60499300

CONTROL DATA

CYBER RECORD MANAGER ADVANCED ACCESS METHODS VERSION 2 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
NOS 1
NOS 2
NOS/BE 1

60499300

CONTROL DATA

CYBER RECORD MANAGER ADVANCED ACCESS METHODS VERSION 2 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
NOS 1
NOS 2
NOS/BE 1

REVISION RECORD

Revision	Description
A (03/31/78)	Original release.
B (07/20/79)	This revision reflects CYBER Record Manager Advanced Access Methods Version 2.1. Major changes include the implementation of extended direct access and actual key files, and the operation of the FLBLOK utility.
C (02/13/81)	This revision moves most information on initial file organizations to appendix J, describes the creation and use of system data compression routines, and reflects miscellaneous technical and editorial changes at PSR level 528.
D (05/14/82)	This revision reflects the support of NOS Version 2. Major changes include the alteration of FLBLOK utility output and an addition to appendix G on buffer allocation. This revision also reflects miscellaneous technical and editoral changes at PSR level 564.
E (07/15/83)	This revision drops reference to initial file organizations and documents only what was called extended file organizations. Changes include an added feature to the FLSTAT utility. This revision also reflects miscellaneous technical and editorial changes at PSR level 577.
F (12/16/85)	This revision reflects the support of the CYBER 170 800 Series models and the CYBER 180 Computer Systems. This revision also reflects miscellaneous technical and editorial changes at PSR level 647.

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

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

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	Revision
Front Cover	-
Title Page	-
ii	F
iii/iv	F
v	F
vi	F
vii	F
viii	F
ix	C
1-1	E
1-2	E
2-1 thru 2-3	E
2-4	F
2-5 thru 2-9	E
3-1 thru 3-11	E
4-1	F
4-2	F
4-3	E
4-4	E
4-5	C
4-6 thru 4-9	E
4-10	C
4-11	E
4-12	E
4-13	C
5-1	F
5-2	F
5-3 thru 5-7	E
6-1 thru 6-7	E
7-1	E
7-2	C
7-3 thru 7-11	E
A-1	C
A-2	C
A-3	B
A-4	B
B-1	C
B-2	C
B-3	D
B-4	E
B-5	D
B-6	E
B-7	E
B-8	C
B-9 thru B-15	E
B-16	F
B-17	C
B-18	C
C-1 thru C-4	E
D-1	C
D-2	E
D-3 thru D-6	C
D-7	E
D-8	C
D-9 thru D-12	E
E-1	E
E-2	E
F-1	C
G-1	E
G-2	D
H-1	E
H-2 thru H-6	C
I-1	E
J-1	F
J-2	F
K-1	F
K-2	F
Index-1 thru -7	F
Comment Sheet/Mailer	F
Back Cover	-

PREFACE

CONTROL DATA® CYBER Record Manager Advanced Access Methods (AAM) Version 2.1 operates under control of the following operating systems:

- NOS 1 and NOS 2 for the CONTROL DATA CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.
- NOS/BE 1 for the CDC® CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems.

NOS 2 supports only those file organizations previously known as extended indexed sequential, extended direct access, and extended actual key. Also, NOS 2 supports only the Multiple-Index Processor (MIP) previously called extended MIP.

This manual documents only those file organizations previously called extended file organizations and refers to them as indexed sequential, direct access, and actual key. Only the Multiple-Index Processor (MIP) previously called extended MIP is documented in this manual.

FORTRAN user programs communicate with AAM through calls to AAM subroutines. COMPASS programs communicate with AAM directly through macro calls. COBOL, PL/I, FORM, and Query Update automatically access AAM input/output facilities. AAM input/output facilities are available under the CDCS and TAF data management environments.

This manual, which is intended as a primary document for COMPASS programmers, presents background information and operational specifications for AAM. Programmers accessing AAM indirectly can use this manual as a source for AAM terminology and concepts and can find specific language interfaces in the appropriate reference manuals. The user is assumed to be familiar with the operating system at the installation and with file organization and manipulation.

The following manuals are of primary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
COMPASS Version 3 Reference Manual	60492600	X	X	X
CYBER Record Manager Advanced Access Methods Version 2 User's Guide	60499400	X	X	X
NOS Version 1 Reference Manual Volume 1 of 2	60435400	X		
NOS Version 1 Reference Manual Volume 2 of 2	60445300	X		
NOS Version 2 Reference Set Volume 3, System Commands	60459680		X	
NOS Version 2 Reference Set Volume 4, Program Interface	60459690		X	
NOS/BE Version 1 Reference Manual	60493800			X

The following manuals are of secondary interest:

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
Common Memory Manager Version 1 Reference Manual	60499200	X	X	X
CYBER Loader Version 1 Reference Manual	60429800	X	X	X

Publication	Publication Number	NOS 1	NOS 2	NOS/BE 1
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700	X	X	X
CYBER Record Manager Basic Access Methods Version 1.5 User's Guide	60495800	X	X	X
NOS Version 1 Installation Handbook	60435700	X		
NOS Version 2 Installation Handbook	60459320		X	
NOS/BE Version 1 Installation Handbook	60494300			X
NOS/BE Version 1 System Programmer's Reference Manual	60494100			X

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

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

NOTATIONS

The following notations are used throughout the manual with consistent meaning:

UPPERCASE	In language syntax, uppercase indicates a statement keyword or character that is to be written as shown.
lowercase	In language syntax, lowercase indicates a name, number, or symbol that is to be supplied by the programmer.
[ ]	In language syntax, brackets indicate an item that can be used or omitted.
{ }	In language syntax, braces indicate that only one of the vertically stacked items can be used.
. . .	In language syntax, a horizontal ellipsis indicates that the preceding optional item in brackets can be repeated as necessary.
⋮	In program examples, a vertical ellipsis indicates that statements or parts of the program have not been shown.

Numbers that appear without a subscript are decimal values. Other value formats are denoted as:

n . . . n	Value is decimal
n . . . nB	Value is octal
n . . . nW	Value is decimal, specified in words

AAM FEATURES 1

An interface between user programs and system input/output routines is provided by the Advanced Access Methods (AAM). AAM routines exist in the NOS and NOS/BE operating systems. AAM provides consistent error processing and maintenance of different file organizations.

AAM routines are used by some compilers and are available for user programs. Use of AAM by compilers and user programs extends input/output compatibility to both the system and application program levels.

The primary task of AAM is record input/output for files on supported devices. The various types of records and file organizations must be identified for AAM. These and other file characteristics must be set by the user in the file information table (FIT). The FIT is divided into fields that describe certain aspects of the file. Refer to appendix D for the exact structure of the FIT.

The following terms are relevant to AAM and related systems:

- AAM (Advanced Access Methods)

  A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor.

- BAM (Basic Access Methods)

  A file manager that processes sequential and word addressable file organizations.

- CRM (CDC CYBER Record Manager)

  A generic term relating to both BAM and AAM as they run under the NOS and NOS/BE operating systems.

- MIP (Multiple-Index Processor)

  A processor that allows AAM files to be accessed by alternate keys.

CDC offers guidelines for the use of the software described in this manual. These guidelines appear in appendix I. Before using the software described in this manual, the reader is strongly urged to review the contents of this appendix. The guidelines recommend use of this software in a manner that reduces the effort required to convert application programs to future hardware or software systems.

FILE ORGANIZATIONS

Three file organizations are supported by AAM:

- Indexed sequential

  Records are in order by primary key and can be accessed sequentially or randomly.

- Direct access

  Records are not in any recognized order and are accessed by key manipulation.

- Actual key

  Records are accessed by a primary key containing the block and record number within the file.

AAM MACROS

The file information table is established for the file by the FILE macro encountered at assembly time. The FILE macro establishes the FIT in the field length of the user program at the point at which it is called. This macro can contain only the file name and file organization or it can have user-specified parameters describing the particular file. FIT fields are assumed by AAM through default values when not specified as macro parameters. AAM macros and functions are listed in table 1-1. Macros are grouped according to their associated functions.

The applicability of some AAM macros depends on the file organization established by the user. Table 1-2 indicates the applicability of each macro to the various file organizations and to files processed by MIP.

Macros are discussed according to each file organization in section 4, File Processing. Consequently, material is presented redundantly for the benefit of a programmer who uses this manual to reference a particular file organization. The format of each macro and a general description are presented in section 5, File Processing Macros.

Macro statements are coded in COMPASS format. Each statement can contain a location field, a macro name in the operation field, a variable field, and a comment field. Any field is terminated by one or more blanks. A macro statement begins in character position 1 of an 80-column card image and continues through column 72. Columns 73 through 80 are used for sequencing. COMPASS coding conventions are shown in figure 1-1.

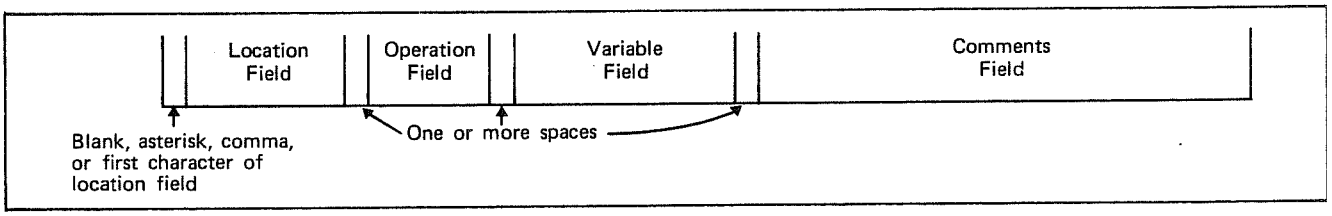


Figure 1-1. COMPASS Format

TABLE 1-1. AAM MACROS

Function	Macro	Action Taken
File creation and maintenance	FILE	Creates the file information table (FIT). In addition to this macro, a FILE control statement is available to supply FIT information.
	FETCH	Retrieves the value of a specified field in the FIT.
	FITDMP	Dumps the contents of a FIT to the error file.
	STORE	Sets a value in a FIT field.
	SETFIT	Sets values in fields of the FIT with values supplied through the FILE control statement.
File initialization and termination	OPENM	Prepares a file for processing.
	FLUSHM.	Flushes buffers to bring mass storage files into a state of equilibrium.
	CLOSEM	Terminates file processing.
Data transfer	GET	Transfers data from a file to the working storage area.
	PUT	Transfers data from the working storage area to a file.
File updating	DELETE	Deletes a record from a file.
	REPLACE	Replaces a record in a file.
File positioning	SKIP	Repositions a file backward or forward.
	REWINDM	Rewinds a file to beginning-of-information (BOI).
	SEEK	Provides an overlap between input/output and processing by positioning while processing.
	START	Positions a file to a record that satisfies a specific condition.

TABLE 1-2. APPLICABILITY OF MACROS

Macro	File Organization			
	Indexed Sequential	Actual Key	Direct Access	MIP
CLOSEM	X	X	X	X
DELETE	X	X	X	
FETCH	X	X	X	X
FILE	X	X	X	X
FITDMP	X	X	X	X
FLUSHM	X	X	X	
GET	X	X	X	X
GETN	X	X	X	X
GETNR	X	X	X	X
OPENM	X	X	X	X
PUT	X	X	X	
REPLACE	X	X	X	
REWINDM	X	X	X	X
RMKDEF	X	X	X	X
SEEK	X	X	X	X
SETFIT	X	X	X	X
SKIP	X	X		X†
START	X			X
STORE	X	X	X	X

†SKIPFL macro only.

Suggested column conventions are as follows:

1	Comma (continuation), asterisk (comment line), or other (beginning of new statement)
2 thru 9	Location field entry, left-justified
10	Blank
11 thru 16	Operation field entry, left-justified
17	Blank
18 thru 29	Variable field entry, left-justified
30	Beginning of comment

FILE STRUCTURES 2

A hierarchical data structure is recognized in a progression from the character level to the largest grouping of data, the file. The AAM user can describe file structure by file organization and record type.

LOGICAL STRUCTURE

The logical structure of an AAM file is user-controlled. The following terms are applicable to the logical file structure and are used throughout this manual:

- Record

  A record is a group of related characters. A character is represented in six bits as internal display code. A record is the smallest collection of information passed between AAM and the user. The user defines the structure and characteristics of records within a file by declaring a record format. The beginning and ending points of a record are implicit within each format.

- Block

  A block contains one or more records. Block structure is interwoven with the physical recording format; unlike other logical file structure declarations, the block structure is transparent in use. AAM constructs blocks from the records supplied by the user and supplies the user with records as requested. The user is unaware of block boundaries.

- File

  A file is a logically connected set of information; it is the largest collection of information that can be addressed by a file name. All data in a file is stored between beginning-of-information (BOI) and end-of-information (EOI).

PHYSICAL STRUCTURE

The following terms pertain to the physical means used to record files:

- Input/output device

  Any storage medium supported by the operating system.

- Rotating mass storage (RMS)

  Disk or disk pack.

- Mass storage device

  Disk, disk pack, or extended memory.

- PRU device

  All mass storage devices. The operating system superimposes a physical structure over the user-declared AAM file structure on all files that reside on PRU devices.

- Physical record unit (PRU)

  The smallest unit of information that can be transferred between a peripheral storage device and central memory. The PRU size is 640 characters.

- Short PRU

  A PRU containing fewer than the 640 characters defined for a PRU.

- System-logical-record

  A group of PRUs terminated by a short or zero-length PRU.

AAM controls the physical file position; the user controls only the logical file position.

FILE ORGANIZATIONS

The following paragraphs describe the structure of each file organization.

NOTE

Refer to appendix I for recommended access methods.

INDEXED SEQUENTIAL FILE STRUCTURE

An indexed sequential file consists of a file statistics table, index blocks, and data blocks. Each block is an integral number of PRUs less two central memory words and is treated as a system-logical-record. Both index and data blocks are fixed-length blocks and must be the same size.

Each record in the indexed sequential file is identified by a unique primary key value. Records are stored in data blocks in increasing primary key sequence. Index blocks contain primary key information used to retrieve any record in the file.

The file statistics table (FSTT) maintains file integrity by preventing user actions that would destroy the file. When the file is created, the user defines the file and key structure, which must remain the same for the life of the file. This information is stored in the FSTT and is used to guide processing as long as the file exists. If the user sets a field in the FIT to a value that does not conform to the FSTT, the value is rejected and the job is terminated. The FSTT stores accumulated statistics related to the file access; it also stores a default or user-supplied collating sequence for ranking symbolic keys.

The logical structure of an indexed sequential file is shown in figure 2-1. The blocks identified as D01 through D09 are data blocks; those identified as I01 through I03 are the first level index blocks and I11 is the primary or second level index block.

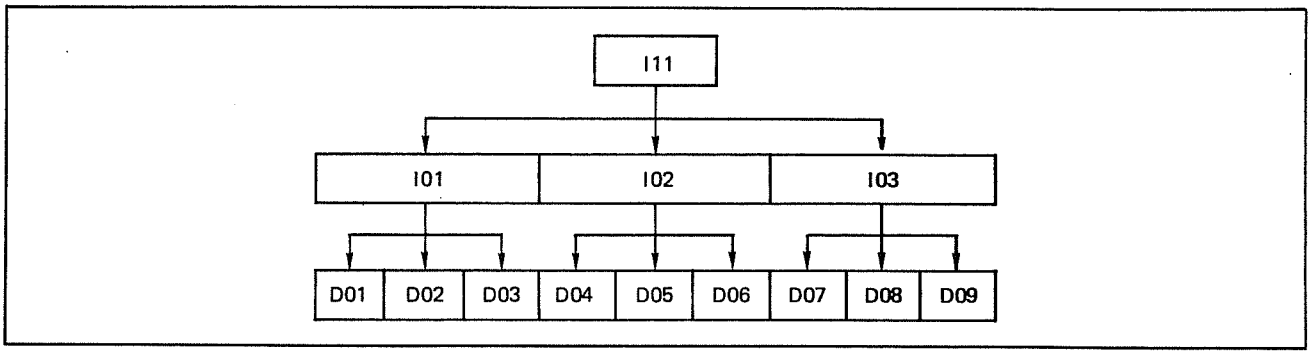


Figure 2-1. Logical Structure of a Two-Level Indexed Sequential File

The physical structure of an indexed sequential file is shown in figure 2-2. FSTT is the file statistics table, D01 through D09 are the data blocks, and I01 through I03 are the index blocks.

When an indexed sequential file is created, data and index block size, record size, and key characteristics must be specified for AAM to construct the data blocks, index blocks, and key entries for the file.

Data Blocks

A data block in an indexed sequential file contains a header, user records, record pointers, and padding. The size and characteristics of the data block are determined by the setting of various fields in the FIT when the file is created. The specific FIT fields that are used during file creation are discussed in section 4, File Processing. The formats of the fields are detailed in section 3, File Information Table.

The two-word header in a data block on disk contains a pointer that chains the block in a forward direction to permit sequential reading without an index. It also contains the size of the unused space, a record count, and other flags and counts. An optional checksum can also be included in the header.

User records in a data block can be fixed or variable length. Only whole records can be in a data block; records cannot span blocks. Records are stored in ascending primary key sequence. The first record in the first data block has the lowest primary key value in the file, and the last record in the last data block has the highest key value.

One or more record pointers are stored in a data block. The record pointer is a 30-bit field; two record pointers are stored in a word. The pairs of record pointers are stored at the end of the data block beginning with the last word. The record pointer contains the last word address plus 1 of the record; the address is relative to the beginning of the first record in the block. It also contains the number of trailing characters that are not part of the record and processing flags. If all records in the block are the same length, only one record pointer is needed.

Padding in a data block is the amount of space that is not to be used for writing records during file creation. This space can then be used to insert new records during subsequent runs that update the file. The amount of padding is specified as a percentage of the total block size. The default value of zero percent can be used for files that are expected to grow mainly by sequential inserts or by adding records at the end of the file.

Index Blocks

An index block in an indexed sequential file is structured the same as a data block with a system-supplied header, records, a record pointer, and padding. The size of an index block is the same as the size specified for a data block. Other index block characteristics are specified through the FIT. Refer to section 4, File Processing, for the specific FIT fields and to section 3, File Information Table, for the format of the fields.

Index block records are created and maintained by AAM. A record consists of a primary key value and a PRU number. The primary key value is the lowest key value in the next lower level index block or in a data block; the PRU number points to the beginning of the block. Records within an index block are in ascending primary key sequence.

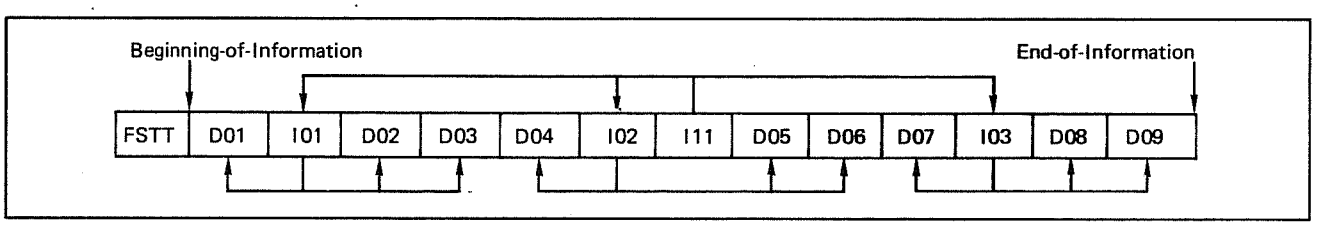


Figure 2-2. Physical Structure of an Indexed Sequential File

Index blocks are organized into as many levels as necessary to ensure only one index block at the highest or primary level. The maximum number of levels that can exist for a file is specified in the FIT; no more than 15 levels can be specified.

An index block requires only one record pointer because all records in the block are the same length. The record pointer, which is the same as described for data blocks, is stored in the last word of the index block.

Padding in an index block is the same as in a data block. Data blocks and index blocks, however, do not have to have the same percentage of padding. The default index block padding factor is zero percent.

ACTUAL KEY FILE STRUCTURE

An actual key file consists of a file statistics table and a number of data blocks. New data blocks are created at end-of-information as the file grows. Block size can be specified by the user or a default size can be determined by the system. Padding can be defined for data blocks, or block size can be defined to allow for an increase in record size.

The data block contains a fixed number of slots for data records that can be fixed or variable length. The block number and slot number assigned to each record as it is written are computed from the primary key of the record. When a record is written on the file, the primary key can be specified by the user or it can be determined by AAM. If a primary key value of zero is specified by the user, AAM determines where to write the record and returns the primary key to the user.

The logical structure of an actual key file is shown in figure 2-3. The first block is the 126-word file statistics table containing file information and a pointer to end-of-information. The remaining blocks are fixed-length data blocks. Data block format is shown in figure 2-4.

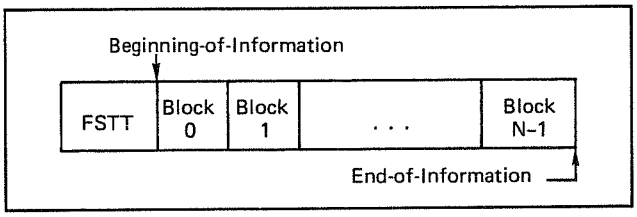


Figure 2-3. Logical Structure of an Actual Key File

A block unable to accommodate one of its member records will, in place of the record, contain an overflow pointer to its physical location in the file. Data records within the block are ordered by slot number with the smallest number being the first record. Record pointers are placed at the bottom of the block in inverse order of the data records. A block checksum, if specified for the file, is contained in the block header.

Actual Keys

Records are stored and retrieved by an actual key, which is the primary key. The actual key specifies a record number which is converted by AAM to a block and slot number.

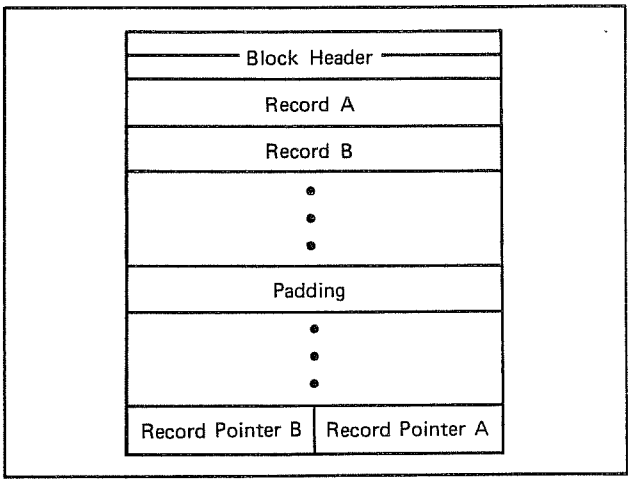


Figure 2-4. Actual Key Data Block Format

Key length is specified by the user when the file is created. Key length can range from 1 to 8 characters. The key length determines the maximum file size. For example, a key length of 2 limits the file to 4095 records.

Larger blocking factors (64 versus 8) provide better storage density for files with variable-length records; eight records per block is the default blocking factor. Actual keys need not be contained within the records.

Overflow

Overflow occurs in two ways:

- The user specifies the actual key for a write operation and the specified block has insufficient empty space to contain the new record.
- The user attempts to replace a record with a new larger record and the block containing the old record has insufficient empty space to contain the new record.

In either case, the record is inserted into a different block that has enough empty space. This requires two record slots; one in the original block containing the overflow pointer and the second one in the other block containing the record.

Logically, the overflow slot that contains the record is still empty. If a GET macro is issued to retrieve a record from that overflow slot, an error is issued. If a PUT macro is issued to write a record in that slot, the overflow record is moved to its proper location, if possible, or to another empty slot. Thus the overflow pointer will either be updated or replaced by the record. Overflow records will, in all cases, require two accesses to retrieve the record.

The formats of the block headers on disk and overflow record headers are shown in figure 2-5. The format of a record header is shown in figure 2-6.

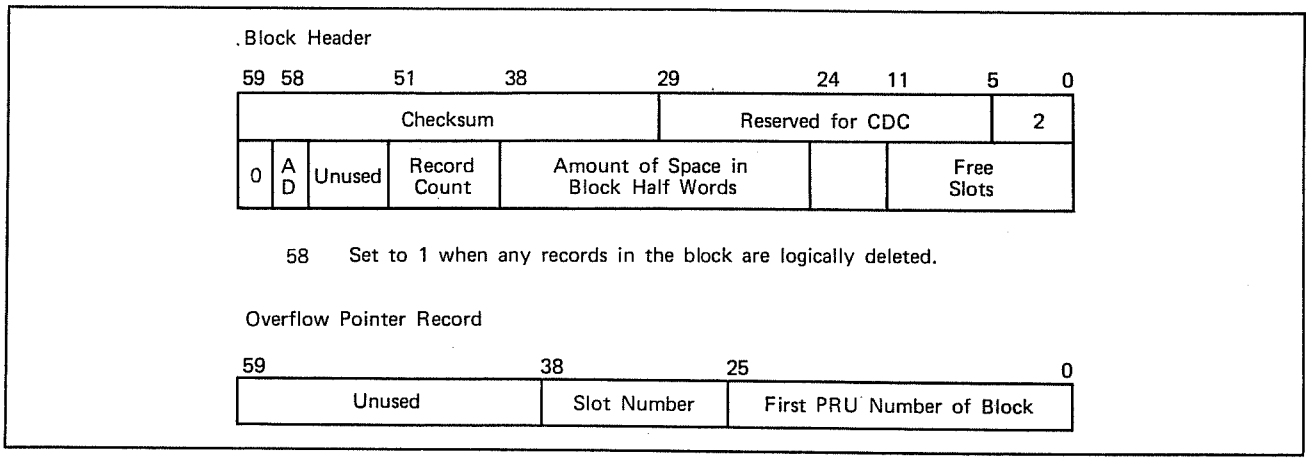


Figure 2-5. Actual Key Block and Overflow Record Header Formats

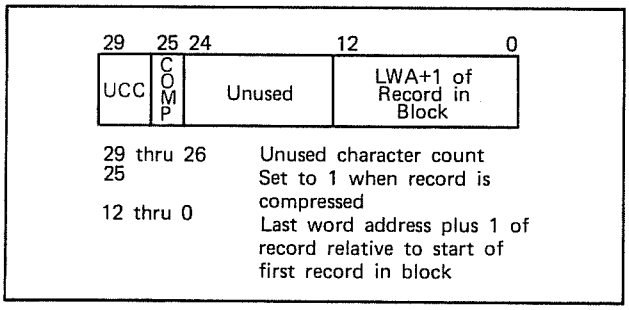


Figure 2-6. Extended Actual Key Record Pointer Format

DIRECT ACCESS FILE STRUCTURE

A direct access file contains a file statistics table, home blocks, and (under certain conditions) overflow blocks. All blocks are fixed length. The following terms have specific meaning in relation to direct access files:

- Primary key

  A primary key is a character string that is hashed to produce the location of the home data block containing the record (1 to 255 characters).

- Hashing

  Hashing denotes the method of using primary keys to search for relative home block addresses of direct access records.

- Home block

  A home block is a block whose relative address is computed by hashing primary keys. A home block contains synonym records whose keys hash to that relative address.

- Synonyms

  Synonyms are records whose keys hash to the same home block.

- Overflow record

  An overflow record is a record whose key has been hashed to a home block that is already filled.

- Overflow block

  An overflow block is the second or subsequent block in a chain that starts at a home block. It contains overflow records and can contain records belonging to more than one overflow chain.

- Chain

  A chain consists of a home block and possibly overflow blocks connected by forward pointers.

The relative position of records within a direct access file is not important. A record is stored and retrieved by hashing its primary key to produce the relative address of a home block. When a home block is filled, the record is placed in a system-generated overflow block.

The logical structure of a direct access file is shown in figure 2-7. FSTT is the file statistics table. H1 through H6 are the home data blocks, and OV1 through OV3 are the overflow blocks.

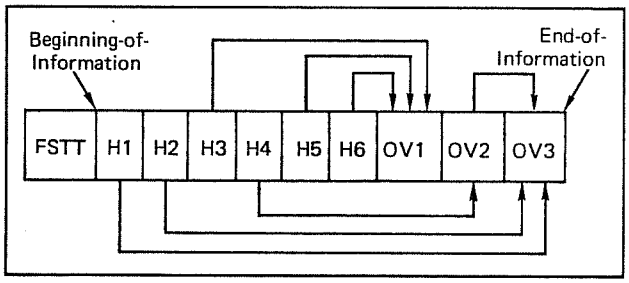


Figure 2-7. Logical Structure of a Direct Access File

File Storage Allocation

Mass storage space is preallocated when a direct access file is opened. Record storage and retrieval are by primary key; the location of a record within a file is determined by hashing the primary key to a relative block address.

Records are grouped in fixed-size blocks according to the results of the primary key hashing. When more records hash to a home block than the block can contain, overflow blocks are created and linked to form a chain.

Extensive analysis of the record key structure, key range, and key distribution is necessary to implement a randomly organized file in an optimum manner. An ideal hashing algorithm distributes records uniformly across all home blocks. Because no single hashing routine can produce optimum results for all data, a user-supplied hashing routine can be used. Hashing routines are discussed in further detail in section 4, File Processing.

File Blocking

Each direct access block (home or overflow) is an integral number of PRUs less two central memory words and is treated as a system-logical-record. Direct access block header format is shown in figure 2-8. The two-word block header is followed by the records ordered by ascending key value. Half-word record headers are entered from the end of the block.

The structure of a direct access record header is the same as shown in figure 2-6 for actual key files.

RECORD TYPES

Eight external record types are supported; these record types are listed in table 2-1. Except for S type records and W type records, each record type is described in detail in the following paragraphs. AAM processes S and W type records the same as U type records.

NOTE

Refer to appendix I for recommended record types.

When records are written on an AAM file, the record type specification is used to compute the record length in characters. This length is recorded in the header word that accompanies each record in these files. When the record is read, record type is ignored and the number of characters indicated by the length field in the header is returned to the program.

The numbering conventions for describing a record or the position of a control field or key field in a record are summarized in figure 2-9. All record lengths are specified by character count. Values are normally unsigned positive integers, counting in a decimal system. For AAM files, the maximum record length (MRL) field in the FIT must not exceed 81870 characters.

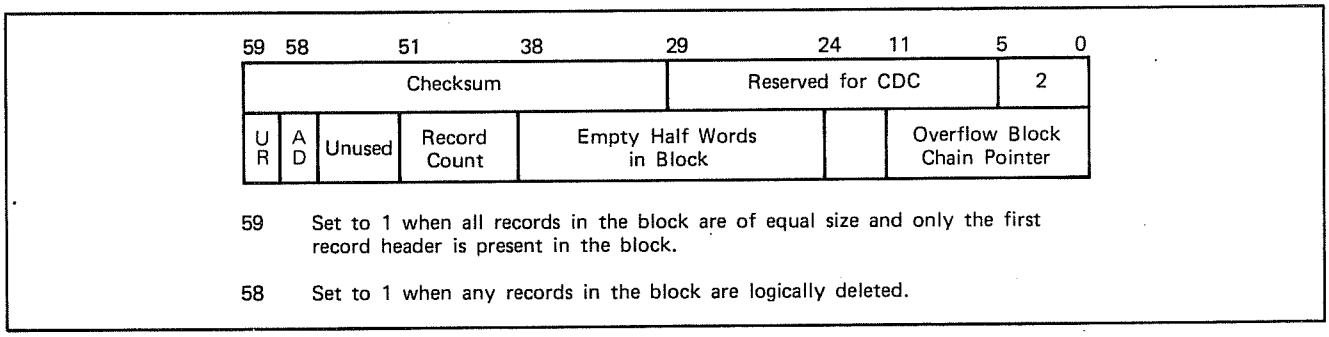
TABLE 2-1. RECORD TYPES AND LENGTH DESCRIPTIONS

Record Type	Length Description
D - Decimal Character Count	A length field within the record gives the length as character count.
F - Fixed Length	All records are the same fixed length.
R - Record Mark	A record mark character specified by the user terminates the record.
S - System Record	The length is defined by the user.
T - Trailer Count	The fixed-length header contains a trailer count field that specifies the number of fixed-length trailers for the record.
U - Undefined	The length is defined by the user.
W - Control Word	The length is defined by the user.
Z - Zero Byte	The length is determined using the RL or FL field and removing all full words of blanks.

DECIMAL CHARACTER COUNT, D TYPE RECORDS

Records in a file with D type records vary in length. The length of an individual record is specified in a record length field located within the record. The position of the record length field is specified by two fields in the FIT. The length field beginning character position (LP) field indicates the character position (numbering from 0) in which the record length field begins. The length field length (LL) field specifies the number of characters in the record length field, which cannot exceed six characters.

When a D type record is written, the record length field cannot contain a value greater than the value of the maximum record length (MRL) field in the FIT. The



59 Set to 1 when all records in the block are of equal size and only the first record header is present in the block.

58 Set to 1 when any records in the block are logically deleted.

Figure 2-8. Direct Access Block Header Format

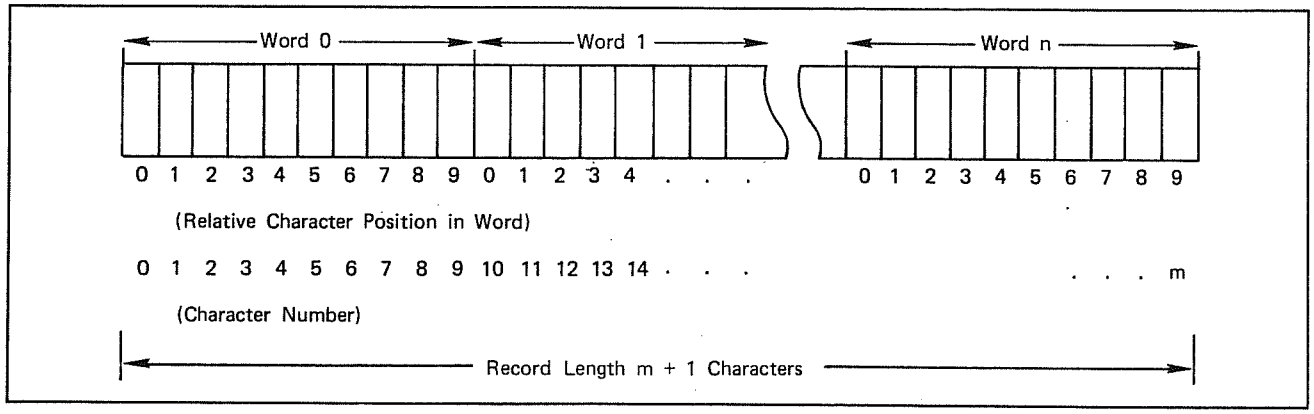


Figure 2-9. Numbering Conventions

maximum length that can be specified in the MRL field is 81870 characters for AAM files. The length value specified in the record length field is given as right-justified display code filled with zeros or blanks. If the COMP-1 (C1) field in the FIT is set to YES, the record length field is a COMP-1 (binary) field. If the sign overpunch (SB) field in the FIT is set to YES, the record length field is a sign-overpunch field.

The minimum record length (MNR) field in the FIT specifies the minimum number of characters for the D type record. Minimum record length must be large enough to contain the field specifying total record length and should be at least 10 characters to ensure a correct detection of end-of-data conditions. The default value for the MNR field is the sum of the values in the LP and LL fields; however, the MNR field can be set to a greater value.

Figure 2-10 shows an example of a D type record. The record length field is three characters in length (the LL field is set to 3) beginning in character position 22 (the LP field is set to 22). The minimum number of characters in a record is 25 (the sum of the values in the LL and LP fields).

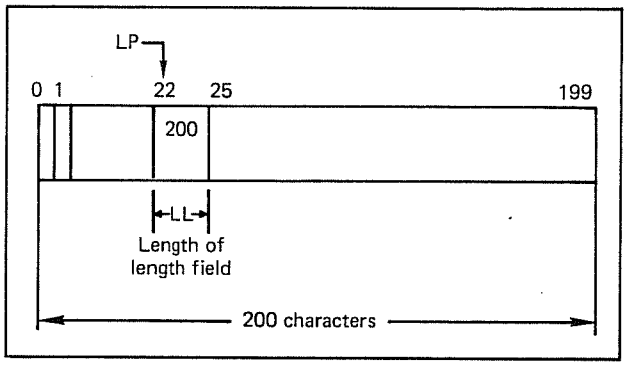


Figure 2-10. D Type Record Example

FIXED LENGTH, F TYPE RECORDS

In a file with F type records, all records are the same length. The number of characters in the F type records is specified by the fixed length (FL) field in the FIT. The maximum record length that can be specified for F type records is 81870 characters for AAM files. Minimum record length is 10 characters. An example of an F type record is shown in figure 2-11; each record in the file contains 200 characters as specified by the FL field.

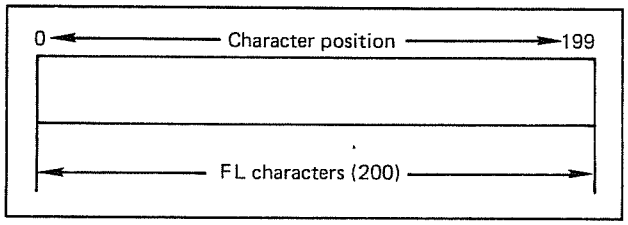


Figure 2-11. F Type Record Example

Any value in the record length (RL) field in the FIT is ignored. When a GET or PUT macro is issued, the value of the fixed length (FL) field in the FIT determines the number of characters that are transferred. A value must be supplied for the FL field before the file can be successfully opened.

RECORD MARK, R TYPE RECORDS

A special delimiting character, called a record mark, terminates R type records. The record mark character, which can be any character of the character set, is selected by the user. The delimiting character is specified in the record mark character (RMK) field in the FIT.

The size of an R type record cannot exceed the number of characters specified by the value of the maximum record length (MRL) field in the FIT. Maximum length that can be specified for R type records is 81870 characters for AAM files. Minimum record length should be at least 10 characters to ensure a correct detection of end-of-data conditions.

When a GET macro is issued, all characters up to and including the record mark character are transferred to the working storage area. If the record mark character is not found within the specified maximum record length, the maximum number of characters is transferred and an excess data error is given.

Issuing a PUT macro causes all characters up to and including the record mark character to be written on the file. If the record mark character is not found within the specified maximum record length, no data is written on the file and an excess data error is given.

Figure 2-12 illustrates the use of R type records. The maximum record length (MRL) field is set to 120 and the record mark character (RMK) field is set to 62_8, which is the default right bracket (]) character. For a file read or write operation, the right bracket character terminates the record.

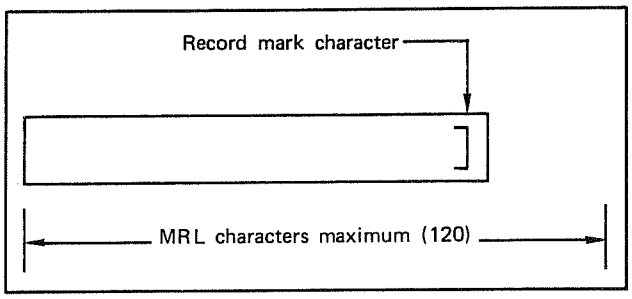


Figure 2-12. R Type Record Example

SYSTEM RECORD, S TYPE RECORDS

When S type records are specified, AAM processes the records the same as U type records. Refer to the description of U type records.

