGERET)=TOTREC(AGERET)+1
C PRINT REPORT OF RECORD
      .              .
      .              .
      .              .
C INCREMENT NUMBER RECORDS PROCESSED
      NPROC=NPROC+1
C GO BACK TO PROCESS NEXT RECORD
      GO TO 275
C WAS THERE AN END-OF-FILE INDICATION
  500 IF (AND(ISTAT.$0100).NE.0) GO TO 800
  600 ASSIGN 650 TO RETREV
  650 CALL GETS (JBUF.JREC.IAGE.ISTAT)
      GO TO 250
C REPORT PRINTING COMPLETE
  800 CONTINUE
```

Figure 2-13. Example of Repeated GETS Request, Access by Key Value, Initial Positioning by READR (FORTRAN) (Sheet 1 of 2)

```
C RELINQUISH ACCESS PERMIT TO FILE J.
      CALL CLOSFL (JBUF,ISTAT)
      IF (ISTAT.LT.0) GO TO 9998
      GO TO 950
C READR REQUEST REJECTED OR GETS REQUEST REQUEST REJECTED
  850 CONTINUE
C WAS REJECT DUE TO SOME ERROR
      IF (AND(ISTAT,$6020).NE.0) GO TO 9995
C WAS THERE AN END-OF-FILE INDICATION
      IF (AND(ISTAT,$0100).NE.0) GO TO 800
C REQUEST WAS REJECTED BECAUSE RECORD TO BE ACCESSED IS LOCKED.
C HAS REQUEST BEEN RETRIED NMAX TIMES
      IF (NTRY.GE.MAXTRY) GO TO 9010
      NTRY=NTRY+1
C RETRY READR REQUEST OR RETRY GETS REQUEST AFTER 1-SECOND DELAY
      CALL TIMER (RETREV,IFLAG,ITIME,ITEMP)
      CALL DISPAT
C PROCEDURE COMPLETE
  950 CONTINUE
      .     .
      .     .
      .     .
```

Figure 2-13. Example of Repeated GETS Request, Access by Key Value, Initial Positioning by READR (FORTRAN) (Sheet 2 of 2)

Records in File	
Relative Record Number	Key Value (Key 2)
1	0
2	0
3	1
4	3
5	3
6	4
7	4
8	4
9	4
10	4
11	4
12	6
13	8
14	8
15	8
16	9
17	9
18	9
19	9
20	9
⋮	⋮

Request in Figure 2-13	Execution	Relative Record No. of Records Retrieved	Initial Contents of KEY2	Contents of KEY2 Array on Completion
READR		4	2	3
GETS	1st	5,6,7,8	3	4
GETS	2nd	9,10,11,12	4	6

Figure 2-14. Example of File Records Retrieved by Code in Figure 2-13

istat is the file request word as defined in Status Indicator Word, section 1. (See also error considerations below.)

In assembly language, the store updated record request has one of the following forms:

```
EXT*   UPDREC              EXT    UPDREC
 :       :                  :       :
RTJ    UPDREC      or      RTJ    UPDREC
ADC    reqbuf              ADC    reqbuf
 :       :                  :       :
```

The use of UPDREC as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to UPDREC if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error has occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- Neither the file nor the records to be updated are locked (bit 7 is also set).
- The file request buffer, reqbuf, was altered after the completion of the preceding retrieval before this UPDREC call (bit 13 is also set).
- The file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).
- The preceding retrieval was by key value and more than one record was retrieved (bit 14 is also set). The number of records retrieved is governed by the preceding OPENFL request. The OPENFL request gives an error indication if record locking or file locking is specified for access by key value and the number records specified is greater than one. However, the UPDREC request can be made with no lock indication in the preceding OPENFL request if the file is locked between the OPENFL request and the UPDREC request.

  In this case, the previous OPENFL request would give no error indication if the number records specified exceeds one. It is in this way that this bit 14 error indication can be generated.

An example of FORTRAN code using the UPDREC request is shown in figure 2-15.

DELETED RECORD (DELREC)

Only one record may be deleted per delete record request. Each execution of a DELREC request must be preceded by either a GETS request that retrieves only one record, or a READR request that retrieves only one record. The single record retrieved by the preceding GETS or READR request is the record deleted by the DELREC request.

Either the file must have been previously locked or the record to be deleted must have been locked on retrieval.

In deleting a record from a file, the file manager stores the record delete code, FMRDEL, in word 1 of the record buffer and then stores the record back into the file. (More information on the code FMRDEL may be found in Main-Memory-Resident Volume Description Parameters, appendix B.) In addition, if the file is indexed, the record's pointers are deleted from the file's key indices.

In FORTRAN, the delete record request has the following form:

CALL DELREC (reqbuf,recbuf,istat)

```
C PURPOSE OF THIS CODE IS TO CHANGE WORDS IFIRST THROUGH ILAST OF RECORD
C WITH RELATIVE RECORD NUMBER IRNM.
C NEW VALUES ARE PASSED VIA COMMON IN ARRAY CHGREC.
      INTEGER CHGREC(96)
      COMMON IRNM,IFIRST,ILAST,CHGREC
      DIMENSION KBUF(24),KDATA(15),KREC(96),IRNM(2)
      DATA KBUF /24*0/
      DATA KDATA /"FILE K   PALMER   ","SYSVOL20",0,1,-1/
C OBTAIN ACCESS TO FILE K.  REQUEST RECORD ACCESS BY RELATIVE
C RECORD NUMBER, SINGLE RECORD RETRIEVAL AND FILE LOCKING.
      CALL OPENFL (KBUF,KDATA,KSTAT)
      IF (KSTAT.LT.0) GO TO 8900
C RETRIEVE RECORD IRNM
      CALL READR (KBUF,KREC,IRNM,ISTAT)
      IF (ISTAT.LT.0) GO TO 8950
C CHANGE RECORD
C STORE INDICES IN LOCAL STORAGE
      KFIRST=IFIRST
      KLAST=ILAST
      DO 400 IND=KFIRST,KLAST
  400 KREC(IND)=CHGREC(IND)
C STORE RECORD BACK INTO FILE.
      CALL UPDREC (KBUF,KREC,KSTAT)
      IF (KSTAT.LT.0) GO TO 8970
        .                    .
        .                    .
        .                    .
```

Figure 2-15. UPDREC Example (FORTRAN)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in the retrieval of the record to be deleted. Contents of reqbuf may not be altered between completion of the preceding record retrieval and the initiation of the delete record request.

recbuf is the record buffer containing the record to be deleted. (Length is the length of the record.) Upon completion of the request, the first word of this buffer contains the record deleted code, FMRDEL.

istat is the file request status word as defined in Status Indicator Word, section 1. (See also error considerations below.)

In assembly language, the delete record request has one of the following forms:

```
EXT*    DELREC              EXT     DELREC
 :        :                  :        :
RTJ     DELREC      or      RTJ     DELREC
ADC     reqbuf              ADC     reqbuf
 :        :                  :        :
```

The use of DELREC as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

An error is indicated on the return from a call to DELREC if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error has occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- Neither the file nor the record to be deleted is locked (bit 7 is also set).
- The file manager is unable to delete one or more of record's key values from the key structure because one or more errors exist in the key structure (bit 11 is also set).
- The file request buffer, reqbuf, was altered after the completion of the preceding retrieval before this DELREC call (bit 13 is also set).
- The file is closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).
- More than one record was retrieved by the retrieval preceding the DELREC call (bit 14 is also set).

An example of FORTRAN code using the DELREC request is shown in figure 2-16.

COMPRESS FILE (COMFIL)

A file should be compressed when space for more records is needed in the file and some records have been marked as deleted.

Compression of a given file involves physically moving the nondeleted records together, writing over any records that have been marked as deleted. This allows more new records to be stored into the file. If the file is indexed, the associated key structures are rebuilt during compression.

Before compressing a file, an access permit must be obtained to compress the file. This involves executing an OPENFL request (see the Open File (OPENFL) section). The file manager locks the file at the time the compression access permit is granted.

To compress a file, the user must repeatedly execute a COMFIL request. Each execution causes a set of the records in the file to be compressed. The size of the set is the number specified in the OPENFL request. (The size must be one record for an indexed file.) When an end-of-file indication has been received, the entire file has been compressed. The reason for requiring a series of COMFIL calls instead of only one to compress the entire file is that in this way other file manager requests may be interspersed between the COMFIL calls in the request processor queues (see Reentrant Request Processors; Serial Request Processors, section 1). This avoids holding out other file manager users for a long period of time during file compression.

In FORTRAN, the compress file request has the following form:

CALL COMFIL (reqbuf,recbuf,istat)

Where:

reqbuf is the file request buffer, a 24-word array. This must be the same array as the one used in opening the file for compression. It must not have been altered by the user.

recbuf is the record buffer used by the file manager for temporary storage of records to be compressed. The length of recbuf is determined as for the PUTS request (see the Store New Records Sequentially (PUTS) section), except that four extra words are needed by the file manager at the end of the buffer instead of only two extra words. For example, using the symbols defined in the Store New Records Sequentially (PUTS) section, suppose n = 3, r = 189, and s = 96, and records are sector-aligned. Then let b' = the size of recbuf for a COMFIL request.

$$\text{Then } b' = 3 \times \left\lceil \frac{189}{96} \right\rceil \times 96 + 4$$

$$= 576 + 4 = 580$$

Where $\lceil y \rceil$ is the least integer greater than or equal to y.

istat is the file request startup word as defined in Status Indicator Word, section 1. (See also error considerations below).

In assembly language, the compress file request has one of the following forms:

```
EXT*    COMFIL              EXT     COMFIL
 :        :                  :        :
RTJ     COMFIL      or      RTJ     COMFIL
ADC     reqbuf              ADC     reqbuf
 :        :                  :        :
```

```
C PURPOSE OF THIS CODE IS TO RETRIEVE EACH RECORD FROM A MAILING
C LIST FILE AND DELETE THOSE RECORDS WHICH SHOW DATE OF LAST PURCHASE
C BEFORE A SPECIFIED YEAR.
C
C RECORD FORMAT FOR FILE "MAILLIST"
C
C     WORD  CONTENTS
C     ----  --------
C     1-4   CUSTOMER IDENTIFICATION CODE
C     5-44  ADDRESS
C     45    YEAR OF LAST PURCHASE (ASCII)
C
      DIMENSION IRQBUF(24),IODATA(15),IRECBF(45)
      DATA IRQBUF/24*0/
C PRESET IODATA TO SPECIFY RETRIEVAL BY RELATIVE RECORD NUMBER, ONE
C RECORD PER RETRIEVAL, WITH RECORD LOCKING.
      DATA IODATA /"MAILLIST","MIDSOUTH","VOLUME 1",0,1,1/
C PRESET PURGE DATE TO ASCII CODE FOR 1972
      DATA IYEAR/$3732/
C OBTAIN ACCESS PERMIT TO RETRIEVE 1 RECORD PER REQUEST FROM
C MAILING LIST FILE WITH RECORD ACCESS BY RELATIVE RECORD NUMBER.
C RECORD LOCKING IS SPECIFIED IN IODATA.
      CALL OPENFL(IRQBUF,IODATA,IOSTAT)
C TEST FOR REJECT
      IF (IOSTAT.LT.0) GO TO 9000
C RETRIEVE ONE RECORD
  110 CALL GETS(IRQBUF,IRECBF,IDUM,IGSTAT)
C TEST FOR REJECT
      IF (IGSTAT.LT.0) GO TO 9020
C TEST FOR PREVIOUS DELETION OF RECORD
      IF (AND(IGSTAT,$10).NE.0) GO TO 110
C IS THIS RECORD TO REMAIN IN FILE
      IF (IRECBF(45).GE.IYEAR) GO TO 110
C THIS RECORD IS FOR CUSTOMER WITH LAST PURCHASE PRIOR TO SPECIFIED
C DATE.  DELETE THIS RECORD.
      CALL DELREC(IRQBUF,IRECBF,IDSTAT)
      IF (IDSTAT.LT.0) GO TO 9300
C GO BACK TO RETRIEVE NEXT RECORD
      GO TO 110
C DISTINGUISH END-OF FILE FROM OTHER ERRORS
 9020 IF (AND(IGSTAT,$0100).NE.0) GO TO 5000
      •       •
      •       •
      •       •
C END OF PROCESS
 5000 CONTINUE
      CALL CLOSFL (IRQBUF,ICSTAT)
      IF (ICSTAT.LT.0) GO TO 9500
      •       •
      •       •
      •       •
```

Figure 2-16. Example of DELREC Request (FORTRAN)

The use of COMFIL as an absolute external or as a relative external is discussed in File Manager Interceptor Module, section 1.

A status indication is that the file is currently locked by this user if bit 2 of istat is set.

An error is indicated on the return from a call to COMFIL if bit 15 of istat is set. The particular error condition(s) are indicated in istat as follows:

- A mass memory error has occurred (bit 5 is also set).
- The file manager data structures on mass memory or in main memory contain one or more errors (bits 5 and 14 are also set).
- An end-of-file has been reached. The file should now be closed (bit 8 is also set).
- The file request buffer, reqbuf, was altered by the user before this COMFIL call (bit 13 is also set).
- The file was closed by executive forced file close due to hardware failure or operator shutdown of the volume (bit 13 is also set).

An example of a FORTRAN subroutine that performs file compression using the COMFIL request is shown in figure 2-17.