TRAILER COUNT, T TYPE RECORDS

Records in a file with T type records consist of a fixed-length header and a variable number of fixed-length trailer items. The fixed-length header contains a count field that specifies the number of fixed-length trailer items in the record.

Five fields in the FIT are applicable to T type records and must be specified.

HL Header length specifies the number of characters in the fixed-length header.

TL Trailer length specifies the number of characters in one fixed-length trailer item.

CP Starting character position specifies the character position (numbered from 0) in which the count field begins.

CL Count field length specifies the number of characters (one through six) in the count field.

MRL Maximum record length specifies the maximum number of characters in any record.

The value in the count field is right-justified display code with zero or blank fill. The COMP-1 (C1) field or the sign overpunch (SB) field in the FIT can be set to YES to change the count field to a COMP-1 or sign-overpunch field.

The count field, which is identified by the CP and CL fields in the FIT, must be located in the fixed-length header portion of the record. The value in the header length (HL) field, therefore, cannot be less than the sum of the values in the CP and CL fields.

The value in the HL field is the logical minimum record length. The maximum length for a record is specified by the maximum record length (MRL) field in the FIT; the value in the HL field cannot exceed the value in the MRL field. Maximum length that can be specified for T type records is 81870 characters for AAM files. Minimum record length indicated by HL should be at least 10 characters to ensure a correct detection of end-of-data conditions. The logical structure of a T type record is shown in figure 2-13.

UNDEFINED, U TYPE RECORDS

Files with U type records have records that are not formatted according to any of the supported record types. The maximum record length (MRL) field in the FIT indicates the maximum length for any record in the file. The maximum record length that can be specified for U type records is 81870 characters for AAM files.

When a GET macro is executed, the record length (RL) field in the FIT is updated to indicate the number of characters read. When a PUT macro is executed, the RL field in the FIT must be set to indicate the number of characters to be written. The value in the RL field cannot exceed the specified maximum record length. AAM maintains record pointers that define the length of the stored record.

CONTROL WORD, W TYPE RECORDS

When W type records are specified, AAM processes the records the same as U type records. Refer to the description of U type records.

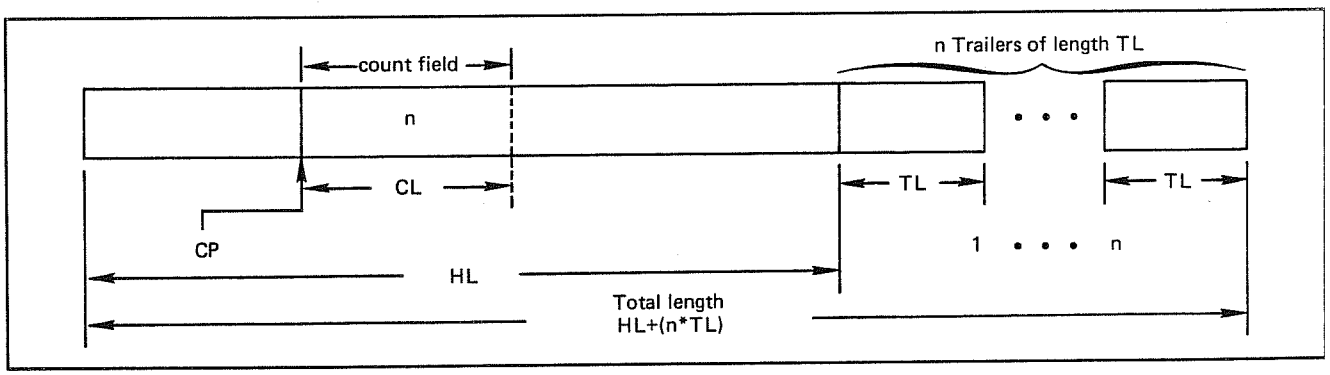


Figure 2-13. T Type Record Format

ZERO BYTE, Z TYPE RECORDS

A Z type record has trailing blank suppression. Maximum record size is indicated by the full length (FL) field in the FIT; maximum length that can be specified for Z type records is 81870 characters for AAM files.

When a record is read, the zero byte is stripped from the record and blank padding is added to fill the working storage area to (MRL+9)/10. If end-of-information is encountered before the zero byte is found, it is possible the file does not contain Z type records. At the conclusion of a read operation, the record length (RL) field in the FIT is set to indicate the number of characters read, not including blank padding.

When a record is written, the value of the RL field determines the processing that takes place. If the RL field is set to a value greater than zero, the end of the record is determined by searching backward from the character position specified by the value of the RL field and removing all full words of blanks.

If the RL field is set to zero when a record is being written, the end of the record is determined by a backward search for the last nonblank character in the working storage area. The search begins in the character position indicated by the FL field in the FIT; all full words of blanks are removed.

ALTERNATE KEY INDEX FILE STRUCTURE

An index file is created and maintained by the Multiple-Index Processor (MIP) whenever a data file has alternate keys defined. The index file is automatically created when the data file is created and updated whenever an update to the data file affects the index file.

The index file created and maintained by MIP contains an index for each alternate key position defined for the file. Within an index, each alternate key value is associated with a primary key list of records containing that value. The index file is created when the data file is created, or the MIPGEN utility can be used to create the index file for an existing data file.

When the index file is created, the user can specify the size of the index file blocks in a field in the data file FIT. The block size is increased if necessary to the nearest multiple of 640 characters minus 20. The default size for index file blocks is the data block size.

Each alternate key index is ordered in ascending sequence of alternate key values. The ordering of primary key values within the primary key list associated with an alternate key value can be controlled by the user. The structure of primary key lists can be indexed sequential or first-in first-out. Indexed sequential structure is most efficient. The user can also specify that alternate key values are unique, in which case each primary key list contains only one value.

The index file is structured into three levels: a level 1 main file, level 2 subfiles, and level 3 subfiles. The level 1 main file contains descriptions of the alternate keys. A level 2 subfile contains values for an alternate key and a level 3 subfile contains primary key values for a specific alternate key value. The logical structure of a MIP index file is shown in figure 2-14.

The level 1 main file contains descriptions of all the alternate keys defined for the data file. The description of an alternate key includes the position, length, and type of the key as well as information related to sparse keys. Normally, all the descriptions can be contained in one block; however, if more than one block is required, the main file has an indexed sequential structure.

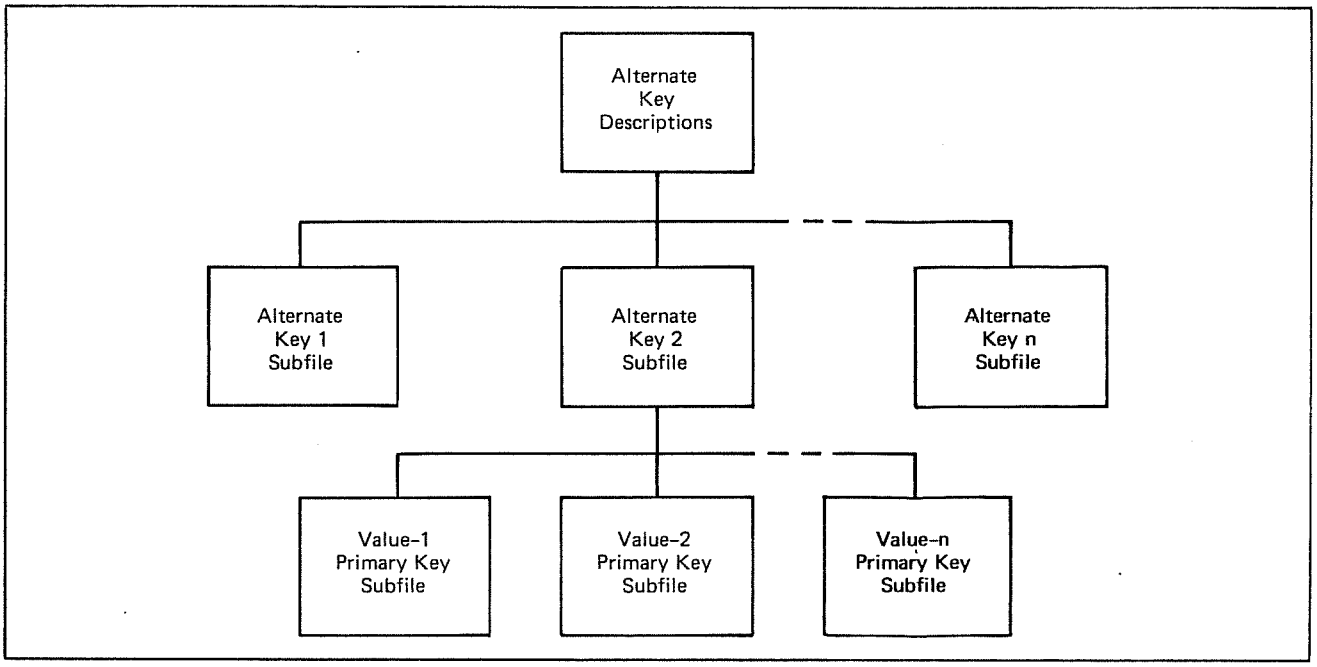


Figure 2-14. Index File Logical Structure, MIP

Each level 2 subfile contains all the values for one of the alternate keys. The level 2 subfiles have indexed sequential file organization with index blocks and data blocks. Each record in a data block contains an alternate key value and the first primary key value associated with it. Depending on the amount of available space in the data block and the size of the primary key list, the data block might contain additional primary key values.

A level 3 subfile contains primary key values that cannot be accommodated in the level 2 subfile. If alternate key values are unique, level 3 subfiles are not needed. The structure of the level 3 subfile is either indexed sequential or first-in first-out as specified when the alternate key is defined. Indexed sequential subfiles have index blocks and data blocks. First-in first-out level 3 subfiles have data blocks chained in a forward direction.

The physical structure of a MIP index file is shown in figure 2-15. A block in the figure can be either an index block or a data block. The block structure is identical to block structure in an indexed sequential file. All blocks within a subfile are chained together in a forward direction.

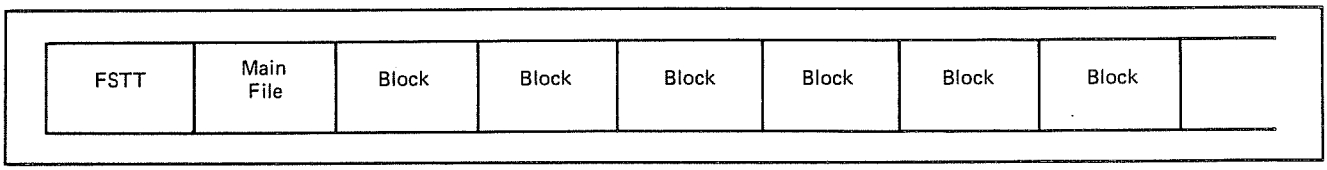


Figure 2-15. Index File Physical Structure, MIP

A file information table (FIT) is required for all AAM files. Information in the table defines the file and specifies how it is accessed. The FILE macro and the FILE control statement are used to create and update the FIT. The FILE macro assembles the FIT in the COMPASS program at the address where the macro is encountered. Pertinent information from the FILE control statement is saved until the file is opened; the saved information is then stored in the FIT and takes precedence over any corresponding preexisting information. A blank FIT, except for addressing information, file organization, and logical file name, could be set up in the user program with definition of file characteristics deferred until the file is opened.

The STORE macro or the FILE control statement can be used to change the setting of fields in the FIT. The fields are identified by the keywords of the FILE macro. The FETCH macro is used to retrieve the contents of a field in the FIT; a FILE macro keyword identifies the field being retrieved.

AAM macros that request file operations can result in amendment of FIT fields. Certain macro operands are stored in FIT fields before the request is performed and values can be stored in FIT fields as a result of processing the request. AAM also maintains certain fields in the FIT to reflect the current state of the file.

FILE MACRO

The FILE macro constructs the file information table at the address where the macro is encountered during assembly; the FIT must be built before the file is opened. The format of the FILE macro is shown in figure 3-1. The interaction between lfn and LFN=axxxxxx is shown in table 3-1.

The FILE macro does not check fields for validity or consistency. If the option specified for a field exceeds the maximum specified size, it is truncated and an assembler warning message is produced.

Misspelled or unrecognized parameters generate null parameters; the referenced fields are set to zero. Null parameters are ignored. Warning messages are generated when overlapping fields are specified.

[lfn] FILE [LFN=axxxxxx] [,keyword=option,] ...

lfn	Symbolic address where the FIT is assembled in the COMPASS program; if the LFN=axxxxxx is omitted or is the same name, logical file name by which the file can be referenced.
LFN	FIT field mnemonic for logical file name; if lfn is omitted, LFN must be specified with axxxxxx.
axxxxxx	Logical file name by which the file can be referenced; if lfn is omitted, symbolic address where the FIT is assembled in the COMPASS program.
keyword	Symbolic name of the FIT field.
option	Selected option of the FIT field.

Figure 3-1. FILE Macro Format

TABLE 3-1. LFN AND lfn INTERACTION

Statement	COMPASS Location Value	Contents of LFN Field in FIT
A FILE	A	A
FILE LFN=A	A	A
A FILE LFN=A	A	A
A FILE LFN=B	A	B

The FILE macro must specify the file organization (FO) mnemonic for an AAM file. Any parameter not applicable to the specified file organization is ignored and an error type 4 is generated during assembly.

The values specified for the FILE macro parameters are assembled into the FIT; parameters can be specified in any order. Table 3-2 shows the FILE macro parameters applicable to each AAM file organization. A detailed explanation of each FIT field that can be specified by the FILE macro parameters follows. The default value is indicated for each field.

The option aexp represents an absolute expression; exp is any COMPASS expression.

TABLE 3-2. FILE MACRO PARAMETERS BY FILE ORGANIZATION

Parameter	Indexed Sequential	Actual Key	Direct Access
BCK	X	X	X
BFS	X	X	X
CDT			
CL	X	X	X
CP	X	X	X
CPA	X	X	X
C1	X	X	X
DCA	X	X	X
DCT	X		
DFC	X	X	X
DP	X	X	
DX	X	X	X
EFC	X	X	X
EMK	X	X	X
ERL	X	X	X
EX	X	X	X
FL	X	X	X
FLM	X	X	X
FO	X	X	X
FWB	X	X	X
FWI	X	X	X
HL	X	X	X
HMB			X
HRL			X
IP	X		
KA	X	X	X
KL	X	X	X
KP	X	X	X
KT	X		
LFN	X	X	X
LL	X	X	X
LP	X	X	X
MBL	X	X	X
MKL	X		
MNR	X	X	X
MRL	X	X	X
NDX	X	X	X
NL	X		
OF	X		
ON	X	X	X
ORG	X	X	X
PD	X	X	X
PKA	X	X	X
RB	X	X	X
REL	X	X	X
RKP	X	X	X
RKW	X	X	X
RMK	X	X	X
RT	X	X	X
SB	X	X	X
TL	X	X	X
WSA	X	X	X
XBS	X	X	X
XN	X	X	X

BCK

Block checksum

BCK=NO (default) -

Checksums are not computed during file creation. For a file created with checksums, no checksumming is done during a read operation; however, checksums are computed for blocks written.

BCK=YES -

A checksum is computed before each block is written and after it is read. The checksum is part of the block.

BFS

Buffer size

BFS=0 (default) -

AAM provides the buffer space; the amount of common buffer space is increased by an amount determined by AAM.

BFS=aexp -

The buffer size is the number of words specified; maximum is $2^{17}-1$ or 131 000 words. If the FWB field is set to zero, AAM increases the amount of common buffer space allocated by BFS.

CDT

Collating sequence to display code conversion table; ignored if the DCT field is zero (initial indexed sequential files)

CDT=0 (default) -

Conversion table is generated from the table specified by the DCT field.

CDT=exp -

Conversion table is at the specified address.

CL

Trailer count field length (T type records)

CL=0 (default) -

For T type records, this field must be defined before the file is opened.

CL=aexp -

The length of the trailer count field is the specified number of characters; maximum is 6.

CP

Trailer count beginning character position (T type records)

CP=0 (default) -

The trailer count field begins in character position 0.

CP=aexp -

The specified number is the beginning character position, numbered from 0 on the left; maximum is 81870 for AAM files.

CPA

Compression/encryption routine number or address

CPA=0 (default) -

Records are not compressed unless a system routine was specified when the file was previously opened.

CPA=aexp -

The specified number identifies the system compression routine to be used; must be less than 100_8.

CPA=exp -

The user-supplied compression routine is at the specified address; must not be less than 100_8.

C1

COMP-1 format for length field (D or T type records)

C1=NO (default) -

The length field is in coded format.

C1=YES -

The length field is in binary (COBOL COMP-1) format.

DCA

Decompression/decryption routine address; required if the CPA field specifies a user routine

DCA=0 (default) -

If the CPA field specifies either no compression or a system compression routine, AAM sets DCA at open time if needed.

DCA=exp -

The user-supplied decompression routine is at the specified address; DCA must be specified if a user-supplied compression routine is specified.

DCT

Display code to collating sequence conversion table (indexed sequential files)

DCT=0 (default) -

CDC conversion tables are used.

DCT=exp -

The user-supplied table is at the specified address. For indexed sequential files, AAM generates the collating sequence to display code conversion table from the user-supplied table.

DFC

Dayfile control

DFC=0 (default) -

Only fatal error messages are written on the dayfile.

DFC=1 -

Error messages are written on the dayfile.

DFC=2 -

Statistics/notes are written on the dayfile.

DFC=3 -

Error messages and statistics/notes are written on the dayfile.

DP

Data block padding factor (indexed sequential and actual key files)

DP=0 (default) -

The installation default value is used for indexed sequential files; 0 for actual key files.

DP=aexp -

Padding for the data block is the specified percentage; maximum is 99.

DX

End-of-data exit

DX=0 (default) -

No end-of-data exit is specified.

DX=exp -

The routine at the specified address is enterec when an end-of-data condition occurs. The system stores a jump at the first address of the routine and control passes to the first executable statement, which is routine+1.

EFC

Error file control (file ZZZZZEG); The FITDMP macro forces EFC=0 to 2 and EFC=1 to 3

EFC=0 (default) -

No messages are written on the error file.

EFC=1 -

Error messages are written on the error file.

EFC=2 -

Statistics/notes are written on the error file.

EFC=3 -

Error messages and statistics/notes are written on the error file.

EMK

Embedded key; examined when the file is opened for creation

EMK=NO (default for indexed sequential and actual key files) -

The key is not part of the record.

EMK=YES (default for direct access files) -

The key is embedded in the record; the RKW, RKP, and KL fields define the position and length of the key.

ERL

Trivial error limit

ERL=0 (default) -

An indefinite number of trivial errors is permitted; no limit is specified.

ERL=aexp -

The specified number is the maximum number of trivial errors allowed before a fatal error occurs; maximum is 511.

EX

Error exit

EX=0 (default) -

No routine is entered if an error occurs; control is returned to the user's in-line code.

EX=exp -

The routine at the specified address is entered when an error occurs. The system stores a jump at the first address of the routine and control passes to the first executable statement, which is routine+1.

FL

Fixed length (F type records) or full length (Z type records)

FL=0 (default) -

The field must be defined before the file is opened.

FL=aexp -

For F type records, the specified number is the record length in characters; 10 through 81870 for AAM files.

For Z type records, the specified number establishes the upper limit of characters or blank padding moved to the working storage area.

FLM

File limit

FLM=0 (default) -

The file limit is not checked.

FLM=aexp -

The file limit cannot exceed the specified number of records.

FO

File organization (no default value)

FO=AK -

The file has actual key file organization.

FO=IS -

The file has indexed sequential file organization.

FO=DA -

The file has direct access file organization.

FWB

First word address of user-supplied buffer

FWB=0 (default) -

AAM provides the buffer space needed.

FWB=exp -

The user buffer is at the specified address.

FWI

Forced write indicator

FWI=NO (default) -

Each buffer is written only when the buffer space is needed for another input/output operation.

FWI=YES -

All buffers are written immediately after each operation that modifies the buffer content. This option increases file integrity by keeping the file current; however, performance is degraded as more input/output transfers are required.

HL

Header length (T type records)

HL=0 (default) -

For T type records, this field must be defined before the file is opened.

HL=aexp -

The fixed-length portion of the T type records is the specified number of characters; maximum is 81870 for AAM files. Minimum is the sum of the CP and CL fields.

HMB

Number of home blocks (direct access files)

HMB=0 (default) -

The field must be defined to open the file.

HMB=aexp -

The file contains the specified number of home blocks; maximum is $2^{24}-1$ or 16777215.

HRL

Hashing routine location; cannot be changed after file creation (direct access files)

HRL=0 (default) -

The system-supplied hashing routine is used.

HRL=exp -

The user-supplied hashing routine is at the specified address.

IP

Index block padding factor (indexed sequential files)

IP=0 (default) -

For indexed sequential files, the default value zero is used.

IP=aexp -

The index block padding is the specified percentage; maximum is 99.

KA

Key address

KA=0 (default) -

No address is specified for a key.

KA=exp -

The key value for the record to be processed is at the specified address; for the GETN macro, the key of reference is returned to the specified address.

KL

Key length

KL=0 (default) -

This field must be defined before the file is opened.

KL=aexp -

The key length is the specified number of characters. The positive integer that can be specified for the key length in indexed sequential files depends on the key type defined by the KT field.

KT=S or KT=U -
Symbolic key, maximum is the installation-defined key limit (default is 255).

KT=I -
Integer key, 10 characters must be specified for the signed binary key.

KP

Beginning key position (all AAM files except indexed sequential with KT=I)

KP=0 (default) -

The key is word-aligned.

KP=aexp -

The key begins in the specified character position within the KA field, numbered from 0 on the left; maximum is 9.

KT

Key type (indexed sequential files)

KT=S (default) -

A collated symbolic key is a string of 6-bit characters; sorted according to the sequence indicated by the display code to collating sequence conversion table.

KT=I -

An integer is a 10-character signed binary key; sorted by the magnitude of the key values.

KT=U -

An uncollated symbolic key is a string of 6-bit characters; sorted by the magnitude of their binary display code values.

LFN

Logical file name (no default value)

LFN=axxxxxx -

The data file logical file name is one to seven characters in length beginning with a letter.

LL

Length field length (D type records)

LL=0 (default) -

The field must be defined before the file is opened.

LL=aexp -

The length of the length field is the specified number of characters; maximum is 6.

LP

Length field beginning character position (D type records)

LP=0 (default) -

The length field begins in character position 0.

LP=aexp -

The length field begins in the specified character position, numbered from 0 on the left; maximum is 81870 for AAM files.

MBL

Maximum block length; should not be changed after the file is opened

MBL=0 (default) -

The installation default size is used.

MBL=aexp -

The data block is the specified number of characters in length. The specified size is increased to an integral multiple of PRU size minus two words. MBL should not be specified if a value for the RB field is given for indexed sequential files. If both are set, the value of the RB field is ignored. For indexed sequential files, MBL also specifies the length of the index blocks.

MKL

Major key length (indexed sequential files, symbolic key type); must be set with STORE macro after the file is opened.

MKL=0 (default) -

Major key length processing is not specified.

MKL=aexp -

The major key length is the specified number of characters; maximum is the KL value. The file is positioned at the first record with a key in which the first specified number of characters matches the major key. MKL is reset to zero after each START or GET.

MNR

Minimum record length

MNR=0 (default) -

The minimum record length is zero characters. Zero length records are not accepted in direct access and actual key files.

MNR=aexp -

The minimum record length is the specified number of characters; maximum is the MRL value.

MRL

Maximum record length

MRL=0 (default) -

This field must be defined before the file is opened for creation.

MRL=aexp -

The maximum record length is the specified number of characters; maximum is 81870 for AAM files. This establishes the upper limit of characters moved to the working storage area. The field must be specified for OPENM NEW and is returned for OPENM OLD.

NDX

Index flag (multiple-index files)

NDX=NO (default) -

The data file can be accessed by primary or alternate key.

NDX=YES -

Only the index file is accessed.

NL

Number of index block levels; used only when files are created (indexed sequential files)

NL=0 (default) -

The installation default value is used.

NL=aexp -

The number specified is the expected number of levels for the file; maximum is 15 for indexed sequential files.

OF

Open flag; file positioning at OPENM time (indexed sequential files)

OF=R (default) -

The file is rewound.

OF=E -

The file is positioned at end-of-information for extend.

ON

Old or new file

ON=OLD (default) -

The file is an existing file (FSTT exists).

ON=NEW -

The file is being created (FSTT to be established). The PD field must also be set to OUTPUT.

ORG

Currently supported file organization

ORG=NEW (default) -

The file organization is currently supported (previously known as extended).

PD

Processing direction

PD=INPUT (default) -

The file is open for input (read).

PD=OUTPUT -

The file is open for output (write).

PD=IO -

The file is open for input/output (read and/or write).

PKA

Primary key address

PKA=0 (default) -

The primary key is not returned on an alternate key read operation.

PKA=exp -

The primary key is returned to the specified address on an alternate key read operation.

RB

Records per block; used in block size calculation

RB=0 (default) -

RB is set to 1; the installation default is used if MBL is also zero.

RB=aexp -

Blocking factor limit is 4095. For indexed sequential files, RB should not be specified if the MBL field is specified.

REL

Key relation; relation of record key to key value at location KA. REL is significant only for START operations and index-only operations.

REL=EQ -

This specifies an equal relation.

REL=GE -

This specifies a greater than or equal relation.

REL=GT -

This specifies a greater than relation.

RKP

Relative key position (required if EMK is set to YES)

RKP=0 (default) -

The key is word-aligned starting at RKW position.

RKP=aexp -

The key begins in the specified position within RKW, numbered from 0 on the left; maximum is 9.

RKW

Relative key word (required if EMK is set to YES)

RKW=0 (default) -

The key begins in the first word of the record.

RKW=aexp -

The key starts in the specified word (numbered from 0) within the record.

RMK

Record mark character (R type records)

RMK=0 (default) -

The record mark character is the right bracket (]), which is 62_8.

RMK=ccB -

The record mark character is the specified octal value (cc); maximum is 77_8.

RMK=1Rx -

The record mark character is the specified character (x); any character is valid.

RMK=cc -

The record mark character is the specified decimal value (cc); maximum is 63.

RT

Record type

RT=W (default) -

This specifies a control word record; however, AAM processes this the same as RT=U.

RT=F -

This specifies a fixed length record.

RT=R -

This specifies a record mark record.

RT=Z -

This specifies a zero byte terminated record.

RT=D -

This specifies a decimal character count record.

RT=T -

This specifies a trailer count record.

RT=U -

This specifies an undefined record.

RT=S -

This specifies a system-logical-record; however, AAM processes this the same as RT=U.

SB

Sign overpunch format for length field (D or T type records)

SB=NO (default) -

The length field is in unsigned display code.

SB=YES -

The length field uses a COBOL sign overpunch scheme.

TL

Length of the trailer portion (T type records)

TL=0 (default) -

This field must be defined before the file is opened.

TL=aexp -

The length of the trailer portion is the specified number of characters; maximum is 131071.

WSA

Working storage area address

WSA=0 (default) -

No working storage area is specified.

WSA=exp -

The working storage area is at the specified address. This field must be set before any file processing macro uses the working storage area. It can be set by the GET, PUT, GETN, GETNR, and REPLACE macros.

XBS

Index file block size (multiple-index files, MIP)

XBS=0 (default) -

The index file blocks are the same size as the data file blocks.

XBS=aexp -

The index file blocks are the specified number of characters.

XN

Index file name (multiple-index files)

XN=0 (default) -

No accesses or updates by alternate key can be performed.

XN=lfn -

The index file for alternate key access is the file with the specified logical file name.

FILE CONTROL STATEMENT

The FILE control statement is used to specify file information to update the FIT either when the SETFIT macro is issued or the first time the file is opened in the job step. FILE control statements are not processed if NOFCP is set to YES by the FILE or STORE macro. This run-time control over file specification allows a single program to process files with different record types. Corresponding FIT fields have the value specified on the last control statement encountered.

FILE control statements must be placed before any program call in which the information in the statements is to be used. Because processing of the FILE control statement involves calling a central processor program, it should not be placed within a load set sequence. For example, the FILE control statement should not be placed between the LOAD and EXECUTE control statements.

A FILE control statement cannot be continued to a second card or card image, but the same logical file name can appear on more than one FILE control statement. If more than one FILE control statement appears for a given file, the data on the first control statement can be overwritten by the data on a subsequent control statement when overlapping fields occur in those statements. The FILE control statement conforms to operating system coding conventions.

When an error diagnostic is produced by FILE control statement processing, the entire statement is ignored. FILE control statement diagnostics are written on the dayfile as soon as the error is encountered; diagnostics name the faulty parameter and are self-explanatory. Control is then passed to the next EXIT control statement. No EXIT is taken for advisory diagnostics.

The format of the FILE control statement is shown in figure 3-2. Keywords can be specified in any order. Keywords have the same meanings as described for the FILE macro.

FILE(lfn[=axxxxxx] [,keyword=option] ...)

lfn	Name of the FIT; required.
=axxxxxx	Optional new name for the FIT; allows a file to be requested by a new name without reassembly.
keyword=option	Symbolic name of the FIT field and the option selected.

Figure 3-2. FILE Control Statement Format

If only the lfn and FO parameters appear in the FILE control statement and no subsequent FILE control statement references that file, FIT fields for all succeeding job steps are those specified in the program. If the FILE control statement appears without any parameters, FIT fields for all files revert back to those specified in the program for all succeeding job steps until another FILE control statement is encountered. Except for the USE and OMIT parameters, all parameters valid in a FILE control statement are valid in a FILE macro.

The FILE control statement parameters are listed in table 3-3. The various options for a keyword are separated by the | symbol. If the keyword is selected, one of the options must be selected and the others must be omitted. Parameter values are absolute and generally reference a number of characters. Value formats are denoted as:

n...n	Decimal value
n...nB	Octal value
n...nW	Decimal value, specified in words

Descriptions of the FILE control statement parameters are the same as for the corresponding FILE macro parameters.

TABLE 3-3. FILE CONTROL STATEMENT PARAMETERS

Keyword	Options	Keyword	Options	Keyword	Options
BCK	NO|YES	FWI	NO|YES	OF	R|N|E
BFS	0|n...n|n...nB	HL	0|n...n|n...nB|n...nW	OMIT	macro name/macro name/...
CF	R|N|U|RET|DET	HMB	0|nn	ON	OLD|NEW
CL	0|n...n|n...nB|n...nW	IP	0|nn	ORG	NEW
CP	0|n...n|n...nB|n...nW	KL	0|n...n|n...nB|n...nW	PD	INPUT|OUTPUT|IO
CPA	0|n...n|n...nB	KP	0|n...n|n...nB	RB	0|n...n|n...nB
C1	NO|YES	KT	S|I|F|U	RKP	0|n...n|n...nB
DFC	0|1|2|3	LFN	lfn	RKW	0|n...n|n...nB
DP	0|nn	LL	0|n...n|n...nB	RMK	0|ccB|1Rx|cc
EFC	0|1|2|3	LP	0|n...n|n...nB|n...nW	RT	W|F|R|Z|D|T|U|S
EMK	NO|YES	MBL	0|n...n|n...nB|n...nW	SB	NO|YES
ERL	0|n...n|n...nB	MNR	0|n...n|n...nB|n...nW	TL	0|n...n|n...nB|n...nW
FL	0|n...n|n...nB|n...nW	MRL	0|n...n|n...nB|n...nW	USE	macro name/macro name/...
FLM	0|n...n	NDX	NO|YES	XN	lfn
FO	IS|DA|AK	NL	0|n...n|n...nB	XN	lfn

RUN-TIME MANIPULATION

The user can communicate with AAM through the FIT without knowing the exact format of the FIT. This is done with the FETCH, STORE, and SETFIT macros; FIT field mnemonics are used in the FETCH and STORE macros.

FETCH MACRO

The FETCH macro retrieves the contents of a specified FIT field by a reference to its mnemonics. The format of the FETCH macro is shown in figure 3-3.

FETCH fit,keyword,xi,f,m

fit	Logical file name address of the FIT, or any COMPASS expression giving the FIT address.
keyword	Any of the keywords in the FILE macro, FILE control statement, or any of the following:

BN	Block number
BZF	Busy FET address
ECT	Trivial error count
ES	Error status
FNF	Fatal/nonfatal flag
FP	File position
LOP	Last operation code
OC	Open/close flag
RC	Record count
RL	Record length
WPN	Write bit

Xi	X register to receive the value of the requested field.
f	Number of the X register used to fetch the FIT word; must be 1 through 5 (default is 5).
m	Number of the X register used as a mask (default is 7).

Figure 3-3. FETCH Macro Format

If the specified keyword represents a 1-bit field, the value of the field is returned in the sign bit of the X register; the contents of the remainder of the X register are undefined. File names are returned left-justified with zero fill. All other fields are returned right-justified with zero fill.

FIT field mnemonics can be any of the keywords used with the FILE macro or any of the fields listed in figure 3-3. The macro generates code to extract the requested value from the FIT. The code expansion destroys values in user registers Xf, Xm, Af, and Xi (which can be Xf or Xm).

STORE MACRO

The STORE macro places a user-determined value in a specified FIT field at execution time. The format of the STORE macro is shown in figure 3-4.

STORE fit,keyword= { value / option / Ri } ,f,s,m

fit	Address of the FIT or any COMPASS expression giving the address.
keyword	Any keyword described in connection with the FILE macro.
value	Integer value associated with the keyword; when the keyword represents a length, it is specified in characters.
option	Option associated with the keyword.
Ri	Any register containing the proper value for the keyword.
f	Number of the X register used to fetch the FIT word; must be 1 through 5 (default is 5).
s	Number of the X register used to store the FIT word; must be 6 or 7 (default is 6).
m	Number of the X register used as a mask (default is 7).

Figure 3-4. STORE Macro Format

Most FIT fields listed in appendix D can be set symbolically by the STORE macro. Some fields, such as the file structure parameters, are protected against being changed by the STORE macro. Other fields are not protected but should not be changed after the file has been opened.

A field can be set by using the option with the keyword or by using a register to hold the option as shown in figure 3-5. Examples a and b have the same effect.

```
a.  STORE   fit,RL=10

b.  SX1     10
    STORE   fit,RL=X1

c.  STORE   fit,FO=IS
```

Figure 3-5. STORE Macro Examples

The STORE macro generates code to store the requested value in the FIT. This code expansion destroys the values in user registers Xf, Xs, Xm, Af, As, and Xi (which can be Xf, Xs, or Xm).

SETFIT MACRO

The SETFIT macro sets fields in the FIT according to information provided in the FILE control statement. This normally occurs when the OPENM macro is executed. The SETFIT macro makes it possible for system routines to obtain information, such as run-time buffer requirements, needed by other system routines. The format of the SETFIT macro is shown in figure 3-6.

```
SETFIT  fit

fit     Address of the FIT or register containing the
        address of the FIT.
```

Figure 3-6. SETFIT Macro Format

The SETFIT macro is valid only for a closed file. Any attempt to execute this macro for an open file results in an error. Once the FILE control statement values are placed in the FIT, the macro sets the processed flag (PDF) field to inhibit further FILE control statement processing when the OPENM macro is executed. The flag is cleared during subsequent OPENM processing.

If the buffer size (BFS) field is zero for an existing file, the parameters from the file statistics table are placed in the FIT; the buffer size returned to the BFS field is based on these values. After a buffer is calculated, the open/close (OC) field and first word address of the buffer (FWB) field are cleared.

For a new file, the SETFIT macro should not be issued unless sufficient information exists for buffer calculations. Parameters needed for buffer calculation are shown in table 3-4.

TABLE 3-4. BUFFER CALCULATION PARAMETERS

File Organization	User Must Supply	Parameter	User Can Supply or Default is Used	Parameter
Indexed Sequential	Key length	KL	Maximum block length	MBL
	Key type	KT	Index block padding factor	IP
	Maximum record length	MRL	Data block padding factor	DP
			Index block specification	NL
			Embedded key	EMK
			Compression routine	CPA
Direct Access	Home block number	HMB	Blocking factor	RB
	Key length	KL	Embedded key	EMK
	Maximum block length	MBL or	Compression routine	CPA
	Maximum record length	MRL and		
	Minimum record length	MNR		
Actual Key	Maximum block length	MBL or	Blocking factor	RB
	Maximum record length	MRL and		
	Minimum record length	MNR		

FILE PROCESSING 4

This section provides general processing information and explains by file organization the logical operations of processing AAM files. Macros and FIT fields are discussed as applicable to the type of processing for each file organization. The macros and their parameters are described in general in section 5, File Processing Macros. Detailed explanations of the FIT fields are in section 3, File Information Table. Processing of multiple-index files is discussed in section 6, Multiple-Index Files.

GENERAL PROCESSING INFORMATION

Certain processing procedures are common to all AAM file organizations. These procedures are explained in the following paragraphs. Processing unique to each file organization is discussed by file organization.

FILE INFORMATION TABLE

Before an AAM file can be processed, the file information table (FIT) must be established. This provides the name by which the file can be referenced and defines the file structure and processing limitations. The FIT contains fields that are referenced whenever AAM processes the file. FIT fields can be set before file processing by the FILE control statement, FILE macro, SETFIT macro, or STORE macro.

FILE STATISTICS TABLE

A separate creation run is necessary for AAM files. This creation run establishes the file statistics table (FSTT), which becomes a permanent part of the file. The FSTT contains FIT fields that cannot be changed for the life of the file. When the file is opened for processing after its creation run, the FIT fields are automatically established from information in the FSTT of the file.

OPENM MACRO

All files must be initialized using the OPENM macro. Applicable default values are inserted into FIT fields for certain values not supplied before executing the OPENM macro. AAM also performs certain consistency checks on FIT fields when the file is opened. Refer to the OPENM macro description in section 5 for the FIT fields that are checked.

INPUT/OUTPUT MACROS

The GET, GETN, and GETNR macros read records from a file. A working storage area must be established to pass data to the program from a file storage device. The user defines the working storage area (WSA) by supplying an address for the WSA field in the FIT. A GET macro transfers data from the buffer area, which is set up either by the user or by AAM when the file is opened, to the working storage area.

The PUT macro is used to write records to the file. A working storage area must be established to pass data from the program to a file storage device. The PUT macro transfers data from the working storage area to the buffer area, which is set up either by the user or by AAM when the file is opened. The maximum record length (MRL) field in the FIT must be set by the user on a file creation run and becomes a permanent part of the file. The value specified in the MRL field becomes the upper limit on the number of characters that can be transferred.