```
      SUBROUTINE COMPRS (NAME,IOWNER,IVOL,RECLEN,ALIGN,INDEXD,ERRFL1,
     1                    ERRFL2     )
C PURPOSE OF THIS SUBROUTINE IS TO COMPRESS FILE WITH NAME, OWNER
C AND VOLUME PASSED IN SUBROUTINE PARAMETER LIST.  RECLEN IS RECORD
C LENGTH.  ALIGN=0 FOR NON-SECTOR-ALIGNED RECORDS, =1 FOR SECTOR-ALIGNED
C RECORDS.
      INTEGER RECLEN,ALIGN,RECBUF(964),REQLEN,ERRFLG
      INTEGER ERRFL1,ERRFL2
      DIMENSION NAME(4),IOWNER(4),IVOL(4),IDATA(15),IBUF(24)
      DATA IDATA(13)/-1/
C IDATA(15) NEED NOT BE PRESET, AS IT IS IGNORED WHEN FILE IS OPENED
C FOR COMPRESSION.
C INITIALIZE ERROR FLAGS
      ERRFL1=0
      ERRFL2=0
C TEST FOR RECORD LENGTH WITHIN RANGE ALLOWED
      IF (RECLEN.GT.960) GO TO 9000
C MOVE PASSED FILE NAME, OWNER, AND VOLUME TO IDATA ARRAY
      DO 20 I=1,4
      IDATA(I)=NAME(I)
      IDATA(I+4)=IOWNER(I)
      IDATA(I+6)=IVOL(I)
   20 CONTINUE
C OPEN FILE FOR COMPRESSION
C COMPUTE NUMBER RECORDS WHICH WILL FIT INTO RECBUF.
C COMPUTE REQUIRED LENGTH FOR ONE RECORD.
C IF INDEXED FILE, NUMBER RECORDS PER COMFIL CALL MUST BE ONE.
      IF (INDEXD.EQ.0) GO TO 100
      NUMREC=1
      GO TO 300
C ARE RECORDS NOT SECTOR-ALIGNED
  100 IF (ALIGN.EQ.0) GO TO 200
C RECORDS ARE SECTOR ALIGNED
C SECTOR LENGTH IS 96 WORDS.
      REQLEN= (RECLEN/96)*96
      IREM= RECLEN-REQLEN
      IF  (IREM.GT.0) REQLEN= REQLEN+96
      NUMREC = 960/REQLEN
      GO TO 300
C RECORDS ARE NOT SECTOR-ALIGNED
  200 NUMREC =  960/RECLEN
C STORE NUMBER RECORDS INTO IDATA ARRAY.
  300 IDATA(14)=NUMREC
      CALL OPENFL (IBUF,IDATA,ISTAT)
      IF (ISTAT.LT.0) GO TO 9000
  500 CALL COMFIL (IBUF,RECBUF,ISTAT)
      IF (ISTAT.GE.0) GO TO 500
C TEST FOR END-OF-FILE
      IF (AND(ISTAT,$0100).NE.0) GO TO 900
C OTHER ERROR.  PASS ERROR FLAG BACK TO CALLER IN ERRFL1 PARAMETER
      ERRFL1=ISTAT
  900 CALL CLOSFL (IBUF,ISTAT)
      ERRFL2=ISTAT
      RETURN
      .    .
      .    .
      .    .
```

Figure 2-17. Compress File Example (FORTRAN)

GLOSSARY

A

BACKGROUND – Processing with relatively low priority that is executed in unprotected main memory as foreground processing permits. Background priority levels are 0, 1, and 2.

BYTE – Basic unit of data; specifically for CDC CYBER 18 computers, a byte is eight bits, either bits 0 through 7 of a word or bits 8 through 15 of a word.

CLEAR – To clear a bit is to cause its value to become zero.

CYLINDER – A set of tracks in a drum or disk that can be read without repositioning the read/write heads (see figure A-1).

EXECUTIVE – See System executive.

FCB – See File control block.

FCBT – File control block table.

FDB – File definition block.

FDD – File definition directory.

FDS – File definition segment.

FIAT – File control block index allocation table.

FILE – A collection of related-records treated as a unit.

FILE CONTROL BLOCK (FCB) – The set of definition and control parameters for a file.

FILE DEFINITION DIRECTORY – A collection of pointers; for each file on a volume, there is one pointer, each pointing to the file control block for the file.

FOREGROUND – Processing that is time-critical. Foreground processing is executed in protected main memory.

INDEX – An ordered set of pointers.

KEY – Specific attribute of a record, such as age, birthdate, social security number, etc.

KIB – Key information block.

KIS – Key information segment.

MODULO (MOD) – A function such that if $x = r \pmod{k}$, there exists an integer n such that $x = n \cdot k + r$.

PARTITION – A collection of subsets of a set such that any pair of subsets are disjoint and the union of all the subsets is the entire set.

PROTECTED MAIN MEMORY – That part of main memory that is protected from erroneous storage or entry by unprotected programs. Attempted storage into a protected word or transfer of control to a protected instruction by an instruction in unprotected main memory causes a protect violation interrupt.

PROGRAM LIBRARY – A set of commonly used routines available to background programs.

RECORD – A set of data that is input or output at one time.

RELATIVE RECORD NUMBER – Position of the record within a file, expressed as an ordered pair of integers (n,m). A relative record number is stored in a two-word array with n stored in the first word and m in the second word of the array. If the relative record number r = (n,m), then

$$r = n \times 65{,}536 + m$$

Where: $0 \le m \le 65{,}535$ and

$0 \le n \le 255$

That is, m is a 16-bit (two-byte) positive integer and n is an eight-bit (one-byte) positive integer.

SCATTER CODE – A function that maps the integers into a specified subset; for example, the integer's modulo n is a hash code where n is any positive integer. See File Definition Directory section of appendix B for use of the scatter code.

SECTOR – One of the equal parts of a disk track (see figure A-1). A set of words on a drum defined by software to be a sector for drum/disk software compatibility, even though sectors do not exist physically on a drum.

SET (BIT) – Causes the bit to have a value of one.

SYSTEM EXECUTIVE – A set of program modules that controls the operation of other programs within the system.

TRACK – One of the concentric rings on a disk such that the entire ring (track) of data passes a read/write head every time the disk completes one revolution (see figure A-1).

UCT – User control table.

UNPROTECTED MAIN MEMORY – That area of main memory used by background programs (see Protected main memory).

USER AREA – A block of partitioned main memory that is controlled by the ITOS executive and used to execute user programs under ITOS control. This area is unprotected memory based on the setting of the protect bounds registers and page registers (see the ITOS 1 Reference Manual). User programs execute at a priority level controlled by the ITOS executive.

VOLUME – A single physical unit of a peripheral storage device; a volume that can be used for file manager file storage is a removable disk cartridge, a disk pack, a nonremovable disk cartridge, or a drum.

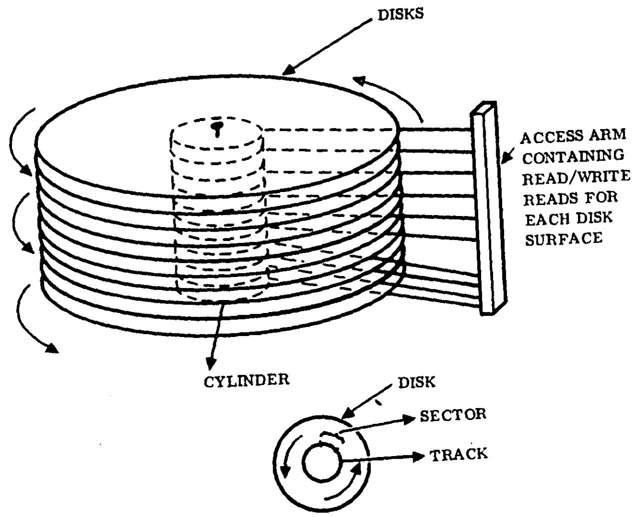


Figure A-1. A Disk Pack

Associated with each created file is a file control block (FCB) that contains the file's definition and control parameters. File control block (FCB) structure is described in the File Control Block Table section. The FCBs for the files on a given mass memory volume are stored together on that volume in a table called the file control block table (FCBT). Each FCB within this table has an FCBT index.

The FCB for a given file may be found on mass memory if the file's FCBT index is known. To ascertain the FCBT index for a given file, a search may be made in the file definition directory (FDD). There is an FDD on each volume used in the system. Each FDD contains a pointer to an FCB for each file located on the particular volume. File definition directory structure is described in the File Definition Directory section. Information needed to access the FDD for a given volume is found in main memory, as described in the Main-Memory-Resident Volume Description Parameters section.

In addition, on each volume, there is a file control block index allocation table (FIAT) used in assigning an FCB index to a file when it is created. The FIAT structure is described in the File Control Block Index Allocation Table section. An example of the above structures appears in the File Structure Example section.

MAIN-MEMORY-RESIDENT VOLUME DESCRIPTION PARAMETERS

When a volume is mounted and ready for use, volume file control parameters are stored into words 1 through 19 of the volume information table (see figure D-4). The following parameters are from the volume label as described in appendix E:

Mnemonic	Parameter Definition
VIFDDM	File definition directory address, most significant bits
VIFDDL	File definition directory address, least significant bits
VIMAXF	Maximum number of files permitted for a volume (defined at the time of system installation or volume initialization)
VICURF	Current number of files existing on the volume
VINFDB	Number of blocks in the file definition directory
VINXTB	Next block available for overflow in the file definition directory
VIWPS	Number of words in each sector on this volume
VINAME	Volume name; four words; eight ASCII characters

The location of the main file structures on a volume is shown in figure B-1.

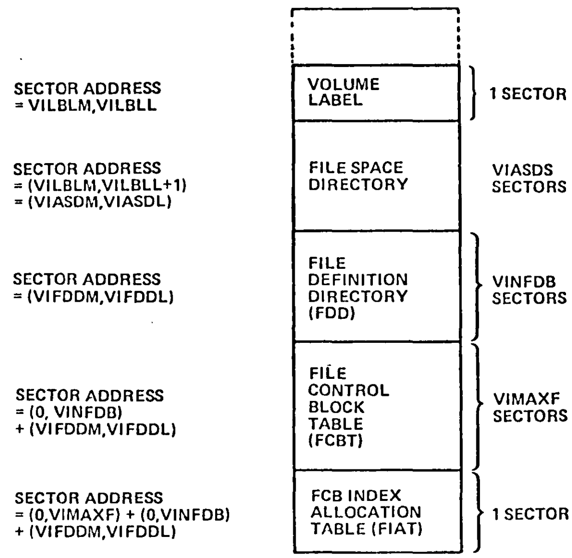


Figure B-1. Location of Main File Control Structures on a Volume

FILE DEFINITION DIRECTORY

The file definition directory for a given volume contains one file definition segment (FDS) for each file on that volume.

The FDSs are grouped into blocks so that all FDSs in a given block are for files with the same scatter code. The scatter code for a file is defined as follows. A file on a given volume is determined uniquely by its file name/file owner character string as defined in File Identification, section 1.

Let $(n_i \mid i=1,2,3,4)$ be the file's name where each n_i is the binary representation of two ASCII characters. Let $(w_i \mid i=1,2,3,4)$ be the file owner's name where each w_i is the binary representation of two ASCII characters. Let MINFDB be the number of blocks in the file definition directory's main part. (The value of MINFDB is dependent on the maximum number of files permitted on the volume. The formula for MINFDB appears later in this section.) If c is the scatter code for the file, then:

$$c = 1 + \sum_{i=1}^{4} \left[(n_i + w_i) \right] \pmod{\text{MINFDB}}$$

where the summation is computed as a 16-bit positive integer with the overflow handled as in a CDC CYBER 18 computer; that is, bit 15 is considered a part of the number and not a sign bit; overflow handling is such that $FFFF_{16} + 1 = 1$.

For example, suppose the file name is FILE A (left-justified), the file owner is SMITH (left-justified), and MINFDB = 100_{16}. Then:

$n_1 = 4649_{16}$ $\qquad w_1 = 534D_{16}$

$n_2 = 4C45_{16}$ $\qquad w_2 = 4954_{16}$

$n_3 = 2041_{16}$ $\qquad w_3 = 4820_{16}$

$n_4 = 2020_{16}$ $\qquad w_4 = 2020_{16}$

and:

$$c = 1 + \left[\sum_{i=1}^{4} (n_i + w_i) \right] \pmod{100_{16}}$$

$$= 1 + \left[D7D1_{16} \right] \pmod{100_{16}}$$

$$= 1 + D1_{16} = D2_{16} = 210$$

This file would be grouped with other files with scatter code 210.

Each file definition segment (FDS) has the following format:

Word	Contents
1 through 4	File name, eight ASCII characters
5 through 8	File owner, eight ASCII characters
9	File control block table index

The FDSs are grouped by scatter code into file definition blocks (FDBs). Each block is one sector long and consists of a one-word header together with as many existing FDSs for that scatter code as can fit into the block. Let NUMFDS be the maximum number of FDSs per FDB. Then:

$$\text{NUMFDS} = \left\lfloor \frac{(\text{VIWPS}-1)}{9} \right\rfloor^{\dagger}$$

where VIWPS is number of words per sector (see the Main-Memory-Resident Volume Description Parameters, appendix B). The header of each FDB contains the index to an overflow block if there are more than NUMFDS files with this scatter code. Otherwise, the header contains 0000_{16}.

The number of FDBs in the file definition directory's main part is:

$$\text{MINFDB} = \left\lceil \frac{\text{VIMAXF}}{\text{NUMFDS}} \right\rceil^{\dagger\dagger}$$

The file manager allows a maximum of $\lceil \text{MINFDB}/4 \rceil$ file definition overflow blocks. The maximum number of FDSs in the FDD is:

$$\text{VINFDB} = \text{MINFDB} + \left\lceil \frac{\text{MINFDB}}{4} \right\rceil$$

This provides ample directory space for VIMAXF files with a normal scattering of file name/owner strings.

Location of the FDD on a volume is shown in figure B-1. Structure of the FDD is shown in figure B-2.

† $\lfloor y \rfloor$ = The greatest integer less than or equal to x.
†† $\lceil y \rceil$ = The least integer greater than or equal to x.

NOTE

In a system including ITOS, a dump of the file definition directory may show the existence of files that were not created by any system user. ITOS creates a number of files for its own use.

FILE CONTROL BLOCK TABLE

The file control block table (FCBT) consists of one file control block (FCB) for each file on the volume. A maximum of VIMAXF FDBs are contained in the table. The FCBT immediately follows the space allowed for the file definition directory on the volume. Therefore, the sector address of the FCBT is the sum of the ordered pairs (VIFDDM, VIFDDL) and (0, VINFDB). (See figure B-1 and the Main-Memory-Resident Volume Description Parameters section.) Each FCB is one sector long; thus the length of the FCBT is VIMAXF sectors. The range of the FCB index is from one to VIMAXF.

When a file is created, thirty-three words are stored into the file's FCB. These words are defined as follows (words 6 through 10 are modified as the file is used):

Word	Mnemonic	Definition
1	RECLEN	Record length in words
2	TDATRM	Maximum number of records, most significant bits
3	TDATRL	Maximum number of records, least significant bits
4	DATBAM	Sector address of first record, most significant bits
5	DATBAL	Sector address of first record, least significant bits
6	FCBIND	FCB indicators as follows:
		Bit 15 — Record alignment indicator 0 — Records need not be sector-aligned 1 — Records must be sector-aligned
		Bit 14 — Storage mode for indexed file 0 — Records presented and stored randomly with respect to primary key

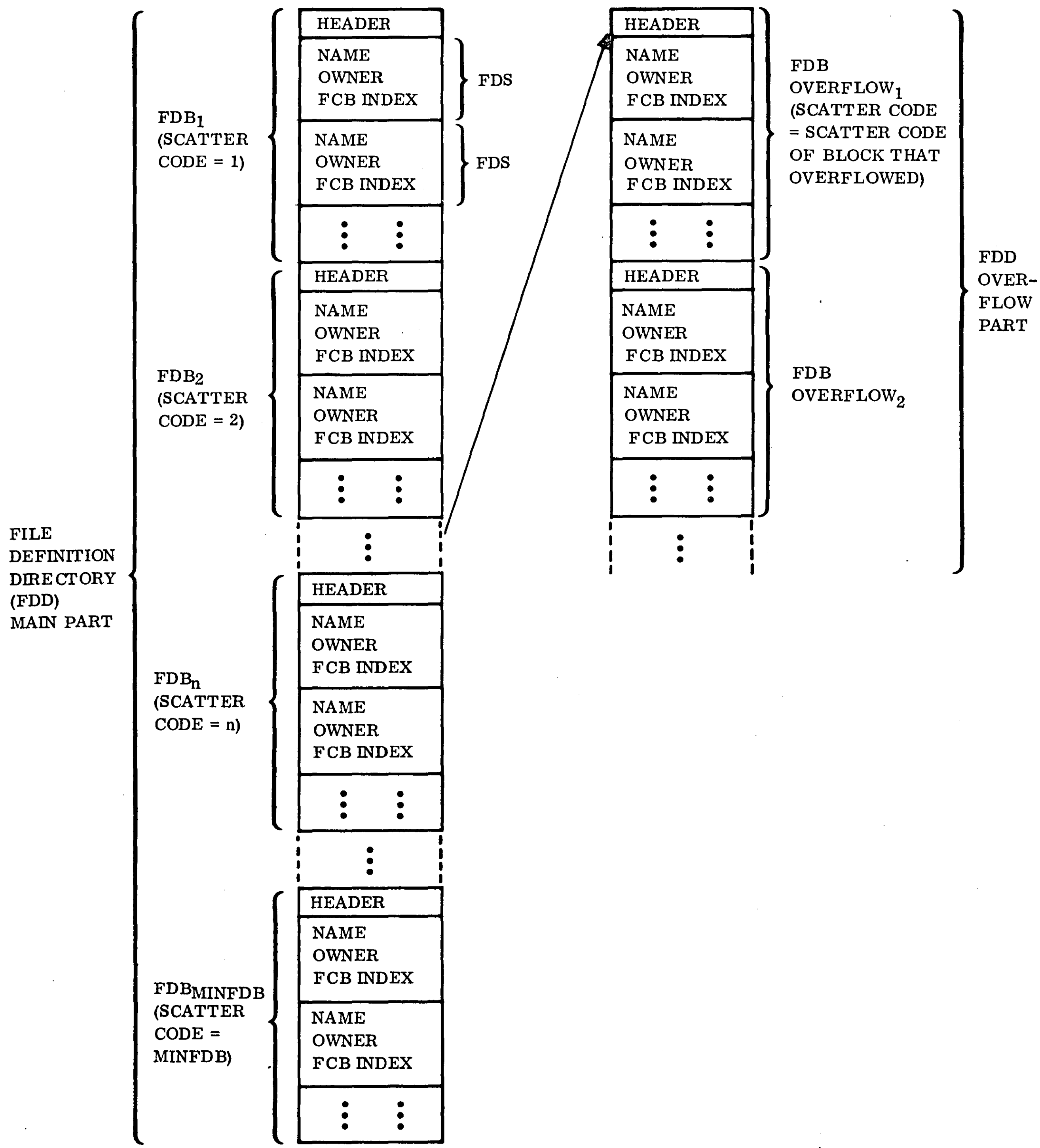


Figure B-2. File Definition Directory Structure

		1 Records presented and stored in order with respect to primary key
		Bit 13 Open/close indicator
		0 File closed
		1 File open
		Bit 12 File compression indicator
		0 File not currently being compressed
		1 File currently being compressed
		Bit 11 File special processing indicator
		0 File not currently open for special processing
		1 File currently open for special processing
		Bit 8 Binary data indicator
		0 File does not contain essentially binary data
		1 File contains essentially binary data
		Bit 0 File type
		0 Sequential file
		1 Indexed file
7	NEDATM†	Number of existing records, most significant bits
8	NEDATL†	Number of existing records, least significant bits
9	LINKFM	Next free key index block, most significant bits of block number
10	LINKFL	Next free key index block, least significant bits of block number
11	TNKEYM	Total number of key index blocks, most significant bits
12	TNKEYL	Total number of key index blocks, least significant bits
13	KEYBAM	Key index sector address, most significant bits
14	KEYBAL	Key index sector address, least significant bits
15	LENKY1	Length of key number 1 in bytes
16	POSKY1	Byte position of key number 1
17	LENKY2	Length of key 2 in bytes
18	POSKY2	Byte position of key 2
19	LENKY3	Length of key 3 in bytes
20	POSKY3	Byte position of key 3
21	LENKY4	Length of key 4 in bytes
22	POSKY4	Byte position of key 4
23	TSFILM	Total sectors allocated for file, most significant bits
24	TSFILL	Total sectors allocated for file, least significant bits

NOTE

The file has been deleted only if word 25 contains all binary zeroes.

25	NAME12	File name, characters 1 and 2
26	NAME34	File name, characters 3 and 4
27	NAME56	File name, characters 5 and 6
28	NAME78	File name, characters 7 and 8
29	OWNR12	Owner name, characters 1 and 2
30	OWNR34	Owner name, characters 3 and 4
31	OWNR56	Owner name, characters 5 and 6
32	OWNR78	Owner name, characters 7 and 8

†The ordered pair (NEDATM, NEDATL) includes any record marked as deleted that has not been written over by file compression.

33	BYTLEN	Record length in bytes. This is the originally specified length and, therefore, may be either an odd or even number of bytes.

Words 34 through 37 of the file control block are used only during file compression. These words are defined as follows:

Word	Mnemonic	Definition
34	PRSRNM	Relative record number of the last processed record, most significant bits
35	PRSRNL	Relative record number of the last processed record, least significant bits
36	NEWRNM	Relative record number of the last record in a set of compressed records, most significant bits
37	NEWRNL	Relative record number of the last record in a set of compressed records, least significant bits

Whenever a sequential file is open, the first ten words of its FCB reside in main memory. Words 9 and 10 are required because of the manner in which FCB subsets are moved into/out of FCBs (see Main-Memory-Resident File Control Block Tables in appendix D). Whenever an indexed file is open, the first twenty-two words of its FCB reside in main memory.

A five-word header is appended to an FCB when the FCB is in main memory. A main memory FCB header is composed of the following:

Word	Mnemonic	Definition	
1	FILEID	File identifier	
		Bits 0-10	FCB index
		Bits 11-15	File manager unit number. (This is an index into the volume information table in main memory. It corresponds to the drive on which the volume is mounted. It is not a system logical unit number.)
2	FCBFLG	FCB flag	
		Bits 0-7	Number of current file users (number of users to which file is currently open)
		Bits 8-15	Unused
3	FILOCK	File lock flag (if nonzero, contains user identification)	
4	NUMSET	Number of sets of locked records in the file (not currently used)	
5	NUMNEW	Number of new records stored on mass memory since file control block was updated on mass memory	

FILE CONTROL BLOCK INDEX ALLOCATION TABLE

The location of the file control block index allocation table (FIAT) on a volume is shown in figure B-1. The FIAT is a bit table used to control the assignment of file control blocks within the file control block table (FCBT). The FIAT occupies a sector of mass memory, but only the first VIMAXF bits are used.

The correspondence of bits in the FIAT to FCBs in the FCBT is as follows:

FIAT Word	Bit	Corresponding FCB Index
1	15	1
1	14	2
.	.	.
.	.	.
.	.	.
1	0	16
2	15	17
2	14	18
.	.	.
.	.	.
.	.	.

At the time of system installation, all bits in the FIAT are zeroed. When a bit in the FIAT is one, it signifies that the corresponding file control block defines a file that has been created and has not been deleted. When a new file is created, the file manager searches the FIAT for the first zero bit. It sets this bit to one and uses the corresponding FCB for the file.

FILE STRUCTURE EXAMPLE

A FORTRAN program, FSDR, was written to demonstrate the file structures generated on mass memory for a set of sample files. The demonstration routine was written to give an example of each of the following:

- A file definition directory including an overflow file definition block
- File control blocks
- File records including records marked as deleted

Routine FSDR is shown in figure B-3. To generate an overflow file definition block, a number of files with the same scatter code were generated. The number of files with the same scatter code must exceed the number of file definition segments (FDSs) that can be stored in one file definition block (FDB). The number of FDSs per FDB was computed as follows. The system installation parameter, VIWPS (number of words per sector on this volume), equals 96. Therefore, NUMFDS, the number of FDSs per file definition block, is computed from the formula in the File Definition Directory section as follows:

$$\text{NUMFDS} = \left\lfloor \frac{(\text{VIWPS}-1)}{9} \right\rfloor = \left\lfloor \frac{(96-1)}{9} \right\rfloor = \left\lfloor \frac{95}{9} \right\rfloor = 10$$

Thus, more than 10 files with the same scatter code must be generated to cause the use of a file definition overflow block.

The formula for a file's scatter code is given in the File Definition Directory section. For a file with file name n_1, n_2, n_3, n_4 and owner w_1, w_2, w_3, w_4, the scatter code c is defined as follows:

$$c = \left[\sum_{i=1}^{4} (n_1 + w_i) \right] \quad (\text{mod MINFDB}) + 1$$