CLOSEM MACRO

At completion of processing, all files must be closed by the CLOSEM macro. Any remaining records of an output file are written from the buffer to the file storage device, the open/close (OC) field in the FIT is set to closed, and control is returned to the user. Execution of the CLOSEM macro causes the FSTT to be updated; if requested, file statistics are written to the error file ZZZZZEG.

END-OF-DATA ROUTINE

The end-of-data exit (DX) field in the FIT specifies the address of a user routine for processing an end-of-data condition. End-of-data occurs when beginning-of-information (BOI) or end-of-information (EOI) is encountered while attempting a data transfer or positioning operation.

Control is passed to the address (DX)+1; a jump back to the user in-line return code is stored at the DX address. The file position (FP) field indicates the specific end condition (BOI or EOI).

When file position is at EOI, the GETN macro transfers control to the end-of-data exit. If continued GETN macros are issued without repositioning the file, the GETN macro issues an error and transfers control to the error exit (if specified) instead of to the end-of-data exit. No GETN macro that passes control to the end-of-data exit causes data to be transferred to the working storage area. Control is passed to the end-of-data exit only when end-of-information is encountered. The FP field is not set until the file is logically at the end of information.

For indexed sequential and actual key files, control is transferred to the end-of-data exit whenever a SKIP macro encounters EOI or BOI. A trivial error condition is produced by successive SKIP macros after end-of-data has been encountered.

INDEXED SEQUENTIAL FILES

The indexed sequential file organization is well suited for applications that require reasonably efficient storage and retrieval of records both randomly and sequentially by primary or alternate key. A primary key is a unique

identifier defined by the user for each record within an indexed sequential file. Primary and alternate keys can be in any of the following forms:

- 60-bit signed binary (10 characters)
- Symbolic (1 to 255 contiguous 6-bit characters)
- Uncollated symbolic (1 to 255 contiguous 6-bit characters)

The value of the primary key determines the location of the record in the file. Characters within a symbolic key are collated according to the standard CDC collating sequence or according to a user-supplied collating sequence. Any user collating sequence has meaning for ranking keys only; it is stored with the user file in the FSTT. Numeric keys are ordered by value. Keys within an indexed sequential file can be a part of the record (embedded) or not a part of the record (nonembedded). Refer to the CYBER Record Manager Advanced Access Methods Version 2 User Guide for information on defining a user-supplied collating sequence.

Refer to the CYBER Record Manager Advanced Access Methods Version 2 User Guide for information on defining a user-supplied collating sequence.

FILE CREATION RUN

A separate creation run is necessary for an indexed sequential file. This can be done through the FORM utility or through a source program. The FSTT is created when the indexed sequential file is created.

The efficiency with which an indexed sequential file can be processed is influenced by two fields in the FIT: maximum block length (MBL) and buffer size (BFS). On a creation run, the user has the option of specifying these values directly or accepting system defaults calculated by AAM. The FLBLOK utility, which is described in section 7, can be used to calculate suggested values for the MBL and BFS fields.

If the MBL field is not specified directly, the value is calculated from the values of the following fields in the FIT:

DP	Data block padding
KL	Key length
MNR	Minimum record length
MRL	Maximum record length
RB	Records per block

A number of fields in the FIT determine the size and characteristics of data and index blocks during file creation. Data and index blocks must be the same size; padding percentages, however, can be different. The following FIT fields are used to calculate data block size:

DP	Data block padding
KL	Key length
KT	Key type
MBL	Maximum block length
MNR	Minimum record length
MRL	Maximum record length
RB	Records per block

The FIT fields used to calculate index block size are as follows:

IP	Index block padding
KL	Key length
MBL	Maximum block length
MNR	Minimum record length
MRL	Maximum record length
NL	Number of index levels
RB	Records per block

Certain FIT fields must be set by the user before the file is opened on a creation run; otherwise, a fatal error occurs. These fields can be specified in the FILE control statement, FILE macro, or STORE macro. Any attempt to change these fields after file creation is ignored without comment. The FIT fields that must be set are as follows:

FO	File organization
KL	Key length
KT	Key type
LFN	Logical file name
MRL	Maximum record length
ORG	Currently supported file organization. Must be set to NEW (default is NEW)

If the primary key is embedded in the record, the following FIT fields must also be set:

EMK	Embedded key, set to YES
RKP	Relative key position; character position within RKW in which the key begins
RKW	Relative key word; word in which the key begins

Other FIT fields that must be defined before the file is opened on a creation run can be set by the user or can assume default values. These fields remain the same for the life of the file and attempts to change them are ignored.

BCK	Block checksum; default is no checksums
DCT	Display code to collating sequence conversion table; default is CDC conversion table
DP	Data block padding percentage; release default is 0
IP	Index block padding percentage; release default is 0
MBL	Maximum block length, data and index blocks; default is calculated by AAM

MNR Minimum record size; cannot exceed value of MRL; default is 0

NL Number of index levels; maximum is 15; release default is 1

RB Records per block; should not be specified if MBL is specified; release default is 2

XBS Index file block size; default is data file block size (MBL)

Some FIT fields that can be specified before the file is opened for creation are in effect only until another OPENM macro is executed. Attempted changes are ignored without comment or error until the file is opened again; the values in the FIT are then used to accomplish the open. Default values are assumed without comment if the following fields are not set:

BFS Buffer size; default is buffer size calculated by AAM

FWB First word address of the buffer; default is buffer address provided by AAM

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time (whether new or old) and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.

2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-new time, the user has the following choices for CPA and DCA:

1. Set CPA=0 (the default value), which excludes all compression and decompression for the life of the file; DCA will never be significant thereafter. CPA must never be set nonzero during the life of the file; this could cause an abort, or at the very least a nonfatal error, whenever a PUT or REPLACE is attempted.

2. Set CPA to the actual address of a user-supplied compression routine, and DCA to that of the corresponding user-supplied decompression routine. The fields should not be changed until the file is closed. It is the user's responsibility to ensure that whenever the file is opened in the future as an old file the same routines are available and are pointed to by CPA and DCA. The only exception is that CPA can at any time be set to 0 to stop compression until the next CLOSEM.

3. Set CPA to an integer in the range 1 through 63 to select system-supplied compression and decompression routines. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. When the file is opened, AAM loads the routines and stores their addresses into CPA and DCA. The user must not alter CPA or DCA until the file is closed, except to stop compression by setting CPA=0.

Two FIT fields have no default value and must be set before being used by a file processing macro. If the following fields are not set before required, a fatal error occurs:

KA Key address

WSA Working storage area

When records are written to a file on a creation run, the primary keys should be in ascending sequence for a more efficient run.

The old/new file (ON) field must be set to NEW and the processing direction (PD) field to OUTPUT for a file creation run. These fields can be set by using FIT manipulation statements or by setting the processing direction (pd) parameter to NEW in the OPENM macro. Setting the pd parameter to NEW sets the ON field to NEW and the PD field to OUTPUT.

EXISTING FILE PROCESSING

Indexed sequential files must reside on mass storage devices for processing. After file creation, however, the file can be dumped to tape with a COPYBF statement or a permanent file dump routine. The file can be returned later to mass storage for processing.

Open Processing

Before an existing file can be opened, the user must call for construction of the FIT by specifying the logical file name and the file organization. When the file is opened, values from the FSTT are returned to the following FIT fields:

KL Key length

KT Key type

MBL Maximum block length

MNR Minimum record length

MRL Maximum record length

NL Number of index levels

RKP Relative key position

RKW Relative key word

The RKW and RKP fields are set to 0 and 10, respectively, if the key is not embedded in the record.

A default value is assumed without comment if the following FIT fields are not set before opening the file:

BCK Block checksum; default is no checksums

BFS Buffer size; default is buffer size calculated by AAM

FWB First word address of the buffer; default is buffer address provided by AAM

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.

2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-old time, the user should set these fields depending on what was done when the file was created:

1. If CPA was 0 when the file was created, compression was excluded for the life of the file. Whenever the file is opened, AAM automatically sets CPA and DCA to 0.

2. If CPA was an integer in the range 1 through 63 when the file was created, system-supplied compression and decompression routines were specified for the file. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. Whenever the file is opened, these routines are automatically loaded, and CPA and DCA are automatically set to point to them.

3. If CPA was any value greater than 63, it was assumed to be the address of a user-supplied compression routine. The same routine and its corresponding decompression routine must be supplied by the user at every subsequent open, and CPA and DCA must be set by the user to point to them. AAM calls the compression subroutine and checks to ensure it is the same routine that was used during creation; if it is not, a fatal error is declared.

Two FIT fields have no default value and must be set before being used by a file processing macro. If the following fields are not set before required, a fatal error occurs:

KA	Key address
WSA	Working storage area

Other fields that can be set before the file is opened but need not be set until required by a file processing macro are as follows:

DFC	Dayfile control
EFC	Error file control
ERL	Trivial error limit
EX	Error exit
FLM	File limit
FWI	Forced write indicator
KP	Beginning key position
MKL	Major key length

The MKL field is reset to zero after execution of a GET, SEEK, or START macro. The other fields remain in effect until changed.

The first time an existing file is opened after its creation run, the old/new file (ON) field in the FIT must be changed from NEW to OLD. This can be done through the FILE macro, FILE control statement, or STORE macro or by specifying any option except NEW in the processing direction (pd) parameter of the OPENM macro.

An existing file can be positioned at end-of-information during open processing. This position is established by specifying the E option in the open flag (of) parameter of the OPENM macro or by setting the open flag (OF) field in the FIT to E through the STORE macro, FILE control statement, or FILE macro before the file is opened.

Read Processing

Records can be read from the file randomly by key value or sequentially by position. The key of reference for a read operation can be the primary key or any alternate key defined for the file. The file must be open for input or for input/output.

The GET macro is used for a random read operation. The relative key word (RKW), relative key position (RKP), and key length (KL) fields in the FIT determine whether the read operation is by the primary key or by one of the alternate keys. For a nonembedded primary key, RKW and RKP must be set to 0 and 10, respectively. The key value at the address specified by the key address (KA) field is used to locate the record to be read. The user must set the KA field to the address of the key value. A trivial error condition results if the specified key is not found in the file; however, the file position is altered to point to where the record should exist.

Sequential reading is accomplished by the GETN and GETNR macros. The GETN macro returns the next sequential record to the working storage area. The GETNR macro performs this same function; however, control returns immediately to the user if input/output is required to complete the request. The macro can be issued repeatedly until the transfer of the record is complete, or the input/output status can be monitored for completion before issuing the GETNR macro again.

If KA is not zero, both GETN and GETNR return the key value at the location specified by KA and KP upon completion of the record transfer.

Write Processing

New records are added to an existing indexed sequential file with the PUT macro. Records are inserted by primary key value. For a nonembedded primary key, the user must set the KA field in the FIT to the address of the key value. Execution is faster if the records to be inserted are sorted by primary key in ascending order.

Random Processing

Random processing implies index block manipulation as well as record processing. If the user cannot allow AAM to use the Common Memory Manager (CMM), maximum efficiency is gained by allowing buffer space for one index block for each index level and space for two data blocks. This number of index blocks allows the primary index block to remain in memory while processing the other index and

data blocks. Two data blocks provide input/output/compute overlap. The user can direct AAM to allocate this amount of buffer space by setting the buffer size (BFS) field and by not setting the first word address of the buffer (FWB) field. Refer to appendix G for a detailed description of buffer allocation.

If no input/output is in progress for the file, a write is initiated for any data block that satisfies the following conditions:

- The block was altered by the preceding macro.
- The block is not the object of the current macro.

This permits a high degree of input/output/compute overlap; however, if the forced write indicator (FWI) field in the FIT is set, each modified block is written immediately.

Major Key Processing

The major key feature is available with the GET, SEEK, and START macros. It allows the user to perform a search on the leading characters of a symbolic key. Major key processing on primary keys applies only to collated symbolic keys. Major key processing on alternate keys applies to collated or uncollated symbolic keys. When the major key length (mkl) parameter is specified in the GET macro, the record returned to the working storage area is the first one encountered with a major key that matches the specified major key value. Presumably, the user wishes to examine a subset of records defined by the major key; the subset is processed using the GETN or GETNR macro to access the records belonging to the subset.

The START macro can also include the mkl parameter. When it is specified, the file is positioned at the first record containing a major key that matches the specified major key value. A record is not returned to the working storage area by the START macro.

When the mkl parameter is specified in the SEEK macro, AAM initiates transfer into the buffer of an index block or the data block containing the first occurrence of the major key. Other program processing can occur while the transfer is taking place.

The file position (FP) field in the FIT can be checked for the status of the block transfer. The FP field has the value 0 if an index block is being transferred or the value 20_8 if a data block is being transferred. If the value of the FP field is 0, another SEEK macro can be issued and a check made of the FP field. This can be done repeatedly until the data block is transferred into the buffer. The GET macro can then be issued to transfer the record containing the first occurrence of the major key from that data block in the buffer to the working storage area. The GET macro can be issued when the FP field contains 0, but then there is no overlap in processing.

File Updating

The DELETE macro physically removes the key and its associated record from the file. The key address (KA) field in the FIT must be set to point to the address of the primary key value for the record to be deleted. If the deleted record is the only one in the data block, the block is linked into a chain of deleted blocks to be used when new blocks are required for file expansion. If the delete operation results in an empty index block, the block is linked into the chain of deleted blocks.

The REPLACE macro replaces an existing record with the record in the working storage area. The primary key value for the record in the working storage area must be the same as the primary key value for an existing record. For a nonembedded primary key, the KA field must be set to point to the primary key for the working storage area record.

File Positioning

When the OPENM macro is executed, positioning of the file depends on the open flag (of) parameter in the macro. If R (rewind) is specified, the file is positioned at the first record, which is the record with the lowest primary key value. If E (end-of-information) is specified, the file is positioned after the last record, which is the record with the highest primary key value. Omitting the parameter causes the current value of the OF field in the FIT to be used. File positioning remains unchanged until one of the following macros is executed: GET, GETN, GETNR, REWINDM, SKIP, or START.

The GET macro, which accesses a record randomly, alters the file position to the record returned by the macro. The GETN macro, which accesses a record sequentially, advances the file position one logical record and returns that record unless the file is positioned at end-of-information. The GETNR macro also advances the file position one logical record when it returns a record.

The REWINDM macro positions the file to beginning-of-information; execution of the GETN or GETNR macro then returns the first record in the file. The SKIP macro positions the file forward or backward the specified number of records; the file is positioned at beginning-of-information or end-of-information if the skip count is too large.

The START macro positions the file according to a specified key value and key relation; the file is positioned at the record with a key value that is equal to (EQ), greather than or equal to (GE), or greater than (GT) the specified key value. If the specified key value does not exist in the file, the file is positioned at the record with the next greater key value.

Overlap Processing

In response to a user program request for a record, AAM locates the data block by searching the index blocks and transfers the data block from mass storage to the buffer area. The record is then transferred to the working storage area. The execution time to do this can be overlapped with program processing by using the SEEK macro or the GETNR macro.

The SEEK macro initiates the transfer of an index or data block from mass storage to the buffer; the macro then returns control to the user program, which can perform other operations while the read is in progress. To find out when the read is complete, the user program can check the completion bit in the FET to which the busy FET address (BZF) field of the FIT points. Bit 0 of the word at address BZF will change from 0 to 1 at completion. To find out whether an index or a data block is being (or has just been) read, the user program can test the FP field of the fit

FP=0 for an index block or 20_8 for a data block. The basic procedure for using SEEK with GET to achieve overlap is as indicated in the following steps:

1. Issue a SEEK with the given key.

2. Wait until the completion bit is 1. Presumably this time is partly used for work not related to I/O, or for I/O on a different file.

3. Test FP. If it is 0, return to step 1.

4. If FP is not 0, it should be 20_8. The data block containing the wanted record (if the file contains a match to the given key) is already in memory.

5. Issue a GET with the same key. This accesses the proper data block, which is already in memory, and returns the wanted record or reports the error.

The SEEK itself never returns a record to the working storage area.

The following list illustrates the use of SEEK to overlap GET operations on two different files:

```
SEEK    key A on FILE1
SEEK    key X on FILE2
GET     key A on FILE1
SEEK    key B on FILE1
GET     key X on FILE2
SEEK    key Y on FILE2
```

Before each SEEK or GET, the user program would wait until the completion bit for that file was 1. Then FP would be tested, and if it was nonzero, the final GET, instead of another SEEK, would be done.

In the following list, however, the effect of the first SEEK on each file would be nullified by the second SEEK on that file, which uses a different key; this illustrates that two GETs on a single file cannot be overlapped with each other by using SEEK:

```
SEEK    key A on FILE1
SEEK    key X on FILE2
SEEK    key B on FILE1
SEEK    key Y on FILE2
GET     key A on FILE1
GET     key X on FILE2
GET     key B on FILE1
GET     key Y on FILE2
```

All the SEEKs would be completely wasted. The effect of SEEK would also be nullified by a later PUT, DELETE, or REPLACE.

The GETNR macro can be used to read records sequentially, while overlapping the block reads with central processor activity, or with I/O on a different file.

GETNR, like SEEK, returns control to the user program as soon as a block read is initiated. The procedure for using GETNR is as follows:

1. Issue a GETNR.

2. If FP is 0, wait until the completion bit (the rightmost bit of the word to which BZF points) is 1 and then return to step 1.

3. If FP is not 0, the action is complete. The record and key value have been returned (or not) and FP has been set to 20_8 or 100_8, as if the preceding operation had been a GETN rather than a GETNR.

ACTUAL KEY FILES

The actual file organization provides fast random access to records in the file. Random access usually requires one access per record. The primary key for a record is its storage location (record number within the file). The user must preserve primary keys if the file is to be accessed randomly by primary key.

FILE CREATION RUN

A separate creation run is necessary for an actual key file. This can be done through the FORM utility or a source program. The FSTT is created when the actual key file is created.

Certain FIT fields must be set by the user before the file is opened on a creation run; otherwise, a fatal error occurs. These fields can be specified in the FILE control statement, FILE macro, or STORE macro. Any attempt to change these fields after file creation is ignored without comment. The FIT fields that must be set are as follows:

FO	File organization
KL	Key length (in characters)
LFN	Logical file name
MNR	Minimum record length
MRL	Maximum record length
RT	Record type

Three FIT fields that must be defined for file creation can be specified by the user or can assume default values.

BCK	Block checksum; default is no checksums
DP	Data block padding percentage; release default is 0
MBL	Maximum block length; default is calculated by AAM

If the primary key is embedded in the record, the following FIT fields must also be set:

EMK	Embedded key, set to YES
RKP	Relative key position; character position within RKW in which the key begins
RKW	Relative key word; word in which the key begins

If the MBL field is not specified directly, the value is calculated from the values in the following fields:

MNR	Minimum record length
MRL	Maximum record length
RB	Records per block

The value specified for the MBL field should be at least (15*RB)+(10*RB)*((MNR+MRL)/2). AAM increases the block size, if necessary, to use mass storage efficiently. Resulting blocks are an integral multiple of physical record unit (PRU) size minus two central memory words.

The following FIT field must be selected before the file is created if the option is to be used during the life of the file:

RB Records per block

Some FIT fields that can be specified before the file is opened for creation are in effect only until another OPENM macro is executed. Attempted changes are ignored without comment until the file is opened again; the values in the FIT are then used to accomplish the open. Default values are assumed without comment if the following fields are not set:

BFS Buffer size; default is buffer size calculated by AAM

FWB First word address of the buffer; default is buffer address provided by AAM

The old/new file (ON) field must be set to NEW and the processing direction (PD) field to OUTPUT for a file creation run. These fields can be set by using FIT manipulation statements or by setting the processing direction (pd) parameter to NEW in the OPENM macro. Setting the pd parameter to NEW sets the ON field to NEW and the PD field to OUTPUT.

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time (whether new or old) and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.

2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-new time, the user has the following choices for CPA and DCA:

1. Set CPA=0 (the default value), which excludes all compression and decompression for the life of the file; DCA will never be significant thereafter. CPA must never be set nonzero during the life of the file; this could cause an abort, or at the very least a nonfatal error, whenever a PUT or REPLACE is attempted.

2. Set CPA to the actual address of a user-supplied compression routine, and DCA to that of the corresponding user-supplied decompression routine. The fields should not be changed until the file is closed. It is the user's responsibility to ensure that whenever the file is opened in the future as an old file the same routines are available and are pointed to by CPA and DCA. The only exception is that CPA can at any time be set to 0 to stop compression until the next CLOSEM.

3. Set CPA to an integer in the range 1 through 63 to select system-supplied compression and decompression routines. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. When the file is opened, AAM loads the routines and stores their addresses into CPA and DCA. The user must not alter CPA or DCA until the file is closed, except to stop compression by setting CPA=0.

If EMK=NO the key value at the address specified by the key address (KA) and key position (KP) fields in the FIT determines where the record is written. If EMK=YES, the key value in the record pointed to by RKW and RKP determines where the record is written. The user can do either of the following:

- The key value can be set to zero; this allows AAM to determine the key value associated with the record.
- The key value can be set to a properly formatted key; this tells AAM where to store the record.

Only the following macros can be used during a creation run:

OPENM

REWINDM

PUT

CLOSEM

EXISTING FILE PROCESSING

Actual key files must reside on mass storage for processing. After file creation, however, the file can be dumped to tape with a COPYBF statement or a permanent file dump routine. The file can be returned later to mass storage for processing.

Open Processing

Before an existing file can be opened, the user must call for construction of the FIT by specifying the logical file name and the file organization. When the file is opened, values from the FSTT are returned to the following FIT fields:

KL Key length

MBL Maximum block length

MNR Minimum record length

MRL Maximum record length

RB Records per block

RKP Relative key position

RKW Relative key word

If EMK=NO, AAM sets RKW=0, RKP=10.

A default value is assumed without comment if the following FIT fields are not set before the file is opened:

BCK Block checksum; default is no checksums

BFS Buffer size; default is buffer size calculated by AAM

FWB First word address of the buffer; default is buffer address provided by AAM

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.

2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-old time, the user should set these fields depending on what was done when the file was created:

1. If CPA was 0 when the file was created, compression was excluded for the life of the file. Whenever the file is opened, AAM automatically sets CPA and DCA to 0.

2. If CPA was an integer in the range 1 through 63 when the file was created, system-supplied compression and decompression routines were specified for the file. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. Whenever the file is opened, these routines are automatically loaded, and CPA and DCA are automatically set to point to them.

3. If CPA was any value greater than 63, it was assumed to be the address of a user-supplied compression routine. The same routine and its corresponding decompression routine must be supplied by the user at every subsequent open, and CPA and DCA must be set by the user to point to them. AAM calls the compression subroutine and checks to ensure it is the same routine that was used during creation; if it is not, a fatal error is declared.

Other FIT fields that can be set before the file is opened but need not be set until required by a file processing macro are as follows:

DFC	Dayfile control
DX	End-of-data exit
EFC	Error file control
ERL	Trivial error limit
EX	Error exit
FLM	File limit
FWI	Forced write indicator
KP	Beginning key position

The first time an existing file is opened after its creation run, the old/new file (ON) field in the FIT must be changed from NEW to OLD. This can be done through a FIT manipulation macro or by specifying any option except NEW for the processing direction (pd) parameter in the OPENM macro.

Read Processing

The GET macro is used to read records randomly by key value. The relative key word (RKW), relative key position (RKP), and key length (KL) fields in the FIT determine whether the read operation is by the primary key or by one of the alternate keys. For a nonembedded primary key, RKW and RKP must be set to 0 and 10, respectively. The key value at the address specified by the key address (KA) and key position (KP) fields is used to locate the record to be read. The user must set the KA field to the address of the key value and the KP field to the character position within KA that begins the key value.

When a record is read, the number of characters retrieved is returned to the record length (RL) field in the FIT. If the requested record is not found, a trivial error results.

Execution of the GETN macro causes the next sequential record to be placed in the working storage area. The first time the GETN macro is issued after the file is opened or after any rewind request, the first record in the file is retrieved. The next GETN macro retrieves the next sequential record. Any empty record position is ignored. Overflow records are returned as they are encountered. An overflow record occupies two slots; the first slot is the one where the record should be and the second slot is the one that actually contains the record. The record is returned when the first slot is encountered. If the key address (ka) parameter is specified in the GETN macro, the primary key of the record retrieved is returned to the specified address.

The GETNR macro performs the same function; however, control returns immediately to the user while an input process may be concurrently taking place. The macro can be issued repeatedly until the transfer of the record is complete, or the status can be monitored for completion through the busy FET address (BZF) field before issuing the GETNR macro again.

Write Processing

The PUT macro is used to add a record to an existing actual key file. The key value at the address and position specified by the key address (KA) and key position (KP) fields in the FIT must be unique or zero; otherwise, the request is ignored. When a key value is specified, it must not extend the file by more than one block. A key value of zero causes AAM to determine the location for the record; the key value is returned to the user at location KA. If a block cannot accommodate a record with a user-specified key, the record is placed elsewhere by AAM; the value of the original key does not change for user program purposes.

The maximum acceptable key value depends on KL and RB as follows:

- The key value must not result in increasing the file length by more than one block. If KL=n, a maximum key value of 2 to the nth power -1 is automatically effective.

- If KL is greater than 4, hardware considerations prevent the file from approaching the upper limit of key value.

- If KL is 1, 2, 3, or 4, the maximum possible key values are 63, 4095, 262143, and 16777215, respectively. Dividing this maximum by the blocking factor (RB) yields a quotient and a remainder; the quotient is the maximum number of data blocks possible, and the remainder is the number of key values at the high end of the range that are not acceptable.

- If, for example, KL is 2 and RB is 4, the maximum possible key value is 4095. Dividing this by 4 yields a quotient of 1023 and a remainder of 3. This means that the highest 3 values are unacceptable, or in other words the range of usable key values is 1-4092.

- Considering an extreme case in which KL is 1 and RB is 80, dividing 63 by 80 yields a quotient of 0 and a remainder of 63, which implies that no keys are acceptable; in this case, RB would be automatically adjusted to 63 to allow keys 1-63 to be used. Considering another extreme case in which KL is 1 and RB is 32, only key values 1-32 could be used.

An index for actual key files is not maintained by AAM. For subsequent random reading by primary key value, the user is responsible for preserving primary keys of records written on the file. A multiple-index file can be created to maintain an index for actual key files.

File Updating

After a file has been created, records in the file can be deleted or replaced. The DELETE and REPLACE macros are used to update an actual key file.

A record can be eliminated from an existing file with the DELETE macro. The record indicated by the key value at location KA is logically removed from the file. If the requested record cannot be found, the request is ignored and a trivial error results.

The REPLACE macro is used to replace an existing record with a new record. The existing record is specified by the key value at location KA and position KP. The new record is in the working storage area. The new record need not be the same size as the record being replaced.

File Positioning

When the OPENM macro is executed, the file is positioned at the first record in the file. File positioning remains unchanged until one of the following macros is executed: GET, GETN, GETNR, REWINDM, or SKIP. The GET, GETN, and GETNR macros, which are used to read records, position the file at the record retrieved.

The SKIP macro positions the file forward or backward the specified number of records to the beginning of another record. Only small skips should be made because each intervening record is read and counted. The SKIP macro does not return a record to the working storage area.

Skipping stops if beginning-of-information or end-of-information is reached. An informative message is issued if skipping or sequential reading is attempted past the file boundary, but no error exit is taken. Any end-of-data exit is executed only if end-of-information is encountered. A skip count of zero is interpreted as a no-op.

The use of the REWINDM macro is more efficient than extensive backward skipping of records. This macro positions the file to beginning-of-information, which is the start of the user data record with the lowest key.

Overlap Processing

In response to a user program request for a record, AAM determines the block needed and transfers it from mass storage to the buffer area. The specified record is then transferred to the working storage area. The execution time to do this can be overlapped with program processing by using the SEEK and GETNR macros.

The SEEK macro initiates the transfer of a data block from mass storage to the buffer; the macro then returns control to the user program, which can perform other operations while the read is in progress. To find out when the read is complete, the user program can check the completion bit in the FET to which the busy FET address (BZF) field of the FIT points. Bit 0 of the word at address BZF will change from 0 to 1 at completion.

Upon completion of a SEEK, the user should issue a GET with the same key, either to get the wanted record or to learn that the file contains no record with that key.

Similarly, SEEK could be used as a preliminary to a REPLACE or DELETE operation to overlap some of the I/O time with other work.

The GETNR macro can be used to read records sequentially while overlapping the block reads with central processor activity or with I/O on a different file. GETNR, like SEEK, returns control to the user program as soon as a block read is initiated. The procedure for using GETNR is as follows:

1. Issue a GETNR.

2. If FP is 0, wait until the completion bit (the rightmost bit of the word to which BZF points) is 1, and then return to step 1.

3. If FP is not 0, the action is complete. The record and key value have been returned (or not) and FP has been set to 20_8 or 100_8, as if the preceding operation had been a GETN rather than a GETNR.

DIRECT ACCESS FILES

The direct access file organization is well suited for applications that require rapid access by key value. Direct access files can be accessed either randomly or sequentially by primary or alternate key; however, records accessed sequentially by primary key are not logically ordered.

FILE CREATION RUN

A separate creation run is necessary for a direct access file. This can be done through the FORM utility, the CREATE utility, or a source program.

Mass storage for a direct access file is preallocated. Before the file is opened on a creation run, the user must specify the size and number of home blocks to be preallocated. The number of home blocks is specified by setting the number of home blocks (HMB) field in the FIT. The key analysis utility, which is described in section 7, can be used to test various values of HMB.

The user has the option of specifying the home block size directly or accepting a system default. The maximum block length (MBL) field is set by the user to specify home block size. If the MBL field is not set by the user, AAM calculates the value for the MBL field from the values in the following FIT fields:

MNR Minimum record length

MRL Maximum record length

RB Records per block; default is 2

A number of fields must be set by the FILE control statement, the FILE macro, or the STORE macro before the file is opened for a creation run. If these fields are specified for an existing file, the new values are ignored without comment. A fatal error occurs if the following fields are not set on a creation run:

HMB Number of home blocks

KL Key length

LFN Logical file name

MNR Minimum record length

MRL Maximum record length

If the primary key is not embedded in the record, the embedded key (EMK) field must be set to NO.

The EMK field set to YES (default) indicates the primary key is embedded in the record. In this case, the primary key is assumed to begin in the first character position of the record. If the primary key is in another position, the position must be specified before the file is opened. The following FIT fields, which cannot be changed after the file is opened for a creation run, describe the key position within the record:

RKP Relative key position

RKW Relative key word

Default values are used without comment if certain FIT fields are not set before the file is opened for a creation run. These fields are effective only until the file is opened again; attempted changes are ignored without comment until another OPENM macro is executed. At that time, the values in the FIT are used to accomplish the open. These fields are as follows:

BCK Block checksum; default is no checksums

BFS Buffer size; default is buffer size calculated by AAM

FWB First word address of the buffer; default is buffer address provided by AAM

The old/new file (ON) field must be set to NEW and the processing direction (PD) field to OUTPUT for a file creation run. These fields can be set by using FIT manipulation statements or by setting the processing direction (pd) parameter to NEW in the OPENM macro. Setting the pd parameter to NEW sets the ON field and the PD field to OUTPUT.

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time (whether new or old) and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.

2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-new time, the user has the following choices for CPA and DCA:

1. Set CPA=0 (the default value), which excludes all compression and decompression for the life of the file; DCA will never be significant thereafter. CPA must never be set nonzero during the life of the file; this could cause an abort, or at the very least a nonfatal error, whenever a PUT or REPLACE is attempted.

2. Set CPA to the actual address of a user-supplied compression routine, and DCA to that of the corresponding user-supplied decompression routine. The fields should not be changed until the file is closed. It is the user's responsibility to ensure that whenever the file is opened in the future as an old file the same routines are available and are pointed to by CPA and DCA. The only exception is that CPA can at any time be set to 0 to stop compression until the next CLOSEM.

3. Set CPA to an integer in the range 1 through 63 to select system-supplied compression and decompression routines. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. When the file is opened, AAM loads the routines and stores their addresses into CPA and DCA. The user must not alter CPA or DCA until the file is closed, except to stop compression by setting CPA=0.

The user has the option of specifying a user hashing routine. The following FIT field gives the location of a routine that cannot be changed for the life of the file:

HRL Hashing routine location; default is the system hashing routine

Two fields in the FIT have no default value and must be set before being used by a file processing macro. If the following fields are not set before required, a fatal error occurs:

KA Key address

WSA Working storage area

Only the following macros can be used on a file creation run:

OPENM

REWINDM

PUT

CLOSEM

Overflow

Overflow blocks are created to handle any overflow records occurring.

User Hashing Routine

At file creation time, the user has the option of selecting a user hashing routine instead of the system hashing routine. This option is controlled by the hashing routine location (HRL) field in the FIT.

If the symbolic entry point name of the user hashing routine is MYHASH, the user should code HRL==XMYHASH in the FILE macro. Parameters needed by the user hashing routine are passed as follows:

```
SA1     ARRAY

RJ      =XMYHASH
```

The array contains the addresses of the following:

ARRAY	Key length (KL)
ARRAY+1	Key value
ARRAY+2	Number of home blocks (HMB)
ARRAY+3	Hashing result

The key is presented to the user hashing routine left-justified and zero-filled to (KL+9)/10 words.

AAM then converts the value to a relative physical record unit (PRU) number. The hashing routine could be coded as shown in figure 4-1. Upon return to AAM from any hashing routine, the remainder of the hashed key divided by the value of the HMB field is used as the ordinal of a home data block.

```
MYHASH  DATA    0
        Computation
        BX6     Xi          STORE HASH RESULT
        SA2     A1+3        GET ADDRESS FOR
                            HASHED RESULT
        SA6     X2          STORE HASH RESULT
        EQ      MYHASH      RETURN TO AAM/DA
```

Figure 4-1. User Hashing Routine Example

Supplied Hashing Routine

When the HRL field is not set to the address of a user hashing routine, the system-supplied hashing routine is used. The system hashing routine folds the word-aligned key into one word using the integer add instruction. If the folded key is an 18-bit integer or an 18-bit packed integer, no further hashing is done; otherwise, the folded key is hashed using the shift and divide instructions to produce a 48-bit result. This hashed key is the ordinal of a home data block on mass storage.

A prime number of home blocks is recommended when the supplied hashing routine is used. This generally produces a more uniform distribution of records than a nonprime number.

Direct Access File Records

All record types are allowed for direct access files. When creating the file through a source language, W type records are the default. When using the FORM utility, W type records are the default.

EXISTING FILE PROCESSING

Direct access files must reside on mass storage for processing. After file creation, however, the file can be dumped to tape with a COPYBF statement or a permanent file dump routine. The file can be returned later to mass storage for processing.

Open Processing

Before an existing file can be opened, the user must call for construction of the FIT by specifying the logical file name and the file organization. When the file is opened, values from the FSTT are returned to the following FIT fields:

HMB	Number of home blocks
KL	Key length
MBL	Maximum block length
MNR	Minimum record length
MRL	Maximum record length
RKP	Relative key position; character position within RKW in which the key begins
RKW	Relative key word; word in which the key begins

If a user hashing routine was specified at file creation time, the following field must be set before the file is opened:

HRL	Hashing routine location

If the following fields are not set before the file is opened, the default value is assumed without comment:

BCK	Block checksum; default is no checksums
BFS	Buffer size; default is buffer size calculated by AAM
FWB	First word address of the buffer; default is buffer address provided by AAM

Two FIT fields, CPA and DCA, specify compression and decompression routines that are to be used with the file. Generally they should be set, or allowed to default to 0, at open time and not be altered until the next CLOSEM. Any change would risk causing an abort, except for two possibilities:

1. CPA could be set to 0 at any time between OPENM and CLOSEM to prevent any further compression of records.
2. If no compression was selected when the file was created, DCA could be altered at any time without causing any effect whatever.

At open-old time, the user should set these fields depending on what was done when the file was created:

1. If CPA was 0 when the file was created, compression was excluded for the life of the file. Whenever the file is opened, AAM automatically sets CPA and DCA to 0.

2. If CPA was an integer in the range 1 through 63 when the file was created, system-supplied compression and decompression routines were specified for the file. CPA=1 specifies the release routines; CPA=2 through 63 specifies installation-defined routines. Whenever the file is opened, these routines are automatically loaded, and CPA and DCA are automatically set to point to them.

3. If CPA was any value greater than 63, it was assumed to be the address of a user-supplied compression routine. The same routine and its corresponding decompression routine must be supplied by the user at every subsequent open, and CPA and DCA must be set by the user to point to them. AAM calls the compression subroutine and checks to ensure it is the same routine that was used during creation; if it is not, a fatal error is declared.

Two FIT fields that have no default value must be set before being used by a file processing macro; otherwise, a fatal error occurs. These fields are as follows:

KA	Key address
WSA	Working storage area

A number of FIT fields can be set before the file is opened but need not be set until required by file processing macros. These fields are as follows:

DFC	Dayfile control
DX	End-of-data exit
EFC	Error file control
ERL	Trivial error limit
EX	Error exit
FWI	Forced write indicator
KP	Beginning key position

The first time an existing file is opened after its creation run, the old/new file (ON) field in the FIT must be changed from NEW to OLD. This can be done through a FIT manipulation macro or by specifying any option except NEW in the processing direction (pd) parameter of the OPENM macro.

Read Processing

A direct access file can be read randomly by primary or alternate key using the GET macro. It can also be read sequentially by the GETN and GETNR macros. The file must be open for input or input/output.

For the GET, GETN, and GETNR macros, the number of characters read is that of the actual length of the record as it is carried in the file. The value of the record length (RL) field in the FIT is ignored. At the completion of a read operation, the RL field is set to the length of the record returned.

With the GET macro, records are located using the key value at the address indicated by the key address (KA) and key position (KP) fields in the FIT. If the requested record cannot be found, a trivial error occurs.

The first GETN macro executed after an OPENM or REWINDM macro retrieves the first record in the file. A subsequent GETN macro retrieves the next sequential record. If the key address (ka) parameter is specified in the macro, the primary key of the record retrieved is returned to the specified address. All home blocks are processed first and then any overflow blocks. Intervening GET and DELETE macros are allowed and do not alter the sequential position of the file. REPLACE macros are allowed if the new record is the same length as the old one. If a PUT macro or a REPLACE macro with a different size record is followed by a GETN or GETNR macro, a trivial error results and the file position is lost. Any other function has no effect on sequential reading or file positioning.

Write Processing

Records are written to a direct access file with the PUT macro. The user must set the key of the record to be unique. If the EMK field was set to YES at creation time, the key must be in the same position within the record as when the file was established.

File Updating

The DELETE macro logically removes an existing record from a file. The record associated with the specified key is flagged as deleted and the space is available to store another record in the file. If the requested record is not found, a trivial error results and the request is ignored.

The REPLACE macro can be used to replace a record in the direct access file with a record in the working storage area. The record to be replaced is located by hashing the key specified by the relative key word (RKW), relative key position (RKP), and key length (KL) fields in the FIT. If the file was created with EMK set to NO, then the record to be replaced is located by hashing the key specified by the key address (KA), key position (KP), and key length (KL) fields in the FIT. Replacement records need not be the same size as the records replaced unless the file is being processed sequentially. A REPLACE macro that changes the record size invalidates further sequential processing.

File Positioning

The REWINDM macro is used to position the file to beginning-of-information. The file must be open when the macro is executed. The REWINDM macro resets the sequential position so that the next GETN macro returns the first record in the direct access file.

Overlap Processing

In response to a user program request for a record, AAM locates the desired home block by hashing the key and then transfers the home block to the buffer area. The specified record is then transferred to the working storage area. The execution time to do this can be overlapped with program processing by using the SEEK macro.

The SEEK macro initiates the transfer of a data block from mass storage to the buffer; the macro then returns control to the user program, which can perform other operations while the read is in progress. To find out when the read is complete, the user program can check the completion bit in the FET to which the busy FET address (BZF) field of the FIT points. Bit 0 of the word at address BZF will change from 0 to 1 at completion.

Upon completion of a SEEK, the user should issue a GET with the same key, either to get the wanted record or to learn that the file contains no record with that key.

Similarly, SEEK could be used as a preliminary to a REPLACE or DELETE operation to overlap some of the I/O time with other work.

The GETNR macro can be used to read records sequentially while overlapping the block reads with central processor activity or with I/O on a different file. GETNR, like SEEK, returns control to the user program as soon as a block read is initiated. The procedure for using GETNR is as follows:

1. Issue a GETNR.
2. If FP is 0, wait until the completion bit (the rightmost bit of the word to which BZF points) is 1, and then return to step 1.
3. If FP is not 0, the action is complete. The record and key value have been returned (or not) and FP has been set to -20_8 or 100_8, as if the preceding operation had been a GETN rather than a GETNR.

The macros described in this section are used for processing the AAM files established with the FILE macro and FILE control statement. All macros reside on COMPASS system text IOTEXT, which must be specified by the S=IOTEXT parameter on the COMPASS control statement at assembly time. The macros conform to COMPASS syntax; the location, operation, and variable fields are separated by one or more blanks. When using the COMPASS system texts, SYSTEXT and IOTEXT, it is necessary for you to reconstruct your program so that you have separate IDENTS for the SYSTEXT and IOTEXT macros. Once your programs is restructured, you can then use SYSTEXT to assemble the module containing the macros. Information shared by the modules should be placed in a common block.

When using the COMPASS system texts, SYSTEXT and IOTEXT, it is necessary for you to reconstruct your program so that you have separate IDENTS for the SYSTEXT and IOTEXT macros. Once your program is restructured, you can then use SYSTEXT to assemble the module containing the macros. Information shared by the modules should be placed in a common block.

In the macro parameter strings, the fit parameter is required; all others are optional and positional. When an optional parameter is omitted, the parameter position must be marked by a comma; however, trailing commas can be omitted. For example, the format of the OPENM macro is:

```
.OPENM      fit,pd,of
```

If the pd parameter is not specified in the OPENM macro, the format is:

```
OPENM       fit, ,of
```

The first parameter of every macro (fit) identifies the file information table for the referenced file. If the address specified by the fit parameter is invalid, the results are indeterminate. The fit parameter can specify any of the following:

lfn	Location field name of the first word of the FIT; one through seven alphabetic or numeric characters
Rn	Any A, B, or X register containing the FIT address
exp	Any COMPASS expression giving the FIT address

Only parameters applicable to the file organization specified in the FIT should be set. Supplying parameters applicable to other file organizations could cause erroneous results.

MACRO EXECUTION

The current contents of the FIT are used for macro execution. If a parameter is omitted, the default value is valid only if the respective FIT field has not been previously set to a different value. A field in the FIT can be set by any of the following:

- FILE macro parameter
- FILE control statement parameter, which can override defaults during open processing
- SETFIT macro, which can call for FILE control statement processing without full open processing
- STORE macro, which can set individual fields before or after open processing.
- Default, which can be set during open processing
- Macro parameter that is moved to the FIT before file processing occurs (a zero value in a parameter list moves a zero to the FIT field; a null value does not affect the FIT field)

Registers are not saved or restored. It should be assumed that all registers are destroyed during macro execution.

Static loading for AAM uses the STLD.RM macro and new parameters in the FILE control statement. Refer to appendix E for details on static loading.

The user macros, with the exception of the FETCH, FILE, STLD.RM, and STORE macros, generate code as follows:

- When syntax error checking is completed, all nonnull parameters following the FIT address are placed in registers.
- Register B6 is set to the end of the macro expansion as the return address.
- A jump to the proper AAM entry point is generated in the top of a word; bits indicating which parameters were specified with the macro are set in the bottom of the word.
- The FIT address is placed in register A0; if it is already in A0, no code is generated.
- Register B1 is set to 1; if B1=1 pseudo-op is in effect, no code is generated.

PROCESSING MACROS

Several macros are available for processing AAM files. These macros are described in this section. The FETCH, FILE, SETFIT, and STORE macros are described in section 3, File Information Table.

CLOSEM MACRO

The CLOSEM macro terminates file processing and positions the file as specified. It should be the last macro issued for a file. The format of the CLOSEM macro is shown in figure 5-1.

When the CLOSEM macro is executed for a file opened for output, any information in the file buffer is written on the file as part of file termination. If end-of-information has moved and the file is a NOS/BE permanent file, an

EXTEND is issued. If unload (U) or return (RET) is specified in the CLOSEM macro, close processing is as follows:

- If it is a permanent file, it is detached from the job and returned to the permanent file manager.
- If it is not a permanent file, mass storage space assigned to the file is released.

CLOSEM fit,cf

fit Address of the FIT.

cf File positioning after close processing:

R	Rewind (default)
N	No rewind
U	Unload; release buffer space and remove name from the active file list.
RET	Return; release buffer space and remove name from the active file list.
DET	Detach; release buffer space and remove name from the active file list.

Only the fit parameter can be specified as a register.

Figure 5-1. CLOSEM Macro Format

Close processing for a file varies according to the value specified for the cf parameter of the CLOSEM macro.

- Rewind (R)

  The file is rewound.

- No rewind (N)

  The file is not rewound.

- Unload (U)

  The file is rewound. The open/close flag (OC) field in the FIT is cleared. If the file is a permanent file, it is detached from the job and returned to the permanent file manager. Any scratch mass storage space assigned to the file is released.

- Return (RET)

  The processing is the same as for unload.

- Detach (DET)

  The file is not rewound. The open/close flag (OC) field in the FIT is cleared. The DET option should be used when there is a possibility the FIT could be destroyed. The DET option removes the FIT from the list of effective FITs.

A CLOSEM request for a file that has never been opened, or a file that has been closed but not unloaded or reopened, has the following effects:

- The FIT error status redundant close is set.
- File positioning is the same as for an open file.
- Control is returned to the error exit.

DELETE MACRO

The DELETE macro removes a record from the file. If the requested record is not found, a trivial error results and the request is ignored. The format of the DELETE macro is shown in figure 5-2.

DELETE fit,ex,ka,kp

fit Address of the FIT.

ex Address of the error routine.

ka Address of the primary key for the record to be deleted.

kp Beginning character position of the primary key.

Parameters can be specified as registers.

Figure 5-2. DELETE Macro Format

Applicable parameters by type of file organization for the DELETE macro are as follows:

Indexed sequential	fit,ex,ka,kp
Direct access	fit,ex,ka,kp
Actual key	fit,ex,ka,kp

When the DELETE macro is executed, the specified record is either flagged as deleted or physically removed from the file. If the requested record is not found, a trivial error results and the request is ignored.

FLUSHM MACRO

The FLUSHM macro processes one or more file buffers as if a CLOSEM macro had been issued; the files, however, remain open. Blocks with pending writes and the updated FSTT are written on the file. If end-of-information has moved and the file is a NOS/BE permanent file, an EXTEND is issued. The format of the FLUSHM macro is shown in figure 5-3.

FLUSHM fitlist

fitlist Address of the list of FIT address entries.

Figure 5-3. FLUSHM Macro Format

The list referenced by the fitlist parameter contains a one-word entry for each file to be flushed. A word of binary zeros terminates the list. The one-word entry is formatted as shown in figure 5-4.

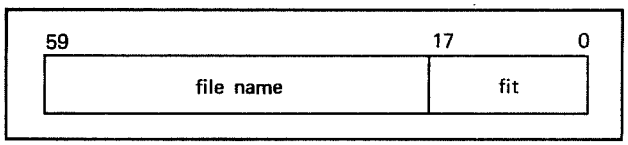


Figure 5-4. Entry in List Referenced by FLUSHM Macro

The file name, which is specified in display code, is used as a consistency check. The address of the FIT is specified in the lower bits of the word.

GET MACRO

The GET macro retrieves data from a file and delivers it to the working storage area. The file must be open for input or for input/output. The GET macro retrieves a record randomly by key value. The GETN and GETNR macros retrieve records sequentially by file position. The formats of the macros are shown in figure 5-5.

GET	fit,wsa,0,ex,ka,kp,mkl
GETN	fit,wsa,ex,ka
GETNR	fit,wsa,ex,ka
fit	Address of the FIT.
wsa	Address of the working storage area to which the user record is returned.
ex	Address of the error routine.
ka	Address of the key for the record to be read.
kp	Beginning character position of the key.
mkl	Major key length in characters; can be used only for a symbolic key in an indexed sequential file. Reset to zero after each GET.

Parameters can be specified as registers; if parameters are not specified, values in appropriate FIT fields are used.

Figure 5-5. GET, GETN, and GETNR Macro Formats

Applicable parameters by type of file organization for the GET macro are as follows:

Indexed sequential	fit,wsa,0,ex,ka,kp,mkl
Direct access	fit,wsa,0,ex,ka,kp
Actual key	fit,wsa,0,ex,ka,kp

The GET macro transfers a record from a file to the specified working storage area. The location referenced by the ka parameter contains the key value for the record to be read. If no record in the file has a matching key value, a nonfatal error occurs. The record length (RL) field in the FIT is updated to indicate the number of characters in the record retrieved from the file. If the operation is not successful, the RL field is not defined.

If the record is longer than specified by the maximum record length (MRL) field in the FIT, an excess data error occurs. Control is passed to the error exit after transferring to the working storage area the number of characters specified by the MRL field. A record greater than the maximum record length is prevented from overwriting a portion of the calling program or other preserved information. Control is transferred to the user end-of-data exit (DX field in the FIT) by a GET request that detects end-of-information.

The GETN macro is used to read records sequentially. The next record in sequence by position on the file is retrieved and transferred to the specified working storage area. If the ka parameter is an address, the primary key value is returned to the location pointed to by ka; if ka is 0, the primary key value is not returned. The RL field is updated to indicate the number of characters in the record retrieved from the file. If the operation is not successful, the RL field is not defined.

Applicable parameters by type of file organization for the GETN macro are as follows:

Indexed sequential	fit,wsa,ex,ka
Direct access	fit,wsa,ex,ka
Actual key	fit,wsa,ex,ka

The GETNR macro causes the next sequential record to be transferred to the working storage area the same as the GETN macro. The difference is that the GETNR macro returns control to the user if the request initiates block transfer to the buffer. The user can continue issuing the GETNR macro until transfer is complete. The file position (FP) field in the FIT is set to 20_8 (EOR) when transfer of the record is complete. While intermediate reads are being performed (index blocks or MIP index file blocks), the FP field is set to 0.

Unnecessary GETNR requests can be avoided by monitoring the status of the input/output processing. The busy FET address (BZF) field in the FIT contains the address of the input/output status word. When the low order bit of the status word is set to 1, input/output processing is complete and a GETNR macro will start a new block read or return a record to the working storage area. If the low order bit is set to 0, a GETNR macro immediately returns control to the user.

OPENM MACRO

Before a file can be read or written, the file must be made available by the OPENM macro. Macros that affect the FIT (FILE, STORE, FETCH, and SETFIT) can be executed before the file is opened. Any file manipulation macro, however, is valid only after the file has been opened. Error procedures are initiated if attempts are made to access an unopened file. The format of the OPENM macro is shown in figure 5-6.

OPENM	fit,pd,of		
fit	Address of the FIT.		
pd	Type of processing:		
		INPUT	File is opened for read only (default).
		OUTPUT	File is opened for write only.
		I-O	File is opened for read and write.
		NEW	A new file is being created; sets the PD field to OUTPUT and the ON field to NEW.
of	Open flag; file positioning at open time:		
		R	File is rewound before any other open procedures are performed (default).
		E	File is positioned immediately before the end-of-information.

Only the fit parameter can be specified as a register.

Figure 5-6. OPENM Macro Format

Applicable parameters by type of file organization for the OPENM macro are as follows:

Indexed sequential	fit,pd,of
Direct access	fit,pd
Actual key	fit,pd

The OPENM macro prepares a file for processing by creating and linking all required system tables for a file and by translating user-supplied parameters into appropriate values in the relevant tables. When the OPENM macro is executed, the following events occur:

- FILE control statement processing occurs unless it has been suppressed by previous execution of the SETFIT macro. The PDF field in the FIT is set by the SETFIT macro to inhibit reprocessing of the FILE control statement. The PDF field is cleared by the OPENM macro.
- The FIT is checked for logical consistency; depending on the file organization, additional checks are made for required fields and defaults are supplied where needed.
- Buffer parameters are processed.
- If no error has been detected, the open/close flag (OC) field in the FIT is set to open and control transfers to the user.

Complete open processing occurs when the first OPENM macro in a job step is issued. If a file is closed and then reopened, FIT verification and FILE control statement processing are not repeated if the close flag (CF) field in the FIT is set to R or N.

Any error detected during open processing sets the error status (ES) field in the FIT. If a user error routine is specified by the error exit (EX) field in the FIT, control is transferred to the routine. If the user routine corrects the condition that caused the error and executes another OPENM macro, processing of the file can continue; otherwise, the open/close (OC) field in the FIT indicates the file is not open (set to 0) and further file access is prohibited.

Conditions investigated during FIT consistency checks are listed in table 5-1. Buffer fields are also investigated. The settings of the first word address of the buffer (FWB) field and buffer size (BFS) field determine the method of buffer allocation. If the FWB field is zero, the Common Memory Manager (CMM) must be present or an error occurs. For an indexed sequential file, the buffer pool limit is increased by the default amount if the BFS field is also zero; otherwise, it is increased by the amount specified by the BFS field.

When the FWB field is not zero, an error occurs if the BFS field is zero. If the BFS field is also nonzero, the specified buffer space is partitioned into table areas for AAM, blocks for the data file, and (if needed) blocks for the MIP index file. A minimum of two blocks must be allocated for each file or CMM must be present; otherwise, an error occurs. The buffer pool amount must be increased to accommodate two blocks per file.

TABLE 5-1. FIT CONSISTENCY CHECKS

Condition	Action
RT=D, LL=0	Error
RT=T, and CL, HL, or TL=0	Error
RT=Z, FL=0	Error
RT=F, FL=0	Error
RT=T, HL not greater than CL+CP	Error
MRL, MBL=0, BT=K, E	Error

Data compression can be established for an AAM file at the time it is created. Once data compression is selected, it must be specified for the life of the file. The compression routine address (CPA) and decompression routine address (DCA) fields in the FIT point to the routines to be used for data compression. These fields can originally be specified when the file is created or at any subsequent time the file is opened. Whenever the file is opened after that time, the routine addresses must be supplied in the CPA and DCA fields. Refer to appendix H, Data Compression, for more detailed information.

The timing of the setting of the parameters for the processing of each file organization in relation to the OPENM macro is important. These parameters differ for each file organization. The requirements for specific parameters are discussed under open processing for each file organization; refer to section 4, File Processing. The following shows the possible relationships between the OPENM macro and the FIT parameters:

- For file creation, certain parameters must be set before executing the OPENM macro; otherwise, a fatal error occurs. If these parameters are specified for an existing file, the new values are ignored without comment.
- Certain parameters must be selected before the file is created if the option is to be used during the life of the file.
- Certain parameters are optional at file creation. If these parameters are not specified, default values are used. Values specified after file creation are ignored.
- Certain parameters must be set prior to open time; otherwise, default values are assumed without comment. These parameters are effective only until another OPENM macro is executed.
- Certain parameters need not be set until they are required by file processing commands. Once set, these parameters remain in effect until changed.
- Certain parameters have no default and must be set prior to use by a file processing command; otherwise, a fatal error occurs.

PUT MACRO

The PUT macro transfers data from the working storage area to a file. The file must be open for output or input/output. The format of the PUT macro is shown in figure 5-7.

PUT fit,wsa,rl,ex,ka,kp	
fit	Address of the FIT.
wsa	Address of the working storage area from which the user record is transferred.
rl	Number of characters to be written.
ex	Address of the error routine.
ka	Address of the primary key for the record to be written.
kp	Beginning character position of the primary key.
Parameters can be specified as registers; if parameters are not specified, values in appropriate FIT fields are used.	

Figure 5-7. PUT Macro Format

Applicable parameters by type of file organization for the PUT macro are as follows:

Indexed sequential	fit,wsa,rl,ex,ka,kp
Direct access	fit,wsa,rl,ex,ka,kp
Actual key	fit,wsa,rl,ex,ka,kp

Any errors detected during PUT macro execution cause transfer to the error routine if one is specified. If the error is excess or insufficient data, no data has been transferred; for other errors, the data is unreliable.

The length of a record being written is determined by the record length (RL) field in the FIT. For U, S, and W type records, the RL field can be set by the rl parameter in the PUT macro. For F, Z, R, T, and D type records, AAM uses certain FIT fields and the content of the record in the working storage area to determine record length for the RL field; the value of the RL field is determined as follows:

- F type records

  Record length is taken from the FL field in the FIT.

- Z type records

  Record length is taken from the RL field in the FIT or from the FL field if the RL field is set to zero. The end of the record is determined by searching backward from the character position specified by the value of the RL or FL field and removing full words of blanks.

- R type records

  Record length is determined by scanning the record in the working storage area for the terminating record mark character, which is specified by the record mark (RMK) field in the FIT. An error occurs if the record mark is not found within the maximum record length.

- T type records

  Decimal count is extracted from the record and used to calculate the record length. Length and location of the count field in the record (CL and CP fields), length of the header (HL field), and length of the trailers (TL field) are obtained from the FIT.

- D type records

  Decimal character record length is extracted from the record. Length and location of the character count field in the record (LL and LP fields) are obtained from the FIT.

In all preceding cases, the length of the record transferred is stored in the RL field in the FIT at the end of the PUT macro operation.

REPLACE MACRO

An existing record in a file is replaced by a record from the working storage area when the REPLACE macro is executed. The new record can be smaller or larger than the original record; however, record length cannot exceed the size specified by the maximum record length (MRL) field in the FIT. The format of the REPLACE macro is shown in figure 5-8.

REPLACE fit,wsa,rl,ex,ka,kp	
fit	Address of the FIT.
wsa	Address of the working storage area with the new record.
rl	Length (in characters) of the new record.
ex	Address of the error routine.
ka	Address of the primary key for the record to be replaced.
kp	Beginning character position of the primary key.
Parameters can be specified as registers.	

Figure 5-8. REPLACE Macro Format

Applicable parameters by type of file organization for the REPLACE macro are as follows:

Indexed sequential	fit,wsa,rl,ex,ka,kp
Direct access	fit,wsa,rl,ex,ka,kp
Actual key	fit,wsa,rl,ex,ka,kp

Replacement records need not be the same size as the record being replaced except for a direct access file being processed sequentially. A larger replacement record in a direct access file can cause overflow of records, which leaves the sequential position undefined. If the requested record is not found, a trivial error results and the request is ignored.

REWINDM MACRO

The REWINDM macro positions a file to beginning-of-information, which is the beginning of the first data record in the file. The file must be open when the macro is issued. A GETN macro issued immediately after the REWINDM macro returns the first record. The format of the REWINDM macro is shown in figure 5-9.

REWINDM fit

fit	Address of the FIT or register containing the address.

Figure 5-9. REWINDM Macro Format

SEEK MACRO

Program execution time can be shortened through the use of the SEEK macro, which allows overlapping of central memory processing and input/output activity. The SEEK macro initiates block transfer to the buffer; it does not return a record to the user. The user can then continue processing while the transfer occurs. The format of the SEEK macro is shown in figure 5-10.

SEEK fit,ex,ka,kp,mkl

fit	Address of the FIT.
ex	Address of the error routine.
ka	Address of the key for the desired record.
kp	Beginning character position of the key.
mkl	Major key length in characters.

Parameters can be specified as registers. If the ex, ka, kp, and mkl parameters are not specified, values in appropriate FIT fields are used.

Figure 5-10. SEEK Macro Format

Applicable parameters by type of file organization for the SEEK macro are as follows:

Indexed sequential	fit,ex,ka,kp,mkl
Direct access	fit,ex,ka,kp
Actual key	fit,ex,ka,kp

When the SEEK macro is executed, control returns to the user program once a read is initiated. The FP field is set to zero if the transfer of an index block has been initiated; it is set to 20_8 (EOR) if a data or home block is being transferred.

The user can monitor the FP field or the busy FET address (BZF) field. By monitoring the BZF field, the user can avoid issuing SEEK macros with the same key, which would return immediately because the file was busy. The BZF field in the FIT is set by AAM and points to an input/output status word. When the low order bit of the status word is set, the current SEEK macro input/output is complete and another operation can be profitably issued for the file.

Normally, the SEEK macro is followed by a macro such as GET or DELETE accessing the record referenced by the SEEK macro. An operation on some other record not already in the buffer can negate the action of the SEEK macro by writing over the data transferred by it. The record is not moved into the working storage area until a GET macro is executed. If a call is made before the seek operation is complete, processing continues reading blocks from the point where the SEEK calls were discontinued.

SKIP MACRO

The SKIP macro repositions an indexed sequential or actual key file in a forward or backward direction a specified number of logical records. It does not return a record to the working storage area. Only small skips are recommended because each record must be read and counted for proper positioning. The format of the SKIP macro is shown in figure 5-11.

SKIPdL fit,count

d	Direction of skip:
	F Forward
	B Backward
fit	Address of the FIT.
count	Number of logical records to be skipped. A null parameter results in a zero count.

The fit and count parameters can be specified as registers.

Figure 5-11. SKIP Macro Format

When the SKIP macro is executed, user parameters are checked, records in the file are read, the file is positioned according to the number of records to be skipped, and control returns to the user. A negative skip count is not allowed; the request is ignored and an error is issued. If the skip operation encounters end-of-information or beginning-of-information before the skip count is exhausted, control is transferred to the end-of-data routine with the appropriate file position set.

START MACRO

The START macro positions an indexed sequential file or an alternate key index file to a record that meets a specific condition, but does not transfer the record to the

working storage area. The START macro also updates the area pointed to by the KA field. The file is positioned in the same manner as for a GET macro except that GET uses REL=EQ even if the REL field has been set to a different value. The format of the START macro is shown in figure 5-12.

The file is positioned according to the key relation (REL) field in the FIT and the current value at the key address (KA) location. The REL field specifies the desired relation between the value at location KA and the key of the record at which the file is to be positioned. Relations that can be specified are EQ (equal to), GT (greater than), and GE (greater than or equal to). The file is positioned at the beginning of the record that satisfies the relation. If the mkl parameter is specified, the file is positioned relative to the major key specified for an indexed sequential symbolic key.

Apart from index-only processing, START is the only operation for which REL is significant.

START	fit,ex,ka,kp,mkl
fit	Address of the FIT.
ex	Address of the error routine.
ka	Address of the key for positioning the file.
kp	Beginning character position of the key.
mkl	Major key length in characters; can be used only for a symbolic key in an indexed sequential file. Reset to zero after each START.

Parameters can be specified as registers. If the ex, ka, kp, and mkl parameters are not specified, values in appropriate FIT fields are used.

Figure 5-12. START Macro Format

MULTIPLE-INDEX FILES 6

All AAM files have a primary key associated with each record to provide random access to the file. In addition, alternate keys can be defined for records in an AAM file. Alternate keys provide the means to access records by more than one field in a record.

Primary key values must be unique within the file. Alternate keys, which can overlap each other and the primary key, need not have values unique to the record or to the file. Alternate keys must be contained within the minimum record size.

The original data file structure is not affected by alternate key processing. The Multiple-Index Processor (MIP) creates an index file on the creation run for a multiple-index file. On subsequent runs, the index file is updated as necessary when the data file is updated. The index file must be made available to the updating program.

For existing AAM files, the MIPGEN utility is available to assist in creating the index file for alternate key processing and to add new fields or delete fields for an existing index file. Refer to section 7 for a description of the MIPGEN utility.

INDEX FILE

The index file is created and updated automatically by MIP. It is identified by the index file name (XN) field in the FIT. The index file, which is defined when the file is created, must be made available whenever the data file is updated or is accessed by alternate key. Alternate keys can be defined by the user on the creation run.

STORAGE STRUCTURE

The index file contains an index for each alternate key defined for the data file. Within an index, each alternate key value is associated with a keylist of the primary keys for records containing that value.

Ordering Keys

Each alternate key index is ordered by alternate key value. The ordering of the primary key list for a given index is controlled by the user through a parameter that can be specified when the alternate key is defined by the RMKDEF macro or directive. The ordering of the list is as follows:

- If the parameter is omitted or U is specified, each value of the alternate key must be unique. The primary key list for each alternate key value consists of only one primary key value.
- If F is specified for the parameter, the ordering of primary key values is first-in first-out. The primary keys are stored in the order in which their corresponding records are created.
- If I is specified for the parameter, the primary keys are stored in ascending sequence of primary key values. Numeric keys are in numeric order; symbolic keys are in collating sequence order.

Block Size

The size of the index file blocks is determined when the data file is created. The index block size (XBS) field in the data file FIT specifies the number of characters in a block. A value specified for the XBS field is rounded upward if necessary to the nearest multiple of 640 characters minus 20. The default index file block size is the data file block size.

ALTERNATE KEY SPECIFICATION

A record can be accessed by the primary key or by any alternate key defined for the file. Alternate keys can be defined when the data file is first created or after the file is created.

Alternate Key Definition

There are two ways to define alternate keys:

For a new file:

Using the RMKDEF macro in a COMPASS program; or using the RMKDEF call statement in a FORTRAN program; or using the ALTERNATE RECORD KEY CLAUSE in a COBOL program. Alternate keys can be defined by these methods only when the file is first created.

For an existing file:

Using the RMKDEF directive in the MIPGEN utility to define alternate keys and to create the index file. The MIPGEN utility is described in section 7.

The paragraphs that follow describe the RMKDEF macro only.

RMKDEF Macro

The RMKDEF macro is used to describe an alternate key field on a file creation run. The macro must be used once for each alternate key field in the record. The RMKDEF macros must be executed after the OPENM macro and before the first PUT macro. An RMKDEF macro that defines the primary key is ignored without comment. The format of the RMKDEF macro is shown in figure 6-1.

RMKDEF fit,kw,kp,kl,ki,kf,ks,kg,kc,nl,ie,ch

fit — Address of the FIT for the data file.

kw — Word of the record where the key starts, counting from zero; default is zero.

kp — Beginning character position of the key:

- 0 to 9 for symbolic key
- 0 for signed binary key

kl — Key length, in characters:

- 1 to 255 for symbolic key
- 10 for signed binary key

ki — 0 (reserved).

kf — Key type:

- 0 or S — Symbolic
- 1 or I — Integer
- 2 or U — Uncollated symbolic

ks — Substructure for each primary key list in the index:

- U — Unique (default)
- I — Indexed sequential
- F — First-in first-out

kg — For a repeating group, number of characters in the group where the key resides.

kc — For a repeating group, number of occurrences; zero for T type records, and a number for a repeating group embedded within the record.

nl — Null suppression:

- 0 — Null values are recorded (default)
- N — Null values are not recorded

A null value is all spaces (symbolic key) or all zeros (signed binary key).

ie — Include/exclude sparse control character:

- E — Exclude alternate key value if the record contains a sparse control character (default)
- I — Include alternate key value if the record contains a sparse control character

ch — Characters that qualify as sparse control characters; up to 36 letters and digits can be specified as a character string.

Figure 6-1. RMKDEF Macro Format

The kg and kc parameters refer to an alternate key that is a repeating group. For example, a repeating group is described in COBOL by an OCCURS n TIMES clause. If the same alternate key value occurs more than once in a data record, the primary key is entered in the index only once for that value; therefore, a primary key associated with an alternate key value indicates that the value occurs at least once in the record. Alternate key fields can overlap in a record; for example, first name, last name, and whole name can all be defined as alternate keys.

The nl, ie, and ch parameters are used to define sparse keys. These are alternate keys for which only certain values are of interest to the user. A sparse key is defined by specifying null suppression or sparse control characters.

Null suppression is specified by the nl parameter. The primary key for a record that has a null alternate key value is not included in the alternate key index. A null value is all spaces for a symbolic key or all zeros for an integer key.

The ie and ch parameters are used when indexing of alternate key values is to be controlled by a sparse control character. The one-character field containing the sparse control character must be in the fixed-length portion of the record. The ie parameter specifies whether to include or exclude the alternate key values for records that contain a sparse control character. The ch parameter specifies the sparse control characters applicable to the alternate key being defined.

The sparse control character field is identified by an RMKDEF macro that must appear before the macro defining the alternate key and its sparse control characters. This macro is specified in the following format:

```
RMKDEF fit,kw,kp,0
```

The kw and kp parameters specify the position of the sparse control character. The zero kl parameter indicates that the field is a sparse control character field.

APPLICABLE FIT FIELDS

Several FIT fields are applicable to multiple-index file processing. These fields and their respective uses are as follows:

FP — File position; when the index file is being accessed, 10_8 indicates the end of primary keys associated with a given alternate key value. 100_8 indicates the end of the alternate key list.

KL — Key length; number of characters in a primary or alternate key.

KNE — Key not equal; 1 indicates the key in process is not the same key specified by the KA field. KNE is set only after an operation for which a GE relation was specified.

MRL — Maximum record length; when the primary key lists are being retrieved, MRL indicates the length of the working storage area.

NDX — Index flag; 1 indicates an index only operation; 0 indicates a data record operation.

PKA — Primary key address; when accessing records by alternate key, the primary key for a record is returned to the specified address.

RC — Record count; number of records containing the value of the key at location KA.

REL — Key relation; relation of the key value at location KA to the key at which the file is positioned; can be EQ, GT, or GE.

RL Current record length.

RKP Relative key position; character position of a primary or alternate key within the word specified by the RKW field. For a nonembedded primary key, setting RKP=10 and KL equal to the number of characters in the primary key before performing a REWIND, GET, or START switches the key being referenced from an alternate key back to the primary key.

RKW Relative key word; word in which a primary or alternate key begins.

XBS Index file block size; number of characters in an index file block.

XN Index file name; logical file name of the index file.

ALTERNATE KEY PROCESSING

Defining alternate keys for a file allows the user to access records by fields other than the primary key. Two files are involved with alternate key processing. The data file contains records that have unique primary keys. The index file contains alternate key values and their associated primary keys. Both files must be made available to the program. Reading by alternate key can be random or sequential.

ALTERNATE KEY ACCESS

To access a data record by an alternate key, the alternate key position must first be established in the FIT. The relative key word (RKW), relative key position (RKP), and key length (KL) fields must be set for the desired alternate key. These three fields are set for the primary key by open processing; thereafter, the user is responsible for setting them when changing access from primary to alternate key or from one alternate key to another. The index flag (NDX) field in the FIT must be set to zero to access a data record.

The alternate key defined by the RMKDEF macro refers to a position within a record. The GET macro is used to retrieve a record with a specific value in the alternate key position. When the GET macro is executed, the RKW, RKP, and KL fields in the FIT define the alternate key position in the record. The ka, kp, and mkl macro parameters establish the alternate key location that contains the value for the record to be retrieved. The first primary key associated with the alternate key value determines the record returned to the working storage area. The format of the GET macro is:

```
GET     fit,wsa,0,ex,ka,kp,mkl
```

When the GET macro is executed successfully, a record is returned to the location specified by the wsa parameter, the index file is positioned, and the following FIT fields are set:

PKA Primary key address; address of location that contains the primary key of the record retrieved.

RC Record count; number of records that contain the alternate key value.

RL Record length; number of characters in the record returned to the working storage area.

If the operation is unsuccessful, the fields are not defined.

Once a GET macro has been executed to establish an index file position, the record for the next primary key in the index can be accessed by the GETN macro. When the index file is positioned past the last primary key in the index, no record is returned to the working storage area, the file position (FP) field is set to EOI, and any specified end-of-data exit is taken. An informative error message is written on the error file ZZZZZEG.

After execution of a GETN, the file position (FP) field in the FIT can have one of three values:

- 10_8 A good record was returned to the working storage area. Either the next record will have a different alternate key value, which means the end of a keylist (EOK) has been reached, or end-of-information (EOI) will be encountered.
- 20_8 A good record was returned to the working storage area and the next record has the same alternate key value as this record.
- 100_8 End-of-information (EOI) was encountered.

If a different alternate key value is encountered during execution of the GETN macro, that value is moved to program location KA.

The format of the GETN macro is:

```
GETN    fit,wsa,ex,ka
```

Execution of the GETN macro returns a record to the working storage area. The primary key for the record is moved to the program location indicated by the primary key address (PKA) field in the FIT.

FILE UPDATING

Updating a multiple-index file is basically the same as updating any other AAM file. The only difference is that the logical file name of the alternate key index file must be specified in the FILE control statement by the XN parameter. The index file is automatically updated when a data file update affects the index file.

The PUT and REPLACE macros are used to write and rewrite records. For MIP when the primary key is embedded, it is not necessary to set FIT fields for the primary key; that is, the RKW, RKP, KL, KA, and KP fields do not have to be set. The KA and KP fields must be set for nonembedded keys. The position of the primary key in the record is constant for the file and the address in the working storage area (WSA) field is the address of the record to be written or rewritten.

The DELETE macro is used to delete a record from the file. The RKW, RKP, and KL fields in the FIT do not have to be set; however, the key address (KA) and key position (KP) fields must be set for the primary key because the WSA field is not required for the DELETE macro.

The index file position and the RKW, RKP, and KL fields are not changed by execution of the PUT, REPLACE, or DELETE macro. A series of GETN macro requests can be interrupted by update requests without losing alternate key sequence.

INDEX FILE POSITIONING

The alternate key index file is positioned when a GET macro accesses a record by alternate key. The index file can also be positioned without returning a record. The START, SKIP, and REWINDM macros change the position of the index file.

START Macro

The START macro positions the index file to the first primary key for a given alternate key value. The value is at the location specified by the key address (KA) field in the FIT. The format of the START macro is:

```
START     fit,ex,ka,kp,mkl
```

The key relation (REL) field in the FIT determines the positioning of the index file in relation to the value at location KA. The REL field has three possible values:

- EQ — The index file is positioned at the alternate key value equal to the value at location KA. The default for the REL field is EQ. If an equal key value is not in the index, trivial error 506 results.
- GT — The index file is positioned at the first alternate key value greater than the value at location KA.
- GE — The index file is positioned at the first alternate key value greater than or equal to the value at location KA. If an equal key value is not in the index, the key not equal (KNE) field in the FIT is set to 1.

After the START macro is executed, the record count (RC) field in the FIT is set to the number of primary keys for the alternate key at which the index file is positioned.

Apart from index-only processing, START is the only macro for which REL is significant.

Other Positioning Macros

In addition to the START and GET macros, the index file position is changed by the SKIP and REWINDM macros. When a change is made from one alternate key index to another, the index position must be established by a REWINDM, GET, or START macro.

The SKIP macro is used to skip forward a number of primary keys from the current position. The format of the SKIP macro is:

```
SKIP      fit,n
```

The index file is positioned at the first primary key in the alternate key index by the REWINDM macro. The format of the REWINDM macro is:

```
REWINDM   fit
```

PROCESSING ONLY THE INDEX FILE

The alternate key index file can be accessed to retrieve information related to the alternate keys. Primary key lists or counts of primary keys for either a single alternate key value or a range of values can be retrieved. Obtaining this information from the index file has no effect on the data file.

In order to access the index file, the index flag (NDX) field in the data file FIT must be set to YES. If the OPENM macro is executed with NDX set to YES, only the index file is opened for processing. The index file must be an existing file at open time. If the NDX field is set to YES when the file is opened, it cannot be reset to NO until after the file has been closed.

MACRO PROCESSING

The index file is accessed through execution of various macros. Only those macros described in the following paragraphs can be used with the index file.

The OPENM macro and the CLOSEM macro open and close the index file. Execution of these macros does not affect the data file.

The REWINDM macro positions the index file at the beginning of the alternate key index from which information is to be retrieved. The alternate key is determined by the relative key word (RKW), relative key position (RKP), and key length (KL) fields in the data file FIT. The file is positioned at the first value for the designated alternate key.

The index file can be positioned at a specific value of an alternate key through execution of the START macro. The RKW, RKP, and KL fields in the FIT specify the alternate key for file positioning. The alternate key value at the location indicated by the key address (KA) field in the FIT and the condition designated by the key relation (REL) field determine the positioning at a specific value within the alternate key index. When the relational condition is EQ, the file is positioned at the alternate key value equal to the value at location KA; if an equal value cannot be found in the index, the file is positioned at the next higher value. For the GT relational condition, the file is positioned at the next higher value than the value at location KA. The GE relational condition causes the file to be positioned at a value equal to or greater than the value at location KA.

The GET macro is used to retrieve the primary keys for an alternate key value. The alternate key to be accessed is determined by the RKW, RKP, and KL fields in the FIT. The alternate key value at location KA and the condition specified in the REL field determine the positioning of the index file. Execution of the GET macro positions the index file at the desired alternate key value and returns as many of its associated primary key values as the working storage area can contain.

The GETN macro can be executed after the GET macro to retrieve additional primary key values associated with the alternate key value. It can also be executed after a REWINDM, START, or SKIPFL macro to begin returning primary key values from the position established by the previous macro. Primary keys are returned to the working storage area until one of the following conditions occurs:

- The working storage area is full.

- The end of the list of alternate key values is reached (end-of-information).
- The index file is positioned at the beginning of a primary key list for an alternate key that is greater than the key at location KA when the value of the REL field in the FIT is GT or GE, or the index file is positioned at the beginning of a primary key list for an alternate key that is equal to the key at location KA when the value of the REL field is GE or EQ.

The key address (KA) field in the FIT must be set for the GETN macro when the index file is being accessed. If primary key list retrieval is to be terminated according to a key value, the KA field must point to the location containing the key value. If the KA field is set to 0, primary key list retrieval terminates only if the working storage area is filled or if end-of-information is reached. This is the same as if the key value at location KA is greater than any possible value for the alternate key.

The SKIPFL macro is used to count the number of primary key values for one or more alternate key values; the primary key values are not returned to the working storage area. The counting can be terminated by a key value in the same manner as the GETN macro. Counting can also be specified for a number of alternate key values or to end-of-information.