```
      PROGRAM FSDR
C FILE STRUCTURE DEMONSTRATION ROUTINE.
C CREATE  20 SEQUENTIAL FILES, 13 WITH SAME SCATTER CODE.
C STORE 5 RECORDS INTO EACH FILE.  DELETE A RECORD IN EACH OF THE
C FIRST 5 FILES.
      DIMENSION NAME(20)
      DATA NAME/" A5-$G70(M*P,S-?.V/$0Y933*B C D E F G H "/
      DIMENSION IRQBUF(24),IDATA(24),IREC(42),IRECN(2),IODATA(15)
      DATA IDATA /"                    ","SYSVOL  ",16,0,600,0,8*0/
      DATA IODATA/"                    ","SYSVOL  ",0,1,-1/
C INITIALIZE STATUS INDICATORS
      DATA ISTAT,IOSTAT,IPSTAT,ICSTAT/4*0/
      DATA IRSTAT,IDSTAT/2*0/
C FOR RECORD RETRIEVAL PRECEDING RECORD DELETION,
C SET FIRST WORD OF RELATIVE RECORD NUMBER TO ZERO.
      DATA IRECN(1)/0/
      DO 1000 NFILE=1,20
C STORE FIRST TWO CHARACTERS OF FILE NAME INTO DEFINITION ARRAYS
      IDATA(1)= NAME(NFILE)
      IODATA(1)=NAME(NFILE)
      CALL CREATE (IRQBUF,IDATA,ISTAT)
      IF (ISTAT.LT.0) GO TO 9000
C INITIALIZE REQUEST BUFFER
      DO 20 I= 1,24
   20 IRQBUF(I)=0
      CALL OPENFL  (IRQBUF,IODATA.IOSTAT)
      IF(IOSTAT.LT.0) GO TO 9000
C PREPARE RECORD BUFFER
      DO 200 I=1,40
      IREC(I)=NAME(NFILE)+I-1
  200 CONTINUE
      CALL PUTS (IRQBUF,IREC,5,IPSTAT)
      IF (IPSTAT.LT.0) GO TO 9000
C DELETE RECORD "NFILE" IF NFILE LESS THAN OR EQUAL TO FIVE.
      IF (NFILE.GT.5) GO TO 900
C STORE NFILE INTO SECOND WORD OF RELATIVE RECORD NUMBER.
      IRECN(2)=NFILE
C RETRIEVE RECORD TO BE DELETED
      CALL READR (IRQBUF.IREC,IRECN,IRSTAT)
      IF (IRSTAT.LT.0) GO TO 9000
C DELETE RETRIEVED RECORD
      CALL DELREC (IRQBUF,IREC,IDSTAT)
      IF (IDSTAT.LT.0) GO TO 9000
  900 CALL CLOSFL (IRQBUF,ICSTAT)
      IF (ICSTAT.LT.0) GO TO 9000
 1000 CONTINUE
      GO TO 9090
 9000 CONTINUE
C PRINT ERROR MESSAGE
      WRITE (12,7000)   ISTAT,IOSTAT,IPSTAT,IRSTAT,IDSTAT,ICSTAT,
     1                 (IRQBUF(IW),IW=1,24)
      CALL CLOSFL (IRQBUF,ICSTAT)
      GO TO 9095
C DEMONSTRATION ROUTINE COMPLETE

 9090 WRITE (12,7070)
 9095 CONTINUE
      CALL PGMOUT
 7000 FORMAT ( 5X,7HISTAT =,$4,/
     1          5X,7HIOSTAT=,$4,/
     2          5X,7HIPSTAT=,$4,/
     3          5X,7HIRSTAT=,$4,/
     4          5X,7HIDSTAT=,$4,/
     5          5X,7HICSTAT=,$4,/5X,7HIRQBUF=,$4,/23(5X,$4,/)   )
 7070 FORMAT (5X,*TWENTY FILES CREATED*)
      END
```

Figure B-3. File Structure Demonstration Routine (FORTRAN)

For the system used in the demonstration, VIMAXF (number of files permitted on the volume) equals 1024. Therefore, using the formula in the File Definition Directory section:

$$\text{MINFDB} = \left\lceil \frac{\text{VIMAXF}}{\text{NUMFDS}} \right\rceil = \left\lceil \frac{1024}{10} \right\rceil = 103$$

$$= 67_{16}$$

If file 1 and file 2 are two files in this system such that the name of file 1 is:

$$n_{1_1}, n_{1_2}, n_{1_3}, n_{1_4}$$

the owner of file 1 is:

$$w_{1_1}, w_{1_2}, w_{1_3}, w_{1_4}$$

the name of file 2 is:

$$n_{2_1}, n_{2_2}, n_{2_3}, n_{2_4}$$

and the owner of file 2 is:

$$w_{2_1}, w_{2_2}, w_{2_3}, w_{2_4}$$

then the scatter code for the two files is equal only if there exists an integer m such that:

$$\sum_{i=1}^{4} (n_{1_i} + w_{1_i}) - \sum_{i=1}^{4} (n_{2_i} + w_{2_i})$$

$$= m \times 103, (= m \times 67_{16}).$$

Each file in routine FSDR was constructed with an owner string of all ASCII blanks and a name string of all ASCII blanks except the first two characters.

The first two characters of the first file name are defined to be blank, A (equals ASCII code 2041_{16}). The first two characters of the next twelve file names are generated by adding multiples of 67_{16} to 2041_{16} and selecting those sums that represent legitimate ASCII characters.

The first two characters of each file name are stored in the NAME array (figures B-3 and B-4). For example:

$$2041_{16} + 34_{16} \times 67_{16} = 352D_{16}$$

=ASCII code for 5-

$$2041_{16} + A_{16} \times 67_{16} = 2447_{16}$$

=ASCII code for $G

```
3   0002  0002          .00001  ORG   NAME
    0002  2041                  NUM    8257
    0003  352D                  NUM   13613
    0004  2447                  NUM    9287
    0005  3730                  NUM   14128
    0006  284D                  NUM   10317
    0007  2A50                  NUM   10832
    0008  2C53                  NUM   11347
    0009  2021                  NUM   11553
    000A  2E5B                  NUM   11862
    000B  2F24                  NUM   12068
    000C  3059                  NUM   12377
    000D  3933                  NUM   14643
    000E  332A                  NUM   13098
    000F  4220                  NUM   16928
    0010  4320                  NUM   17184
    0011  4420                  NUM   17440
    0012  4520                  NUM   17696
    0013  4620                  NUM   17952
    0014  4720                  NUM   18208
    0015  4820                  NUM   18464
```

Figure B-4. Assembly List of Name Array, FSDR Routine

After running the FSDR routine, the location of the file definition directory (FDD) is obtained from the main memory volume information table for this volume. (See the sample in figure D-4.) The sector address of the FDD is (0,7017). Portions of the FDD as dumped by ODEBUG are shown in figure B-5.

The address of the file control block (FCB) table is determined by the formula in figure B-1 using the value of VINFDB from the appropriate main memory volume information table (see the sample in figure D-4). In the system used in the example, VINFDB equals 81_{16}. Thus, the sector address of the FCB table is:

$$(0,7017_{16}) + (0, 81_{16}) = (0, 7098_{16}).$$

The first six files created by the routine, FSDR, have FCB indices $C5_{16}$, $C7_{16}$, $C8_{16}$, $C9_{16}$, CA_{16}, and CB_{16}, respectively. (Refer to FDB number 54_{16}, figure B-5.)

The sector addresses of the FDBs for these files are $7098_{16} + C5_{16} = 715D_{16}$, $715F_{16}$, 7160_{16}, 7161_{16}, 7162_{16}, and 7163_{16}, respectively. These FCBs are shown in figure B-6.

Words 4 and 5 of each FCB give the sector address of the first record of the file (see the FCB format in the File Control Block Table section).

The sector address of the first record of each of the first six files created by the FSDR program may be obtained from the FCBs shown in figure B-6. For example, file A has sector address (1,314D).

Using these addresses, the file records for these files are dumped as shown in figure B-7. Records deleted in the FSDR routine are indicated in the figure.

```
SECTOR ADDRESS ->                                   0000 7017
                                                    0000 5052 464F 4530 3631 2020 2020 2020
                                                    2020 0078 5341 4F55 5420 2020 2020 2020
                                                    2020 2020 00AF 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000   FIRST FDB
                                                    0000 0000 0000 0000 0000 0000 0000 0000   IN FDD
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
SECTOR ADDRESS ->                                   0000 7042
FDB HEADER; ZERO IMPLIES NO OVERFLOW             -> 0000 5052 4649 4E30 3033 2020 2020 2020
BLOCK FOR THIS SCATTER CODE                         2020 0008 5445 524D 5320 2020 2020 2020
                                                    2020 2020 0095 4720 2020 2020 2020 2020 <- FDS FOR FILE G, FCB
                                                    2020 2020 2020 00D8 0000 0000 0000 0000    INDEX = D8(16)
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000   FDB NUMBER 2C(16)
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
SECTOR ADDRESS ->                                   0000 706A
FDB HEADER CONTAINS 68(16). THIS IMPLIES FDB     -> 0068 5052 4641 5230 3137 2020 2020 2020
NUMBER 68(16) (SECTOR 68(16) OF FDD) CONTAINS       2020 003A 4152 5452 414E 5320 2020 2020
OVERFLOW BLOCK FOR THIS SCATTER CODE.               2020 2020 009F 2041 2020 2020 2020 2020 <- FDS FOR FILE A; FCB
                                                    2020 2020 2020 00C5 352D 2020 2020 2020 <- INDEX = C5(16)
                                                    2020 2020 2020 2020 00C7 2447 2020 2020    FDS FOR FILE 5-; FCB
                                                    2020 2020 2020 2020 2020 00C8 3730 2020    INDEX = C7(16)
                                                    2020 2020 2020 2020 2020 2020 00C9 284D
                                                    2020 2020 2020 2020 2020 2020 2020 00CA    FDB NUMBER 54(16)
                                                    2A50 2020 2020 2020 2020 2020 2020 2020
                                                    00CB 2C53 2020 2020 2020 2020 2020 2020
                                                    2020 00CC 2D21 2020 2020 2020 2020 2020 <- FDS FOR FILE -!; FCB
                                                    2020 2020 00CD 0000 0000 0000 0000 0000    INDEX = CD(16)
SECTOR ADDRESS ->                                   0000 7074
FDB HEADER; ZERO IMPLIES NO OVERFLOW             -> 0000 5052 4649 4E30 3133 2020 2020 2020
BLOCK FOR THIS SCATTER CODE.                        2020 0012 4820 2020 2020 2020 2020 2020 <- FDS FOR FILE H; FCB
                                                    2020 2020 00D9 0000 0000 0000 0000 0000    INDEX = D9(16)
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000   FDB NUMBER 5E(16)
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
                                                    0000 0000 0000 0000 0000 0000 0000 0000
```

Figure B-5. Example of File Definition Directory Entries (Sheet 1 of 2)

SECTOR ADDRESS

FDS FOR FILE B

```
0000  707D
0000  5052  464F  4530  3630  2020  2020  2020
2020  0077  4220  2020  2020  2020  2020  2020
2020  2020  00D3  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

FDB NUMBER 67_{16}

FDB HEADER; ZERO IMPLIES NO MORE OVER-FLOW BLOCKS FOR THIS SCATTER CODE

```
0000  707E
0000  2E56  2020  2020  2020  2020  2020  2020
2020  00CE  2F24  2020  2020  2020  2020  2020
2020  2020  00CF  3059  2020  2020  2020  2020
2020  2020  2020  00D0  3933  2020  2020  2020
2020  2020  2020  2020  00D1  332A  2020  2020
2020  2020  2020  2020  2020  00D2  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

FDS FOR FILE 0Y

FDS FOR FILE 93

FDB NUMBER 68 (OVER-FLOW BLOCK FOR FDB 54)

FDS FOR FILE 3*

Figure B-5. Example of File Definition Directory Entries (Sheet 2 of 2)

SECTOR ADDRESS OF FIRST RECORD

```
0000 715D
0008 0000 0258 0001 314D 0000 0000 0005
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0032
2041 2020 2020 2020 2020 2020 2020 2020
0010 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 715F
0008 0000 0258 0001 35E1 0000 0000 0005
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0032
352D 2020 2020 2020 2020 2020 2020 2020
0010 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 7160
0008 0000 0258 0001 3614 0000 0000 0005
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0032
2447 2020 2020 2020 2020 2020 2020 2020
0010 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 7161
0008 0000 0258 0001 3647 0000 0000 0005
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0032
3730 2020 2020 2020 2020 2020 2020 2020
0010 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