FIT FIELDS FOR INDEX FILE PROCESSING

Index file processing involves user setting of several fields in the FIT. In addition, AAM sets certain FIT fields during macro execution. The following FIT fields can be set by the user:

KA — Key address; location of the user-supplied key value for START and GET macros and for GETN and SKIPFL macros that use a key.

KL — Key length; number of characters in the alternate key being accessed.

KP — Key position; position of user-supplied key value at location KA.

MKL — Major key length; number of characters, which is less than the full length of the alternate key, in the user-supplied symbolic key value. For primary key processing in MIP, MKL applies only to collated symbolic keys (KT=S). For alternate key processing in MIP, MKL applies to collated or uncollated symbolic keys (KT=S or U).

MRL — Maximum record length; length of the working storage area in characters; should be a multiple of 10 characters because each primary key value returned begins on a new word boundary.

NDX — Index flag; must be set to YES for index file access only.

REL — Key relation; indicates the relation to be satisfied between the user-supplied key value and the index file key value; possible relations are EQ, GE, and GT; for initial MIP, LE and LT can also be used.

RKP — Relative key position; beginning character position of the alternate key within the word specified by the RKW field.

RKW — Relative key word; word in which the alternate key being accessed begins.

WSA — Working storage area; location into which primary key lists are returned.

The following FIT fields are set by AAM during execution of the START, GET, GETN, and SKIPFL macros:

FP — File position; set to indicate the position of the index file when control returns to the user:

0	Middle of primary key list
10_8	End of primary key list
100_8	End-of-information

KNE — Key not equal; for an operation involving a key, indicates whether or not the current alternate key value matches the user-supplied key value:

0	Equal key values
1	Higher user-supplied key value or end-of-information

MKL — Major key length; reset to 0 after a user-supplied value has been noted. For primary key processing in MIP, MKL applies only to collated symbolic keys (KT=S). For alternate key processing in MIP, MKL applies to collated or uncollated symbolic keys (KT=S or U).

PTL — Primary key total; number of primary key values transferred to the working storage area during execution of the GET or GETN macro.

RC — Record count. After a START or after a GET that did not run to completion (FP=0), RC contains the number of primary keys associated either with the given alternate key value or, if there is no match in the file or REL=GT, with the next higher alternate key value. After a GET that did run to completion, the number of associated primary keys is found in RL instead of RC.

RL — Set by the GET, START, SKIPFL, and GETN macros as follows:

GET	Set to the value in the PTL field.
START	Set to zero.
SKIPFL	Set to the number of primary key values that have been skipped.
GETN	Increased by the number of primary key values transferred to the working storage area; cleared on entry only if the file position from the last operation was end-of-keylist (EOK).

COUNT RETRIEVAL

The primary key values associated with a given alternate key value are counted by executing the START macro. The RKP, RKW, and KL fields in the FIT must be set to identify the alternate key. Because a specific alternate key value is involved, the major key length (MKL) field is set to 0 for full length key comparison and the key relation (REL) field is set to equal (EQ). The format of the START macro is as follows:

```
START     fit,ex,ka,kp
```

The fit parameter specifies the address of the data file FIT with which the index file is associated. The file is positioned at the alternate key value that is equal to the value at the location specified by the ka parameter; the record count (RC) field in the FIT contains the number of primary keys associated with the alternate key value. The key not equal (KNE) field is set to 0 or 1 depending on whether or not the desired value has been found.

The file position (FP) field in the FIT is set during execution of the START macro. It is set to 10_8 if the index file is positioned at an alternate key value. If, however, the user-supplied key value is greater than all existing values for the alternate key, the FP field is set to 100_8.

RANGE COUNT RETRIEVAL

The number of primary keys associated with a range of consecutive alternate key values can be determined by executing a REWINDM or START macro and then a SKIPFL macro. The beginning and end of the range can be specified in various ways.

The beginning of the range indicates the first alternate key value for which primary keys are to be counted. The key value is specified as one of the following:

- The first alternate key value in the file; execution of the REWINDM macro positions the index file to this point.
- The first alternate key value that is not less than a specified value; the REL field in the FIT is set to GE and the START macro is executed to reach this position in the index file.
- The first alternate key value that is greater than a specified value; the REL field in the FIT is set to GT and the START macro is executed to reach this position in the index file.

If a major key is specified for the START macro, only the number of characters in the major key are used for comparison. If the REL field is set to EQ or GE, the file is positioned at the first alternate key value with leading characters that match the major key. If no such key exists, the file is positioned at the next logical alternate key value. If the REL field is set to GT, the file is positioned at the first alternate key value with leading characters greater than the major key value.

The end of the range, which is the last alternate key value to be included in the range count, is specified by setting various FIT fields before executing the SKIPFL macro. The last key value is determined as follows:

- If the key address (KA) field is set to 0, the last alternate key value in the index is the end of the range.
- If the KA field points to a location that contains an alternate key value and the key relation (REL) field is set to GT, all key values not exceeding the value at location KA are included in the count.
- If the KA field points to a location that contains an alternate key value and the REL field is set to GE, all key values less than the value at location KA are included in the count.

After the SKIPFL macro is executed, the RL field in the FIT contains the number of primary key values for all the alternate key values within the specified range.

PRIMARY KEY LIST RETRIEVAL

The list of primary keys for a specific alternate key value can be retrieved by executing the GET macro. The major key length (MKL) field in the FIT should be set to 0 for a full-length alternate key comparison and the key relation (REL) field should be set to EQ for an equal comparison. When the GET macro is executed, the key not equal (KNE) field is set to 0 if the alternate key value is found in the index file or to 1 if it is not found. The format of the GET macro is:

```
GET     fit,wsa,0,ex,ka,kp,mkl
```

Execution of the GET macro causes the primary key values associated with the alternate key value to be transferred to the working storage area. Transfer of primary key values terminates when the last primary key value has been transferred or when the working storage area has been filled. The following FIT fields indicate the status of a successful primary key list retrieval:

FP — File position; set to 10_8 when all primary keys have been transferred; otherwise, set to 0.

PTL — Primary key total; number of primary keys transferred to the working storage area.

RC — Record count; the total number of primary keys associated with the alternate key value if not all of these keys have been delivered (FP=0). If all the primary keys have been delivered (FP=10_8), the total number of primary keys is found in RL.

RL — Same as the PTL field after a GET operation (the number of primary key values delivered by the GET). If all the primary keys have been delivered (FP=10_8), RL contains the total number of primary keys.

If the operation is unsuccessful, the fields are not defined.

If the FP field is set to 10_8, the entire primary key list has been retrieved. In this case, the PTL and RL fields contain the same value. The index file is positioned at the beginning of the primary key list for the next alternate key value.

The FP field set to 0 indicates that additional primary keys are associated with the alternate key value. The RC field contains a value greater than the PTL and RL fields, which contain equal values. The number of additional primary keys is given by the difference between RC and RL. The remaining primary keys can be retrieved by executing the GETN macro after making the working storage area available for use again. Primary keys are transferred until

the end of the list is reached or the working storage area is filled. Additional GETN macros can be executed to complete transfer of the primary key list. The FP field in the FIT is set to 10_8 when the last primary key in the list is transferred to the working storage area.

The normal purpose of primary key list retrieval is to determine the primary key values for a specific alternate key value. If the major key length (MKL) field in the FIT is set to a value other than 0, more than one alternate key value could satisfy the condition of the REL field. The GETN macro execution would then continue until the index file is positioned at an alternate key value that does not satisfy the condition specified by the REL field.

Whenever a GETN macro is executed, the RL field is incremented by the number of primary keys transferred to the working storage area; the PTL field indicates the number of primary keys transferred during execution of the macro most recently executed (GET or GETN). The final value in the RL field (when the FP field contains 10_8) should equal the value in the RC field after execution of the GET macro, which should also equal the total of the values in the PTL field after execution of the GET macro and all subsequent GETN macros.

RANGE LIST RETRIEVAL

The primary key lists for a range of consecutive alternate key values can be retrieved through execution of a START macro followed by execution of one or more GETN macros. The beginning of the range of alternate key values is established in the same manner as for the range count retrieval; that is, the REWINDM macro can be used to position the index file to the first alternate key value, or the START macro can be used to position the file to a specific alternate key value.

Once the beginning of the range has been established, the GETN macro is executed to transfer primary keys to the working storage area. To determine when the primary key lists for the range of alternate key values have all been transferred, the file position (FP) field in the FIT must be checked for a value of 10_8 (end-of-keylists) or 100_8 (end-of-information) after execution of the GETN macro.

The end of the range of alternate key values is determined by the setting of certain fields in the FIT:

- If the key address (KA) field is set to 0, the end of the range is end-of-information.
- If the KA and key position (KP) fields are set to indicate an alternate key value and the key relation (REL) field is set to GT, the end of the range is the last alternate key value that is not greater than the one indicated by the KA and KP fields.
- If the KA and KP fields are set to indicate an alternate key value and the REL field is set to GE, the end of the range is the last alternate key value that is less than the one indicated by the KA and KP fields.

Whenever the GETN macro is executed, the FP field in the FIT should be checked. If it is equal to 10_8 or 100_8, all the desired primary key lists have been retrieved. If FP is equal to 0, however, the working storage area should be made available for retrieval of more primary keys and another GETN macro should be issued.

Several utility routines are provided for use with AAM files. Utilities are available for:

- Printing statistics
- Displaying alternate key fields (as defined by MIPGEN) for definition checking
- Estimating the optimal block and buffer sizes for indexed sequential files
- Performing key analysis for direct access files
- Creating direct access files
- Creating an index file for alternate key access to an existing file

The utilities are called by operating system control statements. File dumping and reloading functions are handled by FORM and permanent file utilities.

INDEXED SEQUENTIAL FILES

Two utilities are provided for use with indexed sequential files. These utilities print statistics and suggest block and buffer sizes.

FLSTAT UTILITY

The FLSTAT utility is primarily used for displaying statistical information about an AAM file. This utility has a secondary function; it displays those fields in records of an AAM file that have been designated by a set of RMKDEF directives. (RMKDEF directives work through the MIPGEN utility, which is explained later in this section.)

FLSTAT Statistical Information

The FLSTAT can be used to display statistical information about an indexed sequential, direct access, actual key, or index file. When used for this purpose, the format of the FLSTAT control statement is shown in figure 7-1.

```
FLSTAT(lfn,sfn)

lfn     Logical file name of the AAM file
        about which the information is wanted.

sfn     Logical file name of the file on which
        the information is to be written;
        default is OUTPUT.
```

Figure 7-1. FLSTAT Control Statement Format for Statistical Information

FLSTAT(lfn) means the same as FLSTAT(lfn,OUTPUT).

The amount of information output by the FLSTAT utility depends on whether or not an installation option is selected. (Refer to the Installation Handbook for details.) Figure 7-2 shows the output generated for a data file and an index file when the installation option is not selected. Figure 7-3 shows the output generated for the same two files when the option is selected.

FLSTAT uses values from the file statistics table (FSTT). The FSTT can be updated only if the AAM file is attached with write or write modify permission. If the file is attached with read or read modify permission, those FLSTAT statistics relating to file usage (transactions, calls) might not be accurate. Those statistics describing size of file or key information are not affected.

FLSTAT Alternate Key Information

For existing AAM files, an FLSTAT option can display which fields will become alternate keys as a result RMKDEF directive definitions.

The purpose of this secondary FLSTAT function is to aid a user who is about to use the MIPGEN utility on an indexed sequential, direct access, or actual key file. Part of the input to MIPGEN is a set of RMKDEF directives, defining the fields of each record that are to become alternate keys. The second, third, and fourth parameters on a RMKDEF directive are RKW, RKP, and KL, defining the position and length of a field. The user might not be certain that these values define the desired fields. In that case, FLSTAT can be used to check the fields. When used for this purpose, the format of the FLSTAT control statement is as shown in figure 7-4.

FLSTAT(lfn,,RMKDEF) is equivalent to:

FLSTAT(lfn,OUTPUT,RMKDEF,INPUT).

When the statement is entered interactively and the default INPUT is allowed to occur, the user is prompted to enter RMKDEF directives.

FLSTAT first lists the RMKDEF directives that were specified, up to nine directives. The utility then copies the first 20 records of the file, 100 characters per line.

FLSTAT labels the fields that are defined by the RMKDEF directives using the following scheme: 1's are placed under the field defined by the first RMKDEF directive, 2's are placed under the field defined by the second RMKDEF directive, and so on. A sample output is shown in figure 7-5.

```
STATISTICS FOR FILE RELFIT


ORGANIZATION------- IS
CREATION DATE------ 07/06/78
DATE OF LAST CLOSE- 07/06/78
TIME OF LAST CLOSE- 18.35.37.

FILE IS NOT MIPPED
COLLATION IS STANDARD

PRIMARY KEY INFORMATION
  KEY IS NOT EMBEDDED
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 10

MAXIMUM RECORD SIZE  100
MINIMUM RECORD SIZE    0

TOTAL TRANSACTIONS
  NUMBER OF PUTS ------ 10
  NUMBER OF GETS ------ 0
  NUMBER OF DELETES --- 0
  NUMBER OF REPLACES -- 0
  NUMBER OF GETNEXTS -- 2

CIO CALLS FOR FILE
  NUMBER OF READS ----- 2
  NUMBER OF WRITES ---- 2
  NUMBER OF RECALLS --- 0
  NUMBER OF REWRITES -- 2

NUMBER OF BLOCKS------- 1
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 2
NUMBER OF DATA RECORDS- 10

FILE LENGTH IN PRUS  4
NUMBER OF INDEX LEVELS IN USE    0
```

```
STATISTICS FOR FILE INDEXF


ORGANIZATION------- MIP
CREATION DATE------ 08/23/78
DATE OF LAST CLOSE- 08/23/78
TIME OF LAST CLOSE- 15.44.48.

PRIMARY KEY INFORMATION
  KEY IS NOT EMBEDDED
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 5

ALTERNATE KEY INFORMATION
  CHARACTERS IN LARGEST KEY-- 20

PRIMARY KEY SUBSTRUCTURES
  NUMBER OF UNIQUE -- 4
  NUMBER OF -IS-   -- 2
  NUMBER OF FIFO   -- 1

NUMBER OF BLOCKS------- 8
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 4
NUMBER OF DATA RECORDS- 7

FILE LENGTH IN PRUS  34
MAX NUMBER OF LEVEL 2 INDEX LEVELS  4
MAX NUMBER OF LEVEL 3 INDEX LEVELS  4
```

Figure 7-2. FLSTAT Utility Regular Output

```
STATISTICS FOR FILE RELFIT


ORGANIZATION------- IS
CREATION DATE------ 07/11/78
DATE OF LAST CLOSE- 07/11/78
TIME OF LAST CLOSE- 07.22.11.

FILE IS NOT MIPPED
COLLATION IS STANDARD

PRIMARY KEY INFORMATION
  KEY IS NOT EMBEDDED
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 10

MAXIMUM RECORD SIZE  100
MINIMUM RECORD SIZE  0

TOTAL TRANSACTIONS
  NUMBER OF PUTS ------ 10
  NUMBER OF GETS ------ 0
  NUMBER OF DELETES --- 0
  NUMBER OF REPLACES -- 0
  NUMBER OF GETNEXTS -- 2
  NUMBER OF SEEKS ----- 2
  NUMBER OF GETNRS ---- 2

CIO CALLS FOR FILE
  NUMBER OF READS ----- 2
  NUMBER OF WRITES ---- 2
  NUMBER OF RECALLS --- 0
  NUMBER OF REWRITES -- 2

NUMBER OF BLOCKS ------ 1
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 2
NUMBER OF DATA RECORDS- 10

FILE LENGTH IN PRUS  4
NUMBER OF INDEX LEVELS IN USE    0
```

```
STATISTICS FOR FILE INDEXF


ORGANIZATION------- MIP
CREATION DATE------ 09/13/78
DATE OF LAST CLOSE- 09/13/78
TIME OF LAST CLOSE- 14.35.21.

PRIMARY KEY INFORMATION
  KEY IS NOT EMBEDDED
  TYPE -- COLLATED SYMBOLIC
  LENGTH IN CHARACTERS -------- 5

ALTERNATE KEY INFORMATION
  CHARACTERS IN LARGEST KEY-- 20

PRIMARY KEY SUBSTRUCTURES
  NUMBER OF UNIQUE -- 4
  NUMBER OF -IS-   -- 2
  NUMBER OF FIFO   -- 1

NUMBER OF BLOCKS------- 8
NUMBER OF EMPTY BLOCKS- 0
BLOCK SIZE IN PRUS----- 4
NUMBER OF DATA RECORDS- 7

FILE LENGTH IN PRUS  34
MAX NUMBER OF LEVEL 2 INDEX LEVELS  4
MAX NUMBER OF LEVEL 3 INDEX LEVELS  4
```

Figure 7-3. FLSTAT Utility Expanded Output

```
FLSTAT(lfn,sfn,RMKDEF,dfn)

lfn       Logical file name of the AAM file
          about which the information is wanted.

sfn       Logical file name of the file on which
          the information is to be written;
          default is OUTPUT.

RMKDEF    The value RMKDEF.  When this is speci-
          fied, FLSTAT displays alternate key
          information; it suppresses statistical
          information that would be produced by
          FLSTAT(lfn).  Any explicit value other
          than RMKDEF results in an error, and
          no output is produced.

dfn       Logical file name of the file that
          contains the set of RMKDEF directives;
          default is INPUT.
```

Figure 7-4. FLSTAT Control Statement Format for Alternate Key Information

The following should be noted:

On a record, information beyond 1000 characters is ignored.

Up to nine directives are shown. All those after the ninth are ignored.

Where keys overlap, the field of overlap is marked with the higher digit.

Any specification of a repeating group by an RMKDEF directive is ignored.

FLBLOK UTILITY

The FLBLOK utility is an aid to the user in creating an indexed sequential file. The utility provides a method of comparing the effects of different values in some essential FIT fields. The output information is based on stepping the maximum block length (MBL) field through its possible values and calculating file characteristics. In indexed sequential files, MBL applies to both index and data blocks; therefore, the content of MBL determines how many index and data records can fit in a block. MBL has major effects on both the physical structure and the performance characteristics of the file. The main purpose of the utility is to indicate the best MBL value.

```
                RMKDEF(FLRECS,0,0,2,0,S,I)
                RMKDEF(FLRECS,3,3,20,0,S,I)
                RMKDEF(FLRECS,4,3,7,0,S,I)
                RMKDEF(FLRECS,5,3,11,0,S,I)
                RMKDEF(FLRECS,6,3,11,0,S,I)
                49 ADAM'S RIB                    2 TRACY     HEPBURN
                11                                 22222222223333333222444444444455555555555
                38 BRINGING UP BABY              2 HEPBURN   GRANT
                11                                 22222222223333333222444444444455555555555
                42 CASABLANCA                    4 HENREID   BERGMAN   RAINES     LORRE
                11                                 22222222223333333222444444444455555555555
                46 DECEPTION                     3 DAVIS     HENREID   RAINES
                11                                 22222222223333333222444444444455555555555
                39 ELIZABETH AND ESSEX           3 FLYNN     DAVIS     DEHAVILLAND
                11                                 22222222223333333222444444444455555555555
                44 IN OUR TIME                   2 LUPINO    HENREID
                11                                 22222222223333333222444444444455555555555
                41 JOAN OF PARIS                 1 HENREID
                11                                 22222222223333333222444444444455555555555
                39 MR SMITH GOES TO WASHINGTON 2 RAINES    STEWART
                11                                 22222222223333333222444444444455555555555
                42 NOW VOYAGER                   3 HENREID   DAVIS     RAINES
                11                                 22222222223333333222444444444455555555555
                34 OF HUMAN BONDAGE              4 DAVIS     HOWARD    PARKER     HENREID
                11                                 22222222223333333222444444444455555555555
                38 THE ADVENTURES OF ROBIN HOOD4 FLYNN     RAINES    DEHAVILLANDRATHBONE
                11                                 22222222223333333222444444444455555555555
                51 THE AFRICAN QUEEN             2 BOGART    HEPBURN
                11                                 22222222223333333222444444444455555555555
                33 THE INVISIBLE MAN             1 RAINES
                11                                 22222222223333333222444444444455555555555
                41 THE LITTLE FOXES              2 DAVIS     MARSHALL
                11                                 22222222223333333222444444444455555555555
                34 THE LITTLE MINISTER           1 HEPBURN
                11                                 22222222223333333222444444444455555555555
                40 THE PHILADELPHIA STORY        3 HEPBURN   GRANT     STEWART
                11                                 22222222223333333222444444444455555555555
          RM NOTE  1010 ON LFN FLRECS          640
```

Figure 7-5. FLSTAT Utility Alternate Key Output

The FLBLOK utility increases MBL (stepping by PRU) until the file can be built in the number of index levels indicated by the value of the NL parameter on the FLBLOK control statement. As MBL is increased through its maximum possible value of 128 PRUs, the file characteristics are computed.

A variety of file description parameters are input to the FLBLOK utility by the user. The format of the FLBLOK control statement is shown in figure 7-6. For best results, all input parameters should be specified. Special care should be taken when specifying NR and RL as follows:

NR — NR is the total number of records the file is expected to contain. This is not necessarily the file limit (FLM) that is related to the output field record capacity.

RL — RL is the average record length. Note that for nonembedded keys, AAM appends the key to the record; therefore, KL should be added to the RL specified to FLBLOK. While the number is usually an estimate, FLBLOK results depend heavily on this parameter. If the user has a knowledge of the data, a better number than that calculated by AAM can be used (given only the maximum and minimum record length, AAM uses a mean average). If most of the records are of a specific length, a mode average should be used by setting RL to that specific length. If the records are well distributed, a median average should be used by setting RL so half the records are larger and half are smaller.

FLBLOK,lfn,keyword=value,keyword=value, ...

lfn	Output file name; default is OUTPUT.
keyword	Any of the following:
NR	An integer indicating the number of records; default is 100 000. Under NOS, a maximum of 7 digits can be specified; under NOS/BE, a maximum value of 1 073 341 823 can be specified.
KL	An integer indicating the key length in characters; default is 10.
RL	An integer indicating the average record length in characters: default is 80. Add KL for nonembedded keys.
IP	An integer indicating index padding percentage; default is 0.
DP	An integer indicating data padding percentage; default is 0.
NL	An integer indicating the highest number of index levels to be printed; default is 3. Possible values smaller than or equal to NL are printed.
MRL	An integer indicating the maximum record length in characters; add KL for nonembedded keys. Default is RL (fixed length records).

Figure 7-6. FLBLOK Control Statement Format

The input deck structure consists only of a job statement, the FLBLOK control statement, and a 6/7/8/9 card image.

FLBLOK utility output in batch mode is shown in figure 7-7. When the output file is connected to a terminal, only essential information is provided as shown in figure 7-8. For batch and interactive mode, the printout indicates when the number of index levels decreases (MBL is at a minimum for that NL). For batch mode only, the printout also indicates when disk usage decreases within a specific NL.

Unless disk usage is critical, performance and buffer information should provide the criteria for choosing MBL. The smallest MBL for a given number of index levels yields the best random performance and the smallest buffers. If moving to a smaller number of index levels improves random performance, the smaller number of index levels should be used. If random performance between two levels is close, sequential performance and buffers should be taken into consideration. Although scarce system resources can reorder the list, the usual hierarchy for selecting MBL is:

1. Random performance
2. Sequential performance
3. Buffer size
4. Disk usage (batch mode only)

Columnar information printed by the FLBLOK utility in both batch and interactive modes is listed and described in table 7-1.

ACTUAL KEY FILES

One utility is provided for use with actual key files. The FLSTAT utility displays either statistical information concerning an actual key file since file creation time, or it displays information about alternate key definitions. This utility is discussed in this section under indexed sequential files.

DIRECT ACCESS FILES

Three utilities are provided for use with direct access files. These utilities analyze the effectiveness of a hashing routine and create a direct access file. All three utilities require the Common Memory Manager (CMM) to be present within the current version of the operating system.

FLSTAT UTILITY

The FLSTAT utility displays either statistical information concerning a direct access file since file creation time, or it displays information about alternate key definitions. This utility is discussed in this section under indexed sequential files.

KEY ANALYSIS UTILITY

The key analysis utility tests hashing routines for effectiveness in producing uniform distribution of record keys in a file. A uniform distribution optimizes processing time. The key analysis utility can be called by reading the input file and calling the key analysis utility to process the file on a record-by-record basis.

```
                                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE

          CONTROL STATEMENT : FLBLOK,OUTPUT,RL=200,KL=25,NR=30000,DP=0,IP=5,NL=2,MRL=400.

             PARAMETER VALUES USED THIS RUN :

             TOTAL NUMBER OF RECORDS(NR) =     30000        KEY LENGTH(CHARACTERS)(KL) =      25
             AVERAGE RECORD LENGTH(RL)   =       200        MAXIMUM RECORD LENGTH(MRL) =     400
             DATA BLOCK PADDING(DP)      =         0        INDEX BLOCK PADDING(IP)    =       5
             NUMBER OF INDEX LEVELS(NL)  =         2

                 ********************************************************************************

  (1)†     (2)          (9)          (4)          (10)         (7)           (5)          (6)       (3)
  NUMBER   BLOCK        MINIMUM      AVERAGE      RECORDS      FILE          NON-POOLED   MINIMUM   844-41 FULL
  INDEX    LENGTH       DISK USAGE   ACCESSES     PER          CAPACITY      PERF. BUFFER BUFFER    TRK. DISK GET
  LEVELS   (MBL)        WITH NR      PER GET      BLOCK        IN RECORDS    SIZE (WORDS) SIZE      TIME APROX.
  (NL)     (CHARS/PRUS) RECORDS(PRUS) (SEQ./RANDOM) (DATA/INDEX) (MAX./PADDED) (SEQ./RANDOM) (WORDS) (SEQ./RANDOM)

     2      1870/3          10178      .111/2       9/59         34596/31329   733/925      541      3.6/64
     2      2510/4          10134      .083/2       12/79        82668/74892   925/1181     669      2.7/65
     2      3150/5          10107      .067/2       15/100       165375/150000 1117/1437    797      2.2/66
     2      3790/6          10094      .056/2       18/120       285768/259200 1309/1693    925      1.9/68
     2      4430/7          10089      .048/2       21/140       453789/411600 1501/1949    1053     1.7/69
     2      5070/8          10074      .042/2       24/160       685464/614400 1693/2205    1181     1.5/71
     2      6350/10          9742      .032/2       31/201       1380151/1252431 2077/2717  1437     1.2/73
     2      11470/18         9704      .018/2       56/363       8171744/7379064 3613/4765  2461     .8/85
                 ----------------------------------------------------------------------------
     1      14030/22         9748      .015/1       68/444       31756/30192   4381/4381    2973     .7/45
     1      17230/27         9695      .012/1       84/545       48216/45780   5341/5341    3613     .6/48
     1      17870/28         9690      .011/1       87/566       51765/49242   5533/5533    3741     .6/49
     1      22990/36         9686      .009/1       112/728      85792/81536   7069/7069    4765     .5/55
     1      27470/43         9677      .007/1       134/870      122610/116580 8413/8413    5661     .4/60
        (8) PROBABLE BEST MBL = 17870

                                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE

          CONTROL STATEMENT : FLBLOK,,NR=1000,RL=150,DP=0,IP=5,MRL=150,KL=15,NL=4.

             PARAMETER VALUES USED THIS RUN :

             TOTAL NUMBER OF RECORDS(NR) =      1000        KEY LENGTH(CHARACTERS)(KL) =      15
             AVERAGE RECORD LENGTH(RL)   =       150        MAXIMUM RECORD LENGTH(MRL) =     150
             DATA BLOCK PADDING(DP)      =         0        INDEX BLOCK PADDING(IP)    =       5
             NUMBER OF INDEX LEVELS(NL)  =         4

                 ********************************************************************************

  (1)      (2)          (9)          (4)          (10)         (7)           (5)          (6)       (3)
  NUMBER   BLOCK        MINIMUM      AVERAGE      RECORDS      FILE          NON-POOLED   MINIMUM   844-41 FULL
  INDEX    LENGTH       DISK USAGE   ACCESSES     PER          CAPACITY      PERF. BUFFER BUFFER    TRK. DISK GET
  LEVELS   (MBL)        WITH NR      PER GET      BLOCK        IN RECORDS    SIZE (WORDS) SIZE      TIME APROX.
  (NL)     (CHARS/PRUS) RECORDS(PRUS) (SEQ./RANDOM) (DATA/INDEX) (MAX./PADDED) (SEQ./RANDOM) (WORDS) (SEQ./RANDOM)

     2      590/1            349      .333/2       3/28         2523/2352     347/411      283      10.2/61
     2      1230/2           260      .125/2       8/58         29768/26912   539/667      411      3.9/62
                 ----------------------------------------------------------------------------
     1      1870/3           257      .083/1       12/89        1116/1068     731/731      539      2.7/32
     1      3150/5           247      .048/1       21/150       3297/3150     1115/1115    795      1.6/33
        (8) PROBABLE BEST MBL = 3150
```

† See Table 7-1.

Figure 7-7. FLBLOK Utility Sample Output in Batch Mode

```
                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE
     CONTROL STATEMENT : FLBLOK,OUTPUT,RL=200,KL=25,NR=30000,DP=0
                         ,IP=5,NL=2,MRL=400.

MAXIMUM RECORD LENGTH(MRL) =  400   NUMBER OF INDEX LEVELS(NL) =  2
AVERAGE RECORD LENGTH(RL)  =  200   DATA BLOCK PADDING(DP)     =  0
KEY LENGTH(CHARACTERS)(KL) =   25   INDEX BLOCK PADDING(IP)    =  5
                TOTAL NUMBER OF RECORDS(NR) =     30000
        (2)          (3)          (4)          (5)                (7)
 (1)†  BLOCK      844-41       ACCESSES     NON-POOLED    (6)      MAXIMUM
 NO.   LENGTH     DISK GET     PER          BUFFER        MIN.     FILE
 NDX.  (MBL)      TIME EST.    GET          SIZE(WDS)     BUFFER   CAPACITY
 LVLS  (CHARS/    (SEQ./       (SEQ./       (SEQ./        SIZE     IN
 (NL)  PRUS)      RANDOM)      RANDOM)      RANDOM)       (WORDS)  RECORDS

   2   1870/3      3.6/64      .111/2        733/925        541      34596
   1  14030/22      .7/45      .015/1       4381/4381      2973      31756
    (8) PROBABLE BEST MBL = 14030


                INDEXED SEQUENTIAL FILE PARAMETER ESTIMATE
     CONTROL STATEMENT : FLBLOK,,NR=1000,RL=150,DP=0,IP=5,MRL=150
                         ,KL=15,NL=4.

MAXIMUM RECORD LENGTH(MRL) =  150   NUMBER OF INDEX LEVELS(NL) =  4
AVERAGE RECORD LENGTH(RL)  =  150   DATA BLOCK PADDING(DP)     =  0
KEY LENGTH(CHARACTERS)(KL) =   15   INDEX BLOCK PADDING(IP)    =  5
                TOTAL NUMBER OF RECORDS(NR) =      1000
        (2)          (3)          (4)          (5)                (7)
 (1)   BLOCK      844-41       ACCESSES     NON-POOLED    (6)      MAXIMUM
 NO.   LENGTH     DISK GET     PER          BUFFER        MIN.     FILE
 NDX.  (MBL)      TIME EST.    GET          SIZE(WDS)     BUFFER   CAPACITY
 LVLS  (CHARS/    (SEQ./       (SEQ./       (SEQ./        SIZE     IN
 (NL)  PRUS)      RANDOM)      RANDOM)      RANDOM)       (WORDS)  RECORDS

   2    590/1      10.2/61      .333/2        347/411        283      2523
   1   1870/3       2.7/32      .083/1        731/731        539      1116
    (8) PROBABLE BEST MBL = 1870
```

†See Table 7-1.

Figure 7-8. FLBLOK Utility Sample Output in Interactive Mode

TABLE 7-1. FLBLOK UTILITY OUTPUT DESCRIPTIONS

No.	Message	Significance	Description
①	Number of index levels (NL)	Informational	An indication of the number of disk accesses per random get.
②	Block length (MBL)	Needed in the FIT	An indication of the best values for MBL; applies to both index and data block length.
③	844-41 full track disk get time approximation	Informational	An approximate performance estimate calculated from the number of disk accesses per get, using the formulas supplied; provides a good basis for comparison. For best results, users should calculate individual performance using average accesses per get and installation system mass storage characteristics.

TABLE 7-1. FLBLOK UTILITY OUTPUT DESCRIPTIONS (Continued)

No.	Message	Significance	Description
④	Average accesses per get	Informational	A system independent performance number. Sequential accesses per get assume a set of sequential gets. Equations used to derive real performance from these numbers are: random performance = random accesses x block get time sequential performance = sequential accesses x block get time block get time = disk seek time + (disk retrieval time x MBL in PRUs)
⑤	Non-pooled performance buffer size	Informational (can be used in the FIT)	A good approximation of what the buffer will be in a non-pooled situation. Generally, it is best to let AAM set buffer size.
⑥	Minimum buffer size	Informational (can be used to set BFS in the FIT)	Minimum buffer sizes without severe performance degradation.
⑦	Maximum file capacity in records	Can be used in the FIT	The maximum number of records of length RL that will fit into the file. The number is calculated on a file just created from sorted records. If the file grows too large, another level of index will be added and result in severe performance degradation. It might be useful to set FLM to this maximum quatity to help flag the oversize situation.
⑧	Probable best MBL	Informational (can be used in the FIT)	An indication of the best value for MBL from all possible values.
⑨	Minimum disk usage†	Informational	A probable lower bound on disk usage, usually unreachable. The calculation is made on the assumption that the file has just been created using sorted records of length RL. The number is provided for purposes of comparison on FLBLOK runs only.
⑩	Records per block†	Informational	The number of data records in each data block and number of records associated with each index block.

†Output on batch runs only

The same hashing routine can be used for up to five tests varying the number of home blocks for each test. It is also possible to test up to five hashing routines with the same number of home blocks. The number of synonym records produced by each hashing routine is counted and the resulting information written to a file named KEYLIST. The file KEYLIST must be rewound and copied to the file OUTPUT for the results to be printed. Output can show synonym records only, standard deviations only, or both. The format of the output from the key analysis utility is shown in figure 7-9.

The key analysis utility is called through a source program. The format of the KYAN directive is shown in figure 7-10. The directive begins in column 1. All parameters must be declared; no default values are provided.

```
HOME BLOCK              entry1 . . . entry5

    0                    XXX . . .  XXX
    1                    XXX . . .  XXX
    .                      .
    .                      .
    .                      .
    n                    XXX . . .  XXX

STANDARD DEVIATION

                        entry1 . . . entry5

                        XX.XX      XX.XX
```

Figure 7-9. Key Analysis Output

```
KYAN(LFN=axxxxxx,MRL=i,KL=j,RKP=k,RKW=l,
  H1=entry1,hmb1,option1, ... H5=entry5,hmb5,option5)
```

axxxxxx	Logical file name of the file containing the user hashing routines; if the default hashing routine is used, LFN is set to zero.
i	Maximum record length in characters.
j	Key length in characters.
k	Relative key position within relative key word (RKW), counting from 0.
l	Relative key word in which the key begins, counting form 0.
entry1 ... 5	Entry point names of hashing routines to be tested; SDAHASH must be specified to test the system-supplied hashing routine.
hmb1 ... 5	Number of home blocks.
option1 ... 5	Output options: S Synonyms only D Standard deviations only B Synonyms and standard deviations

Figure 7-10. KYAN Directive Format

If a continuation statement is to be used for the first KYAN or subsequent statement, all 80 columns must be filled. A slash (/) in column 80 indicates continuation to a subsequent statement. A maximum of seven statements can be used. Parentheses must enclose the entire parameter list; no embedded blanks are allowed.

Possible error messages that are printed on the user's dayfile are as follows:

- NOT ENOUGH FIELD LENGTH. USE nnnnnn.

  The run is terminated because the field length cannot accommodate the internal tables.

- ILLEGAL PARAMETER IN INPUT CARD

  The run is terminated because the KYAN directive contained a bad parameter.

- ENTRYi - SYNONYM LIMIT EXCEEDED

  More than 4095 records have been hashed to the same home block. Processing terminates on the specific entry but continues on the other entries.

- ENTRYi - BAD KEY ENCOUNTERED

  A specific key hashes outside the home block area. This key is ignored and processing continues.

- MORE THAN 25 BAD KEYS ENCOUNTERED

  The run is terminated.

The key analysis utility can be entered through a source program written in COMPASS, COBOL, or FORTRAN. The field length requirement is the sum of the space needed by the source program, the hashing routines to be tested, AAM, SDAKYAN, and internal tables. The space needed by AAM varies as a function of the input file organization. The number of central memory words required for internal tables is the largest home block value specified.

The key analysis utility has two entry points: SDAKEYH and SDAENDH. The COMPASS user must open the input file and read the records one by one. As each record is read, the user program sets register A1 to point to the location of the key address and issues a return jump to SDAKEYH. A return jump to SDAENDH must be used to terminate use of the KYAN directive.

For a COBOL program, the linkage is as follows:

```
ENTER SDAKEYH USING data-name.

ENTER SDAENDH.
```

The data name contains the record key and must be an elementary item in the Working-Storage or the Common-Storage Section of the COBOL program.

For a FORTRAN program, the linkage is as follows:

```
CALL SDAKEYH (KA)

CALL SDAENDH
```

KA is the address of the record key.

An example of a deck structure using the key analysis utility as an external subroutine is shown in figure 7-11. Hashing routines to be tested are assumed to be in relocatable binary format on a permanent file named MYHASH.

```
Job statement
USER statement
CHARGE statement
ATTACH(MYHASH)
Compiler call
LGO.
REWIND(KEYLIST)
COPYBF(KEYLIST,OUTPUT)
7/8/9
User program source deck
7/8/9
KYAN(LFN=MYHASH, . . .)
6/7/8/9
```

Figure 7-11. Key Analysis as External Subroutine

CREATE UTILITY

The CREATE utility is available only when Sort/Merge has been installed. This utility can be used to create direct access files with embedded keys from a call through a user program. A direct access file is produced more rapidly when the CREATE utility is used than when such a file is produced by reading input and calling AAM to write each record. The CREATE utility should be used for files containing 1000 or more records.

In general, the CREATE utility hashes the key from an input record and prefixes the key to the record. Sort/Merge is then used to sort the hashed keys. After the sort operation, the prefixed keys are removed and the CREATE utility uses AAM to produce the direct access file.

A job using the CREATE utility involves the following:

- FILE control statement to describe input and output files
- Loader control statement to load AAMLIB for Sort/Merge: LDSET, LIB=AAMLIB
- CREATE directive on the file INPUT

The format of the CREATE directive is shown in figure 7-12. The second and third parameters are omitted when the default hashing routine is selected. Any operating system separator, as well as embedded blanks, can be used between parameters.