FCB FOR FILE A

SECTOR ADDRESS OF FIRST RECORD

FCB FOR FILE 5-

SECTOR ADDRESS OF FIRST RECORD

FCB FOR FILE $G

SECTOR ADDRESS OF FIRST RECORD

FCB FOR FILE 70

Figure B-6. Example of File Control Blocks (Sheet 1 of 2)

```
                                    SECTOR ADDRESS OF FIRST RECORD
0000  7162
0008  0000  0258  0001  367A  0000  0000  0005
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0032
284D  2020  2020  2020  2020  2020  2020  2020
0010  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000     FCB FOR FILE (M
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  7163                                         SECTOR ADDRESS OF
0008  0000  0258  0001  36AD  0000  0000  0005     FIRST RECORD
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0032
2A50  2020  2020  2020  2020  2020  2020  2020
0010  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000     FCB FOR FILE *P
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

Figure B-6. Example of File Control Blocks (Sheet 2 of 2)

FILE RECORDS FOR FILE A

RECORD 1 MARKED AS DELETED

END-OF-FILE CODE

0001	314D						
5E5E	2042	2043	2044	2045	2046	2047	2048
2049	204A	204B	204C	204D	204E	204F	2050
2051	2052	2053	2054	2055	2056	2057	2058
2059	205A	205B	205C	205D	205E	205F	2060
2061	2062	2063	2064	2065	2066	2067	2068
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000

FILE RECORDS FOR FILE 5-

RECORD 2 MARKED AS DELETED

0001	35E1						
352D	352E	352F	3530	3531	3532	3533	3534
5E5E	3536	3537	3538	3539	353A	353B	353C
353D	353E	353F	3540	3541	3542	3543	3544
3545	3546	3547	3548	3549	354A	354B	354C
354D	354E	354F	3550	3551	3552	3553	3554
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000

FILE RECORDS FOR FILE $G

RECORD 3 MARKED AS DELETED

0001	3614						
2447	2448	2449	244A	244B	244C	244D	244E
244F	2450	2451	2452	2453	2454	2455	2456
5E5E	2458	2459	245A	245B	245C	245D	245E
245F	2460	2461	2462	2463	2464	2465	2466
2467	2468	2469	246A	246B	246C	246D	246E
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000

FILE RECORDS FOR FILE 70

RECORD 4 MARKED AS DELETED

0001	3647						
3730	3731	3732	3733	3734	3735	3736	3737
3738	3739	373A	373B	373C	373D	373E	373F
3740	3741	3742	3743	3744	3745	3746	3747
5E5E	3749	374A	374B	374C	374D	374E	374F
3750	3751	3752	3753	3754	3755	3756	3757
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000

Figure B-7. Example of File Records Including Records Marked as Deleted (Sheet 1 of 2)

FILE RECORDS FOR FILE M

RECORD 5 MARKED AS DELETED

FILE RECORDS FOR FILE *P (NO RECORDS DELETED IN THIS FILE)

0001	367A						
284D	284E	284F	2850	2851	2852	2853	2854
2855	2856	2857	2858	2859	285A	285B	285C
285D	285E	285F	2860	2861	2862	2863	2864
2865	2866	2867	2868	2869	286A	286B	286C
5E5E	286E	286F	2870	2871	2872	2873	2874
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0001	36AD						
2A50	2A51	2A52	2A53	2A54	2A55	2A56	2A57
2A58	2A59	2A5A	2A5B	2A5C	2A5D	2A5E	2A5F
2A60	2A61	2A62	2A63	2A64	2A65	2A66	2A67
2A68	2A69	2A6A	2A6B	2A6C	2A6D	2A6E	2A6F
2A70	2A71	2A72	2A73	2A74	2A75	2A76	2A77
5F5F	5F5F	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000
0000	0000	0000	0000	0000	0000	0000	0000

Figure B-7. Example of File Records Including Records Marked as Deleted (Sheet 2 of 2)

KEY INDEX STRUCTURE

The general method of key index storage is described in Key Storage, section 1. An indexed file has from one to four keys. Each key defined for a file has its own key index. All the key indices for a given file make up that file's key index structure. For each record stored into an indexed file, a key information segment (KIS) is stored into the lowest level of the key index. This KIS points directly to the record. The KISs in the lowest level are ordered by key value. The lowest level is called a sequence set for the key. The KISs in each level of an index are grouped together in key information blocks (KIBs). Each KIS in an upper level of an index points to a KIB in the next lowest level. The highest level of a key index contains only one KIB.

The format of a KIS and the format of a KIB are shown in the Key Information Block section. Storage within a key index is discussed in the Storage Within a Key Value section. Retrieval by key value is discussed in the Retrieval by Key Value section. Location of the key index structure on mass memory is discussed in the Key Index Control Parameters section. An example of a key index structure is given in the Key Index Example section.

KEY INDEX CONTROL PARAMETERS

The parameters that control a file's key index structure are contained in words 9 through 22 of the file's file control block as defined in the File Control Block Table section. The ordered pair (TNKEYM,TNKEYL) is the total number of key information blocks allocated, not the current number of blocks used. The ordered pair (KEYBAM,KEYBAL) is the beginning sector address of the key index structure.

KEY INFORMATION BLOCK

A key information block consists of a six-word header together with a set of KISs. The format of a KIB header is as follows:

Word	Mnemonic	Description
1	NUMKIS	Number of KISs in this KIB.
2 3	NKIBNM NKIBNL	The ordered pair (NKIBNM, NKIBNL) is meaningful only for KIBs in the lowest level of the key index structure. For the lowest level, (NKIBNM, NKIBNL) is a relative KIB number pointing to the KIB on this level with the set of key values following the key values in this KIB. If (NKIBNM, NKIBNL) = (0,0), this is the KIB with the highest key values stored in the file.
4 5	PKIBNM PKIBNL	The ordered pair (PKIBNM, PKIBNL) is the relative KIB number pointing to the KIB in the next highest level that contains the KIS pointing to this KIB.
6	KIBTYP	KIB type 0 Highest level block for this key 1 Intermediate level block 2 Lowest level block for this key

The format of each KIS is as follows:

- Key value (left-justified).
- A three-byte relative record number pointing to a record or a three-byte relative KIB number pointing to a KIB in the next lowest level of the key index structure.

If KEYLEN equals key length in bytes, and KISLEN is the length of a key information segment, then:

$$\text{KISLEN} = \left\lceil \frac{\text{KEYLEN}}{2} \right\rceil + 2$$

Where $\lceil y \rceil$ is the least integer greater than or equal to y.

The length of a key information block is the system installation parameter, KIBSEC (see the Main-Memory-Resident File Manager Operation Installation Parameters, appendix D). Each KIB in the system is KIBSEC sectors long. The number of KISs that fit into a key information block is computed as follows. Let VIWPS equal the number of words per sector for the volume on which the file resides. Then if KISKIB equals the maximum number of KISs that fit into a KIB:

$$\text{KISKIB} = \left\lfloor \frac{\text{KIBSEC} \bullet \text{VIWPS-6}}{\text{KISLEN}} \right\rfloor$$

Where $\lfloor y \rfloor$ is the greatest integer less than or equal to y.

STORAGE WITHIN A KEY INDEX

When a new record is stored into an indexed file, a key information segment (KIS) is stored into a key information block (KIB) in the sequence set for each key index for the file. The file manager may have to shift other KISs within a KIB, so that the KISs remain in order by key value. The KIS pointing to the KIB is modified if necessary. It may be necessary to create a new sequence set KIB in which to store a new KIS. In this case, the original full KIB is split so that half the existing KISs in the block remain in that KIB and half are moved to the new KIB. An odd number of KISs are split so that the new KIB has one fewer KIS than the old KIB. An exception to the halfway split occurs in the primary key index for an indexed file for which the user specified that records would be presented in order by primary key value. For such a file, only the new KIS is stored in the new sequence set KIB. A second exception to the halfway split occurs when a split at the halfway point would cause KISs for the same key value to be split between two KIBs. In this case, the KISs for the same key value are stored into the same KIB if possible. (The second type of exception is illustrated in figure C-2; see the index for 11 records stored, the index for 13 records stored, and the index for 16 records stored.) Whenever a new sequence set KIB is created, a new KIS is stored in the level above the sequence set to point to the new KIB. Blocks in levels above the sequence set are split in the same way. When a split occurs in the highest level of an index, a new level is created.

The KIB stored at the start of the key index structure has a relative KIB number (0,1). The KIB stored immediately following KIB (0,1) is KIB (0,2), etc. The highest level KIB for the primary key is always stored in KIB (0,1). If a secondary key exists for the file, the highest level KIB for the secondary key is always stored in KIB (0,2). Similarly, if key 3 exists its highest level KIB is stored in KIB (0,3), and if key 4 exists its highest level KIB is stored in KIB (0,4).

Storage within a key index is illustrated in figure C-1. An example of key index storage is shown in figure C-2. In this example, KIBSEC equals 1, VIWPS equals 96, and KEYLEN equals 29 bytes. Therefore:

$$\text{KISLEN} = 2 + \left\lceil \frac{29}{2} \right\rceil = 17$$

$$\text{KISKIB} = \left\lfloor \frac{96-6}{17} \right\rfloor = \left\lfloor \frac{90}{17} \right\rfloor$$

= 5 (there are 5 key information segments per key information block).

Another example of key indices is given in the Key Index Example section. The secondary key in that example has the same set of key values as shown in figure C-2. KIBs in the Key Index Example section split in a different way than shown in figure C-2 because KIBSEC has a different value in that section.

RETRIEVAL BY KEY VALUE

It may be observed from figure C-1 and the example in figure C-2 that for a key index with n levels, it is necessary for the file manager to search n key information blocks to find the pointer to the first record with a given key value. If there is no record for the specified key value, the number of searches required to determine this is less than or equal to n. If there is a record with a higher key value, the file manager makes all n searches, as the file manager is designed to find the next record as ordered by key value when the specified key value is missing.

KEY INDEX EXAMPLE

A FORTRAN program, KIDR, is included to demonstrate the key index structure generated on mass memory for a sample file with two keys. The indexed file EXAMPLE is created by KIDR (see figure C-3). Program KIDR then stores 16 records into the file. Primary key values for these 16 records are 100_{16}, 200_{16}, 300_{16}, etc. The secondary key values for the 16 stored records are shown in figure C-2; that is, the secondary key values are 6, 16, 10, 7, and so forth.

After executing the KIDR routine, the file control block (FCB) for the EXAMPLE file is dumped, as shown in figure C-4. In general, an FCB is located by examining the file definition directory on the appropriate volume. In locating the FCB for file EXAMPLE, it was known that no file has been deleted since the system was initialized. Therefore, the index of file EXAMPLE's FCB within the FCB table is obtained from the main memory volume information table parameter VICURF. Using the location of the key index structure in the FCB in figure C-4, the key index blocks are dumped, as shown in figure C-5. Words 9 and 10 of the FCB indicate the next available key information block (KIB) is KIB (0,5). This means there are four existing KIBs. Each KIB is three sectors long, since the value of KIBSEC for this system is 3. The key index structure for this file is obtained by dumping

$$4 \text{ KIBs} \times \frac{3 \text{ sectors}}{\text{KIB}} = 12 \text{ sectors}$$

starting at the sector address of a key index of 1,708. These 12 sectors are shown in figure C-5. The key index for the primary key consists of KIB (0,1) and KIB (0,3). The key index for the secondary key consists of KIB (0,2) and KIB (0,4).

The location of the EXAMPLE file records is obtained from the file control block in figure C-4. The file's records are dumped as shown in figure C-6.

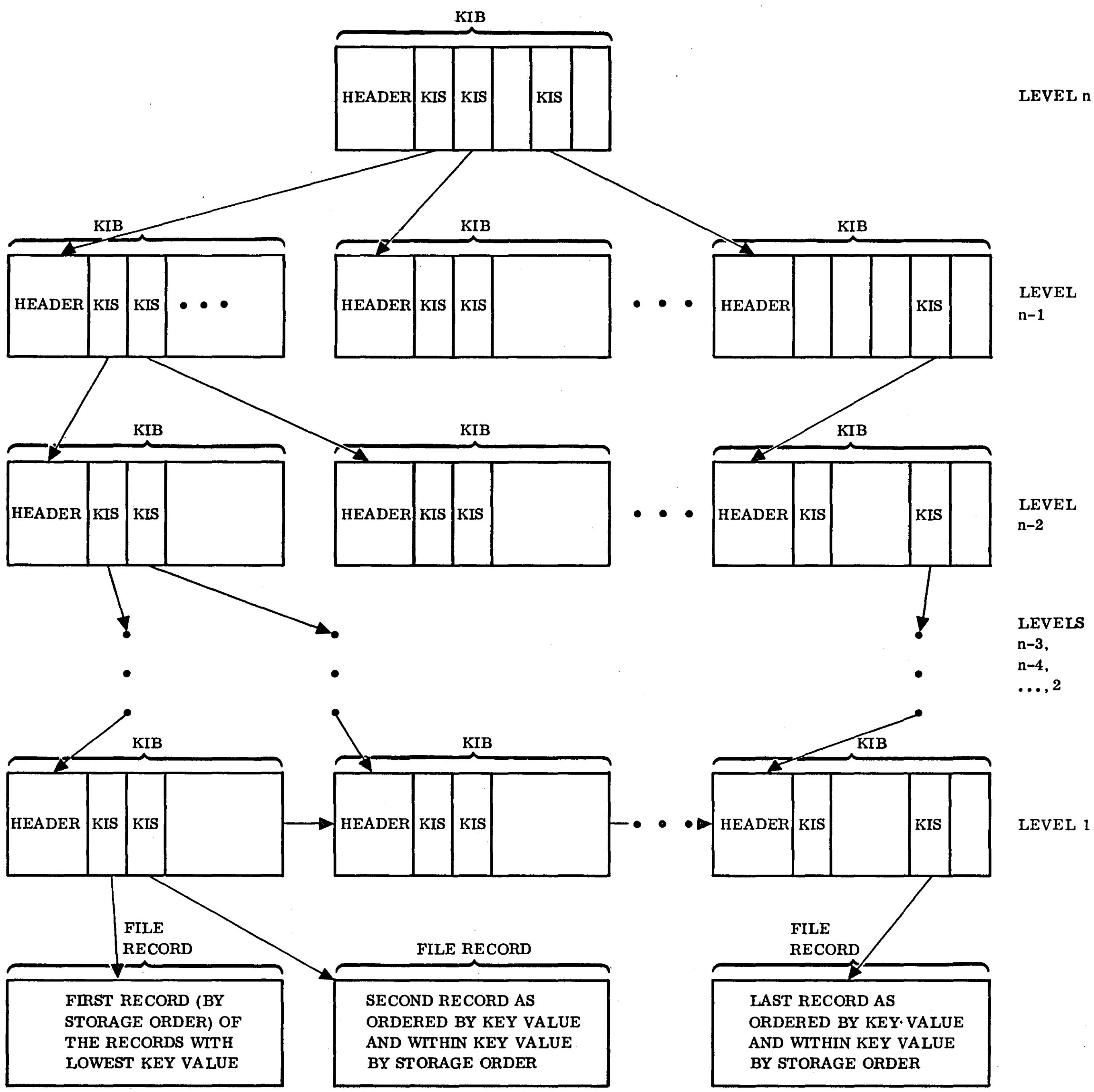


Figure C-1. Key Index Storage

THE FOLLOWING NOTATION REPRESENTS A KIB:

$\boxed{h_i}\boxed{n_1}\boxed{n_2}\boxed{n_3}\boxed{n_4}\boxed{n_5}$

WHERE: (0, 1) IS THE RELATIVE KIB NUMBER AS NUMBERED WITHIN KIBs FOR THIS KEY.

h_i IS THE HEADER.

n_1, n_2, n_3, n_4, n_5 ARE KISs WITH KEY VALUES OF n_1, n_2, n_3, n_4, n_5, RESPECTIVELY.

THE FOLLOWING NOTATION REPRESENTS A FILE RECORD:

$\boxed{r}$

WHERE: (0, r) IS THE RELATIVE RECORD NUMBER OF THE RECORD.

Relative Record Number (0, r)	Key Value (n)
(0, 1)	6
(0, 2)	16
(0, 3)	10
(0, 4)	7
(0, 5)	5
(0, 6)	8
(0, 7)	6
(0, 8)	7
(0, 9)	10
(0, 10)	15
(0, 11)	9
(0, 12)	8
(0, 13)	9
(0, 14)	6
(0, 15)	9
(0, 16)	9

Figure C-2. Example of Key Index Storage (Sheet 1 of 5)

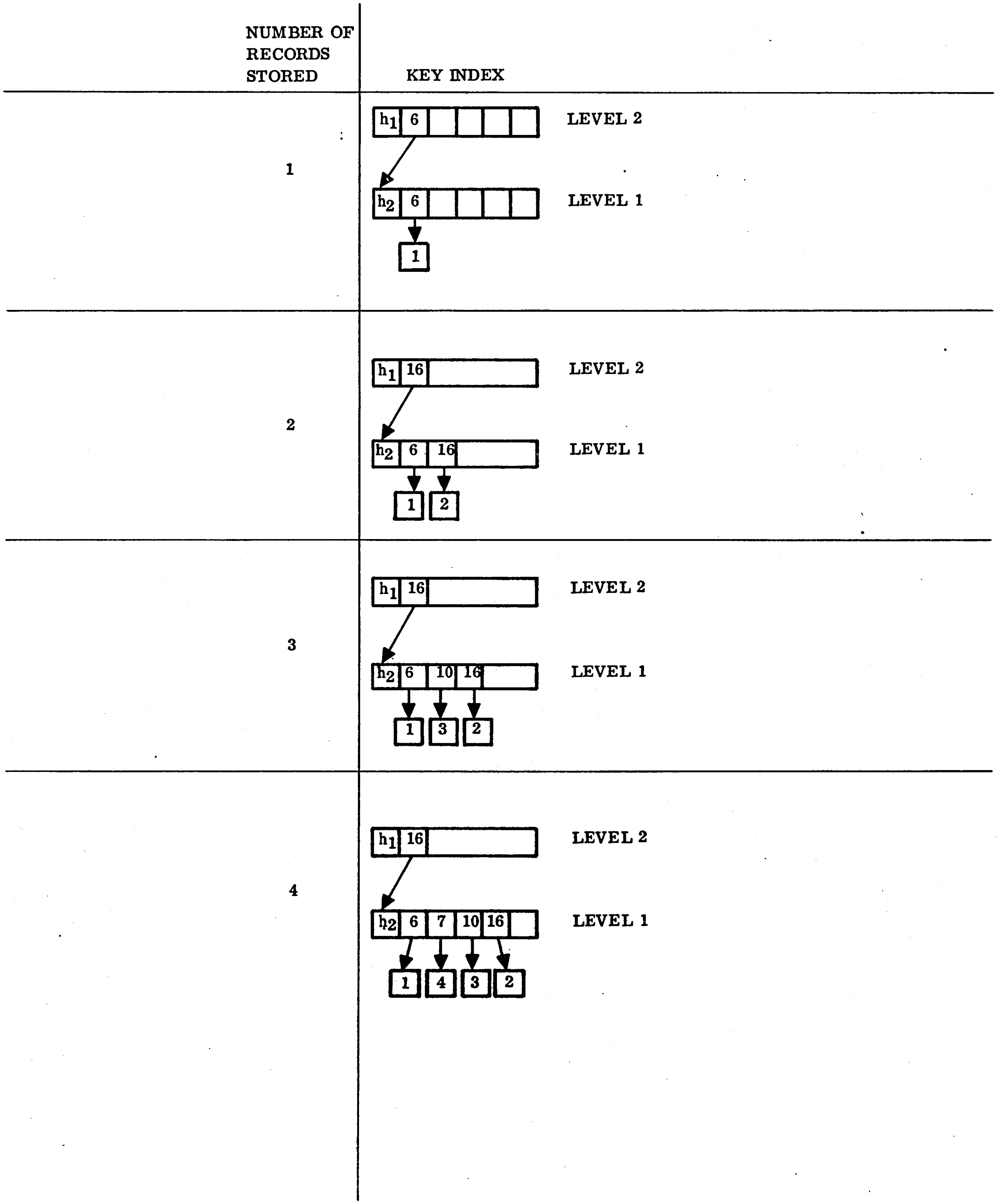


Figure C-2. Example of Key Index Storage (Sheet 2 of 5)

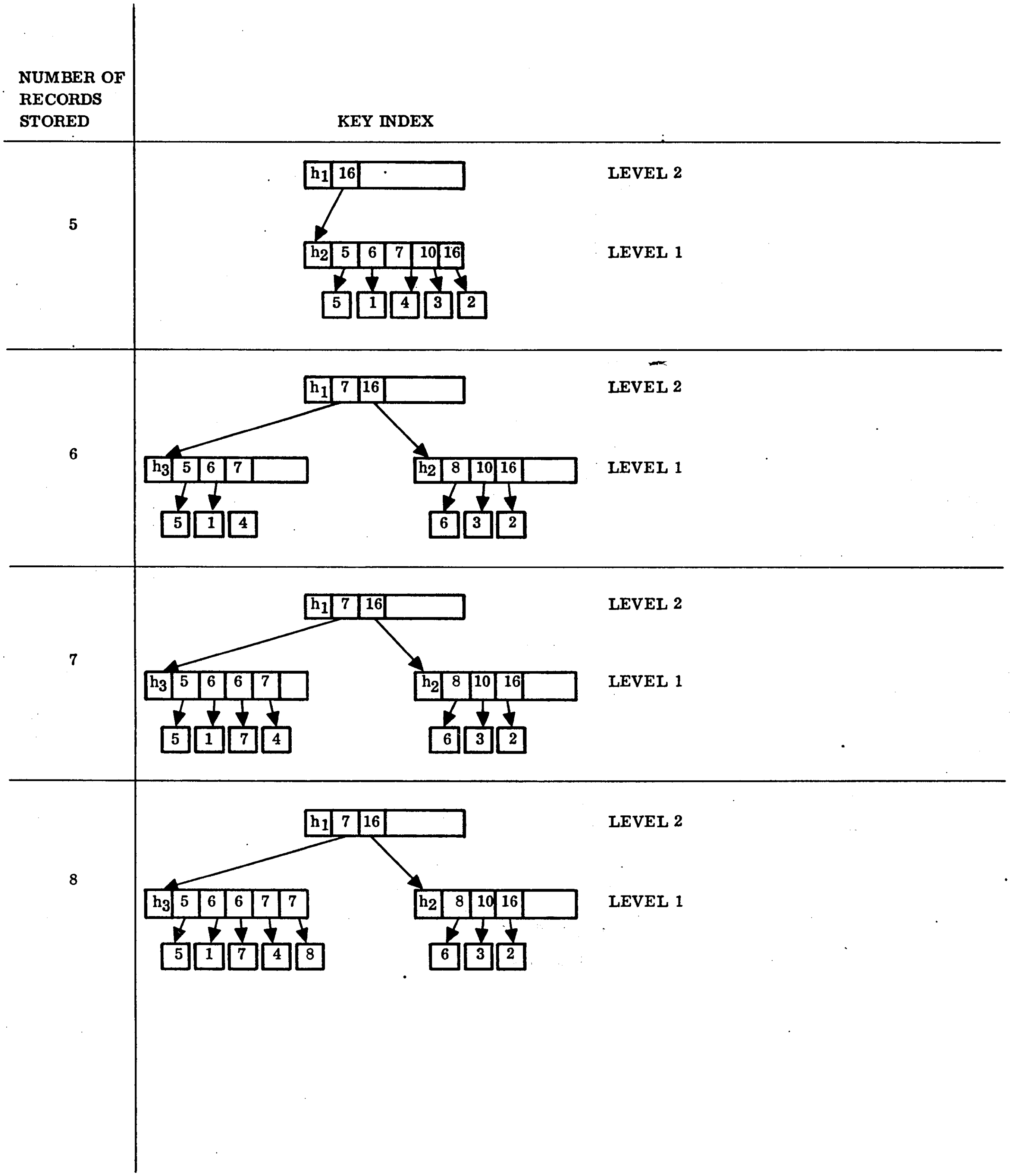


Figure C-2. Example of Key Index Storage (Sheet 3 of 5)

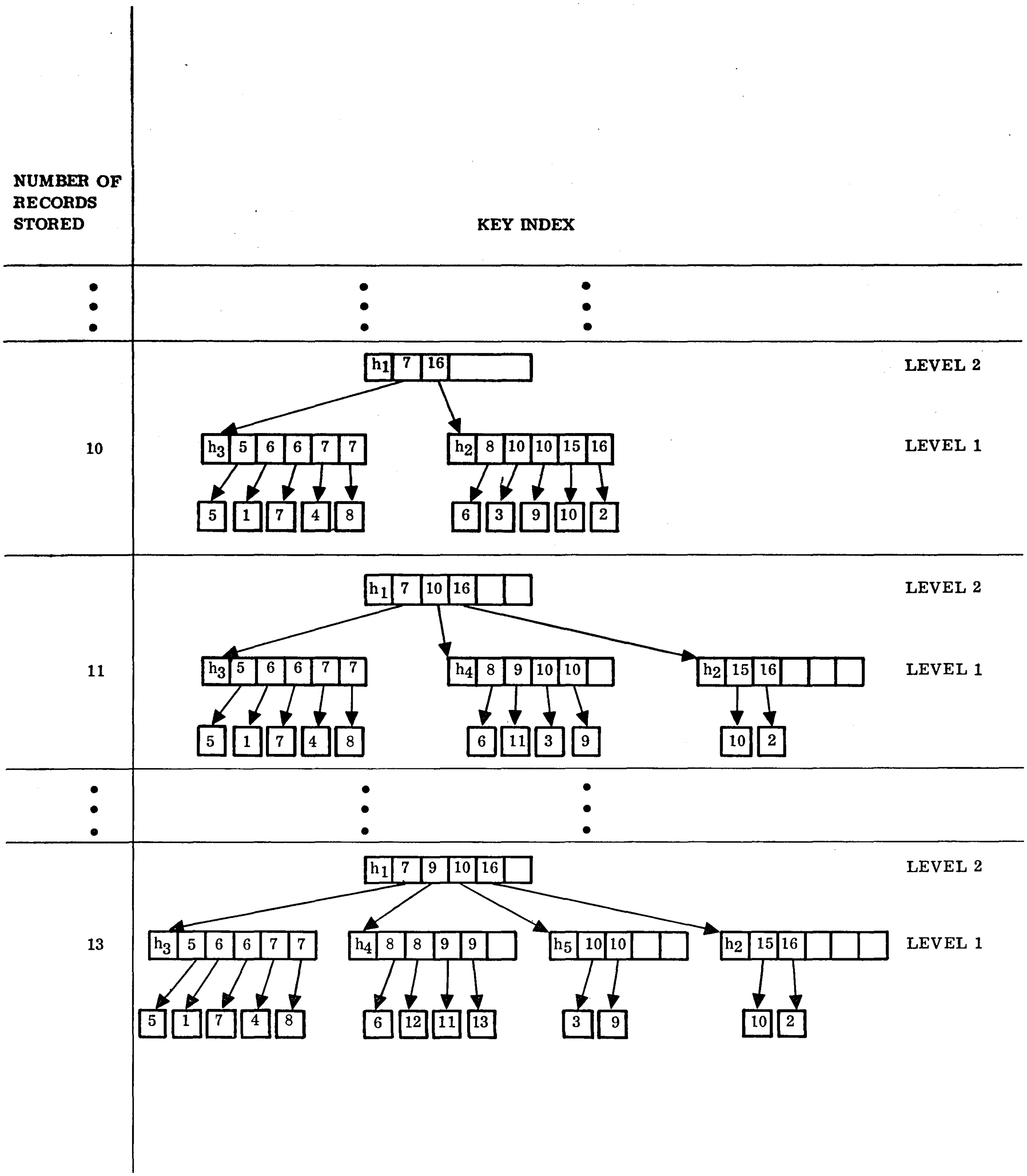


Figure C-2. Example of Key Index Storage (Sheet 4 of 5)

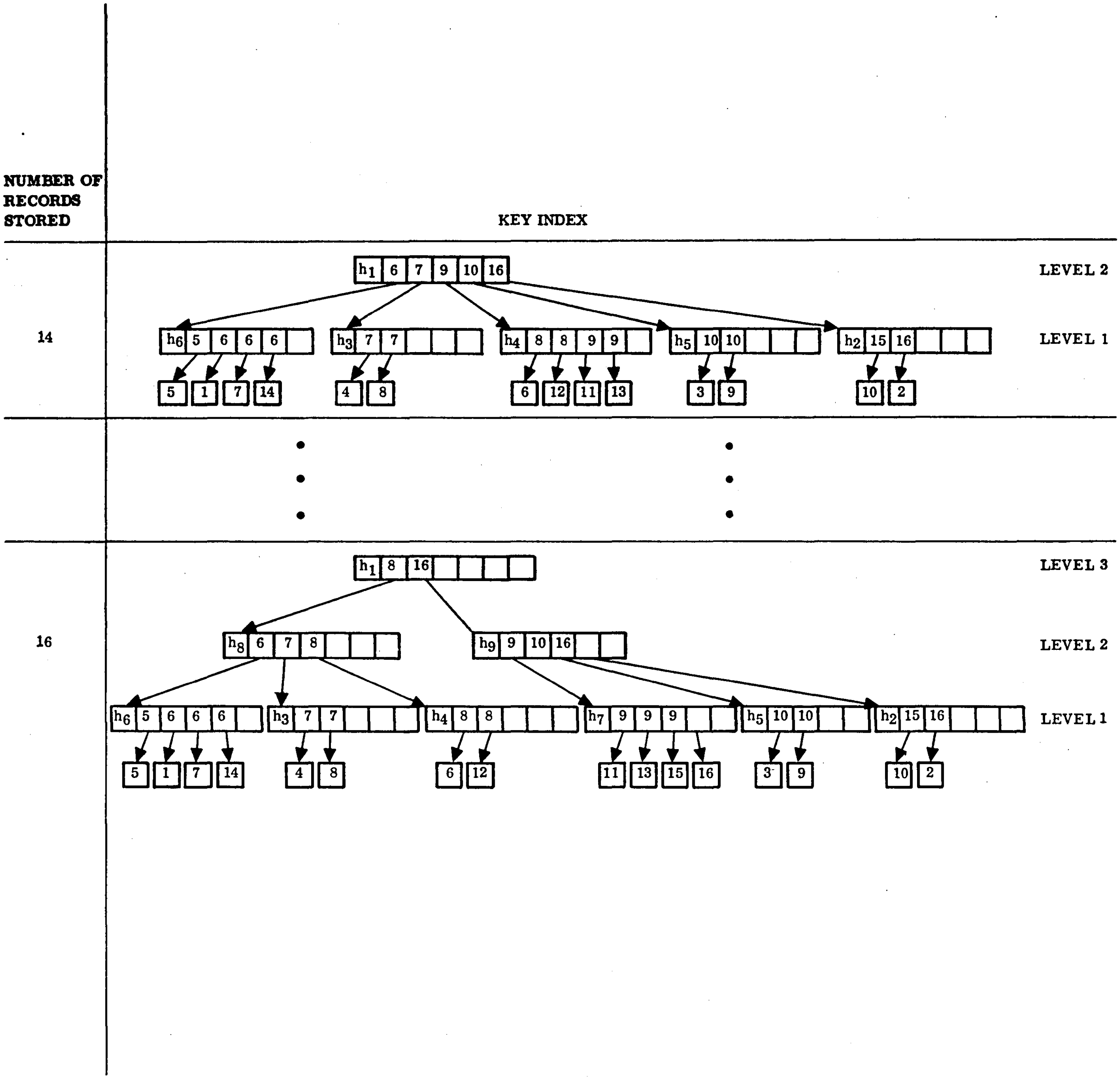


Figure C-3. Example of Key Index Storage (Sheet 5 of 5)