CREATE(lfn,hash,hfl)

lfn	Logical file name of the output file (same as specified in a FILE control statement for a direct access file).
hash	User hashing routine entry point.
hfl	Name of the file containing the hashing routine in relocatable binary form.

Figure 7-12. CREATE Directive Format

All input and direct access file characteristics (other than defaults) must be specified with FILE control statements. AAM modules must be loaded. If a user hashing routine is used, the routine must also be loaded.

The FILE control statement used to define the direct access file structure must specify the following parameters:

lfn	Logical file name
FO	FO=DA file organization
HMB	Number of home blocks
MNR	Minimum number of characters in any record
MRL	Maximum number of characters in any record
KL	Number of 6-bit characters in the key
BFS	Number of words in the buffer; default buffer size is 260 words; the buffer must be able to hold at least one home block

Additional file structure parameters can be included in the FILE control statement.

When the CREATE utility is called, the user must cause the input file to be read. After each record is read, the user must place the key in the record and give control to the utility at entry point SDACRTU. The key address, the working storage address, and the total record length must be passed to the CREATE utility. At the end of file processing, the user calls CREATE at entry point SDAENDC.

The source program must not reference the direct access file being created. A FILE control statement must be used to describe file structure. If key position is not left-justified at location KA, the key position (KP) field must be set by the FILE control statement. The two entry points used in calling the CREATE utility from a source program are SDACRTU and SDAENDC.

The appropriate data name, variable name, or list parameters for working storage area (WSA), key address (KA), and record length (RL) are provided in calls with SDACRTU as follows:

- COBOL

  ENTER SDACRTU USING wsa,ka,rl

- FORTRAN

  CALL SDACRTU wsa,ka,rl

- COMPASS

  A pointer to a comparable three-parameter list is stored in register A1; the call to SDACRTU uses a return jump.

The RL field must be specified as an integer in a COMPASS or FORTRAN program. In a COBOL program, the RL field must be specified by a COMP-1 item.

An example of a COBOL source code call to the CREATE utility is shown in figure 7-13; the Identification, Environment, and Data Divisions are assumed. This procedure illustrates that portion of a job in which the user reads each record, enters SDACRTU for hashing, and enters SDAENDC after all records are read to complete direct access file creation.

```
      .
      .
      .
      PROCEDURE DIVISION.
      START.
      OPEN INPUT lfn.
      PERFORM A n TIMES.
A     READ lfn INTO SDA-WSA AT END GO TO B.
      MOVE xx TO RL.
      .
      .
      .
      ENTER SDACRTU USING SDA-WSA, key,rl.
B     ENTER SDAENDC.
      CLOSE lfn.
      STOP RUN.
```

Figure 7-13. CREATE Call Through COBOL

MULTIPLE-INDEX FILES

Two utilities are provided for use with files processed by the Multiple-Index Processor (MIP).

MIPGEN UTILITY

The MIPGEN utility is used to create an index file for alternate key access to an existing AAM file. This utility can also be used to define additional alternate keys for a file or to remove alternate keys from a file. The existing data file must not be an empty file. Key specifications can define overlapping fields.

A job using the MIPGEN utility involves the following:

- A FILE control statement to identify the existing AAM file, to specify the logical file name of the index file, and to specify the index file block size (if user does not want it the same as data file block size)
- An RFL control statement to specify a field length of 65000_8 plus the size of the buffer to process the file (a larger field length improves efficiency; adding 15000_8 is suggested)
- A MIPGEN control statement to identify the existing data file, the source of additional control information (RMKDEF directives), and a list file for output from the utility
- A set of RMKDEF directives on the file INPUT or other file of card images

When the index file is created, each alternate key must be defined by an RMKDEF directive. Up to 255 alternate keys can be defined.

Alternate key definitions can be added to or purged from an existing index file only through the MIPGEN utility. Each alternate key to be added to or purged from the index file must be specified in an RMKDEF directive.

The format of the MIPGEN control statement is shown in figure 7-14. The format of the RMKDEF directives expected by the MIPGEN utility is shown in figure 7-15. The kg and kc parameters are used together and refer to a key that is a repeating group, such as that which results from the COBOL clause OCCURS n TIMES.

```
MIPGEN(prifile,directs,lfile)

prifile   Logical file name of the existing in-
          dexed sequential, direct access, or
          actual key file.

directs   Name of the file containing the RMKDEF
          directives; optional; default is INPUT.

lfile     Name of the file that contains the
          output listing from MIPGEN; optional;
          default is OUTPUT.
```

Figure 7-14. MIPGEN Control Statement Format

The structure of primary key lists is specified by the ks parameter. For efficiency in processing, indexed sequential structure is recommended. First-in first-out structure can also be specified; however, the ordering of primary keys generated by the MIPGEN utility should not be assumed to be the same order in which the data file records were created.

The nl, ie, and ch parameters are used to define sparse keys. An alternate key is defined as a sparse key when all values of the key are not desired to be indexed. Sparse keys cause short indexing operations that save disk space, computer time for index file maintenance, and search time. A sparse key is a result of either null suppression or sparse control characters.

The nl parameter specifies null suppression for an alternate key. If null suppression is specified, the alternate key index does not include primary keys for records that have null values for the alternate key. All spaces for a symbolic key and all zeros for an integer key are null values.

The ie and ch parameters are used when indexing of alternate key values is to be controlled by a sparse control character. The one-character field containing the sparse control character must be in the fixed-length portion of the record. The ie parameter specifies whether to include or exclude the alternate key values for records that contain a sparse control character. The ch parameter specifies the sparse control characters applicable to the alternate key being defined; up to 36 letters and digits can be specified as a character string.

The sparse control character field is identified by an RMKDEF directive that must appear before the directive defining the alternate key and its sparse control characters. This directive is specified in the following format:

```
RMKDEF(prifile,rkw,rkp,0)
```

RMKDEF(prifile,rkw,rkp,kl,0,kf,ks,kg,kc,nl,ie,ch)

prifile	Logical file name of the existing indexed sequential, direct access, or actual key file; required.
rkw	Relative word in the record in which the alternate key begins, counting from 0; required.
rkp	Relative beginning character position within the relative key word (rkw), counting from 0; required.
kl	Number of characters in the key, 1 to 255; required.
0	Required to mark position for the reserved field.
kf	Key format, required: 0 or S Symbolic 1 or I Integer 2 or U Uncollated symbolic 3 or P Purge alternate key definition from the index
ks	Substructure for each primary key list in the index; optional: U Unique (default) I Indexed sequential; recommended for efficiency in processing F First-in first-out
kg	Length in characters of the repeating group in which the key resides.
kc	Number of occurrences of the repeating group; zero for T type records, and a number for a repeating group embedded within the record.
nl	Null suppression; a null value is all spaces (symbolic key) or all zeros (integer key): 0 Null values are indexed (default) N Null values are not indexed
ie	Include/exclude sparse control character: I Include alternate key value if the record contains a sparse control character E Exclude alternate key value if the record contains a sparse control character
ch	Characters that qualify as sparse control characters; up to 36 letters and digits can be specified as a character string.

Figure 7-15. RMKDEF Directive Format, MIPGEN Utility

The rkw and rkp parameters identify the position of the sparse control character. The zero key length parameter indicates that the field is a sparse control character field.

MIPDIS UTILITY

The MIPDIS utility temporarily or permanently disassociates an index file from its associated AAM data file. If the primary and alternate key fields are not updated during the disassociation, the index file can be reassociated with the data file.

Whenever a data file that has an associated index file is opened, a safety lock in the file statistics table requires the index file to be present. The MIPDIS utility removes this requirement.

Disassociation of an index file from the data file is useful under various circumstances. One instance occurs when a data file that has an associated index file is no longer being accessed by alternate key. In this case, the index file is no longer needed and can be disassociated from the data file.

Indexed sequential files are sometimes reorganized to reclaim extraneous padding caused by block splitting and to redistribute it evenly throughout the file. Actual key files and direct access files also can be reorganized for similar reasons. The reorganization is accomplished through either the FORM utility or a user program. The index file can be disassociated from the data file before the reorganization. After the reorganization, the index file is still valid for the data file and can be associated with the data file again by the MIPDIS utility. This eliminates the need to create a new index file through the MIPGEN utility or during the creation of the restructured data file.

While the data file is disassociated, any changes to the primary or alternate key values are not reflected in the index file. This can result in errors when updating or accessing the file by alternate key.

The format of the MIPDIS control statement is shown in figure 7-16. This control statement can be used to disassociate or associate an index file with its data file.

MIPDIS(lfn1,da,lfn2)

lfn1	Logical file name of the data file.
da	Disassociate/associate index file: D Disassociate from data file A Associate with data file
lfn2	Logical file name of the index file; not required for disassociation.

Figure 7-16. MIPDIS Control Statement Format

STANDARD CHARACTER SETS

CONTROL DATA operating systems offer the following variations of a basic character set:

- CDC 64-character set
- CDC 63-character set
- ASCII 64-character set
- ASCII 63-character set

Table A-1 shows these character sets. The set in use at a particular installation was specified when the operating system was installed or deadstarted.

Depending on another installation option, the system assumes an input deck has been punched either in 026 or in 029 mode (regardless of the character set in use).

Under NOS/BE, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or any 7/8/9 card. The specified mode remains in effect through the end of the job unless it is reset by specification of the alternate mode on a subsequent 7/8/9 card.

Under NOS, the alternate mode can be specified by a 26 or 29 punched in columns 79 and 80 of any 6/7/9 card, as described for a 7/8/9 card. In addition, 026 mode can be specified by a card with 5/7/9 multipunched in column 1, and 029 mode can be specified by a card with 5/7/9 multipunched in column 1 and a 9 punched in column 2.

Graphic character representation appearing at a terminal or printer depends on the installation character set and the terminal type. Characters shown in the CDC Graphic column of table A-1 are applicable to BCD terminals; ASCII graphic characters are applicable to ASCII-CRT and ASCII-TTY terminals.

Several graphics are not common for all codes. Where these differences in graphics appear, assignment of collation positions and translation between codes must be made. Tables A-2 and A-3 show the CDC and ASCII character set collating sequences.

TABLE A-1. STANDARD CHARACTER SETS

Display Code (octal)	CDC			ASCII		
	Graphic	Hollerith Punch (026)	External BCD Code	Graphic Subset	Punch (029)	Code (octal)
00†	: (colon) ††	8-2	00	: (colon) ††	8-2	072
01	A	12-1	61	A	12-1	101
02	B	12-2	62	B	12-2	102
03	C	12-3	63	C	12-3	103
04	D	12-4	64	D	12-4	104
05	E	12-5	65	E	12-5	105
06	F	12-6	66	F	12-6	106
07	G	12-7	67	G	12-7	107
10	H	12-8	70	H	12-8	110
11	I	12-9	71	I	12-9	111
12	J	11-1	41	J	11-1	112
13	K	11-2	42	K	11-2	113
14	L	11-3	43	L	11-3	114
15	M	11-4	44	M	11-4	115
16	N	11-5	45	N	11-5	116
17	O	11-6	46	O	11-6	117
20	P	11-7	47	P	11-7	120
21	Q	11-8	50	Q	11-8	121
22	R	11-9	51	R	11-9	122
23	S	0-2	22	S	0-2	123
24	T	0-3	23	T	0-3	124
25	U	0-4	24	U	0-4	125
26	V	0-5	25	V	0-5	126
27	W	0-6	26	W	0-6	127
30	X	0-7	27	X	0-7	130
31	Y	0-8	30	Y	0-8	131
32	Z	0-9	31	Z	0-9	132
33	0	0	12	0	0	060
34	1	1	01	1	1	061
35	2	2	02	2	2	062
36	3	3	03	3	3	063
37	4	4	04	4	4	064
40	5	5	05	5	5	065
41	6	6	06	6	6	066
42	7	7	07	7	7	067
43	8	8	10	8	8	070
44	9	9	11	9	9	071
45	+	12	60	+	12-8-6	053
46	-	11	40	-	11	055
47	*	11-8-4	54	*	11-8-4	052
50	/	0-1	21	/	0-1	057
51	(	0-8-4	34	(	12-8-5	050
52	)	12-8-4	74	)	11-8-5	051
53	$	11-8-3	53	$	11-8-3	044
54	=	8-3	13	=	8-6	075
55	blank	no punch	20	blank	no punch	040
56	, (comma)	0-8-3	33	, (comma)	0-8-3	054
57	. (period)	12-8-3	73	. (period)	12-8-3	056
60	≡	0-8-6	36	#	8-3	043
61	[	8-7	17	[	12-8-2	133
62	]	0-8-2	32	]	11-8-2	135
63	% ††	8-6	16	% ††	0-8-4	045
64	≠	8-4	14	" (quote)	8-7	042
65	←	0-8-5	35	_ (underline)	0-8-5	137
66	∨	11-0	52	!	12-8-7	041
67	∧	0-8-7	37	&	12	046
70	↑	11-8-5	55	' (apostrophe)	8-5	047
71	↓	11-8-6	56	?	0-8-7	077
72	<	12-0	72	<	12-8-4	074
73	>	11-8-7	57	>	0-8-6	076
74	≤	8-5	15	@	8-4	100
75	≥	12-8-5	75	\	0-8-2	134
76	¬	12-8-6	76	^ (circumflex)	11-8-7	136
77	; (semicolon)	12-8-7	77	; (semicolon)	11-8-6	073

†Twelve zero bits at the end of a 60-bit word in a zero byte record are an end of record mark rather than two colons.

††In installations using a 63-graphic set, display code 00 has no associated graphic or card code; display code 63 is the colon (8-2 punch). The % graphic and related card codes do not exist and translations yield a blank (55_8).

TABLE A-2. CDC CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD	Collating Sequence Decimal	Collating Sequence Octal	CDC Graphic	Display Code	External BCD
00	00	blank	55	20	32	40	H	10	70
01	01	≤	74	15	33	41	I	11	71
02	02	%	63 †	16 †	34	42	v	66	52
03	03	[	61	17	35	43	J	12	41
04	04	→	65	35	36	44	K	13	42
05	05	≡	60	36	37	45	L	14	43
06	06	∧	67	37	38	46	M	15	44
07	07	↑	70	55	39	47	N	16	45
08	10	↓	71	56	40	50	O	17	46
09	11	>	73	57	41	51	P	20	47
10	12	≥	75	75	42	52	Q	21	50
11	13	¬	76	76	43	53	R	22	51
12	14	.	57	73	44	54	]	62	32
13	15	)	52	74	45	55	S	23	22
14	16	;	77	77	46	56	T	24	23
15	17	+	45	60	47	57	U	25	24
16	20	$	53	53	48	60	V	26	25
17	21	*	47	54	49	61	W	27	26
18	22	-	46	40	50	62	X	30	27
19	23	/	50	21	51	63	Y	31	30
20	24	,	56	33	52	64	Z	32	31
21	25	(	51	34	53	65	:	00 †	none†
22	26	=	54	13	54	66	0	33	12
23	27	≠	64	14	55	67	1	34	01
24	30	<	72	72	56	70	2	35	02
25	31	A	01	61	57	71	3	36	03
26	32	B	02	62	58	72	4	37	04
27	33	C	03	63	59	73	5	40	05
28	34	D	04	64	60	74	6	41	06
29	35	E	05	65	61	75	7	42	07
30	36	F	06	66	62	76	8	43	10
31	37	G	07	67	63	77	9	44	11

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.

TABLE A-3. ASCII CHARACTER SET COLLATING SEQUENCE

Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code	Collating Sequence Decimal	Collating Sequence Octal	ASCII Graphic Subset	Display Code	ASCII Code
00	00	blank	55	20	32	40	@	74	40
01	01	!	66	21	33	41	A	01	41
02	02	"	64	22	34	42	B	02	42
03	03	#	60	23	35	43	C	03	43
04	04	$	53	24	36	44	D	04	44
05	05	%	63†	25	37	45	E	05	45
06	06	&	67	26	38	46	F	06	46
07	07	'	70	27	39	47	G	07	47
08	10	(	51	28	40	50	H	10	48
09	11	)	52	29	41	51	I	11	49
10	12	*	47	2A	42	52	J	12	4A
11	13	+	45	2B	43	53	K	13	4B
12	14	,	56	2C	44	54	L	14	4C
13	15	-	46	2D	45	55	M	15	4D
14	16	.	57	2E	46	56	N	16	4E
15	17	/	50	2F	47	57	O	17	4F
16	20	0	33	30	48	60	P	20	50
17	21	1	34	31	49	61	Q	21	51
18	22	2	35	32	50	62	R	22	52
19	23	3	36	33	51	63	S	23	53
20	24	4	37	34	52	64	T	24	54
21	25	5	40	35	53	65	U	25	55
22	26	6	41	36	54	66	V	26	56
23	27	7	42	37	55	67	W	27	57
24	30	8	43	38	56	70	X	30	58
25	31	9	44	39	57	71	Y	31	59
26	32	:	00†	3A	58	72	Z	32	5A
27	33	;	77	3B	59	73	[	61	5B
28	34	<	72	3C	60	74	\	75	5C
29	35	=	54	3D	61	75	]	62	5D
30	36	>	73	3E	62	76	^	76	5E
31	37	?	71	3F	63	77	_	65	5F

†In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

AAM checks user requests to ensure proper processing. If results are not satisfactory, an error condition exists and the following occurs:

- A three-digit octal error code is returned to the error status (ES) field in the FIT.
- For a fatal error, the fatal/nonfatal (FNF) field is set in the FIT.
- The error exit is taken if the user has set the error exit (EX) field in the FIT.

The dayfile control (DFC) field and the error file control (EFC) field in the FIT determine the disposition of error messages and notes/statistics. Depending on the setting of these two fields, error messages and statistics/notes are written to the dayfile and/or the error file ZZZZZEG.

ERROR COMMUNICATION

Regarding errors, AAM and the user communicate through the following FIT fields:

ECT Trivial error count

ERL Trivial error limit

ES Error status

EX Error exit

The ES field is a 9-bit field that is set to an octal value after AAM has attempted error resolution and is ready to return control to the user. When an attempt is made to execute an input/output request after an error, AAM does not clear the ES field. If the request is not legal, AAM increments the ECT field and proceeds with execution. If a subsequent error is detected, the ES field reflects the most recent error. The user is responsible for clearing the ES field when an error exit (EX) is not supplied; the ES field is checked after every macro call.

FIT fields relevant to error processing and their meanings are as follows:

DFC Dayfile control; set by the user to control the listing of error messages on the dayfile.

- DFC=0 Only fatal error messages to the dayfile (default).
- DFC=1 Error messages to the dayfile.
- DFC=2 Statistics/notes to the dayfile.
- DFC=3 Error messages and statistics/notes to the dayfile.

EFC Error file control; set by the user to control the listing of error messages on the error file.

- EFC=0 No error file entries (default).
- EFC=1 Error messages to the error file.
- EFC=2 Statistics/notes to the error file.
- EFC=3 Error messages and statistics/notes to the error file.

ERL Trivial error limit; if not specified, the value is zero, no error account is accumulated, and an indefinite number of trivial errors is permitted; if a value is specified, the job is terminated when the value of the ECT field reaches the value specified for the ERL field.

EX Error exit; an 18-bit field that is interpreted as follows:

- EX=0 No user error routine; control is returned as a normal exit; the ES field is set to an error code. If a fatal error is encountered, the message is output to the dayfile.
- EX≠0 When a fatal or trivial error occurs, control is transferred to EX+1; a jump to the user in-line return address is stored in the EX field and the ES field is set to an error code.

FNF Fatal/nonfatal flag; set to 1 for fatal errors.

ERROR FILE PROCESSING

When the error file control (EFC) field is set to a nonzero value, error messages and/or statistics/notes are written to the error file ZZZZZEG. The error file is always flushed at an abnormal termination. At the completion of a job step, the error file buffer is flushed if all files are closed. The CRMEP control statement can be used to process the error file and control the listing of information from the error file on the output file. The format of the CRMEP control statement is shown in figure B-1.

CRMEP(parameter=option$_1$/option$_2$/ ... /option$_n$, ...)

parameter Mnemonic specifying type of error file processing and listing.

option Selected setting of the specified parameter.

Figure B-1. CRMEP Control Statement Format

The parameters, options, and defaults for the CRMEP control statement are listed in table B-1. The first default listed in the table is set if neither the parameter nor the option is specified. The second default listed is set if the parameter is specified without an option. More than one option can be specified with a parameter; more than one parameter can be specified in one CRMEP control statement. If a parameter is incorrectly specified, the CRMEP control statement ignores the incorrect parameter and those following it.

The capability to dump the contents of the FIT to the error file for subsequent processing is provided by the FITDMP macro. When the FITDMP macro is executed, the FIT is written to the error file ZZZZZEG as note number 1000.

The error file control (EFC) field in the FIT must be set to 2 or 3 to ensure that notes are written to the error file; EFC=0 is forced to 2 and EFC=1 is forced to 3. The CRMEP control statement can then be used to display the FIT on the output file. The format of the FITDMP macro is shown in figure B-2.

```
FITDMP  fit,id

fit     Address of the FIT to be dumped.

id      Address of the FIT display identifier.
```

Figure B-2. FITDMP Macro Format

TABLE B-1. CRMEP CONTROL STATEMENT PARAMETERS

Parameter	Option	First Default	Second Default	Description
LO	N		X	Select notes.
	-N	X		Omit notes.
	F	X	X	Select fatal error messages.
	-F			Omit fatal error messages.
	D	X	X	Select data manager messages.
	-D			Omit data manager messages.
	T		X	Select trivial error messages.
	-T	X		Omit trivial error messages.
SF	$lfn_1/lfn_2/.../lfn_n$	All	All	Select messages associated with specified files.
OF	$lfn_1/lfn_2/.../lfn_n$	None	None	Omit messages associated with specified files.
SN	$mno_1/mno_2/.../mno_n$	All	Hardware and parity errors	Select only specified message numbers.
ON	$mno_1/mno_2/.../mno_n$	None	Error messages 142 and 143 only	Omit only specified message numbers.
L	lfn	OUTPUT	LIST	Specify output file name.
RU	blank			Return unload of error file performed at end of processing. Error file position at EOI at end of processing.
	0	X		
PW	pw	72 (connected file) 132 (unconnected file)	72 (connected file) 132 (unconnected file)	Specify page width for CRMEP output file (range can be 40-160 characters).

The id parameter is an optional parameter that is used to display an identifier for the FIT dump. The FIT display identifier at the location specified by the id parameter consists of 10 characters of displayable information.

ERROR CONDITION PROCESSING

When an error condition is encountered, the error status (ES) field is set to the appropriate error number. For a trivial error, the trivial error limit (ERL) field set to zero allows unlimited trivial errors. If the ERL field is greater than zero, the trivial error count (ECT) field is incremented and compared with the ERL field as follows:

- If the ECT field is less than the ERL field, control is passed to the error exit if specified or to the user's in-line code. If control passes to the in-line code, the user is responsible for checking the error status.
- If the ECT field is equal to the ERL field, the ES field is set to 356 (trivial error limit reached) and the fatal/nonfatal (FNF) field is set. Control is returned to the error exit if specified or to the user's in-line code.

When a file is accessed sequentially and end-of-information is encountered, the file position (FP) field is set to indicate EOI and an informative message is issued. If the end-of-data exit (DX) field has been set, the exit is taken. If another access beyond end-of-information is attempted without repositioning the file, a fatal error status is given for an indexed sequential file and a trivial error status is given for direct access and actual key files. If the error exit (EX) field is set, that exit is taken. If the FNF field is set and any AAM function is attempted on the file, a 115 error is generated and the job is aborted.

CLASSES OF ERRORS

Syntax errors are diagnosed by AAM; the messages are self-explanatory. System errors are detected by the operating system. Execution errors, occurring during execution of input and output requests, are subdivided into call errors and invalid input/output requests.

CALL ERRORS

Call errors are undetectable parameter errors. For example:

```
GET     X1
```

If register X1 does not contain the valid FIT address, an unpredictable AAM error, mode error, or D00 error can result.

INVALID INPUT/OUTPUT REQUESTS

Requests for illegal input/output operations produce the following general types of errors:

FIT	Content of address given as the FIT address does not pass a test for plausibility. It does not contain a legal logical file name in bits 59 through 18, or the FIT has inconsistencies.
File organization	Input/output requests or specifications illegal on the type of file specified by the file organization (FO) field in the FIT.
Record type	Input/output requests illegal for the record type specified by the record type (RT) field in the FIT.
OPENM/CLOSEM	Input/output requests illegal for files opened or closed as specified by the open/close (OC) field and/or the old/new file (ON) field in the FIT.
Processing direction	Input/output requests that would violate the processing direction limitations specified by the processing direction (PD) field in the FIT.
File position	Input/output requests illegal for the file position given by the file position (FP) field in the FIT.
Last operation	Input/output requests illegal in the context of the last operation.
Key	Attempts to access or write records whose keys are not within the range of keys defined for a file.
Data	Errors in data specification, such as inconsistency between the amount of data requested and the amount actually present, illegal field present in the data, required field is absent, or parity error.
Device	Input/output requests illegal on the device upon which the file resides.

All errors are either fatal or nonfatal. Some nonfatal errors are trivial in that no user action is required. Fatal errors usually indicate incorrect parameter specification and incomplete or contradictory information provided by the user as program errors. A fatal error message is always printed on the dayfile.

Trivial errors are usually data errors, such as attempting to insert a record already in the file or to replace or delete a record that does not exist. If a trivial error message is printed, the key and type of error are part of the error message. The record associated with the trivial error is dropped; however, the file position might be altered.

If the error exit (EX) field in the FIT has been set to the address of an error routine, any error causes a transfer of control to the address in EX+1 for a recovery routine after the error has been resolved. Fatal errors inhibit any further attempts to perform input/output on the file using AAM; such attempts cause the job to terminate. If the EX field is not set, an error sets the error status (ES) field and returns control to the calling program. The user should clear the ES field after an error is processed.

AAM is in the user's field length and is subject to destruction by the user.

DIAGNOSTICS

Error messages that can be output by AAM are listed in table B-2. The messages are in order by error code. The table contains the following information:

Code — Octal value corresponding to the error condition.

Message — Diagnostic output; varies depending on the setting of the DFC and EFC fields and the parameters specified in the CRMEP control statement.

Significance — Meaning of the message.

Action — Suggestion for the user to recover from the error condition.

Severity — Type of error; can be any of the following:

- F — Fatal
- T — Trivial
- T/F — Trivial under some conditions, fatal under other conditions

Table B-3 is a list of notes and informative messages that can be output.

TABLE B-2. DIAGNOSTICS

Code	Message	Significance	Action	Severity
001	INVALID FO	File organization must be indexed sequential (IS), direct access (DA), or actual key (AK).	Correct the file organization field.	F
002	FIT/FILE ORGANIZATION MISMATCH	The file organization specified does not match any opened files.	Check to see that the correct file is being processed or that the FO field is specified correctly.	F
006	FIRST BLOCK IS NOT A FSTT	The first block in the file must be the file statistics table (FSTT). For an indexed sequential file, the ORG field must be set for the correct file organization. Possibly a mismatch exists between the currently supported file organization (formerly extended) and an old one (initial).	If a file is being created, check that the pd parameter is specified in the OPENM macro or the ON field is set to NEW. If the problem persists and a mismatch between new and old file organizations does not exist, follow site-defined procedures for reporting software errors or operational problems.	F
030	INVALID RT	Record type must be W, S, Z, F, R, T, D, or U; it must conform to other file specifications, such as FO.	Correct the record type field.	T
031	RT=F/Z AND FL=0	For fixed length F or zero byte terminated Z type records, a maximum record length must be specified for the FL field in the FIT.	Specify the FL field.	T
032	RT=T AND HL OR TL=0	For T type records, the header length (HL) must be large enough to hold the trailer count field defined by the CP and CL fields. The length of the trailer count field must be given in the TL field and must be at least one character long.	Correct the header length or the trailer length field.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
033	RT=D AND LL=0/RT=T AND CL=0	For D type records, the LL field in the FIT must provide the length of the record field that specifies record length.	Specify the length of the D type record length field.	T
		For T type records, the CL field in the FIT must provide the length of the field that specifies the number of trailer items.	Specify the length of the trailer count field of the T type record.	
035	RT=T/D, MRL EXCLUDES CONTROL FIELD	For T and D type records, the record must contain a field identifying record length.	Check that for D type records LP+LL is less than MRL. For T type records, CP+CL must be less than MRL. The position count for LP and CP begins with 0.	T
036	RL INCONSISTENT WITH RECORD DESCRIPTION	For T type records, the fixed header length (HL) must include a field CL characters long, beginning at CP, to identify trailer item count. CL and CP for T type records and LL and LP for D type records must be contained within the value specified by MNR.	For T type records, check that the count field is within HL. For D type records, check that the length field is within MNR. The current record is ignored. Positions CP and LP are counted from 0.	T
037	RT=D/T AND CL/LL > 6	For D and T type records, the length of the count field must be one to six character positions.	Correct the length of the count field.	T
040	REDUNDANT OPEN	A file must be closed before open processing, such as buffer allocation or FILE control statement processing, takes place. A redundant open call is ignored.	Correct the program to close the file before open processing.	T
050	NUMBER OF FILES PERMITTED TO BE OPEN SIMULTANEOUSLY HAS BEEN EXCEEDED	The installation defines the number of AAM files that can be open at one time because buffers are limited by central memory available. Default release value is 10 files of each organization.	Check with a local analyst for the limit on the number of files that can be open at one time.	F
051	SETFIT DISALLOWED ON OPEN FILE	Open processing would have already processed the FILE control statement. The SETFIT macro processes FILE control statements without full open processing.	Change the placement of the SETFIT macro.	T
052	FILE NOT CLOSED AFTER LAST UPDATE/CONDITION QUESTIONABLE	The possibility exists that the file has internal errors. The most likely cause is a system crash that prevented closing of the file.	Rerun the program that updated the file.	T
053	NO HOME RECORD	The OLD parameter has been specified when opening an empty direct access file.	Check that the correct file name has been specified, or change the OLD parameter to NEW.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
054	FILE ILLEGALLY EXTENDED (EOI MOVED)	An existing file has been opened without extend permission and information has been written beyond the old EOI.	Change the program to open with extend permission.	F
055	FILE NONEXISTENT, CANNOT OPEN-OLD	The logical file name specified does not match any existing file or the index file is not attached for a MIP file.	Check that the logical file name is correctly specified.	F
060	REDUNDANT CLOSE	Either a second call to CLOSEM was issued, or an attempt was made to close an unopened file. The operations requested by the CF field are performed before the error is issued.	Correct the program to eliminate the redundant close operation.	T
070	OUTPUT REQUEST, PD=INPUT OR READ ONLY PERMISSION	A file opened with read only permission or with PD set to INPUT cannot be written. The write statement is ignored.	If the file is to be written, set the PD field in the FIT to OUTPUT or IO before opening the file and check the file permissions.	T
071	INPUT REQUEST, PD=OUTPUT	A file opened with PD set to OUTPUT cannot be read. The read statement is ignored.	If the file is to be read, set the PD field in the FIT to INPUT or IO before opening the file.	T
074	MUST HAVE CMM FOR MULTIPLE ACCESS	To have multiple FITs for one file, CMM must be used. The file is not opened.	Correct the program to allow CMM to be loaded.	T
075	UBS MAY NOT BE USED FOR MULTIPLE ACCESS	A file that is to be accessed by more than one FIT cannot have user-supplied buffer space for any of the FITs.	Correct the program to eliminate the user-supplied buffer.	T
100	CANNOT SEQUENTIALLY POSITION BEYOND FILE BOUNDS	A sequential read or SKIPFL is not possible with the file at EOI. A SKIPBL is not possible with the file at BOI.	The file must be repositioned if further access is desired. Repeated access attempts with file at the end cause the fatal error flag to be set.	F
110	FILE NOT OPEN	A file must be opened before it can be read or written. Omission of required FIT field parameters or inconsistencies in specified parameters inhibit open.	Correct the program to open the file before reading or writing, or correct omissions or inconsistencies in FIT fields.	T
115	OUTSTANDING FATAL ERROR ON THE FILE	A fatal error prevents future access to the file with the error, but it does not cause job termination unless the user attempts further operations on the file.	Correct and rerun.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
117	PUT OR REPLACE OF LARGER RECORD ILLEGAL AFTER GETN	Sequential read of a direct access file is possible only if the existing records are not disturbed. Writing any new record or increasing existing record size prevents subsequent sequential access. In a direct access file, replacing an existing record with a smaller one also prevents subsequent sequential access.	Correct the program.	T
130	RT=W, BAD CONTROL WORD, FILE DEFECTIVE OR MISPOSITIONED	Record type was specified as W. This message indicates the records being read are not, in fact, W type records.	Check that the existing file is correctly described, formatted, and positioned.	T/F
135	RMS READ PARITY ERROR	The operating system returned a parity error status after a read, or block length is incorrect.	Recreate the file on a good device.	T/F
136	RMS WRITE PARITY ERROR	The operating system returned a parity error status after a write.	Recreate the file on a good device.	F
142	EXCESS DATA	In a write, no information is written to the file; the user has supplied RL greater than FL/MRL or the record mark character for an R type record was not found before MRL characters. On a read, no information is transferred to the working storage area; the record length exceeds the FL/MRL defined. For GET macro processing, the following conditions cause an error: Z No zero byte found before FL characters R No record mark found before MRL T,D Control field RL > MRL U RL > MRL F Excess data cannot occur	Correct the inconsistency between the RL and FL or MRL fields.	T
143	INSUFFICIENT DATA	Control information in the record being read (length calculated by fields such as CP and CL) specifies a length for each record. The record existing in the file is smaller than the specified length. All characters available are returned.	No action is required.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
146	USER HEADER LENGTH ERROR	The attempted PUT or REPLACE macro is rejected because the user header length is inconsistent with the record length.	Check the user header length and the record length for inconsistencies.	T
147	CHECKSUM ERROR IN DATA OR INDEX BLOCK	There is a conflict between the loading checksum and computed checksum in either the data block or index block.	Follow site-defined procedures for reporting software errors or operational problems.	F
150	FILE NOT ON RMS	Indexed sequential, direct access, and actual key files must be created on a disk, drum, or family pack.	Correct the control statement to ensure a valid device assignment.	T/F
165	ILLEGAL FILE NAME	The LFN does not consist of one to seven letters and digits, the first being a letter.	Correct the LFN or the FIT address.	F
166	FIT INCOMPLETE - CANNOT CREATE FILE	A required parameter is missing, or information for the FIT field is not specified correctly.	Refer to section 4 of this manual for parameters required during file creation.	F
167	RECORD LENGTH OUTSIDE MIN-MAX RANGE -- REQUEST IGNORED	Minimum and maximum record length, MNR and MRL, establish the absolute record limit for the life of the file.	Correct the program to write records within the established limit, or recreate the file changing MNR and MRL.	T
		For D or T type records, the control field specified is outside the value specified by the RL field, or it is not within the values specified by the MNR and MRL fields.	Check to see that the CL/CP fields or the LL/LP fields are specified correctly.	
170	RECORD SIZE EXCEEDS BLOCK SIZE OR IS NEGATIVE	All data blocks or home blocks must hold at least one record plus control information.	Correct the RL or MBL field.	T/F
171	INCORRECT HASHING ROUTINE	The hashing routine used to create a direct access file must be used for all subsequent access.	Check that the correct routine is available to the job or that the HRL field has not been changed. The routine name can be different each time, but the results produced cannot differ.	F
172	ERRONEOUS KL OR RKP FIELD SPECIFIED	The key length (KL) or relative key position (RKP) field is not specified properly for the key type.	Correct the KL or RKP field.	F
174	FIT INCOMPLETE FOR BFS CALCULATION	Record length range MRL and MNR, blocking factor RB, or other key characteristics required for buffer size calculation have been omitted.	Refer to section 3 of this manual for parameters required for BFS calculation.	T
175	REQUESTED DATA OR INDEX BUFFER TOO LARGE	The data block or index block size cannot exceed 131071.	Correct the data or index block size.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
176	MAXRECSZ IN FSTT EXCEEDS MRL IN FIT, WSA MAY BE TOO SHORT	The MRL field in the FIT is less than the maximum record size recorded in the FSTT.	Correct the inconsistency between the current MRL value and the MRL value used when the file was created.	T
200	BAD FSTT LINKED TO FIT	The FSTT field in the FIT does not point to a valid FSTT when the file is being closed.	Correct the program to avoid destroying the FSTT field.	T
201	FILE CONTAINS BAD BLOCKS	Some data blocks in the file have checksum or parity errors. Updating is not allowed.	The file should be recreated as soon as possible.	T
202	FILE IS RUINED	The file structure has been destroyed. The file is no longer usable.	The file must be recreated.	F
203	MIP FUNCTION ATTEMPTED WITHOUT MIP FILE	The file is a multiple-index file, but the index file is not present.	Close the data file. Set the XN field in the FIT to the index file name and reopen. Attach the index file.	T
204	KEY POSITION OUT OF RANGE	The starting character position of a key is defined by positions 0 through 9, counting from the left of a word.	Correct the KP field.	F
205	MINIMUM RECORD SIZE OUT OF RANGE	Minimum record length (MNR) must be at least one character but no more than maximum record length (MRL) and must contain the key.	Correct the MNR field.	F
206	KEY NOT CONTAINED WITHIN RECORD	The embedded key must be within the record.	Check for proper RKW, RKP, KL, MNR, and MRL. Minimum and maximum record lengths (MNR and MRL) are in characters; relative key word (RKW) is in words, starting from 0; relative key position (RKP) is in 6-bit field, 0 through 9, counting from 0 on the left.	F
207	MINIMUM RECORD SIZE EXCEEDS MAXIMUM	Required parameter MRL must be equal to or larger than MNR.	Correct the inconsistency between the MRL and MNR fields.	F
215	OUTSTANDING FATAL ERROR IN ALTERNATE FIT	In a concurrent environment, a fatal error has been detected in an alternate FIT.	Rerun the program when the original problem has been corrected.	F
223	CHECKSUM ERROR IN FSTT	A conflict exists between the loading checksum and the computed checksum in the FSTT.	Follow site-defined procedures for reporting software errors or operational problems.	F
245	FUNCTION NOT VALID FOR THIS FO	The function attempted is not valid for the file organization indicated in the FIT.	Correct the program.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
250	FILE RMS LIMIT EXCEEDED (AK)	The user has exceeded the mass storage limit as specified in the LIMIT control statement or installation-defined limit.	Correct the problem and rerun.	F
252	SYSTEM RMS LIMIT REACHED	No more mass storage was available for the file.	Consult a system analyst; perhaps the installation parameter limit was exceeded.	T/F
253	FILE LIMIT REACHED - RECORD NOT INSERTED	The number of records currently in the file cannot exceed the limit that the user specified with FLM.	Recreate the file increasing the value of FLM.	T
300	NO READ PERMISSION	To be read, a permanent file must be attached with read permission.	Attach the file with the required read permission.	F
301	NO WRITE OR MODIFY PERMISSION	A permanent file requires proper required access permissions. Modify permission is required for any updating operation.	Attach the file with the modify permission.	F
302	NO EXTEND OR ALLOCATE PERMISSION	A permanent file requires extend permission before new records can be inserted.	Attach the file with the required extend permission.	F
304	NOT ALLOWED TO CREATE OVERFLOW BLOCKS (DA)	The OVF option selected requires original home blocks to accommodate all records. New records are ignored because all home blocks are full.	Change the OVF option if overflow blocks can exist.	T
321	DATA STRUCTURES MUST BE WORD ALIGNED	If WSA or KA is provided as a CHARACTER type data structure, it must be word aligned within that structure.	Word align WSA and/or KA.	F
324	PROCESSING DIRECTION NOT CONSISTENT WITH REQUEST	A file opened for INPUT cannot be written; a file opened for OUTPUT cannot be read.	Correct the inconsistency between the PD field and the input/output operation.	F
333	ILLEGAL CALL TO DIAGNOSTIC ROUTINE	An unexpected jump to a diagnostic routine has occurred.	Follow site-defined procedures for reporting software errors or operational problems.	F
335	HIERARCHY TABLE OVERFLOW	Index level has increased too rapidly for AAM; update operation has not been performed.	For indexed sequential files, close and reopen the file. For other files, rerun the program starting with the update transaction that caused the overflow.	T/F
336	BAD FIT ADDRESS	The user or the system has destroyed system tables.	Correct the program and reload it.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
345	INSUFFICIENT CMM SPACE AVAILABLE	Not enough CMM space exists to open the file. To open a file requires enough free CMM space to load any rare capsules required, and to allow two of the largest blocks to be in memory at the same time. The file is not opened.	Release some CMM, if any is being used by the user program, or increase the amount of memory available to the job.	T
346	CMM NOT AVAILABLE AND THERE IS NO LIST OF FILES ADDRESS	A new block for the list-of-files cannot be allocated, and the LOF$RM entry point has been cleared.	Correct the program so that the pointer is not destroyed. A default list with 65_8 entries is supplied.	F
347	FDL ERROR CODE n ON CAPSULE axxxxxx	Either CMM is not loaded when FDL is called to load a capsule or the AAMLIB file is not valid.	Check the load sequence or map to see if CMM is loaded. Fix the static load calls to load the proper routines. If using local libraries, check for a valid AAMLIB file.	T
352	FILE TO BE CLOSED IS NOT KNOWN	The logical file name specified does not match any existing file.	Check that the logical file name is correctly specified.	T
354	BUFFER SPACE SUPPLIED IS INSUFFICIENT FOR I/O	A buffer specified by the BFS field is not large enough to hold at least the larger of one block specified by MBL+2 or one physical record unit for the file's resident device. A record written on a connected file on NOS/BE is larger than the current buffer.	Increase the BFS value.	T/F
355	CODE MODULES REQUIRED FOR I/O NOT LOADED	Routines necessary for processing have not been loaded.	Refer to appendix E for the correct static loading procedures.	T
356	TRIVIAL ERROR LIMIT REACHED	Error count ECT equals the user-defined error limit ERL, resulting in a fatal error.	Correct the errors.	F
357	UNABLE TO OBTAIN SPACE FOR BUFFER	Required space cannot be allocated. CMM is not available and the FWB field is zero.	Supply a value for the FWB field or delete the OMIT=CMM parameter.	F
370	FATAL I/O ERROR	Either a block with an incorrect length was encountered or the operating system detected an error in the file or in the way the file was being used.	Correct the program.	F
372	FO=IS INDEX STRUCTURE FULL 15 LEVELS	The indexed sequential file has filled 15 levels of indexing, which is the maximum allowed. Further updating is not permitted.	Reorganize the file to allow more indexes per block.	F
403	SKIPBL DISALLOWED	A backward skip is not possible for D, R, and T type records.	Correct the program.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
404	SKIPFL DISALLOWED FOR RT=U	No forward record skip is possible for U type records.	Correct the program.	T
415	ONLY PUT ALLOWED DURING INITIAL CREATION	During file creation, only PUT macros are valid between open and close.	Correct the program to eliminate all macros except PUT.	T
417	CANNOT REPLACE WITH LARGER RECORD IN SEQUENTIAL MODE	The REPLACE statement is ignored.	Correct the program.	T
421	WSA NOT SPECIFIED - REQUEST IGNORED	For read or write, the location of the record in the user field length is required.	Specify the WSA field for the read or write operation.	T
422	SEEK NOT ALLOWED IN SEQUENTIAL MODE	The SEEK macro is ignored because it is not allowed during sequential processing.	Close and reopen the file for random processing if SEEK is desired.	T
424	CANNOT GET IN SEQUENTIAL MODE - GETN ASSUMED	The GET macro cannot be used in sequential mode.	Use the GETN macro.	T
425	CANNOT SKIP BACKWARD IN SEQUENTIAL MODE	The SKIP macro is ignored because backward skips are not allowed in sequential mode.	Correct the program.	T
426	GETN NOT ALLOWED DURING FILE CREATION - REQUEST IGNORED	On a file creation run, only the PUT macro is allowed between open and close.	Correct the program to eliminate all macros except PUT.	T
427	GET, SEEK INVALID IN SEQ MODE	Opening an indexed sequential file for INPUT establishes a sequential mode of operation in which access by key is prohibited.	Open the file for input/output if GET and SEEK are desired.	T
441	MAJOR KEY WITH SYMBOLIC KEYS ONLY	Key type (KT) must be S for major key actions.	Correct the KT field.	F
442	INVALID ACTUAL KEY - REQUEST IGNORED	The key is not valid; the request is ignored.	Correct the KA field.	T
444	NEW KEY LESS THAN PREVIOUS KEY IN INITIAL CREATION	Records should be sorted by ascending key before an indexed sequential file is created. An out-of-order key is ignored.	Sort the records into ascending sequence.	T
445	KEY NOT FOUND - FILE POSITION MAY BE ALTERED - REQUEST IGNORED	The key does not exist in the file. Position is changed for indexed sequential, actual key, or MIP only if the operation was GET or START. File position is unchanged for direct access files.	No action is required.	T
446	DUPLICATE KEY FOUND - FILE POSITION ALTERED - REQUEST IGNORED	A duplicate key has been found. The request is ignored.	Change the duplicate key indicator if duplicate keys are allowed, or check the key field of the current record.	T

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
447	KEY ADDRESS NOT SPECIFIED - REQUEST IGNORED	The file cannot be read randomly if a key is not given.	Correct the program to specify the key address (KA) field.	T
452	FILE POSITIONING ERROR	An attempt was made to position the file beyond EOI.	Correct the program to check the FP field or specify the DX field.	F
501	INDEX FILE NOT COMPATIBLE WITH CRM FILE	Information in the file statistics table for a multiple-index file does not agree with index file information.	Check that the proper index file has been specified.	T
502	SPECIFIED KEY NOT DEFINED	The key position specified by the RKW, RKP, and KL fields for an alternate key does not correspond to an alternate key definition in the index file.	Correct the RKW, RKP, or KL field.	T
503	DUPLICATE ALTERNATE KEY ERROR	All alternate key values must be unique if the index structure for a multiple-index file has been specified as unique.	Specify indexed sequential structure if more than one alternate key is to have the same value.	T
504	SEQUENTIAL OPERATION BEYOND EOI ATTEMPTED	End-of-information has been encountered. No further sequential operations, such as GETN or a system search for a key, are possible until the index file is repositioned by a user statement.	Correct the program.	T
505	ERROR IN RMKDEF PARAMETER	The parameters used with the RMKDEF macro have been specified incorrectly.	Check that letters and digits appear properly; also, that the file name given in RMKDEF corresponds to the name of the data file.	F
506	ALTERNATE KEY NOT FOUND	A key value specified does not match any alternate key value in the index file.	Action depends on program processing of keys.	T
507	***AAM MALFUNCTION n ***	For an indexed sequential file, an impossible condition has been encountered. This condition probably occurred when part of the executable code of AAM was altered by an agency other than AAM. The code n specifies the condition that has occurred: n=1 FIAAT POSKEY1 bad n=2 FIAAT POSKEY3 bad - FIFO n=3 Intermediate block reached with all keys too low n=4 Attempt to go up from primary	Follow site-defined procedures for reporting software errors or operational problems.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
		n=5 Error in removing one level of hierarchy n=6 Compression buffer size bad n=7 Running total of CMM too high n=10 Index file not opened n=11 Attempt to use a busy FIX cell n=12 Attempt to chain an already chained block n=13 Attempt to read or write PRU 0 n=14 Attempt to write a block being read n=15 UBS free block count bad n=16 Attempt to unchain block not chained n=17 Empty count less than zero		
511	RMKDEF ONLY AFTER OPEN-NEW - IGNORED	The RMKDEF macro can be used only on a creation run.	Correct the program.	T
512	CRM DATA FILE MODIFICATIONS ILLEGAL WITH NDX=YES	If NDX is set to YES, the PUT, DELETE, and REPLACE macros are not allowed.	Correct the program.	T
515	NO INDEX FILE SPECIFIED	No name has been specified for the XN field on an IXGEN or file creation run for a multiple-index file.	Specify an index file for the XN field.	F
520	CHANGED KEY TYPE	The key type (KT) specified on the file creation run cannot be changed for the life of the file.	Change the KT field.	F
521	CHANGED KEY SIZE	The key length (KL) specified on a file creation run cannot be changed for the life of a file.	Change the KL field.	F
523	NO KEY DEFINED	Key type (KT), key length (KL), and key address (KA) must be defined.	Define the key fields.	F
524	KEY SIZE ILLEGAL	The size of the key is not valid for the file organization or record type.	Correct the KL field.	F
525	MAJOR KEY SIZE ILLEGAL	MKL must be at least 1 and less than the full key defined by KL.	Correct the MKL field.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
526	HASHED KEY OUTSIDE HOME BLOCK AREA	The user has changed hashing routines. The hashing routine in use is limited in the range of keys that it can successfully process.	Check the HRL field in the FIT to verify that the correct hashing routine is in use, otherwise, the user should limit the selection of keys to a narrower range.	F
527	ATTEMPT TO REDEFINE SPARSE CONTROL CHARACTER	An RMKDEF directive attempted to redefine the sparse control character.	Correct the RMKDEF directive.	F
530	PADDING FACTOR OUT OF RANGE	Padding can be specified as 0 to 99 percent.	Correct the padding percentage.	F
532	FILE ALREADY EXISTS, CANNOT OPEN-NEW	Two files in one program cannot have the same name. This can occur during an attempt to open a data file that has a disassociated index file and the index file is still present.	Check the PD field in the FIT or the pd OPENM parameter. The ON field must be changed from NEW for file access after creation run. If a disassociated index file is present when a data file is opened, reassociate the files by MIPDIS; or return the index file; or set XN to 0 in the FIT.	F
534	MRL EXCEEDS MAX ALLOWED RECORD SIZE	The value of the MRL field is greater than 81870 characters. The file is not opened.	Correct the MRL field in the FIT.	T
535	NO DECOMPRESSION ROUTINE SUPPLIED	For files that have user-supplied compression, no DCA value was specified on OPENM OLD. For files that have system-supplied compression, the decompression routine in a user or system library was not made available to job steps referencing the file. The file is not opened.	Correct the DCA field in the FIT or make the routine available.	F
536	NO OR WRONG COMPRESSION ROUTINE SUPPLIED	For files that have user-supplied compression, either no CPA value was specified on OPENM OLD or the CPA value identified a different compression routine from the one specified when the file was created. For files that have system-supplied compression, the CPA value specified a routine in a user or system library that was not made available to job steps referencing the file. The file is not opened.	Correct the CPA field in the FIT or make the routine available.	F
540	FIFO KEY SUBSTRUCTURE NOT ALLOWED IN REPEATING GROUPS	For alternate keys in repeating groups, the key must be unique or stored in an indexed sequential substructure.	Correct the RMKDEF directive.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
541	PURGE ILLEGAL - SPECIFIED ALT KEY NOT KNOWN	An attempt was made through MIPGEN to purge an alternate key that did not exist.	Correct the MIPGEN RMKDEF directive.	F
542	NEW KEYDEF MATCHES ONE ALREADY KNOWN - KEYDEF REJECTED	The key was not defined to be unique.	Correct the MIPGEN RMKDEF directive.	F
544	PADDING REQUESTED TOO LARGE	The block size and padding percentage requested would not allow the data block to contain one maximum record on create or would not allow three index records per index block. The file is not opened.	Correct the DP or IP field in the FIT and reopen the file.	F
545	CANT OPEN NEW FOR INPUT	The processing direction must be set to OUTPUT on a file opened as a new file.	Correct the PD or ON field in the FIT and reopen the file.	T
546	PRIMARY KEY NOT FOUND	A primary key in the alternate key index file cannot be found in the data file. The data file and index file have been modified inconsistently.	Disassociate the data file and create a new index file using the MIPGEN utility.	F

TABLE B-2. DIAGNOSTICS (Contd)

Code	Message	Significance	Action	Severity
547	BAD STRUCTURE FOUND IN FILE	The block being looked at contains an impossible counter or pointer.	Follow site-defined procedures for reporting software errors or operational problems.	F
550	CANNOT COMPRESS - KEY POSITION INVALID	To compress records, the primary key must either be non-embedded or begin in the first character position. The file is not opened.	Change the key position if the file is to be compressed.	T
551	REL MUST BE EQ, GT OR GE	An invalid REL value was detected. The operation is not performed.	Set the REL field to a correct value.	T
552	NO DATA FILE PROCESSING PERMITTED - FILE OPENED FOR INDEX ONLY	The OPENM macro was issued with the NDX field set to 1 (index file processing only).	For data file proccessing, close the file, set the NDX field to 0, and then reopen the file.	F
553	OVERFLOW RECORD NOT FOUND - FILE BAD	A record written to the file is not accessible.	Follow site-defined procedures for reporting software errors or operational problems.	F
554	PROCESSING REQUIRES READ, EXTEND, AND MODIFY PERMISSIONS	Incorrect permanent file usage.	Attach file with correct permissions.	T
555	INDEX FILE NON-EXISTENT	On OPENM OLD, a data file is found to have an associated MIP file, but the MIP file is not present.	Attach the MIP file.	F
556	OPEN FAILURE	System CIO OPEN request failed.	Check the CODE and STATUS field in the FIT.	F
712	NEGATIVE OR OVERSIZED ARGUMENT--WSA, SKP, OR LA	One of the parameters indicated was erroneously specified when a macro was issued.	Correct the program.	F
713	NEGATIVE OR OVERSIZED ARGUMENT--RL, ST, OR LBL	One of the parameters indicated was erroneously specified when a macro was issued.	Correct the program.	F
714	NEGATIVE EX OR DX PARAMETER	A negative value was specified for the EX or DX field.	Correct the program.	F
715	NEGATIVE OR OVERSIZED ARGUMENT--WA OR KA	Either the WA or KA field was erroneously specified.	Correct the program.	F
716	NEGATIVE OR OVERSIZED ARGUMENT--PTL OR KP	Either the PTL or the KP field was erroneously specified.	Correct the program.	F
717	NEGATIVE OR OVERSIZED ARGUMENT--MKL, POS, GPS, OR TRM.	One of the parameters indicated was erroneously specified when a macro was issued.	Correct the program.	F
720	DEVICE CAPACITY EXCEEDED	The CIO read driver has encountered an error.	Check the job dayfile for the specific head driver error.	T
721	ERROR DETECTED BY OPERATING SYSTEM	A system and/or hardware error that cannot be corrected has been encountered.	Check the job dayfile for a system and/or hardware error message.	F

TABLE B-3. NOTES OR INFORMATIVE MESSAGES

Code	Message	Code	Message
1000	FIT DUMP	1025	DATA BLOCK SIZE AND BLOCKING FACTOR BOTH SET
1001	FILE OPENED	1026	EOI ENCOUNTERED ON SKIP OR GETN
1002	FILE CLOSED	1027	THE KEY IS .....
1003	NUMBER OF INDEX LEVELS	1030	ERROR ENCOUNTERED DURING .....
1004	***NUMBER OF GETS THIS OPEN	1031	One of many general comments output by AAM routines
1005	***NUMBER OF PUTS THIS OPEN	1032	THE KEY IS ..... THE KEY IN OCTAL IS .....
1006	***NUMBER OF REPLACES THIS OPEN	1033	***NUMBER OF GET NEXTS THIS OPEN .....
1007	***NUMBER OF DELETES THIS OPEN	1034	***NUMBER OF ACCESSES THIS OPEN .....
1010	***TOTAL DISKAREA*** ..... WORDS	1035	***TOTAL NUMBER OF RECORDS .....
1011	GETN REACHED EOI	1036	***TOTAL NO. OF OVERFLOW RECORDS .....
1012	SKIP REACHED FILE BOUNDARY BEFORE EXHAUSTING SKIP COUNT	1037	***NO. OF AVAILABLE PRIMARY INDEX ENTRIES .....
1013	END OF INFORMATION ENCOUNTERED	1040	***RECORDS/HOME-BLK CREATED THIS OPEN .....
1014	BEGINNING OF INFORMATION ENCOUNTERED	1041	***RECORDS/OVF-BLK CREATED THIS OPEN .....
1015	FILE LIMIT REACHED, LINEAR SEARCH FOR SPACE INITIATED	1042	***OVERFLOW BLOCKS CREATED THIS OPEN .....
1016	ILLOGICAL SUCESSIVE SEEK REQUESTS	1043	***TOTAL NUMBER OF HOME BLOCKS .....
1017	CANNOT CHECKSUM A FILE CREATED WITHOUT CHECKSUMS	1044	***TOTAL NUMBER OF HOME BLOCKS IN USE .....
1020	ILLOGICAL TO CHANGE THE KEY BEFORE SEEK FUNCTION COMPLETED	1046	PADDING NO LONGER HONORED
1021	HOME BLOCKS EMPTY--HASHING ROUTINE NOT VERIFIED	1047	SERIAL PASS OF FILE FOR SPACE
1022	DELETED LAST RECORD	1137	THE FOLLOWING BLOCK CONTAINS A PARITY ERROR .....
1023	EMPTY FILE OPENED		
1024	IS ERROR RECOVERY		

GLOSSARY C

AAM (Advanced Access Methods) -
A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor.

Actual Key -
The primary key for a record in a file with actual key organization, which indicates the storage location of the record.

Actual Key (AK) File -
A mass storage file in which each record is stored at the location indicated by the primary key. For actual key files, the primary key is a record number that AAM converts to the storage location of the record. Access is random or sequential.

Alternate Key -
A key other than the primary key by which an indexed sequential, direct access, or actual key file can be accessed.

BAM (Basic Access Methods) -
A file manager that processes sequential and word addressable file organizations.

Beginning-Of-Information (BOI) -
The start of the first user record in a file.

Block -
A logical or physical grouping of records to make more efficient use of hardware. All files are blocked. See also Data Block, Home Block, Index Block, and Overflow Block.

Block Checksum -
A number used to check that the contents of a data block have not been altered accidentally; a means of ensuring data integrity. Block checksums can be requested for files through use of the BCK parameter in the FILE control statement or FILE macro.

Character -
A letter, digit, punctuation mark, or mathematical symbol forming part of one or more of the standard character sets. Also, a unit of measure used to specify block length, record length, and so forth.

Circular Buffer -
A temporary central memory storage area that contains data during input/output operations. Routines that process I/O treat the first word of the buffer area as contiguous to the last word of the buffer area.

Close -
A set of terminating operations performed on a file when input and output operations are complete. All files processed by AAM must be closed.

Combined Input/Output (CIO) -
An operating system routine that performs input and output.

Compression -
The process of condensing a record to reduce the amount of storage space required. The user can supply a compression routine or use a system-supplied routine. See Decompression.

Concurrency -
Simultaneous access to the same data in a data base by two or more applications programs during a given span of time.

Creation Run -
All processing of a file, from open to close, the first time the file is written or made into an AAM file. Files must be created in a separate creation run during which only write operations on the file being created are allowed.

CRM (CYBER Record Manager) -
A generic term relating to the common products BAM and AAM.

Data Block -
A block in which user records are stored in an indexed sequential or actual key file. Data block structure is defined by the user, or AAM defaults are accepted. Contrast with Index Block for indexed sequential files.

Decompression -
The process of expanding a compressed record to restore it to its original size. The user can supply a decompression routine or use a system-supplied routine. See Compression.

Decryption -
The process of condensing and reformatting an encrypted record to restore it to its original size and format. The user supplies a decryption routine. See Encryption.

Default -
A value assumed in the absence of a user-specified value declaration for the parameter involved. Values for many defaults are defined by the installation.

Direct Access (DA) File -
A file containing records stored randomly in home blocks according to the hashed value of the primary key in each record. Files must be mass storage resident. All allocation for home blocks occurs when the file is opened on its creation run. Access is random or sequential.

Directives -
The instructions that supplement processing defined by a control statement or by a program call for execution of a utility function or member of a product set. Directives do not appear in the control statement record; they are usually in a separate record of the file INPUT or a file referenced in a control statement call. Directives are required for execution of FORM, the CREATE utility, and EDITLIB among others.

Embedded Key -
A primary key that is contained within the record.

Encryption -
The process of expanding and reformatting a record. The user supplies an encryption routine. See Decryption.

End-Of-Information (EOI) -
The end of the last user record in a file.

Extended Memory -
Any extension to central memory.

Field -
A portion of a word or record; a subdivision of information within a record; also, a generic entry in a file information table identified by a mnemonic.

Field Length -
The area in central memory allocated to a particular job; the only part of central memory that a job can directly access. Contrasts with mass storage space allocated for a job and on which user's files reside.

File -
A logically related set of information; the largest collection of information that can be addressed by a file name. It starts at beginning-of-information and ends at end-of-information. Every file in use by a job must have a logical file name.

FILE Control Statement -
A control statement that supplies file information table values after a source language program is compiled or assembled but before the program is executed. In applications such as those with a control statement call to the FORM utility, a FILE control statement must be used. Basic file characteristics such as organization, record type, and description can be specified in the FILE control statement.

File Information Table (FIT) -
A table through which a user program communicates with AAM. For direct processing through AAM, a user must initiate establishment of this table. All file processing executes on the basis of information in this table. The user can set FIT fields directly or use parameters in a file access call that sets the fields indirectly. Some product set members set the fields automatically for the user.

File Statistics Table (FSTT) -
A table generated and maintained by AAM to collect statistics about each file. The FSTT is a permanent part of a file and contains information such as organization type, size of blocks, number of current accesses, and so forth.

Flushing -
The method of processing file buffers and updating the file statistics tables as if close operations had been requested without actually closing the files.

Hashing -
The method of using primary keys to search for relative home block addresses of records in a file with direct access storage structure.

Home Block -
A block in a file with direct access storage structure whose relative address is computed by hashing keys. A home block contains synonym records whose keys hash to that relative address. If all the synonym records cannot be accommodated in the home block, an overflow block can be created by the system. A user creating a direct access file must define the number of home blocks with the HMB parameter in the FILE control statement.

Index -
A series of keys and pointers to records associated with the keys.

Index Block -
For an indexed sequential file, a block with ordered keys and pointers to the data blocks and other index blocks, forming a directory of the records within a file.

Indexed Sequential (IS) File -
A file organization in which AAM maintains files in sorted order by use of a user-defined primary key, which need not be within the record. Keys can be integer or symbolic; access is random or sequential. Files contain index blocks and data blocks.

Installation Option -
One of several alternate means of processing that is selected when AAM is installed at a computer installation. Once an option is selected, all subsequent use of AAM is governed by the selection. For all options or limits defined as installation options, the user should consult with a system analyst to determine the valid limits.

Integer Key -
A 60-bit signed binary key used with indexed sequential files. Integer keys are sorted by magnitude. See Symbolic Key.

Job Step -
The execution of a control statement.

Key -
A group of contiguous characters or numbers the user defines to identify a record in an AAM file.

Key Analysis Utility (KYAN) -
A utility program that provides information about hypothetical record distribution for a file with direct access organization. The utility reads the key of each record in the file and determines the home block where the record would reside.

LDSET -
The loader control statement. Various parameters include:

LIB	Make available the named library
USE	Load the routines named
STAT	Static loading requested
OMIT	Inhibit loading of the routines named

Load Set -
A group of loader control statements beginning with a call that causes information to be loaded into central memory and ending with a call for execution of a loaded program. Nonloader statements must not appear in a load set.

Logical File Name -
The name given to a file being used by a job. The name must be unique for the job and must consist of one to seven letters or digits, the first of which must be a letter.

Macro -
A single instruction that when compiled into machine code generates several machine code instructions.

Maintenance Run -
A program or job to update an existing file; technically refers to that part of the job from file open to file close.

Major Key -
The leading characters of a symbolic key in an indexed sequential file.

Mass Storage -
A disk pack that can be accessed randomly. Extended memory is not considered mass storage.

Master File -
A file containing information about a set of entities. All information about a single entity constitutes a record in the file. A master file is normally kept up to date by a maintenance run.

Multiple-Index File -
An indexed sequential, direct access, or actual key file that has alternate keys defined.

Multiple-Index Processor (MIP) -
A processor that allows AAM files to be accessed by alternate keys.

Nonembedded Key -
A primary key that is not physically contained within the record. A nonembedded key appears before the record when stored in a data block.

Open -
A set of preparatory operations performed on a file before input and output can take place; required for all AAM files.

Overflow Block -
A block added to the file by AAM for use when the home blocks in a direct access file are full.

Owncode -
A routine written by the user to process certain conditions. Control passes automatically to user owncode routines defined in the FIT for:

DX	End-of-data condition
EX	Error condition

Padding -
The free space reserved in a file at creation time to accommodate additional records; specified as a percentage figure.

Permanent File -
A file on a mass storage permanent file device that can be retained for longer than a single job. It is protected against accidental destruction by the system and can be protected against unauthorized access.

Physical Record Unit (PRU) -
The smallest unit of information that can be transferred between a peripheral storage device and central memory. The PRU size is permanently fixed for all mass storage devices.

Primary Key -
A key whose value uniquely identifies a record and determines the location of the record in the file. A primary key must be defined when a file is created. Primary keys must be used to update a file. Contrast with Alternate Key.

PRU Device -
A mass storage device in which information has a physical structure governed by physical record units (PRUs).

Random Access -
Access method by which any record in a file can be accessed at any time in any order; applies only to mass storage files. See Sequential Access.

Record -
The largest collection of information passed between AAM and a user program in a single read or write operation. The user defines the structure and characteristics of records within a file by declaring a record format. The beginning and ending points of a record are implicit in each format.

Record Slot Number -
The position of a record within a block in an actual key file; specified by the low-order bits of the primary key.

Release System -
A software system delivered to a customer. In installing a system, the customer, but not an individual applications programmer, can use default values or parameters that differ from the release system.

Rewind -
To position a file at beginning-of-information.

Sequential Access -
A method in which only the record located at the current file position can be accessed. See Random Access.

Sparse Key -
An alternate key that is used infrequently. Only those alternate key values of interest are included in the index file.

Symbolic Key -
A key consisting of 1 to 255 6-bit characters. These keys are sorted according to the sequence indicated by the display code to collating sequence conversion table. Also called collated symbolic key. See Uncollated Symbolic Key.

Synonym Records -
Direct access file records whose primary keys hash to the same home block.

Uncollated Symbolic Key -
A key consisting of 1 to 255 6-bit characters. These keys are sorted by the magnitude of their binary display code values. See Symbolic Key.

Working Storage Area -
An area within the user's field length intended for receipt of data from a file or transmission of data to a file.

FILE INFORMATION TABLE STRUCTURE

D

A file information table (FIT) must be associated with every file that uses AAM. For normal language requirements, compilers generate the FIT automatically; users writing in high level languages need not be concerned with the FIT and its generation. The COMPASS user is responsible for supplying the FIT; the FILE macro is provided to create the FIT. Word and bit designations are illustrated in figure D-1.

The FIT is activated by an OPENM request for the file. After the file is opened, FIT fields can be updated with the FILE control statement or the STORE macro, with information from the processing macros, or by AAM as a result of processing the file. Information in the FIT can be retrieved with the FETCH macro. In figure D-1, the fields enclosed in parentheses can be accessed by the FETCH macro but cannot be changed. If a STORE macro is attempted on these fields, an assembly diagnostic results. Blank fields are reserved for CRM or CDC.

The FIT fields are listed by word and bit position in table D-1. For the convenience of the user, the COMPASS symbols are included with the applicable FIT field values. Generally, any particular file organization or record type requires only a small portion of the total information specified here. The first ten words of the FIT are used by AAM for communicating with the operating system.

For the reader's convenience, the FIT fields are listed alphabetically with their word positions in table D-2.

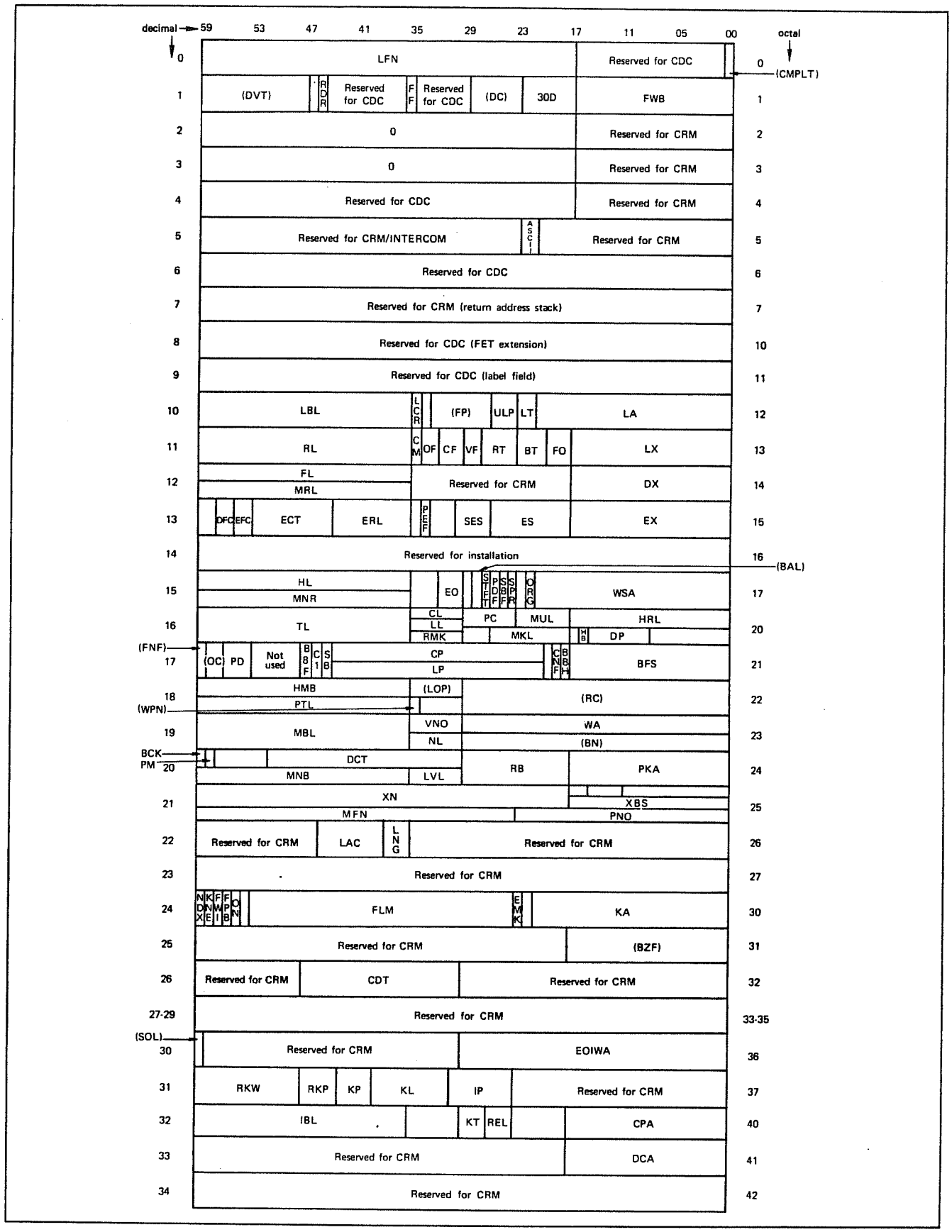


Figure D-1. File Information Table

TABLE D-1. STRUCTURE OF THE FIT

Word	Bits	Fit Field	Description	Contents	COMPASS Symbol
0	59-18	LFN	Logical file name of the data file.		
	17-1		Reserved for CDC.		
	0	CMPLT	FET complete bit; cannot be changed by the user.		
1	59-48	DVT	FET device type; cannot be changed by the user.		
	47		Reserved for CRM.		
	46	RDR	Read release.		
	45-37		Reserved for CDC.		
	36	FF	File flush by operating system on abnormal termination (BAM only).		
	35-30		Reserved for CDC.		
	29-24	DC	Disposition code; cannot be changed by the user. Refer to operating system manual for possible settings.		
	23-18		Length of FIT minus 5; set to 30_{10}.		
	17-0	FWB	First word address of the user buffer.		
2	59-18		Zero-filled field.		
	17-0		Reserved for CRM.		
3	59-18		Zero-filled field.		
	17-0		Reserved for CRM.		
4	59-34		Reserved for CDC.		
	33-0		Reserved for CRM.		
5	59-24		Reserved for CRM/INTERCOM.		
	23-22	ASCII	ASCII character set bits for INTERCOM terminals (BAM only).		
	21-0		Reserved for CRM.		
6			Reserved for CDC.		
7			Reserved for CRM (return address stack).		
8			Reserved for CDC (FET extension).		
9			Reserved for CDC (label fields).		

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents		COMPASS Symbol
10	59-36	LBL	Label area length in characters (BAM only).			
	35	LCR	Label check/creation for input/ output tape (BAM only).			
	34		Reserved for CRM.			
	33-27	FP	File position (in octal); cannot be changed by the user.	0	Mid logical record	
				1	BOI Beginning-of-information	≡BOI≡
				2	BOF Beginning-of-file	≡BOF≡
				10	EOK End-of-keylist	≡EOK≡
				20	EOR End-of-record	≡EOR≡
				100	EOI End-of-information	≡EOI≡
	26-24	ULP	User label processing (BAM only).			
	23-22	LT	Label type (BAM only).			
	21-0	LA	Label area address (BAM only).			
11	59-36	RL	Current record length in characters.			
	35	CM	Conversion mode (EC to IC) (BAM only).			
	34-33	OF	Open flag; positioning of the file at OPENM time.	00	Rewind (default)	≡ ≡
				01	R Rewind	≡R≡
				10	N No rewind	≡N≡
				11	E Extend	≡E≡
	32-30	CF	Close flag; positioning of the file at CLOSEM time.	000	Rewind (default)	≡ ≡
				001	R Rewind	≡R≡
				010	N No rewind	≡N≡
				011	U Unload	≡U≡
				100	RET Return	≡RET≡
				101	DET Detach	≡DET≡
				110	DIS Disconnect (BAM only)	≡DIS≡

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents		COMPASS Symbol
	29-28	VF	End-of-volume flag (BAM only).			
	27-24	RT	Record type.	0000	W Control word	≡WT≡
				0001	F Fixed length	≡FT≡
				0010	R Record mark	≡RT≡
				0011	Z Zero byte	≡ZT≡
				0100	D Decimal character count	≡DT≡
				0101	T Trailer count	≡TT≡
				0111	U Undefined	≡UT≡
				1000	S System-logical-record	≡ST≡
	23-21	BT	Block type (BAM only).			
	20-18	FO	File organization.	000	SQ Sequential (BAM only)	≡SQ≡
				001	WA Word addressable (BAM only)	≡WA≡
				011	IS Indexed sequential	≡IS≡
				101	DA Direct access	≡DA≡
				110	AK Actual key	≡AK≡
	17-0	LX	Label routine exit address (BAM only).			
12	59-36	MRL	Maximum record length in characters; when retrieving primary keys from an alternate key index, working storage area length in characters.			
		FL	Fixed length of an F type record, or full length of a Z type record, in characters.			
	35-18		Reserved for CRM.			
	17-0	DX	End-of-data exit address.			
13	59	NOFCP	No FILE control statement processing.	0	NO FILE control statement processed at SETFIT or OPENM.	≡NO≡
				1	YES FILE control statement not processed.	≡YES≡

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents	COMPASS Symbol
	58		Reserved for CRM.		
	57-56	DFC	Dayfile control for error messages.	0 No dayfile messages except fatal errors 1 Error messages to dayfile 2 Statistics/notes to dayfile 3 Errors and statistics/ notes to dayfile	
	55-54	EFC	Error file control. The FITDMP macro forces EFC=0 to 2 and EFC=1 to 3.	0 No error file messages 1 Error messages to error file 2 Statistics/notes to error file 3 Errors and statistics/ notes to error file	
	53-45	ECT	Trivial error count.		
	44-36	ERL	Trivial error limit.		
	35		Reserved for CRM.		
	34	PEF	Parity error flag (BAM only).		
	33-31		Reserved for CRM.		
	30-27	SES	System parity error severity (BAM only).		
	26-18	ES	Error status (octal value).		
	17-0	EX	Error exit address.		
14			Reserved for installation.		
15	59-36	HL	Header length in characters; T type records.		
		MNR	Minimum record length.		
	35-33		Reserved for CRM.		
	32-30	EO	Error option (BAM only).		
	29		Reserved for CRM.		
	28	BAL	Buffer allocated by CRM; cannot be changed by the user.		
	27	STFT	Internal SETFIT flag used for CRM processing.		