```
      PROGRAM KIDR
C KEY INDEX DEMONSTRATION ROUTINE
C PURPOSE OF THIS PROGRAM IS TO CREATE AN INDEXED FILE AND
C STORE 16 RECORDS IN FILE.
C KEY VALUES FOR PRIMARY KEY ARE $100,$200,$300,ETC.
C KEY VALUES FOR KEY 2 CORRESPOND TO KEY VALUES IN EXAMPLE
C IN FIGURE C-2.
C RECORD FORMAT
C   WORD  CONTENTS
C   ----  --------
C    1    LEFT BYTE OF WORD 1 IS UNUSED
C   1-15  KEY 2      (KEY 2 STARTS IN RIGHT BYTE OF WORD 1)
C    16   KEY 1
C RESERVE SPACE FOR FILE REQUEST BUFFER AND FILE INFORMATION BUFFER
      INTEGER REQBUF(24)
      DIMENSION IDATA(24),IODATA(15)
C SET FILE NAME = EXAMPLE
C SET FILE OWNER= 49504
C VOLUME IS SYSVOL
C SET RECORD LENGTH = 32 BYTES (16 WORDS)
C SET MAXIMUM NUMBER RECORDS TO 500.
C SET FILE TYPE INDEXED WITH RECORDS PRESENTED IN ORDER WITH RESPECT
C TO PRIMARY KEY, RECORDS NOT SECTOR-ALIGNED.
C PRIMARY KEY IS STORED IN BYTES 31-32
C SECONDARY KEY IS STORED IN BYTES 2-30
      DATA IDATA /'EXAMPLE 49504   SYSVOL  ',32,0,500,$4001,2,31,29,2,
     1             4*0/
      DATA IODATA /'EXAMPLE 49504   SYSVOL  ',0,1,0/
      DIMENSION KEYVL2(16)
      DATA KEYVL2 /6,16,10,7,5,8,6,7,10,15,9,8,9,6,9,9/
      INTEGER RECBUF(18)
      DATA RECBUF /18*0/, KEY1/0/
C INITIALIZE STATUS INDICATORS
      DATA ISTAT,IOSTAT,IWSTAT,ICSTAT/4*0/
C CREATE INDEXED FILE
      CALL CREATE (REQBUF,IDATA,ISTAT)
      IF (ISTAT.NE.0) GO TO 9000
C INITIALIZE REQUEST BUFFER
      DO 20 I= 1,24
   20 REQBUF(I)=0
      CALL OPENFL (REQBUF,IODATA,IOSTAT)
      IF ( IOSTAT.LT.0) GO TO 9000
C STORE 16 INDEXED RECORDS INTO FILE
      DO 1000 IND=1,16
C PRIMARY KEY VALUE = RECORD NUMBER * $100
      KEY1 = KEY1 + $0100
      RECBUF(16) = KEY1
C PICK UP SECONDARY KEY VALUE FROM KEYVL2 ARRAY.
      RECBUF(15)= KEYVL2(IND)
      CALL WRITER (REQBUF,RECBUF,KEY1,IWSTAT)
      IF (IWSTAT.LT.0) GO TO 9000
 1000 CONTINUE
      CALL CLOSFL (REQBUF,ICSTAT)
      IF (ICSTAT.LT.0) GO TO 9000
      GO TO 9090
C PRINT ERROR INFORMATION AND EXIT.
 9000 CONTINUE
      WRITE (12,7000)  ISTAT,IOSTAT,IWSTAT,ICSTAT,KEY1,REQBUF
      GO TO 9095
 9090 WRITE (12,7070)
C DEMONSTRATION ROUTINE COMPLETE
 9095 CONTINUE
      CALL PGMOUT
 7000 FORMAT ( 5X,7HISTAT =,$4,/
     1         5X,7HIOSTAT=,$4,/
     2         5X,7HIWSTAT=,$4,/
     3         5X,7HICSTAT=,$4,/
     4         5X,7HKEY1  =,$4,/7HREQBUF=,$4,/23(5X,$4,/) )
 7070 FORMAT (5X,*SIXTEEN INDEXED RECORDS STORED*)
      END
```

Figure C-3. Key Index Demonstration Routine (FORTRAN)

SECTOR ADDRESS OF FIRST RECORD = (1, 3D00)

SECTOR ADDRESS OF KEY INDEX = (1, 3D54)

```
0000  7172
0010  0000  01F4  0001  3D00  4001  0000  0010
0000  0005  0000  0054  0001  3D54  0002  001F
001D  0002  0000  0000  0000  0000  0000  0150
4558  414D  504C  4520  3439  3530  3420  2020
0020  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

Figure C-4. Example of FCB for Indexed File

INDICATES HIGHEST LEVEL BLOCK FOR THIS KEY

KIB HEADER; INDICATES 1 KIS IN THIS KIB

KIS WITH KEY VALUE = 1000_{16}; POINTER TO KIB IN NEXT LOWEST LEVEL = (0, 3)

KIB (0, 1)

```
0001  3D54
0001  0000  0000  0000  0000  0000  1000  0000
0003  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0001  3D55
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0001  3D56
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

LEVEL 1, PRIMARY KEY

Figure C-5. Example of Key Index Blocks (Sheet 1 of 4)

KIB HEADER; FIRST WORD INDICATES 1 KIS IN THIS KIB; WORD 6 INDICATES HIGHEST LEVEL FOR THIS KEY.

KIS WITH KEY VALUE = 16 (=10_{16})†; POINTER TO KIB IN NEXT LOWEST LEVEL = (0, 4)

KIB (0, 2)

```
0001  3D57
0001  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  1000  0000  0004  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0001  3D58
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0001  3D59
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

LEVEL 1, SECONDARY KEY

† NOTE THAT KEY VALUE IS LEFT JUSTIFIED IN KIS

Figure C-5. Example of Key Index Blocks (Sheet 2 of 4)

KIB HEADER; FIRST WORD INDICATES 16 KISs IN THIS KIB; WORDS 4 AND 5 INDICATE KIS IN KIB (0, 1) POINTS TO THIS KIB; WORD 6 INDICATES THIS IS LOWEST LEVEL OF THIS KEY.

KIS WITH KEY VALUE = $F00_{16}$, RELATIVE RECORD NUMBER POINTER = $(0, F_{16})$

KIS WITH KEY VALUE = 100_{16}, RELATIVE RECORD NUMBER POINTER = (0, 1)

```
0001 3D5A
0010 0000 0000 0000 0001 0002 0100 0000
0001 0200 0000 0002 0300 0000 0003 0400
0000 0004 0500 0000 0005 0600 0000 0006
0700 0000 0007 0800 0000 0008 0900 0000
0009 0A00 0000 000A 0B00 0000 000B 0C00
0000 000C 0D00 0000 000D 0E00 0000 000E
0F00 0000 000F 1000 0000 0010 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0001 3D5B
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0001 3D5C
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

KIB (0, 3)

LEVEL 2 (LOWEST LEVEL), PRIMARY KEY

Figure C-5. Example of Key Index Blocks (Sheet 3 of 4)

	0001	3D5D						
KIB HEADER	0010	0000	0000	0000	0002	0002	0000	0000
KIS WITH KEY VALUE 5, RELATIVE RECORD NUMBER POINTER = (0, 5)	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0500	0000	0005	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0600	0000	0001
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0600	0000
	0007	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0600
	0000	000E	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0700	0000	0004	0000	0000	0000	0000	0000
	0001	3D5E						
†	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0700	0000	0008	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0800	0000	0006	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0800	0000	000C	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0900	0000	000B	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0900	0000	000D
	0000	0000	0000	0000	0000	0000	0000	0000
†	0000	0000	0000	0000	0000	0000	0900	0000
	0001	3D5F						
†	000F	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0900
	0000	0010	0000	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0A00	0000	0003	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0A00	0000	0009	0000	0000	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0F00	0000	000A	0000	0000	0000
KIS WITH KEY VALUE = 16 (= 10_{16}), RELATIVE RECORD NUMBER POINTER = (0, 2)	0000	0000	0000	0000	0000	0000	0000	0000
	0000	0000	0000	1000	0000	0002	0000	0000
	0000	0000	0000	0000	0000	0000	0000	0000

KIB (0, 4)

LEVEL 2 (LOWEST LEVEL), SECONDARY KEY

† DASHED LINES INDICATE A SECTOR BOUNDARY WITHIN A KIS

Figure C-5. Example of Key Index Blocks (Sheet 4 of 4)

```
                                  0001 3D00
RECORD 1; PRIMARY KEY             0000 0000 0000 0000 0000 0000 0000 0000
VALUE = 100(16); SECONDARY   ---> 0000 0000 0000 0000 0000 0000 0006 0100
KEY VALUE = 6
RECORD 2 ------------------------> 0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0010 0200
RECORD 3 ------------------------> 0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 000A 0300
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0007 0400
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0005 0500
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0008 0600
                                  0001 3D01
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0006 0700
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0007 0800
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 000A 0900
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 000F 0A00
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0009 0B00
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0008 0C00
                                  0001 3D02
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0009 0D00
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0006 0E00
                                  0000 0000 0000 0000 0000 0000 0000 0000
                                  0000 0000 0000 0000 0000 0000 0009 0F00
RECORD 16 PRIMARY  --------------> 0000 0000 0000 0000 0000 0000 0000 0000
KEY VALUE = 1000(16),             0000 0000 0000 0000 0000 0000 0009 1000
SECONDARY KEY VALUE               5F5F 5F5F 3030 3030 3030 3030 3330 3720
= 9.                              2020 2020 2020 2020 2020 2020 2020 2020
                                  2020 2020 2020 2020 2020 2020 2020 2020
                                  2020 2020 2020 2020 2020 2020 2020 2020
                                      ^
                                      |
                                  END-OF-FILE CODE
```

Figure C-6. Example of Indexed File Records

FILE MANAGER OPERATION PARAMETERS AND MAIN-MEMORY-RESIDENT TABLES

The file manager requires a set of parameter values and main memory tables that are initialized at the time of system installation and are dependent on the particular installation. These parameter values and tables are contained in the file manager portion of SYSDAT. This appendix describes these parameters and tables. Further information on the determination of the initial parameter values may be found in the MSOS ordering bulletin.

MAIN-MEMORY-RESIDENT FILE MANAGER OPERATION INSTALLATION PARAMETERS

The following installation parameters affect the operation of the file manager. Each is an entry point in the file manager area of SYSDAT, evaluated by an equate. Values of these parameters vary from one system to another.

Mnemonic	Description
FMRDEL	Record-deleted code, used by the file manager to mark deleted records. This is usually an infrequently used ASCII code; for example, ($5E5E_{16}$). NOTE Users must define their record formats so that the value of FMRDEL can never occur as a data value for the first word of a record.
FMEOFC	End-of-file code. The file manager stores the value of FMEOFC into the first word and into the second word of the space where the next new record would be stored in a file. The value of FMEOFC must be different from the value of FMRDEL. Usually an infrequently used ASCII code; for example, ($5F5F_{16}$) is used as the value of FMRDEL. NOTE Users must define their record formats so that the value of FMEOFC can never occur as a data value in the first two words of a record.
FMMOSU	Maximum number of open sequential files permitted a single file manager user
FMMOIU	Maximum number of open indexed files permitted a single file manager user
FMNRCD	Maximum number of new records that may be stored into a file before the file manager automatically updates the file control block on mass memory to reflect the new number of records in the file
KIBSEC	Length of a key information block in sectors
ANOFPU	Maximum average number of open files per user
NTSUSR	Maximum number of simultaneous users of file manager
MAXCOP	Maximum number of concurrent open files in the system
ANRLPU	Average number of record locks per user

USER CONTROL TABLE

The user control table (UCT) consists of a six-word entry for each file access permit currently in effect. Each access permit corresponds to a user-file combination. If a given user has n files open, there are n entries in the UCT for that user. If a given file is currently open to q users, there are q entries in the UCT for that file. Entries in the UCT are not ordered. The maximum number of entries in the UCT is the value of MAXCOP (see the Main-Memory-Resident File Manager Operation Installation Procedures section). The address of the UCT is UCTABL within the file manager area of SYSDAT (see figure D-1). When an open file request is made and an entry space is needed in the UCT, the file manager selects the first entry space with a zero-value first word. If no entry space is available in the UCT, the OPENFL request is rejected. When a file is closed to a user, the corresponding UCT entry is cleared. The format of each UCT entry is shown in figure D-1.

The user identifier in word 1 of a UCT entry is one of the following:

- A unique user identification code assigned to the user by the ITOS executive at the time an open file request was intercepted by the ITOS executive.
- The address of the user's request buffer.

The latter definition of user identifier is used whenever an open file request is not intercepted by the ITOS executive.

The file identifier in word 2 of a UCT entry is defined in figure D-1. The file manager unit number in bits 11 through 14 is not the system logical unit number. It is an index into the file manager volume information table, where the first volume entry in the table has the index value 1. For further information, see the Main-Memory-Resident File Control Block Tables section.

MAIN-MEMORY-RESIDENT FILE CONTROL BLOCK TABLES

File control block format is given in the File Control Block Table, appendix B. As noted in appendix B, a certain portion of a file's file control block (FCB) must reside in main

```
*                    USER CONTROL TABLE
*
*           THE USER CONTROL TABLE (UCT) KEEPS AN UP-TO-DATE RECORD OF
*           WHICH FILES ARE OPEN BY WHICH USERS.  THE UCT CONTAINS
*           MAXCOP 6-WORD USER/OPEN FILE ENTRIES.  A 6-WORD ENTRY CON-
*           TAINS THE FOLLOWING INFORMATION.
*
*             WORD 1  USER IDENTIFIER
*             WORD 2  PSEUDO FILE IDENTIFIER
*                      BIT 15      PSEUDO LOCK FLAG
*                                  =1, FILE USERS LOCKED OUT.  MAIN
*                                      MEMORY IS SWAPPED.  SWAPPED
*                                      AREA INCLUDES FCB INFORMATION
*                                      FOR THIS FILE, AND THERE IS
*                                      INSUFFICIENT SPACE TO DUPLICATE
*                                      THIS INFORMATION IN UNSWAPPED
*                                      AREA.
*                                  =0, NO REASON TO LOCK OUT USERS DUE
*                                      TO MAIN MEMORY SWAP.
*                       BITS 14-11  FILE MANAGER LOGICAL UNIT NUMBER
*                       BITS 10-00  INDEX OF FCB IN FCB TABLE
*             WORD 3  FCB CORE ADDRESS
*             WORD 4  FILE SPACE LIMITS TABLE ENTRY ADDRESS, 0 IF NONE
*             WORD 5  FCB SUBSET ADDRESS
*             WORD 6  CONTROL POINT OF USER (CAN BE CHANGED)
*
        ENT   MAXCOP        MAX NO. OF CONCURRENT OPENS PERMITTED
        EQU   MAXCOP(ANOFPU*NTSUSR)
*
        ENT   UCTLEN
        EQU   UCTLEN(MAXCOP*6)   LENGTH OF UCT
*
        ENT   UCTABL        UCT
UCTABL  BZS   UCTABL(UCTLEN)
```

Figure D-1. User Control Table

memory when the file is open to a user. The specific portion of the FCB required in main memory depends on whether the file is sequential or indexed. The required FCB words for each type of file are specified in File Control Block Table, appendix B. Usually required FCB words reside in main memory outside user space. There is a provision for user-space-resident FCB portions, however. This is described in appendix L. For FCBs not stored in user space, two tables exist in the FMTABL portion of SYSDAT to contain required portions of FCBs. One table is for sequential files, and one is for indexed files (see figure D-2).

FCBs in these tables are not ordered. When a new table entry is needed, the file manager uses the first entry with a zero first word. No duplicates occur in these tables; that is, if a file is open to more than one user, its FCB appears only once in the tables.

If two or more users have opened a given file and each user has provided for FCB storage within his own user space, the set of FCB words that can be modified must be stored in one commonly used buffer. The FCB words that can be modified are the five-word header together with words 6 through 10 of the FCB (see File Control Block Table, appendix B). These words are referred to as the file's shared subset. When necessary, the file manager stores a file's shared subset into the subset control table. A sample subset control table from the file manager portion of a sample SYSDAT is shown in figure D-3. There are no duplicates in this table. Entries are not ordered. An entry in this table is empty if its first word is zero.

A file manager user may elect to store FCB words for an open file within his own user space as described in appendix L. When such a file manager user is swapped out, the system executive causes an entry to be stored into the subset control table to enable another user to open the file while the original file user is swapped out. If an entry already exists in the subset control table for this file, or if the necessary FCB words already reside in a file manager FCB table, the entry is not made. If no empty entry space is available, a pseudo file lock bit is set to prevent another user from opening the file while the original user is swapped out. (See figure D-1, word 2 description.)

MASS MEMORY UNITS TABLE; VOLUME INFORMATION TABLES

The mass memory units table is an index to the volume information tables. A sample mass memory units table and a sample set of volume information tables are shown in figure D-4.

FILE SPACE LIMITS TABLE

The file space limits table is used to ensure that all mass memory requests for a file are made within the boundaries of the file. A sample file space limits table is shown in figure D-5. An entry is empty if the first two words are zero. Entries in this table are not ordered. There are no duplicate entries.

RECORD LOCK TABLE

The record lock table is used to maintain a record of locked file records. A sample record lock table is shown in figure D-6. The value of MAXLOC is a system installation parameter. Entries in this table are not ordered. A zero first word indicates an unused entry. When a new entry is needed, the file manager uses the first entry space with a zero first word.

PROCESSOR CONTROL TABLES

The request processor control tables are used in queuing and processing file manager requests. The general method of request queuing is described in Reentrant Request Processors; Serial Request Processors, section 1. There is one processor control table that queues all serial requests. In addition, there is one processor control table to queue reentrant requests for each volume in the system used by the file manager. For example, a system with file space on two volumes would require a total of three processor control tables; a system with file space on three volumes would require a total of four processor control tables, etc. Sample processor control tables for a system with file space on two volumes is shown in figure D-7.

```
*                  SEQUENTIAL FILE CONTROL BLOCK TABLE
*
*           THIS TABLE IS USED FOR STORAGE OF THE FCB.S OF OPENED
*           SEQUENTIAL FILES FOR WHICH A USER SPACE FCB BUFFER WAS
*           NOT PROVIDED BY THE USER WHEN THE FILE WAS OPENED.
*
*           THIS TABLE CONTAINS ROOM FOR FMMOSF SEQUENTIAL FILE FCBS.
*           EACH FCB (WITH ITS HEADER) IS 15 WORDS LONG.  A FCB SPACE
*           IN THE TABLE IS FREE FOR USE IF ITS FIRST WORD IS ZERO.
*           WORDS 14 AND 15 OF EACH FCB SPACE ARE REQUIRED BECAUSE OF
*           THE MANNER IN WHICH FCB SUBSETS ARE MOVED INTO/OUT OF FCBS.
*
        ENT  FMMOSF          MAX NO. OF OPEN SEQ. FILE FCB SPACES
        EQU  FMMOSF(2*NTSUSR)
*
        EQU  FMSLEN(FMMOSF*15)     LENGTH OF TABLE
*
        ENT  FMFCBS          SEQUENTIAL FCB TABLE
FMFCBS  BZS  FMFCBS(FMSLEN)      SEQUENTIAL FCB TABLE


*                  INDEXED FILE CONTROL BLOCK TABLE
*
*           THIS TABLE IS USED FOR STORAGE OF THE FCB.S OF OPENED
*           INDEXED FILES FOR WHICH A USER SPACE FCB BUFFER WAS NOT
*           PROVIDED BY THE USER WHEN THE FILE WAS OPENED.
*
*           THIS TABLE CONTAINS ROOM FOR FMMOIF INDEXED FILE FCBS.
*           EACH FCB (WITH ITS HEADER) IS 27 WORDS LONG.  A FCB SPACE
*           IN THE TABLE IS FREE FOR USE IF ITS FIRST WORD IS ZERO.
*
        ENT  FMMOIF          MAX NO. OF OPEN INDEXED FILE FCB SPACES
        EQU  FMMOIF(2*NTSUSR)
*
        EQU  FMOLEN(FMMOIF*27)    LENGTH OF TABLE
*
        ENT  FMFCBI          INDEXED FCB TABLE
FMFCBI  BZS  FMFCBI(FMOLEN)      INDEXED FCB TABLE
```

Figure D-2. Main Memory File Control Block Tables

```
*                  FCB SUBSET CONTROL TABLE
*
*          EACH ENTRY IN THE FCB SUBSET CONTROL TABLE (FSCT)
*          CORRESPONDS TO AN OPEN FILE.  THE ENTRY FOR A GIVEN FILE
*          CONTAINS THAT SUBSET OF THE FILE"S FCB WHICH IS SUBJECT
*          TO CHANGE WHILE A FILE IS OPEN.  AN OPEN FILE WILL HAVE
*          AN ENTRY IN THE FSCT IF AND ONLY IF THERE IS NO ENTRY IN
*          THE FILE MANAGER MAIN MEMORY FCB TABLES FOR THAT FILE AND
*          ONE OR BOTH OF THE FOLLOWING CONDITIONS HOLDS-
*              (A) THE NECESSARY FCB WORDS FOR THIS FILE ARE CURRENTLY
*                  STORED IN TWO OR MORE USER SPACES.
*              (B) THE NECESSARY FCB WORDS ARE STORED IN USER SPACE
*                  FOR A SWAPPED OUT USER.
*
        ENT  FSCTNE          NO. OF FSCT ENTRY SPACES
        EQU  FSCTNE(NTSUSR*ANOFPU)
*
        EQU  FSCTLN(FSCTNE*10)    LENGTH OF THE FSCT
*
        ENT  FCBSCT
FCBSCT  BZS  FCBSCT(FSCTLN)   FSCT
```

Figure D-3. Sample FCB Subset Control Table

```
*                   MASS MEMORY LOGICAL UNIT TABLE
*
*            THE MASS MEMORY LOGICAL UNIT TABLE IS USED TO DEFINE THE
*            ADDRESSES OF THE VOLUME INFORMATION TABLES.  THE FIRST WORD
*            OF THIS TABLE DEFINES THE NUMBER OF VOLUME INFORMATION
*            TABLES IN THE SYSTEM.  EACH VOLUME DEFINED VIA THE VOLUME
*            INFORMATION TABLE MAY BE USED BY THE FILE MANAGER FOR FILES.
*
       ENT   MMLUTB          MASS MEMORY LOGICAL UNIT TABLE
*
MMLUTB ADC   NUMMLU          NUMBER OF VOLUME INFORMATION TABLES
       ADC   VIT01           VOLUME INFORMATION TABLE  NO. 1
       ADC   VIT02           VOLUME INFORMATION TABLE  NO. 2
NUMMLU EQU   NUMMLU(*-MMLUTB-1)

*                   VOLUME INFORMATION TABLE   NO.  1
*
VIT01  NUM   $8008            0. VISLUN - SYSTEM LOGICAL UNIT NUMBER
       ADC   0                1. VINAME - VOLUME NAME - CHARACTERS 1 AND 2
       ADC   0                2. ****** - VOLUME NAME - CHARACTERS 3 AND 4
       ADC   0                3. ****** - VOLUME NAME - CHARACTERS 5 AND 6
       ADC   0                4. ****** - VOLUME NAME - CHARACTERS 7 AND 8
       ADC   0                5. VINMBR - VOLUME NUMBER (2 ASCII CHARS)
       ADC   0                6. VIBMSM - BEGINNING OF MANAGEABLE SPACE -MSB
       ADC   0                7. VIBMSL - BEGINNING OF MANAGEABLE SPACE -LSB
       ADC   0                8. VIASDM - AVAILABLE SPACE DIRECTORY - MSB
       ADC   0                9. VIASDL - AVAILABLE SPACE DIRECTORY - LSB
       ADC   0               10. VIASDS - # SECTORS IN AVAIL SPACE DIR.
       ADC   0               11. VILBAM - LARGEST BLOCK OF SPACE AVAIL. -MSB
       ADC   0               12. VILBAL - LARGEST BLOCK OF SPACE AVAIL. -LSB
       ADC   96              13. VIWPS  - WORDS/SECTOR FOR VOLUME
       ADC   0               14. VIFDDM - FILE DEFINITION DIRECTORY ADDR-MSB
       ADC   0               15. VIFDDL - FILE DEFINITION DIRECTORY ADDR-LSB
       ADC   0               16. VIMAXF - MAX. NO. OF FILES PERMITTED
       ADC   0               17. VICURF - CURRENT NO. OF FILES ON VOLUME
       ADC   0               18. VINFDB - NUMBER OF BLOCKS IN FILE DEF. DIR.
       ADC   0               19. VINXTB - NEXT AVAILABLE BLOCK IN F.D.R.
       ADC   0               20. VINOOF - NUMBER OF OPEN FILES ON VOLUME
       ADC   0               21. VILBLM - VOLUME LABEL SECTOR - MSB
       ADC   0               22. VILBLL - VOLUME LABEL SECTOR - MSB

*                   VOLUME INFORMATION TABLE   NO.  2
*
VIT02  NUM   $800D            0. VISLUN - SYSTEM LOGICAL UNIT NUMBER
       ADC   0                1. VINAME - VOLUME NAME - CHARACTERS 1 AND 2
       ADC   0                2. ****** - VOLUME NAME - CHARACTERS 3 AND 4
       ADC   0                3. ****** - VOLUME NAME - CHARACTERS 5 AND 6
       ADC   0                4. ****** - VOLUME NAME - CHARACTERS 7 AND 8
       ADC   0                5. VINMBR - VOLUME NUMBER (2 ASCII CHARS)
       ADC   0                6. VIBMSM - BEGINNING OF MANAGEABLE SPACE -MSB
       ADC   0                7. VIBMSL - BEGINNING OF MANAGEABLE SPACE -LSB
       ADC   0                8. VIASDM - AVAILABLE SPACE DIRECTORY - MSB
       ADC   0                9. VIASDL - AVAILABLE SPACE DIRECTORY - LSB
       ADC   0               10. VIASDS - # SECTORS IN AVAIL SPACE DIR.
       ADC   0               11. VILBAM - LARGEST BLOCK OF SPACE AVAIL. -MSB
       ADC   0               12. VILBAL - LARGEST BLOCK OF SPACE AVAIL. -LSB
       ADC   96              13. VIWPS  - WORDS/SECTOR FOR VOLUME
       ADC   0               14. VIFDDM - FILE DEFINITION DIRECTORY ADDR-MSB
       ADC   0               15. VIFDDL - FILE DEFINITION DIRECTORY ADDR-LSB
       ADC   0               16. VIMAXF - MAX. NO. OF FILES PERMITTED
       ADC   0               17. VICURF - CURRENT NO. OF FILES ON VOLUME
       ADC   0               18. VINFDB - NUMBER OF BLOCKS IN FILE DEF. DIR.
       ADC   0               19. VINXTB - NEXT AVAILABLE BLOCK IN F.D.R.
       ADC   0               20. VINOOF - NUMBER OF OPEN FILES ON VOLUME
```

Figure D-4. Sample Mass Memory Unit Table and Volume Information Tables

```
*                 FILE SPACE LIMITS TABLE
*
*          THIS TABLE MAINTAINS A RECORD OF THE BEGINNING WORD ADDRESS
*          AND ENDING WORD ADDRESS + 1 FOR EACH OPEN FILE THAT HAS ITS
*          FCB IN USER SPACE.  THIS TABLE HAS MAXFSL*4 WORD ENTRY
*          SPACES.  EACH FOUR WORD ENTRY SPACE HAS THE FOLLOWING INFOR-
*          MATION WHEN IN USE:
*
*            WORD 1  START WORD ADDRESS, MSB
*            WORD 2  START WORD ADDRESS, LSB
*            WORD 3  ENDING WORD ADDRESS + 1, MSB
*            WORD 4  ENDING WORD ADDRESS + 1, LSB
*
       ENT   FSLIMT          FILE SPACE LIMITS TABLE
       ENT   FSLEND          FILE SPACE LIMITS TABLE ENDING ADDRESS
       EQU   MAXFSL(NTSUSR*ANOFPU)  NUMBER OF ENTRIES
       EQU   FSLLEN(MAXFSL*4)  LENGTH
*
FSLIMT BZS   FSLIMT(FSLLEN)
       EQU   FSLEND(*-1)
```

Figure D-5. Sample File Space Limits Table

```
*                 RECORD LOCK TABLE
*
*          THIS TABLE MAINTAINS A RECORD OF THE RECORD LOCKS IN EFFECT.
*          THIS TABLE HAS MAXLOC 5-WORD ENTRY SPACES, THUS MAXLOC LOCKS
*          MAY BE IN EFFECT CONCURRENTLY.  EACH ENTRY SPACE CONTAINS
*          THE FOLLOWING FIVE WORDS.
*
*            WORD 1  PSEUDO FILE IDENTIFIER
*            WORD 2  1ST WORD OF RECORD,S RELATIVE RECORD NUMBER
*            WORD 3  2ND WORD OF RECORD,S RELATIVE RECORD NUMBER
*            WORD 4  NUMBER OF LOCKED RECORDS IN SET
*            WORD 5  USER IDENTIFIER (OF LOCKING USER)
*
*          A NON-ZERO 1ST WORD INDICATES THAT AN ENTRY SPACE IS IN USE.
*
       ENT   MAXLOC          MAX NO. OF CONCURRENT RECORD LOCKS
       EQU   MAXLOC(NTSUSR*ANRLPU)
*
       EQU   LRTLEN(MAXLOC*5)    TABLE LENGTH
*
       ENT   LRTABL          LOCKED RECORD TABLE
LRTABL BZS   LRTABL(LRTLEN)      LOCKED RECORD TABLE
*
       ENT   NRERLE
NRERLE NUM   0               NUMBER OF RESERVED RECORD LOCK ENTRY SPACES
```

Figure D-6. Sample Record Lock Table

```
*                   PROCESSOR CONTROL TABLE
*
*            THE PROCESSOR CONTROL TABLE IS USED TO DEFINE THE ADDRESSES
*            OF THE REQUEST PROCESSOR CONTROL TABLES.  THE FIRST WORD OF
*            THE TABLE CONTAINS THE ADDRESS OF THE TABLE TO BE USED FOR
*            PROCESSING SERIALLY EXECUTED REQUESTS.  THE REMAINING WORDS
*            CONTAIN THE ADDRESSES OF TABLES TO BE FOR PROCESSING REEN-
*            TRANTLY EXECUTABLE REQUESTS, IN FM LOGICAL UNIT NUMBER ORDER
*
         ENT    PCTABL        PROCESSOR CONTROL TABLE
*
PCTABL   ADC    RPCT0-1       SERIAL PROCESSING CONTROL TABLE
         ADC    RPCT1-1       REENTRANT PROCESSING CONTROL TABLE,  NO. 1
         ADC    RPCT2-1       REENTRANT PROCESSING CONTROL TABLE,  NO. 2

         EQU    LENSCR(20)    LENGTH OF SCRATCH AREA OF RPC TABLE

                    SERIAL PROCESSING CONTROL TABLE
         SPC    1
RPCT0    NUM    0              1. RPLOGU - LU NO. OF MM DEVICE
         ADC    0              2. RPAREQ - ACTIVE REQUEST FLAG, 0 IF NONE
         ADC    0              3. RPWAIT - START OF WAITING REQ. QUEUE.
         ADC    0              4. RPRLEV - REQUEST PRIORITY LEVEL  (CUR REQ)
         ADC    0              5. RPRBF4 - REQBUF ADDRESS - FIRST FOUR WORDS
         ADC    0              6. RPLTEA - LOCK TABLE ENTRY ADDRESS (ABSOLUTE)
         ADC    $5400          7. RPRTNJ - RETURN JUMP TO XQT PROCESSOR
         ADC    0              5. RPPADR - PROCESSOR ADDRESS
         ADC    0              9. RPFCBA - FCB ADDRESS FOR FILE
         ADC    0             10. RPRBMP - REQBUF ADDRESS - MAIN PART
         ADC    0             11. RPPFP1 - REQUEST PARAMETER ADDRESS, FIRST+1
         ADC    0             12. RPPFP2 - REQUEST PARAMETER ADDRESS, FIRST+2
         ADC    0             13. RPPFP3 - REQUEST PARAMETER ADDRESS, FIRST+3
         ADC    0             14. RPPFP4 - REQUEST PARAMETER ADDRESS, FIRST+4
         ADC    0             15. RPMREQ - MONITOR REQUEST CODE WORD
         ADC    0             16. RPMRP1 - COMPLETION ADDRESS
         ADC    0             17. RPMRP2 - THREAD WORD
         ADC    0             18. RPMRP3 - LOGICAL UNIT FOR I/O
         ADC    0             19. RPMRP4 - NUMBER OF WORDS
         ADC    0             20. RPMRP5 - START CORE ADDRESS
         ADC    0             21. RPMRP6 - MASS MEMORY ADDRESS, MSB
         ADC    0             22. RPMRP7 - MASS MEMORY ADDRESS, LSB
         ADC    0             23. RPRCPT - CONTROL POINT FOR I/O REQUESTS
         ADC    0             24. RPRETN - SAVED RETURN ADDRESS
         SPC    1
         BZS    RPSC0(LENSCR)  SCRATCH AREA OF RPC TABLE

*                   REENTRANT PROCESSING CONTROL TABLE, NO. 1
*
RPCT1    NUM    1              1. RPLOGU - LU NO. OF MM DEVICE
         ADC    0              2. RPAREQ - ACTIVE REQUEST FLAG, 0 IF NONE
         ADC    0              3. RPWAIT - START OF WAITING REQ. QUEUE.
         ADC    0              4. RPRLEV - REQUEST PRIORITY LEVEL  (CUR REQ)
         ADC    0              5. RPRBF4 - REQBUF ADDRESS - FIRST FOUR WORDS
         ADC    0              6. RPLTEA - LOCK TABLE ENTRY ADDRESS (ABSOLUTE)
         ADC    $5400          7. RPRTNJ - RETURN JUMP TO XQT PROCESSOR
         ADC    0              8. RPPADR - PROCESSOR ADDRESS
         ADC    0              9. RPFCBA - FCB ADDRESS FOR FILE
         ADC    0             10. RPRBMP - REQBUF ADDRESS - MAIN PART
         ADC    0             11. RPPFP1 - REQUEST PARAMETER ADDRESS, FIRST+1
         ADC    0             12. RPPFP2 - REQUEST PARAMETER ADDRESS, FIRST+2
         ADC    0             13. RPPFP3 - REQUEST PARAMETER ADDRESS, FIRST+3
         ADC    0             14. RPPFP4 - REQUEST PARAMETER ADDRESS, FIRST+4
         ADC    0             15. RPMREQ - MONITOR REQUEST CODE WORD
         ADC    0             16. RPMRP1 - COMPLETION ADDRESS
         ADC    0             17. RPMRP2 - THREAD WORD
         ADC    0             18. RPMRP3 - LOGICAL UNIT FOR I/O
         ADC    0             19. RPMRP4 - NUMBER OF WORDS
         ADC    0             20. RPMRP5 - START CORE ADDRESS
         ADC    0             21. RPMRP6 - MASS MEMORY ADDRESS, MSB
         ADC    0             22. RPMRP7 - MASS MEMORY ADDRESS, LSB
         ADC    0             23. RPRCPT - CONTROL POINT FOR I/O REQUESTS
         ADC    0             24. RPRETN - SAVED RETURN ADDRESS
         BZS    RPSC1(LENSCR)  SCRATCH AREA OF RPC TABLE
```

Figure D-7. Sample Processor Control Tables (Sheet 1 of 2)

```
                          REENTRANT PROCESSING CONTROL TABLE, NO. 2
*
RPCT2  NUM  2                 1. RPLOGU - LU NO. OF MM DEVICE
       ADC  0                 2. RPAREQ - ACTIVE REQUEST FLAG, 0 IF NONE
       ADC  0                 3. RPWAIT - START OF WAITING REQ. QUEUE.
       ADC  0                 4. RPRLEV - REQUEST PRIORITY LEVEL  (CUR REQ)
       ADC  0                 5. RPRBF4 - REQBUF ADDRESS - FIRST FOUR WORDS
       ADC  0                 6. RPLTEA - LOCK TABLE ENTRY ADDRESS (ABSOLUTE)
       ADC  $5400             7. RPRTNJ - RETURN JUMP TO XQT PROCESSOR
       ADC  0                 8. RPPADR - PROCESSOR ADDRESS
       ADC  0                 9. RPFCBA - FCB ADDRESS FOR FILE
       ADC  0                10. RPRBMP - REQBUF ADDRESS - MAIN PART
       ADC  0                11. RPPFP1 - REQUEST PARAMETER ADDRESS, FIRST+1
       ADC  0                12. RPPFP2 - REQUEST PARAMETER ADDRESS, FIRST+2
       ADC  0                13. RPPFP3 - REQUEST PARAMETER ADDRESS, FIRST+3
       ADC  0                14. RPPFP4 - REQUEST PARAMETER ADDRESS, FIRST+4
       ADC  0                15. RPMREQ - MONITOR REQUEST CODE WORD
       ADC  0                16. RPMRP1 - COMPLETION ADDRESS
       ADC  0                17. RPMRP2 - THREAD WORD
       ADC  0                18. RPMRP3 - LOGICAL UNIT FOR I/O
       ADC  0                19. RPMRP4 - NUMBER OF WORDS
       ADC  0                20. RPMRP5 - START CORE ADDRESS
       ADC  0                21. RPMRP6 - MASS MEMORY ADDRESS, MSB
       ADC  0                22. RPMRP7 - MASS MEMORY ADDRESS, LSB
       ADC  0                23. RPRCPT - CONTROL POINT FOR I/O REQUESTS
       ADC  0                24. RPRETN - SAVED RETURN ADDRESS
       BZS  RPSC2(LENSCR)  SCRATCH AREA OF RPC TABLE


*                         REENTRANT PROCESSING CONTROL TABLE, NO. 3
*
RPCT3  NUM  3                 1. RPLOGU - LU NO. OF MM DEVICE
       ADC  0                 2. RPAREQ - ACTIVE REQUEST FLAG, 0 IF NONE
       ADC  0                 3. RPWAIT - START OF WAITING REQ. QUEUE.
       ADC  0                 4. RPRLEV - REQUEST PRIORITY LEVEL  (CUR REQ)
       ADC  0                 5. RPRBF4 - REQBUF ADDRESS - FIRST FOUR WORDS
       ADC  0                 6. RPLTEA - LOCK TABLE ENTRY ADDRESS (ABSOLUTE
       ADC  $5400             7. RPRTNJ - RETURN JUMP TO XQT PROCESSOR
       ADC  0                 8. RPPADR - PROCESSOR ADDRESS
       ADC  0                 9. RPFCBA - FCB ADDRESS FOR FILE
       ADC  0                10. RPRBMP - REQBUF ADDRESS - MAIN PART
       ADC  0                11. RPPFP1 - REQUEST PARAMETER ADDRESS, FIRST+1
       ADC  0                12. RPPFP2 - REQUEST PARAMETER ADDRESS, FIRST+2
       ADC  0                13. RPPFP3 - REQUEST PARAMETER ADDRESS, FIRST+3
       ADC  0                14. RPPFP4 - REQUEST PARAMETER ADDRESS, FIRST+4
       ADC  0                15. RPMREQ - MONITOR REQUEST CODE WORD
       ADC  0                16. RPMRP1 - COMPLETION ADDRESS
       ADC  0                17. RPMRP2 - THREAD WORD
       ADC  0                18. RPMRP3 - LOGICAL UNIT FOR I/O
       ADC  0                19. RPMRP4 - NUMBER OF WORDS
       ADC  0                20. RPMRP5 - START CORE ADDRESS
       ADC  0                21. RPMRP6 - MASS MEMORY ADDRESS, MSB
       ADC  0                22. RPMRP7 - MASS MEMORY ADDRESS, LSB
       ADC  0                23. RPRCPT - CONTROL POINT FOR I/O REQUESTS
       ADC  0                24. RPRETN - SAVED RETURN ADDRESS
       BZS  RPSC3(LENSCR)  SCRATCH AREA OF RPC TABLE
```

Figure D-7. Sample Processor Control Tables (Sheet 2 of 2)

VOLUME LABEL DESCRIPTION

The label for a volume is stored in the first sector of the volume. The entire first sector is reserved for the volume label although only 34 words are used. These 34 words are defined as follows (word 1 resides in the first physical word of the sector):

Word	Mnemonic	Definition
1	VLFLG1	Volume initialize flag 1 (preset to 1400_{16})
2	VLFLG2	Volume initialize flag 2 (preset to 0060_{16})
3	VLNAME	Volume name, ASCII characters 1 and 2
4		Volume name, ASCII characters 3 and 4
5		Volume name, ASCII characters 5 and 6
6		Volume name, ASCII characters 7 and 8
7	VLNMBR	Volume number (two ASCII characters)
8-12	VLSER	Volume serial number (10 ASCII characters)
13-16	VLSEC	Volume security code (eight ASCII characters)
17-20	VLDATE	Date of volume creation (eight ASCII characters)
21	VLBMS1	Sector address of start of space to be used by file manager, most significant bits
22	VLBMS2	Sector address of start of space to be used by file manager, least significant bits
23	VLASDM	Sector address of available space directory, most significant bits
24	VLASDL	Sector address of available space directory least significant bits
25	VLASDS	Number sectors in available space directory
26-27	VLLBA	Sector address of the largest block of space available (32-bit number)
28	VLWPS	Number of words per sector on this volume
29	VLFDD1	Sector address of the file definition directory, most significant bits
30	VLFDD2	Sector address of the file definition directory, least significant bits
31	VLMAXF	Maximum number of files permitted for volume
32	VLCURF	Current number of files existing on volume
33	VLNFDB	Number of blocks in the file definition directory
34	VLNXTB	Next block available for overflow in the file definition directory
35-96		Reserved for future use

STATUS INDICATOR WORD F

TABLE F-1. STATUS INDICATOR WORD (istat)

Bit Set	Requests†																
	CREATE	CLEAR, DELETE	OPENFL	CLOSFL	LOKFIL	UNLFIL	GETFCB, UPDFCB	RENAME	REDUCE	VOLUSE	PUTS	WRITER	READR	GETS	UPDREC	DELREC	COMFIL
0		7	9		25			7	7								
1		8	8					8	8	33							
2			10	17		27			51		28	28	28	28			
3				18	26				53								
4												34	37	37			
5	1 or 11	1 or 11	1, 11, or 12	1 or 11			1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11	1 or 11
6													38	38			
7													39	39	40	40	41
8														43			
9			13									35	44 or 49				
10	2							2									
11	3		14					47				36				50	
12	4		15				29				30	31					
13	5	5	5	19 or 20	19 or 20	19 or 20	21, 22, or 23	5	5	24	19 or 20	19 or 20	19 or 20	19 or 20	19 or 20	19 or 20	19
14	6	11	11 or 16	11			11 or 32	11	11 or 52	11	11	11	11	11	11 or 45	11 or 46	11
15	Request rejected																

†The numbers listed under each request are the status indication numbers, which are defined in table F-2.

TABLE F-2. STATUS INDICATION NUMBERS

Status Indication Number	Causes Request Rejection	Meaning
1	x	A mass memory error occurred.
2	x	The file name/owner string is not unique; that is, there is already a file on this volume with the name/owner string specified.
3	x	There is insufficient space in the specified volume's file definition directory for this file. If an unused file is present on the volume, it may be deleted to provide additional directory space. The deleted file must have the same scatter code as the new file if the new file is to utilize the empty directory entry left by the deleted file (see File Definition Directory, appendix B).
4	x	Insufficient mass memory file space exists on the specified volume for the file's records. If an unused file exists on the volume, it may be deleted to provide additional file space.
5	x	The volume specified for the file is not mounted and ready.
6	x	The file request is illegal. This implies one or more of the following has occurred: • Record length not in the required range • Maximum number of records not in the required range • Length of one or more keys not in the required range • Position of one or more keys not totally within the record • Missing key specification (primary key not specified but key 2 is specified; key 2 not specified, but keys 1 and 3 are specified, etc.) • No keys specified, but indexed file specified
7	x	The file is currently open to one or more users.
8	x	The file could not be located.
9	x	If bit 15 is zero, this file is currently open to another user, but this causes no apparent problem. If bit 15 is set, one of the following has occurred: • File request was to open for compression and some other user currently has file open. (Request may be retried after a delay.) • File request was to open for record access with file lock and some other user currently has file open. (Request may be retried after a delay.) • File is already open to this user.
10		If bit 15 is zero, the file was locked as a part of this OPENFL request. If bit 15 is set, the file was locked at the time this request was made.
11	x	File manager data structures on mass memory or in main memory contain one or more errors. (Both bit 5 and bit 14 of istat are set when this has occurred.)
12	x	A mass memory error occurred. This error may have occurred when the file was previously open and record recovery was not possible because of the timing of the failure. (Refer to appendix K.) If the bit 5 error indication is not accompanied by a MASS MEMORY I/O message on the comment device, the error occurred when the file was previously open. In this case it may be possible to manually restore the file on mass memory by use of ODEBUG. Otherwise, it is necessary to delete and recreate the file.
13	x	This OPENFL request is for record access, not file compression. When the file was last closed, a compression had been initiated but not completed. This compression must be completed before the file can be opened for record access.

TABLE F-2. STATUS INDICATION NUMBERS (Cont'd)

Status Indication Number	Causes Request Rejection	Meaning
14	x	The maximum number of concurrent open files permitted a single user has already been granted to this user.
15	x	The maximum number of open file permits that can be granted to all users in the system has been obtained. In this case, if bit 11 of istat is zero, the request may be retried after a delay.
16	x	Illegal request. This implies one or more of the following has occurred: • Value of idata(14) is invalid. • Indexed file and idata(13) exceeds the number of keys for the file. • Definition of idata(13), idata(14), idata(15) is inconsistent. • File control block storage specified within user area; insufficient number of file control block words were specified.
17		The file was unlocked by the close file request.
18		A set of locked records was unlocked by the close file request. (These records were initially locked by the requestor of the close.)
19	x	The file request buffer (reqbuf) was altered by the user before this file request.
20	x	The file was closed by executive forced file close due to hardware failure or operator shut-down of the volume.
21	x	The first word of array volnam is nonzero and the specified volume is not mounted and ready.
22	x	The first word of array volnam is zero and the file request buffer (reqbuf) was altered by the user before the retrieve file control block request.
23	x	The first word of array volnam is zero and the file was closed by executive forced file close due to hardware failure or operator shutdown of the volume.
24	x	The first word of array volnam is nonzero and the drive specified by vlunit already has a volume enabled.
25	x	The file is currently open to another user.
26	x	Record locking was indicated when the file was open.
27	x	The file is not currently locked by the user.
28		The file is currently locked by this user.
29	x	The file control block index is out of range for the specified volume. (This includes the case of a file control block index equal to 1 and no files created on the volume.)
30		Insufficient room exists in the file to store all numrec records (see reqbuf(15)).
31	x	Insufficient room exists in the file to store the record.
32	z	The file control block index not a positive integer.
33	x	The first word of array volnam is nonzero, and the volnam array does not match the name on the volume label.
34	x	The primary key value is not unique; that is, a record already exists in the file with the primary key value specified in the request.
35	x	The primary key value contained in the record is not the same as that in the keyval array.
36	x	There is insufficient room in the key index structure to store the keys. The record was stored, but it cannot be retrieved by key value.

TABLE F-2. STATUS INDICATION NUMBERS (Cont'd)

Status Indication Number	Causes Request Rejection	Meaning
37		Retrieval was by relative record number and one or more of the records are marked as deleted. The contents of the deleted records have been stored in the buffer recbuf. By testing the first word of each record, the user may determine which records are deleted records. The first word of a deleted record has the value of the external FMRDEL. (See File Identification, section 1; Main-Memory-Resident Volume Description Parameters, appendix B; and figure 2-10.)
38	x	Record locking was requested, but the maximum number of record locks in the system are currently in use. (The request may be retried after a delay.)
39	x	The record is locked by another user. (The request may be delayed and retried if care is taken to avoid the situation described in the note in Update Protection, section 1.)
40	x	Neither the file nor the records to be updated are locked.
41	x	An end-of-file has been reached. The file should now be closed.
42		End-of-file is reached before the number of records specified could be retrieved. At least one record was retrieved if bit 15 is zero. End-of-file indication implies an insufficient number of records in the file to satisfy the conditions specified in recspc array. If retrieval is by key value, no record in the file has a key value greater than or equal to the key value specified by the user. If retrieval is by relative record number, there are not enough records in the file starting at the record number in recspc to retrieve the number records specified in the OPENFL request. If end-of-file is reached before any records were retrieved, bit 15 is also set.
43		End-of-file is reached before the number of records specified in the OPENFL request could be retrieved. At least one record was retrieved if bit 15 is zero. If end-of-file is reached before any records are retrieved, bit 15 is also set.
44		Record retrieval was by key value. The key value specified, k_s, does not equal k_r, the key value retrieved. To test whether or not a record for key value k_s is in the file, it is necessary to test for the simultaneous setting of bits 8 and 15 as well as testing for the setting of bit 9. (See status indication number 42.)
45	x	The preceding retrieval was by key value and more than one record was retrieved. The number of records retrieved is governed by the preceding OPENFL request. The OPENFL request gives an error indication if record locking or file locking is specified for access by key value and the number of records specified is greater than one. However, the UPDREC request can be made with no lock indication in the preceding OPENFL request if the file is locked between the OPENFL request and the UPDREC request. In this case the previous OPENFL request would give no error indication if the number of records specified exceeds one. It is in this way that this error indication can be generated.
46	x	More than one record was retrieved by the retrieval preceding the DELREC call.
47	x	Insufficient file definition directory space exists for the file's new name. (Renaming the file does make available the directory entry space previously used for this file, but the new name/owner string does not hash into that space. Refer to File Definition Directory, appendix B for further information on the directory.)
48	x	Neither the file nor the record to be deleted is locked.
49	x	The relative record number was specified in recspc as (0,0).
50	x	The file manager is unable to delete one or more of the record's key values from the key index structure because one or more errors exist in the file's key index structure.
51		Operation is illegal for indexed files; legal only for sequential files.
52	x	The file header sector contains an error.
53	x	New number of records is either greater than the number defined for the file, is zero, is negative, or is less than the number of records currently stored in the file.

TABLE G-1. REENTRANT/SERIAL REQUEST PROCESSORS

File Request Mnemonic	Reentrant Processor	Serial Processor	Two Distinct Processors: Reentrant Processor for Access by Relative Record Number; Serial Processor for Access by Key Value
CREATE		X	
CLEAR		X	
DELETE		X	
OPENFL		X	
CLOSFL		X	
LOKFIL	X		
UNFIL	X		
GETFCB		X	
UPDFCB		X	
RENAME		X	
REDUCE		X	
VOLUSE		X	
PUTS	X		
WRITER		X	
READR			X
GETS			X
UPDREC	X		
DELREC			X
COMFIL			X

ADDITION OF FILE SPACE TO AN INSTALLED SYSTEM H

If more record space is needed for a given file, a new file may be created with additional records permitted. Records from the old file may then be retrieved from the old file and stored into the new file. The old file may then be deleted.

If the file definition directory for a volume is full, but more space exists on the volume on which file records could be stored, the following steps may be taken to increase the number of files the volume can hold (these steps are an alternative to rebuilding the system):

1. Save the files on an external medium.
2. Modify the volume label (appendix E). For the system volume, the utility ODEBUG may be used (refer to the MSOS Reference Manual). For a nonsystem volume, a file manager utility INIT can be used (refer to the ITOS Reference Manual).
3. Restore the files to the volume.

If more file space is needed than is physically available in the system, more mass storage may be purchased from your Control Data representative (refer to the MSOS Ordering Bulletin).

SUMMARY OF FILE MANAGER REQUEST CALLS

TABLE I-1. SUMMARY OF FILE MANAGER REQUESTS: MNEMONICS DEFINITIONS, FILE OPEN/CLOSE REQUIREMENTS

Mnemonic	Request Description	File must be open to requestor	File must be closed to all users	File lock required	Record lock or file lock required
CREATE	Create a file				
CLEAR	Delete all records in a file		X		
DELETE	Delete a file		X		
OPENFL	Obtain permission to access file (open file)				
CLOSFL	Relinquish permission to access file (close file)	X			
LOKFIL	Prevent other users from obtaining file access permits (lock file)	X			
UNLFIL	Allow other users to obtain access permits for previously locked file (unlock file)	X		X	
GETFCB	Retrieve file control block	†			
UPDFCB	Update file control block	†			
RENAME	Modify file name/ owner string		X		
REDUCE	Reduce number of records in files		X		
VOLUSE	Enable/disable use of volume				
PUTS	Store new record(s) in nonindexed file	X			
WRITER	Store new indexed record	X			
READR	Retrieve specific record(s)	X			
GETS	Retrieve next record(s)	X			
UPDREC	Update retrieved record(s)	X			X
DELREC	Delete a record	X			X
COMFIL	Compress a file	X		X	

†The file must be open only if the file is specified by referencing a request buffer for a particular open file.

TABLE I-2. SUMMARY OF FILE MANAGER REQUEST CALLING LISTS

Mnemonic	Calling List	Size of idata array	Minimum size of recbuf array
CREATE	reqbuf,idata,istat	24	-
CLEAR	reqbuf,idata,istat	12	-
DELETE	reqbuf,idata,istat	12	-
OPENFL	reqbuf,idata,istat	15	-
CLOSFL	reqbuf,istat	-	-
LOKFIL	reqbuf,istat	-	-
UNLFIL	reqbuf,istat	-	-
GETFCB	reqbuf,volnam,index,fcbbfr,istat	-	-
UPDFCB	reqbuf,volnam,index,fcbbfr,istat	-	-
RENAME	reqbuf,idata,newnam,istat	12	-
REDUCE	reqbuf,idata,istat	14	-
VOLUSE	regbuf,volnam,vlunit,istat	-	-
PUTS	reqbuf,recbuf,numrec,istat	-	$base_1$† +2
WRITER	reqbuf,recbuf,keyval,istat	-	$base_1$†
READR	reqbuf,recbuf,recspc,istat	-	$base_2$†
GETS	reqbuf,recbuf,keyval,istat	-	$base_2$††
UPDREC	reqbuf,recbuf,istat	-	$base_2$††
DELREC	reqbuf,recbuf,istat	-	$base_2$††
COMFIL	reqbuf,recbuf,istat	-	$base_2$†† +4

†$Base_1$ is the number of words required for the records accessed. For sector-aligned records, $base_1$ must include any unused words in a sector intersected by an accessed record. (For sector-aligned records, $base_1$ is a multiple of sector length.)

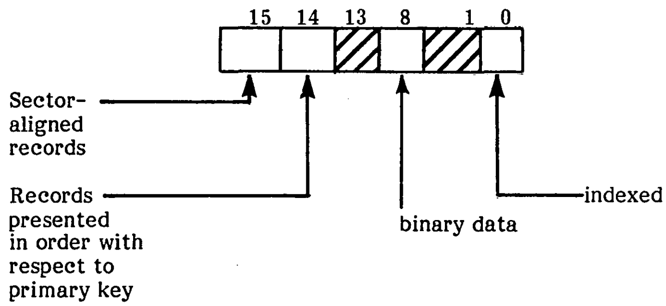
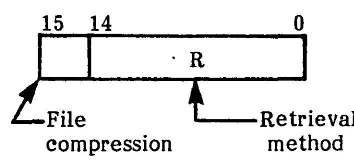
††$Base_2$ is the number of words required for the records accessed. For sector-aligned records, $base_2$ must include any unused words between accessed records, but $base_2$ does not include unused words following the last record accessed.

TABLE I-3. CONSTANT-SIZED ARRAYS

Parameter	Number of Words†
reqbuf	24
volnam	4
index	1
fcbbfr	96
newnam	8
vlunit	1
numrec	1

†Constant for all requests using parameter.

TABLE I-4. SUMMARY OF idata ARRAY (INITIAL VALUES)

<table>
<tr><th>Request</th><th>Word</th><th>Definition</th></tr>
<tr><td>All that specify idata</td><td>1-4</td><td>File name</td></tr>
<tr><td></td><td>5-8</td><td>File owner</td></tr>
<tr><td></td><td>9-12</td><td>Name of volume (idata(9) may initially be 0000_{16} or 2020_{16} for CLEAR,DELETE,OPENFL, or RENAME)</td></tr>
<tr><td>CREATE</td><td>13</td><td>Record length in bytes</td></tr>
<tr><td></td><td>14, 15</td><td>Number of records in file</td></tr>
<tr><td></td><td>16</td><td>File type

</td></tr>
<tr><td></td><td>17</td><td>Length of key 1 (bytes)</td></tr>
<tr><td></td><td>18</td><td>Byte position, key 1</td></tr>
<tr><td></td><td>19</td><td>Length of key 2 (bytes)</td></tr>
<tr><td></td><td>20</td><td>Byte position, key 2</td></tr>
<tr><td></td><td>21</td><td>Length of key 3 (bytes)</td></tr>
<tr><td></td><td>22</td><td>Byte position, key 3</td></tr>
<tr><td></td><td>23</td><td>Length of key 4 (bytes)</td></tr>
<tr><td></td><td>24</td><td>Byte position, key 4</td></tr>
<tr><td>OPENFL</td><td>13</td><td>Access options:


<table>
<tr><th>R Value</th><th>Retrieval Method</th></tr>
<tr><td>0</td><td>Relative record no.</td></tr>
<tr><td>1</td><td>Key 1</td></tr>
<tr><td>2</td><td>Key 2</td></tr>
<tr><td>3</td><td>Key 3</td></tr>
<tr><td>4</td><td>Key 4</td></tr>
<tr><td>Other</td><td>Invalid</td></tr>
</table></td></tr>
<tr><td></td><td>14</td><td>Number of records to retrieve or compress per call (see figure I-5)</td></tr>
<tr><td></td><td>15</td><td>Lock indicator
0 No locking
> 0 Record locking
< 0 File lock</td></tr>
<tr><td>REDUCE</td><td>13-14</td><td>New number of records in file</td></tr>
</table>

TABLE I-5. NUMBER OF RECORDS ACCESSED BY INDIVIDUAL REQUESTS

Request	Number of Records Retrieved or Written†
WRITER	One
PUTS	Number specified in PUTS request
READR	One, if access is by key value Number specified in OPENFL request otherwise
GETS	Number specified in OPENFL request (must equal 1 if access is by key value and record locking or file locking is specified)
COMFIL	Number specified in OPENFL request (must equal 1 for indexed file)
UPDREC	Number of records retrieved by the preceding READR or GETS request (must equal 1 if preceding retrieval was by key value)
DELREC	One

†This assumes sufficient space for storage and sufficient index structure space if needed.

TABLE I-6. VALUES STORED BY FILE MANAGER AVAILABLE TO USER ON COMPLETION OF REQUEST

Request	reqbuf(15)	reqbuf(16) reqbuf(17)	For key value access only		idata(9) idata(12)
			keyval Array	recspc Array	
PUTS	Number of records stored	Relative record number, first record stored	-	-	-
WRITER	Number of records stored	Relative record number of record stored	-	-	-
READR	Number of records retrieved	Relative record number, first record stored	-	Left-justified key value of record retrieved	-
GETS	Number of records retrieved	Relative record number, first record retrieved	Left-justified key value of last record retrieved	-	-
CLEAR, DELETE, OPENFL, RENAME, REDUCE	-	-	-	-	Name of volume on which file was found

SYSTEM FAILURE AND JOB PROCESSOR ERROR MESSAGES RELATED TO IMPROPER USE OF FILE MANAGER

SYSTEM FAILURE

A foreground program that does not run under ITOS control can cause a system hang if the file manager detects an illegal overlapping of parameters in a request calling list. For example, if the record buffer and the request buffer overlap, this is illegal. When an illegal overlap is detected, the file manager transfers control to the main-memory-resident program, SYFAIL. Program SYFAIL saves register values and hangs on a $18FF_{16}$ instruction.

NOTE

A foreground program that runs in partitioned memory under ITOS control and contains a parameter overlap error is aborted by ITOS and does not cause a system hang.

JOB PROCESSOR ERROR MESSAGES

Error message JP02 is printed by the job processor on the system main console when a background program attempts execution of one of the following:

- A file manager request with parameters in the calling list that overlap illegally (for example, record buffer and request buffer overlap).
- A file manager request with a parameter in protected main memory such that the file manager can store into that parameter. (For example, a background program with the request buffer in protected main memory would result in a JP02 error.)

RECOVERY TECHNIQUES

The following technique is used to prevent loss of new records at the time of system failure. When a file closed to all users is initially opened, the file manager modifies the file's control information on mass memory to reflect the open state of the file. Each time the file manager writes a set of records to mass memory, the file manager stores a two-word end-of-file code (as defined by FMEOFC; see Main-Memory-Resident File Manager Operation Installation Parameters, appendix D) into the next record space following the written records. The current number of records in a file is periodically updated in the file control information on mass memory as new records are added to the file. Whenever a file is closed to the last of a set of file users, the file's control information on mass memory is updated to reflect the current number of records in the file and to reflect the closed state of the file.

If the system fails while a file is open, the next time the file is opened for use the file manager detects that the file was left in an open state. The file's record space is then scanned for an end-of-file. The scan commences with the first record space following the space required for the number of records recorded in the file's file control block (FCB) (appendix B) on mass memory. The length of the scan is determined by the system parameter that specifies the number of new records between periodic FCB updates. If an end-of-file is found, the current number of records is updated in the file control information, thus recovering the last new records. If the system fails during a transfer of new records to the file so that no end-of-file can be found upon the next open of the file, these records cannot be recovered. The existence of such irrecoverable data is indicated to the opening user. (See error indication 12, appendix F.)

A similar procedure provides for the recovery of the key index structure accompanying an indexed file.

There is also a procedure to prevent loss of records if a system failure occurs during file compression. File compression is described in Compress File (COMFIL), section 2.

When a file is opened for compression, the file's control information is modified on mass memory to reflect the open-for-compression state of the file. As each set of compressed records is written to mass memory, the number of records processed and a two-word end-of-file is written to mass memory in the next record space following the compressed records. Periodically, the file's control information on mass memory is updated to contain the number of original file records processed and the net number of compressed records. When the compression is completed, the file's control information is updated on mass memory to reflect the new current number of records in the file and to delete the compression-in-progress status. If the system fails before completion of the compression, the file manager detects the previous compression-in-progress state when the file is next opened.

If the system fails during file compression, file compression can be resumed when a compression request is made by a user. The file manager locates the number of records processed and the end-of-file written within the file's record space. It uses the number of records processed to resume the compression process. However, if the system failure occurs during a transfer of compressed records to the file so that the end-of-file is not transferred to mass memory, completion of compression is not possible. The existence of such irrecoverable data is indicated to the opening user. (See error indication 12, appendix F.)

The techniques described above are not utilized if the file was opened for special processing or if the file. If the file is a binary data file, the file control information on mass memory is updated on mass memory as each set of one or more new records is added to the file; thus, if a system failure occurs while the file is open, no special recovery techniques are needed to prevent loss of records. If a file is open for special processing, the file control information on mass memory is not periodically updated as new records are added to the file. Further, no special record recovery techniques are used when a file is first opened for processing following a system failure that occurred while the file was open for special processing.

STORAGE OF FILE CONTROL BLOCKS WITHIN USER SPACE

As specified in the File Control Block Table, appendix B, a portion of a file's file control block (FCB) must be main-memory-resident whenever the file is open. In most systems that include the file manager, users usually elect to let the file manager store main memory FCBs in file manager space as described in Main-Memory-Resident File Control Block Tables, appendix D. However, in systems including ITOS, terminal users and background programs may wish to store FCBs in user space to conserve main memory resources. FCB words stored in user space are stored in a buffer within the user's program. To cause FCB words to be stored in user space, the user must specify this storage when making the OPENFL request. Instead of initializing the request buffer in the OPENFL calling list to all zeroes as described in Open File (OPENFL), section 2, the request buffer is initialized as follows:

Request Buffer Word	Definition
1-9	All binary zeroes
10	Location of buffer to contain FCB header and FCB words
11-12	All binary zeroes
13	Zero, if FCB buffer within user space is to contain the minimum number of words needed by the file manager; that is, 27 words (including file control block header) for an indexed file or 15 words (including FCB header) for a sequential file. or Number of words (not including five-word header) to be retrieved from the FCB and stored in the user's FCB buffer if more than the minimum number of words are to be stored into buffer. (Buffer must allow room for 5-word header in addition to words retrieved from the file control block.) NOTE If word 13 of the request buffer is nonzero but less than the minimum number of words required by the file manager, the OPENFL request is rejected with bits 14 and 15 of the status indicator word set.
14-24	All binary zeroes

A FORTRAN example of FCB storage within user space is shown in figure L-1.

```
      DIMENSION NBUF(24),NDATA(13)
C FILE N IS INDEXED.  RESERVE 22 WORDS FOR FCB,5 WORDS FOR FCB HEADER.
      INTEGER FCBBFR(27)
      DATA NDATA /'FILE N  SGWILLIA','VOLUME 1',1,1,1/
      DATA NBUF /24*0/
C STORE ABSOLUTE ADDRESS OF FCBBFR INTO REQBUF(10)
C       LDA   =XFCBBFR
      ASSEM $C000,+FCBBFR
C       STA*  NBUF+9
      ASSEM $6400,+NBUF(10)
      CALL OPENFL (NBUF,NDATA,NSTAT)
      IF (NSTAT.LT.0) GO TO 9700
       .                        .
       .                        .
       .                        .
```

Figure L-1. FCB Storage Within User Space (FORTRAN Example)

Each volume containing file space contains an allocatable file space directory. The location and maximum size of this directory are given by words 8 through 10 (VIASDM, VIASDL and VIASDS) of the volume information table for this volume. (Refer to Mass Memory Units Table; Volume Information Tables, appendix D, and figure D-4.) On the system volume, the allocatable space directory begins at MAXSEC+1. On all other volumes, the directory begins immediately following the volume label.

Each entry in the directory corresponds to a block of available file space. An entry consists of four words defined as follows:

Words	Contents
1, 2	Size of this block of available space (a 32-bit number)
3, 4	First sector of this block (a 32-bit number)

The value $FFFE_{16}$ is stored in the word following the last entry in the directory.

The system volume is initialized by the SPACE program. All other volumes are initialized by the file manager utilities. At the time of initialization, there is one entry in the directory.

The maximum number of entries in the directory is determined at the time of system installation. The maximum number of entries needed would occur when the following situation develops:

1. The maximum number of files for this volume have been defined with a remaining block of available space at the end of the defined files.
2. Every other file is deleted, leaving an available block of space between each pair of remaining files. Thus the maximum number of entries is:

$$\left\lceil \frac{VIMAXF}{2} \right\rceil$$

The maximum size of the directory in sectors is:

$$\left\lceil \left\lceil \frac{VIMAXF}{2} \right\rceil \text{ entries} \times \frac{4 \text{ words}}{\text{entry}} \times \frac{1 \text{ sector}}{96 \text{ words}} \right\rceil$$

$$= \left\lceil \left\lceil \frac{VIMAXF}{2} \right\rceil \times \frac{1}{24} \right\rceil \text{ sectors}$$

$$= \left\lceil \frac{VIMAXF}{48} \right\rceil \text{ sectors}$$

This size is stored in VIASDS, word 10, of the volume information table (see figure D-4). Words 6, 7, 11, and 12 of the volume information table also refer to the available space directory (see figure D-4).

File space is allocated by reserving n+1 sectors each time n sectors of file space are needed. The extra sector is reserved for the header sector that precedes each file. The header sector has the following format:

	Word	Description
	1	Header identifier containing the ASCII characters AL (equal to 41 $4C_{16}$)
	2-3	Size of this file in sectors (including the header sector), a 32-bit number
	4-5	Starting sector of this block of allocated space (a pointer to the header sector), a 32-bit number
Time of file space allocation	6	Year (two ASCII characters)
	7	Month (two ASCII characters)
	8	Day (two ASCII characters)
	9	Time (binary representation of the four-digit decimal 24-hour time in hours and minutes as retrieved from the SYSDAT word HORMIN)
	10-13	File owner (eight ASCII characters)

Using the information in the header sector for each existing file, a mass storage accounting program could be written.

Available space is managed as follows. When a file is added, the first available block of sufficient size is used. The directory entry for this block of available space is modified to reduce the number of available sectors in this block by the number of sectors allocated. This may temporarily result in an entry in the directory referencing a block of zero length. When a file is deleted, the number of sectors released is added to the available space directory either by inserting a new entry in the directory or by adding the number of newly released sectors to the number of sectors in an adjacent available space. At the time of file release, the available space directory is compressed so that any entries for zero length blocks are deleted and any entries for adjacent blocks are combined.

INDEX

CUT ALONG LINE

COMMENT SHEET

MANUAL TITLE File Manager Version 2 Reference Manual

PUBLICATION NO. 96768040 REVISION C

FROM NAME:

BUSINESS
ADDRESS:

COMMENTS: This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number.

STAPLE STAPLE

FOLD

FIRST CLASS
PERMIT NO. 333

LA JOLLA, CA.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
PUBLICATIONS AND GRAPHICS DIVISION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

FOLD

CUT ALONG LINE

STAPLE STAPLE

CORPORATE HEADQUARTERS, P.O. BOX 0, MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA CORPORATION