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents		COMPASS Symbol
	26	PDF	SETFIT macro FILE statement flag; cannot be changed by the user.			
	25	SBF	Suppressed buffer I/O flag (BAM only).			
	24	SPR	Suppress read ahead (BAM only).			
	23		Reserved for CRM.			
	22	ORG	Currently supported file organization.	0	OLD Initial AAM no longer supported	≡OLD≡
				1	NEW (default) Previously known as extended AAM	≡NEW≡
	21-0	WSA	Working storage area address.			
16	59-36	TL	Trailer length in characters; T type records.			
	35-30	CL	Count field length in characters; T type records.			
		LL	Length field length in characters; D type records.			
		RMK	Record mark character; R type records.			
	29-24	PC	Padding character (BAM only).			
	23-18	MUL	Multiple of characters per K or E type block (BAM only).			
	26-18	MKL	Major key length in characters (indexed sequential files).			
	17-0	HRL	Hashing routine address (direct access files).			
	15-9	DP	Data block padding percent (indexed sequential and actual key files).			
17	59	FNF	Fatal/nonfatal flag; cannot be changed by the user.	0	Nonfatal	
				1	Fatal	
	58-57	OC	Open/close flag.	00	Never opened	≡NOP≡
				01	Opened	≡OPE≡
				10	Closed	≡CLO≡

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents		COMPASS Symbol
	56-54	PD	Processing direction.	000	Input (default)	≡ ≡
				001	INPUT Input	≡INPUT≡
				010	OUTPUT Output	≡OUTPUT≡
				011	IO Input/output	≡IO≡
	53-48		Reserved for CRM.			
	47	B8F	Round PUTs down to *8 bits (BAM only).			
	46	C1	COMP-1; format for the CL/LL field; T or D type records.	0	NO Display code	≡NO≡
				1	YES Binary	≡YES≡
	45	SB	Sign overpunch; overpunch option for CL/LL field; T or D type records.	0	NO No overpunch	≡NO≡
				1	YES Overpunch	≡YES≡
	44-21	CP	Trailer count beginning character position (numbered from 0); T type records.			
		LP	Length field beginning character position (numbered from 0); D type records.			
	20		Reserved for CRM.			
	19	CNF	Connected file flag (BAM only).			
	18	BBH	Buffer below highest high address (BAM only).			
	17-0	BFS	Buffer size in words.			
18	59-36	HMB	Number of home blocks (direct access files).			
		PTL	Partial transfer length (BAM only); number of keys moved to working storage area for a GET or GETN on an alternate key index.			
	35-30	LOP	Last operation code; cannot be changed by the user (BAM only).			
	35	WPN	Write bit; the upper bit of LOP is a 1-bit subfield that can be accessed separately; cannot be changed by the user.	0	Last operation was not a write	
				1	Last operation was a write	

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents	COMPASS Symbol
	29-0	RC	Record count; count of full records read or written since the file was opened. The count is not adjusted for repositioning and backspacing operations. For a multiple-index file, the number of records with this alternate key value. This field cannot be changed by the user.		
19	59-36	MBL	Maximum block length in characters.		
	35-30	VNO	Current volume number of the multi-volume sequential file (BAM only).		
		NL	Number of levels of index blocks (indexed sequential files).		
	29-0	BN	Block number of the current block (sequential files); cannot be changed by the user (BAM only).		
		WA	Current position word address, set by GET and PUT macros (BAM only).		
20	59	BCK	Block checksum.	0 NO No checksumming of blocks	≡NO≡
				1 YES Checksumming of blocks	≡YES≡
	58	PM	Processing mode.	0 Random	≡RPM≡
				1 Sequential	≡SPM≡
	51-30	DCT	Address of the display code to collating sequence conversion table (indexed sequential files).		
	59-36	MNB	Minimum block length in characters (BAM only).		
	29-18	RB	Number of records per block (actual key files) or average number of records (indexed sequential and direct access files).		
	17-0	PKA	Primary key address; address to receive primary key on an alternate key access (AAM files).		
21	59-18	XN	Logical file name of the alternate key index file associated with the data file.		
	17-0	XBS	Index file block size (AAM files).		
	59-24	MFN	Multifile set name (BAM only).		
	23-0	PNO	Multifile position number (BAM only).		

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents		COMPASS Symbol
22	59-46		Reserved for CRM.			
	45-40	LAC	Last action performed on the file; used by compiler languages to communicate with each other.			
	39-36	LNG	Last compiler language to have used the file.	0	Unknown	
				1	FORTRAN	
				2	COBOL	
				3	PL/I	
				4-7	Reserved	
	35-0		Reserved for CRM.			
23			Reserved for CRM.			
24	59	NDX	Index flag.	0	NO Data file is accessed	≡NO≡
				1	YES Index file only is accessed	≡YES≡
	58	KNE	Key not equal (multiple-index files).	0	Key match found	
				1	No key match found	
	57	FWI	Forced write indicator.	0	NO No forced write	≡NO≡
				1	YES Forced write	≡YES≡
	56	FPB	File position bit (system routine use only); or EOI reached random operation (multiple-index files).	0	EOI not reached	
				1	EOI reached	
	55	ON	Old or new file.	0	OLD Old file	≡OLD≡
				1	NEW Creation run	≡NEW≡
	54		Reserved for CRM.			
	53-24	FLM	File limit, records per file.			
	23	EMK	Embedded key flag.	0	NO Key is not part of the record	≡NO≡
				1	YES Key is included in the record	≡YES≡
	21-0	KA	Key address of the key value for record processing.			
25	59-18		Reserved for CRM.			
	17-0	BZF	Busy FET address; cannot be changed by the user.			

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents	COMPASS Symbol
26	59-48		Reserved for CRM.		
	47-30	CDT	Address of the collating sequence to display code conversion table.		
	29-0		Reserved for CRM.		
27			Reserved for CRM.		
28			Reserved for CRM.		
29			Reserved for CRM.		
30	59	SOL	S/L tape bit; cannot be changed by the user (BAM only).		
	58-21		Reserved for CRM.		
	20-0	EOIWA	End-of-information word address (BAM only).		
31	59-48	RKW	Relative key word.		
	47-44	RKP	Relative key position in RKW.		
	43-40	KP	Beginning character position of the key.		
	39-31	KL	Key length in characters.		
	30-24	IP	Index block padding percent (indexed sequential files).		
	23-0		Reserved for CRM.		
32	41-30		Reserved for CRM.		
	29-27	KT	Key type (indexed sequential files).	000 Symbolic (default)	
				001 S Symbolic (if user specified symbolic)	≡SKT≡
				010 I Integer	≡IKT≡
				011 F Floating	≡FKT≡
				011 U Uncollated symbolic	≡UKT≡
	26-24	REL	File position key relation (indexed sequential and multiple-index files). REL is significant only for START operations and index-only operations.	1 EQ Equal	≡EQ≡
				3 GE Greater than or equal	≡GE≡
				5 LT	
				6 GT Greater than	≡GT≡
	17-0	CPA	Compression routine address.		

TABLE D-1. STRUCTURE OF THE FIT (Contd)

Word	Bits	Fit Field	Description	Contents	COMPASS Symbol
33	59-18		Reserved for CRM.		
	17-0	DCA	Decompression routine address.		
34			Reserved for CRM.		

TABLE D-2. ALPHABETIZED SUMMARY OF FIT FIELDS

FIT Field	Word	FIT Field	Word	FIT Field	Word
ASCII	5	FNF	17	OC	17
BAL	15	FO	11	OF	11
BBH	17	FP	10	ON	24
BCK	20	FPB	24	ORG	15
BFS	17	FWB	1	PC	16
BN	19	FWI	24	PD	17
BT	11	HL	15	PDF	15
BZF	25	HMB	18	PEF	13
B8F	17	HRL	16	PKA	20
CDT	26	IP	31	PM	20
CF	11	KA	24	PNO	21
CL	16	KL	31	PTL	18
CM	11	KNE	24	RB	20
CMPLT	0	KP	31	RC	18
CNF	17	KT	32	RDR	1
CP	17	LA	10	REL	32
CPA	32	LAC	22	RKP	31
C1	17	LBL	10	RKW	31
DC	1	LCR	10	RL	11
DCA	33	LFN	0	RMK	16
DCT	20	LL	16	RT	11
DFC	13	LNG	22	SB	17
DP	16	LOP	18	SBF	15
DVT	1	LP	17	SES	13
DX	12	LT	10	SOL	30
ECT	13	LX	11	SPR	15
EFC	13	MBL	19	STFT	15
EMK	24	MFN	21	TL	16
EO	15	MKL	16	ULP	10
EOIWA	30	MNB	20	VF	11
ERL	13	MNR	15	VNO	19
ES	13	MRL	12	WA	19
EX	13	MUL	16	WPN	18
FF	1	NDX	24	WSA	15
FL	12	NL	19	XBS	21
FLM	24	NOFCP	13	XN	21

AAM has been divided into functional capsules that are loaded by relocatable controlling routines at execution time. This method of dynamic loading requires a program to be compatible with the Common Memory Manager (CMM). Static loading is available for programs that are not compatible; however, static loading could involve a field length penalty of as much as 1400_8 words. AAM uses dynamic loading unless static loading is specified through a control statement or a macro.

More information about the Common Memory Manager and the CYBER Loader can be obtained from their respective reference manuals.

DYNAMIC LOADING

For dynamic loading, all AAM macros reference entry points in the controlling routines CTL$RM and CTRL$AA. The controlling routines, which process parameters and diagnose certain types of errors, are loaded at relocatable load time or overlay generation time. The controlling routines load and transfer control to the Fast Dynamic Loader (FDL) capsule containing the proper AAM controller in fixed-position fixed-length blocks. The controller then loads the FDL capsules needed to process the macro.

Do not overlay the controlling routines CTL$RM and CTRL$AA because unknown results, including bad jump addresses to service routines, occur if these routines are overlayed. To prevent the controlling routines from being overwritten, make them part of the (0,0) overlay by specifying the FILE macro in the (0,0) overlay.

The OPENM/SETFIT capsule is loaded when the first OPENM or SETFIT macro is encountered. If the SETFIT macro occurs first, the FILE control statement parameters are processed, the dynamic AAM controller capsule is loaded, and control is transferred to that capsule. The required AAM processor capsule is then loaded, the buffer size is calculated, and control is returned to the user.

When the OPENM macro occurs before a SETFIT macro, the SETFIT functions are performed first. Open processing then occurs. The file is opened, FIT consistency checks are performed, and control is returned to the user. The open processing capsule is unloaded when a macro other than OPENM, SETFIT, STORE, or FETCH is encountered. For optimum efficiency in loading, the open processing for all files should be completed before other processing is specified. The AAM processor capsule remains loaded.

When the first macro that requires a buffer is encountered, a buffer is allocated through CMM in a fixed-position fixed-length block. The capsules required to perform the function specified by the macro are loaded; control transfers to the capsules and then back to the user. Generally, the capsules required to process these functions remain in memory until all files requiring them have been closed. Some capsules are loaded while a series of operations are being performed and are unloaded when additional memory space is needed to load another capsule.

The CLOSEM capsule is loaded when the CLOSEM macro is encountered. An additional AAM capsule might be loaded to close the file and release buffer space. The CLOSEM capsule unloads any capsules no longer needed for processing and unloads itself after closing the last file.

The AAM controller capsule, processing capsules, and dynamic buffers are loaded above the highest high address; however, they are not destroyed by overlay swapping. Because of this, it is possible to swap overlays without first closing the AAM files.

STATIC LOADING

Static loading is provided for the cases where the user is managing memory and the program cannot be compatible with CMM. It should only be used as a short term conversion aid. Long term support of static loading is not to be provided. Two methods are available for designating which capsules need to be statically loaded. One method is control statement oriented and the other method is macro oriented.

CONTROL STATEMENTS

Static loading can be specified through the LDSET and FILE control statements. The STAT option must be specified in the LDSET control statement and the USE and OMIT parameters must be specified in the FILE control statement. One FILE control statement must be included for each file to ensure that all necessary routines are loaded. The file organization (FO), the OLD/NEW FO qualifier (ORG), record type (RT), and index file name (XN) parameters must be specified on the same or a previous FILE control statement as the USE and OMIT parameters. These three parameters cannot be specified in a FILE control statement following the one that specifies the USE and OMIT parameters.

The USE and OMIT parameters are formatted as follows:

$USE=mn_1/mn_2/.../mn_n$

$OMIT=mn_1/mn_2/.../mn_n$

In both parameter formats, mn is a macro name. The functions of the USE and OMIT parameters are listed in table E-1. The USE and OMIT parameters can be used in more than one FILE control statement; the results are cumulative. If the STAT option is specified in the LSDET control statement and the USE parameter is not specified in the FILE control statement, no processing capsules are loaded.

TABLE E-1. USE AND OMIT PARAMETER FUNCTIONS

Parameter	No List of Macros	List of Macros
USE	All capsules are loaded.	Capsules performing functions specified by the macro list are loaded.
OMIT	All previously loaded capsules are unloaded.	Capsules performing functions specified by the macro list are unloaded.

In the example shown in figure E-1, the program to write the file ISFILE uses static loading and contains the OPENM, PUT, and CLOSEM macros. The program to read the file ISFILE also uses static loading. The PUT macro is not contained in that program; the OMIT parameter specifies that the capsule for that macro is to be unloaded. The GET macro is contained in the program and the capsule for that macro is to be loaded. The USE parameter is still in effect for the OPENM and CLOSEM macros.

STLD.RM MACRO FORMAT

Another method of specifying static loading is through the STLD.RM macro. The format of the STLD.RM macro is shown in figure E-2. This macro must be specified once for each file organization.

```
      .
      .
      .
FILE(ISFILE,FO=IS,RT=Z,USE=OPENM/PUT/CLOSEM)
LDSET(STAT=ISFILE)

Load set to write the file.

FILE(ISFILE,OMIT=PUT,USE=GET)
LDSET(STAT=ISFILE)

Load set to read the file.
      .
      .
      .
```

Figure E-1. Static Loading Example

(fo) STLD.RM USERT=(r),USE=(f),OMIT=(c),ORG=(n)

r	Record type list; record types are separated by commas.
f	AAM functions (macro names); functions are separated by commas.
c	CMM or FDL; CMM omits CMM and FDL, FDL omits FDL only.
n	Currently supported AAM file. NEW must be specified for macro to work.

Figure E-2. STLD.RM Macro Format

USE OF LIST-OF-FILES

The NOS and NOS/BE operating systems maintain a pointer to the list-of-files, which is a table of the name and FET or FIT address of all active files for each control point. This pointer is set and accessed by the SETLOF and GETLOF macros. A complete description of this feature can be found in volume 2 of the NOS Reference Manual and the System Programmer's Reference Manual for the NOS/BE operating system.

AAM maintains and uses this list-of-files. To alter this list, a user must follow a procedure that is compatible with AAM.

AAM maintains an entry point in its relocatable loaded routines called LOF$RM. The content of this entry point is the address of the current list-of-files. The purpose of this pointer is to minimize the number of GETLOF monitor calls required. The user is encouraged to use this pointer instead of calling the GETLOF macro.

If a user program that coexists with AAM moves the list-of-files, it must update the LOF$RM pointer in addition to calling the SETLOF macro. Also, if a user program adds a new entry to the end of the list-of-files, it must ensure that the next word is zero because AAM does not initialize the list-of-files block to zero.

BUFFER ALLOCATION

Buffer space for AAM files can be allocated either through Common Memory Manager (pooled buffer space) or by the user (user buffer space). The values at open time in the first word address of a buffer (FWB) and buffer size (BFS) fields of the FIT are used to determine whether Common Memory Manager (CMM) or the user will control buffer allocation.

POOLED BUFFER SPACE

If FWB equals zero, then buffer space is provided through CMM. The file can be open through more than one FIT at the same time, provided that FWB is zero in all of them.

To limit the amount of buffer space AAM requests from CMM, AAM calculates a total called TARGET whenever an OPEN or CLOSE is executed. When a buffer is needed, AAM checks TARGET before requesting more buffer space from CMM. If the size of the new buffer would make the current buffer space total greater than TARGET, old buffers are released to CMM to reduce the current buffer space total enough to accommodate the addition of the new buffer. The method of choosing buffers to be released is described later in this appendix.

CALCULATING TARGET

All files currently open are used in calculating the value of TARGET. The TARGET contribution of each file is calculated by scanning all the FITs through which the file is open as follows:

- Add BFS to the file's TARGET contribution for all the FITs in which BFS does not equal zero. (This gives the user a way of controlling TARGET if necessary.)
- Allocate one block for each FIT in which BFS equals zero. To this number of blocks

  Add 1 block for a direct access or actual key file.

  Add 2 blocks for an indexed sequential file with zero or one level of indexing.

  Add 5 blocks for an indexed sequential file with more than one level of indexing.

  Multiply the new number of blocks by the size of a data file block. Add this result to the file's TARGET contribution.
- Allocate one block for each FIT in which BFS equals zero and an alternate index file (MIP file) is specified. To this number of blocks

  Add 3 blocks if there is only one alternate key.

  Add 5 blocks otherwise.

  Multiply the new number of blocks by the size of a MIP file block, and add the result to the file's TARGET contribution.

Finally, the value resulting from adding each file's TARGET contribution is increased by 2000_8 words to accommodate AAM capsules, which are loaded when needed and unloaded automatically if unused for a long time.

The current field length can impose an upper limit on TARGET. When TARGET is to be adjusted (because of an OPEN or CLOSE operation), AAM calculates the new value of TARGET and uses statistics requested from CMM to adjust TARGET downward, if necessary.

CONTROLLING TARGET

A user program can control TARGET in one of two ways:

- By using BFS. If BFS does not equal zero in every open FIT, then TARGET is simply the total of all the BFS values.
- By storing a value in entry point AAM$BL (an entry point in the statically loaded portion of CRM). AAM$BL normally contains a value of 377777_8; any other value found in AAM$BL is used as an upper bound on TARGET. However, TARGET will never be reduced below the minimum amount of buffer space needed to process the individual file currently open with the largest block requirements. (Execution is very slow if TARGET is equal to the minimum.) The value in AAM$BL, which can be set at any time during program execution, is checked and used by AAM only at file open and close time. If a user sets a value in AAM$BL that is not large enough to allow AAM to process the open files, AAM will increase TARGET to accommodate the worst case file. The user's program is not aborted and the value in AAM$BL is not reset by AAM. AAM will calculate an internal value for AAM$BL based on the minimum number of buffers required to process the worst case file.

TARGET sets the upper bound on the amount of CMM space used for block buffers for the files currently open. The buffer space is pooled; therefore, if only one file is active, all the buffer space could be allocated to the active file's buffers. If all files are active, the files will compete for buffer space. The 2000_8 words for an AAM capsule that is infrequently present in memory will provide additional buffer space when rare capsules are not loaded.

USER BUFFER SPACE

If FWB is not equal to zero when a file is opened, then buffer space is provided directly by the user. The address of the user buffer space is in FWB and the length is BFS words. BFS should be large enough to contain the following:

- The FSTT (file statistics table) -- 130 words.
- If there are alternate keys, the FSTT of the alternate key index file and its FET -- 139 words.
- The FIT extension -- up to 168 words.

- Space for the data file -- 3 block buffers.
- Space for the alternate key index file, if any -- 3 block buffers.
- If compression is used, space for the compression buffer -- 1 block buffer.

If BFS is longer than the minimum, the extra space is divided between extra block buffers for the data file and extra block buffers for the alternate key index file, if any.

Once allocated to a file, the user buffer space (with its alternate key index file, if any) cannot be used by any other file and can be addressed only through this FIT.

If the user explicitly disables CMM, a minimum space of two blocks (three blocks if the file is compressed) must be allocated for each file. If the minimum space is not allocated, an error message is issued and the file is not opened. If CMM is not explicitly disabled by the user, TARGET is increased by the difference between the user buffer space and the minimum requirement.

BUFFER CHAINING AND ALLOCATION

File block buffers, whether provided directly by the user program or obtained from CMM, are bi-directionally chained in two chains. The first chain, an ownership chain, is formed by all block buffers of a given file. This chain associates the block buffers with the file. Unused block buffers, which exist only in user-provided buffer space, are also chained in this way because they belong directly to the file. (Unused block buffers in CMM space are immediately given back to CMM; therefore, the block buffers do not appear in AAM chains). Ownership chains are not part of the strategy of buffer allocation.

The second chain, the kick-out chain, consists of linkages between all block buffers of every open file (including those that have user-provided buffer space).

The basic strategy of buffer management is to move a buffer to the top of the kick-out chain when the buffer is accessed and to release the buffer at the bottom of the kick-out chain as space is needed. Therefore, blocks gradually migrate to the tail of the kick-out chain due to lack of use. The exceptions to this strategy are as follows:

- Whenever any index block of an indexed sequential file is referenced, the index block is moved to the top of the chain, and then the primary index block of the same file, if present in memory, is moved to the top of the chain.
- Whenever AAM's attention moves, within an indexed sequential or actual key file, from one data block to another, the new data block goes to the top of the chain. The old data block is pushed down past all the index blocks and all the capsules that may be in the chain.
- AAM scans the kick-out chain twice, from bottom to top, when choosing the blocks that will be released to make room for the new blocks. On the first scan, only blocks that have not been altered, and, therefore, do not need to be written out, are released. However, if a block exactly the right size is found, it is chosen immediately and written out if necessary, and the search is ended. If no such block is found and the total space from the blocks released is not enough, a second scan is made, and altered blocks are written out and released until there is enough space for the new blocks.
- An AAM capsule goes to the top of the kick-out chain when it is loaded, and whenever it is executed. On loading or executing, the capsule is given a count of three. When the capsule drifts to the bottom of the chain, the count is reduced by one. If the result is zero, the capsule is unloaded. If the result is not zero, the capsule goes to the top of the chain.
- A buffer in the user buffer space can be used only for its own particular file.

PERFORMANCE CONSIDERATIONS

If AAM is allowed to control buffer space, by leaving FWB and BFS equal to zero in all FITs, and not setting a limit in AAM$BL, then performance will not suffer because of insufficient buffer space. Performance will be better when all the open files have the same block size because CMM will not have to be called very often to reallocate space.

DATA COMPRESSION AND DATA ENCRYPTION

AAM supports data compression modules. These modules contain routines that enable the user to manipulate the size and format of data records when the records are passed between the working storage area (WSA) and a data file, as shown in figure H-1. Routine X1 compresses, expands, or reformats the records when they are passed to the file. Routine X2 restores the records to their original size and format when they are retrieved from the file.

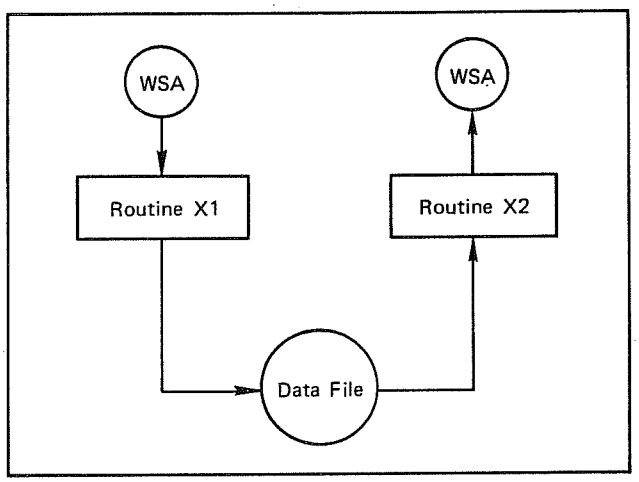


Figure H-1. Routines Used to Manipulate Size and Format of Data Records

Compression modules can conserve disk storage space by deleting consecutive occurrences of specific characters in data records. For example, a compression routine can delete strings of blanks in records before the records are written to the file. A decompression routine restores the blanks to the records when the records are read.

Compression modules can cause disk storage space for a file to increase. AAM conserves space in data blocks that contain records of uniform length by having a single half-word record pointer for the block. If one of the records decreases in size, as by the compression of a string of blanks, the block no longer contains records of uniform length and AAM creates half-word record pointers for every record in the block. This can result in a net loss of space.

A compression module can be coded to perform as an encryption module. Encryption modules are used to reformat and expand data records. The length of the encrypted records must be within the number of characters specified by the maximum record length (MRL) field in the FIT. A decryption routine restores the records to their original size and format when the records are read.

Because AAM handles compression and encryption modules in the same manner, the following discussion is limited to compression modules.

Three types of compression and decompression routines can be provided to AAM for use on a file:

- Release system routines
- Installation-defined system routines
- User-supplied routines

Data compression is specified by setting the compression routine address (CPA) field in the FIT. For a system routine, the CPA field is set to 1 for the release routine or to an integer in the range 2 through 63 for an installation-defined routine. For a user-supplied routine, the CPA field is set to the address of the entry point name of the user subroutine that compresses records. When a user-supplied routine is specified for the CPA field, the decompression routine address (DCA) field must be set to the address of the entry point name of a user subroutine that decompresses records.

Data compression can be established for the life of a file only on the file creation run. Once data compression has been selected, the method of compression must be the same for the life of the file. The same compression and decompression routines must be specified as long as the file exists. However, after data compression has been selected, it can be stopped by setting the CPA field to zero.

The compression and decompression routines are called by AAM with register A1 pointing to the vector shown in figure H-2.

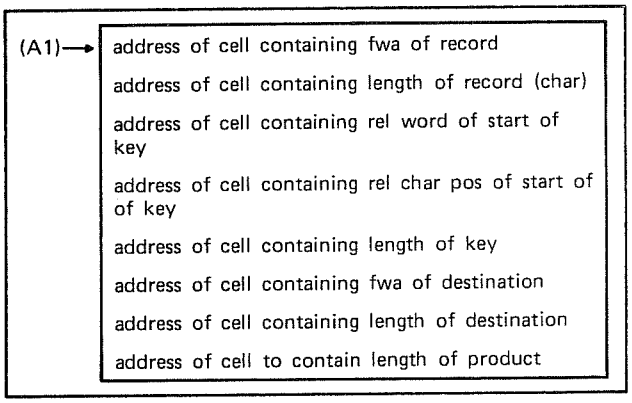


Figure H-2. Vector Used by Compression/ Decompression Routines

The primary key for records in a compressed file can be embedded or nonembedded. If the key is embedded, it must begin in the first character position of the first word. If the key is not embedded, the key length parameter must be zero. The primary key can be any type that is valid for the file organization.

AAM sets up a destination area to receive the record produced by the compression or decompression routine. The compression routine stores an identifier in the first word of the destination area. This identifier is also stored in the COMPACT field in the file statistics table (FSTT) when data compression is first selected. On subsequent runs when the file is opened, the identifier produced by the compression routine is verified against the identifier in the FSTT to ensure that the proper compression routine has been specified. The length of the destination area is the

length indicated by the MRL field plus ten characters for the identifier. A decompression routine does not store an identifier in the destination area; the key/record starts at the first word. Figure H-3 shows the format of the destination area.

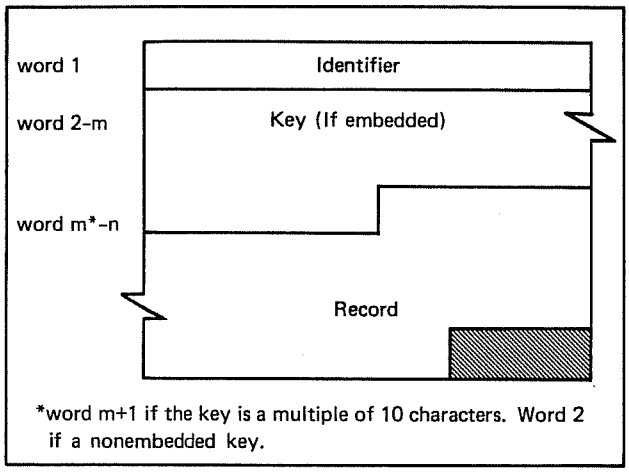


Figure H-3. Destination Area Format

The type of compression is noted in the SYSCOMP field in the FSTT. This field indicates whether system-supplied or user-supplied routines are used for compression and decompression of records.

SYSTEM-SUPPLIED ROUTINES

A system-supplied compression routine is identified by an integer value in the range 1 through 63. The integer is specified for the CPA field; AAM sets the DCA field for the corresponding system decompression routine.

When a system-supplied compression routine is specified at create time, the routine number is stored in the SYSCOMP field in the FSTT, and the identifier is stored in the COMPACT field in the FSTT. The specified compression routine and its associated decompression routine are loaded, and the locations of the routines are stored in the CPA and DCA fields in the FIT. The user must ensure that these FIT fields are not changed as long as the file exists; however, the CPA field can be set to zero to stop compression of records.

When a compressed file is opened, the routine number stored in the FSTT causes the appropriate system routines to be loaded. The CPA and DCA fields are set by AAM to the locations of these routines.

RELEASE ROUTINE

Compression routine number 1 attempts to shorten record length by compressing consecutive occurrences of display coded blanks, zeros, and colons. Up to 18 occurrences of the character are compressed into two characters. The first character is an escape character (<), which indicates the beginning of a compressed string. The six bits of the second character are used as follows:

- A two-bit code indicating the compressed character:

00	Colon	(00_8)
01	Zero	(33_8)
10	Blank	(55_8)
11	Escape character (<)	(72_8)

- A four-bit count indicating the number of occurrences of the compressed character.

Compression is not performed for one or two occurrences of the zero, colon, or blank. The repeat count, therefore, does not reflect the actual count; it designates how many more than three characters appear consecutively.

Compression is performed for all natural occurrences of escape characters (<) in a record to distinguish these characters from those that indicate the beginning of compressed strings in compressed records. One or more occurrences of the escape character anywhere in a record result in compression of that record, even if a longer record is produced. For example, a single occurrence of the escape character in a record is indicated by two characters in the compressed record. One character indicates a compressed string and another indicates the character code and repeat count.

INSTALLATION-DEFINED ROUTINES

The user (system analyst) can create system compression/decompression routines in addition to the standard release routines. After the additional routines are installed, they are called when the file is created by setting the CPA field to an integer between 2 and 63. AAM then dynamically loads and links the appropriate routines.

The source code for installation-supplied compression and decompression routines must be built into a capsule that meets the internal requirements of AAM. The following paragraphs explain the source code for setting up a simple compression/decompression routine. This example assumes a 50-character record with a 5-character key embedded at the beginning of the record. The compression routine discards every other character of the record; the decompression routine replaces the discarded characters with blanks.

Figure H-4 shows the code that sets up the capsule name and linkage to the compression and decompression routines. The code is described as follows:

- The IDENT statement identifies the capsule name. This statement is of the form CMPR$nn where nn is between 02 and 63. The integer nn is specified in the CPA field in the FIT at file creation time. The standard AAM open routine internally recognizes CPA 2 through 10. If CPA 11 through 63 are used, the system analyst must include the numbers used in a table near OPNMDAA.315 in the module OPNM$AA.

- The LCC statements are loader directives that can reside either in the source as LCCs or in the control statement load sequence that builds the capsule. In both cases, the group must be AAM$CTL and the capsule must be CMPR$nn.

```
                                    EXPAND    COMP
                                              IDENT   CMPR$03
                                              LCC     GROUP($AAM$$CTL$)
                                              LCC     CAPSULE($CMPR$$03$)
                                              ENTRY   CMPR$03
0  0315202253333600001                        VFD     42/0LCMPR$03,18/1
1                                   CMPR$03   BSS     0
1  0000000000000000003 +                      VFD     42/0,18/CCOMP
2  0000000000000000010 +                      VFD     42/0,18/CEXPAND
```

Figure H-4. Code Used to Set Up Capsule Name and Linkage to Compression/Decompression Routines

- The remaining statements set up a three-word block that must reside in the capsule. Figure H-5 shows the format of the three-word block. The format is described as follows:

  CMPR$nn (address x) is declared an entry point. The word at x-1 is the AAM capstat word. The capstat word contains the name of the capsule (CMPR$nn) and a capsule activity count (1). AAM uses the capsule activity count to determine when to unload the capsule.

  The words at x and x+1 contain the addresses of the compression and decompression routines, respectively. Note that the loader does not allow nonstandard relocation in capsules. Standard relocation requires 18-bit addresses in one of the three standard parcels of a computer word; therefore, the addresses at x and x+1 are coded as VFD 42/0,18/addr instead of VFD 60/addr. This problem concerns only COMPASS programmers since compilers such as FORTRAN always generate standard relocation.

The formal parameters are as listed in figure H-2. Because the fwa parameters are indirect addresses, any FORTRAN or COBOL compression routine requires a COMPASS interface to adjust the parameter list. Figure H-6 shows code that replaces the indirect pointers in the parameter list used by the FORTRAN compression and decompression routines. The code restores the indirect addresses to the list when control is returned from the FORTRAN routines.

Figure H-7 shows the FORTRAN compression and decompression routines called from the COMPASS portion of the capsule.

Figure H-8 shows the load map for the capsule CMPR$03. The following statements generate the capsule:

```
LOAD,bin.
NOGO,CAPS.
```

The CAPS file contains the modules in capsule format. The capsule can now be added to AAMLIB with the LIBGEN and SYSEDIT statements for NOS or with the EDITLIB statement for NOS/BE. AAM will then load and link the capsule when needed by files with CPA set to the appropriate value at file creation time. Subsequent runs do not require that CPA be specified since CPA information is recorded in the FSTT for the file.

The capsule can also be added to a user library with the LIBGEN or EDITLIB statements. AAM will load and link the capsule from the user library if the library file is declared a global library with the LIBRARY statement. Routines on a user library must be made available to each job step that references the file.

USER-SUPPLIED ROUTINES

User-supplied compression and decompression routines are specified by setting the CPA and DCA fields in the FIT to the routine entry points. AAM communicates with the compression routine through the list of arguments specified in figure H-2.

The compression routine must set the compressed record length parameter. If the record can be compressed, the record length returned to AAM is the number of characters in the compressed record plus ten characters for the identifier. If the compression routine makes the record longer instead of shorter, a negative number must be returned. This informs AAM that the original record should be used.

When a file is compressed by a user-supplied routine, the user is responsible for maintaining the correct routine entry points in the CPA and DCA fields.

When the file is opened, the CPA field is checked for a zero value. If it is set to zero, no compression or decompression is performed.

When a compressed record is retrieved from the file, the decompression routine is called to restore the record to its original state. The record passed to the routine does not include the identifier. The decompression routine must return the decompressed record length. If the decompressed record exceeds the size of the destination area, a negative number must be returned. Returning a negative number produces an error, and the record is not transferred.

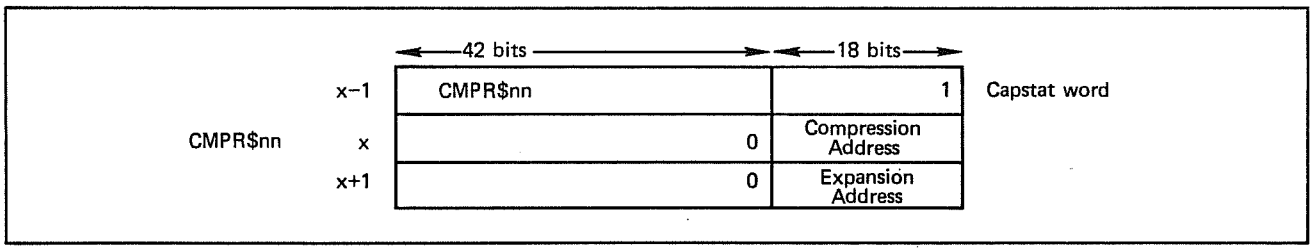


Figure H-5. Three-Word Block Residing in Capsule

```
  3  0400400003 +            CCOMP    EQ     *+400000B
  4  0100000015 +                     RJ     PATCH
  5  0100000000 X                     RJ     =XCOMP
  6  0100000024 +                     RJ     RESTORE
  7  0400000003 +                     EQ     CCOMP


 10  0400400010 +            CEXPAND  EQ     *+400000B
 11  0100000015 +                     RJ     PATCH
 12  0100000000 X                     RJ     =XEXPAND
 13  0100000024 +                     RJ     RESTORE
 14  0400000010 +                     EQ     CEXPAND


                             *
                             *  -PATCH- REMOVES THE INDIRECT POINTERS IN THE -APLIST- FOR THE OLD
                             *  AND NEW RECORD AREAS.
                             *
 15  0400400015 +            PATCH    EQ     *+400000B
 16  74610                            SX6    A1
         5160000031 +                 SA6    SAVEA1
                   53210              SA2    X1              PTR TO PTR TO OLD RECORD
 17  10722                            BX7    X2
         54710                        SA7    A1              PTR REMOVED
               10611                  BX6    X1
 20  5160000032 +                     SA6    SAVEAPO         SAVE OLD REC PTR FOR RESTORE
               54110                  SA1    A1              REFRESH A1 WITH NEW VALUE
 21  5021000005                       SA2    A1+5            PTR TO PTR TO NEW RECORD
               53320                  SA3    X2
                   10733              BX7    X3
 22  54720                            SA7    A2              PTR REMOVED
         10622                        BX6    X2
               5160000033 +           SA6    SAVEAPN         SAVE NEW REC PTR FOR RESTORE
 23  0400000015 +                     EQ     PATCH


                             *
                             *  -RESTORE- PUTS THE INDIRECT POINTERS FOR THE OLD AND NEW RECORD
                             *  AREAS BACK INTO THE -APLIST-.
                             *
 24  0400400024 +            RESTORE  EQ     *+400000B
 25  5120000031 +                     SA2    SAVEA1          OLD APLIST POINTER
               5130000032 +           SA3    SAVEAPO
 26  5140000033 +                     SA4    SAVEAPN
               10633                  BX6    X3
                   10744              BX7    X4
 27  53620                            SA6    X2
         5272000005                   SA7    X2+5
                   53120              SA1    X2
 30  0400000024 +                     EQ     RESTORE


 31                   1   SAVEA1   BSS    1
 32                   1   SAVEAPO  BSS    1
 33                   1   SAVEAPN  BSS    1
 34                       END
```

Figure H-6. Code that Adjusts the Parameter List

```
      SUBROUTINE COMP (OLDREC, OLDRL, KEY, KP, KL, NEWREC, NEWLEN,NEWRL)
      IMPLICIT INTEGER (A-Z)
      CHARACTER OLDREC*100
      CHARACTER NEWREC*100
C
C  -COMP- REMOVES EVERY OTHER CHARACTER FROM EACH RECORD. -COMP- ASSUMES
C  THAT THE KEY IS EMBEDDED AT THE BEGINNING OF THE RECORD.
C
C  SET -PASSWORD- INTO FIRST WORD OF DESTINATION
      NEWREC (1:10) = 'PASSWORD'
      NEWRL = 10 + KL
C  EXIT IF THIS IS A PASSWORD VERIFICATION CALL FROM CRM
      IF (KL .EQ. 0) RETURN
      NEWREC (11 : KL+10) = OLDREC (1 : KL)
      DO 100 I = KL + 1, OLDRL, 2
      NEWRL = NEWRL + 1
  100 NEWREC (NEWRL:NEWRL) = OLDREC (I:I)
      RETURN
C
C  -EXPAND- REPLACES THE CHARACTERS DELETED BY -COMP- WITH BLANKS.
C
      ENTRY EXPAND (OLDREC, OLDRL, KEY, KP, KL, NEWREC, NEWLEN, NEWRL)
      NEWRL = KL
      NEWREC (1 : KL) = OLDREC (1 : KL)
      DO 200 I = KL + 1, OLDRL
      NEWREC (NEWRL+1 : NEWRL+1) = OLDREC (I:I)
      NEWREC (NEWRL+2 : NEWRL+2) = ' '
  200 NEWRL = NEWRL + 2
      RETURN
      END
```

Figure H-7. Compression/Decompression Routines

```
1          LOAD MAP - CMPR$03                              CYBER LOADER 1.5-528


 --------  CAPSULE   CMPR$03      GROUP  AAM$CTL      WRITTEN TO FILE  CAPS

           CAPSULE LENGTH --         3625


           PROGRAM AND BLOCK ASSIGNMENTS.

           BLOCK       ADDRESS   LENGTH    FILE      DATE     PROCSSR VER LEVEL  HARDWARE   COMMENTS

           CMPR$03           3       34     LGO      80/11/14 COMPASS 3.6 530
           COMP             37      266     LGO      80/11/14 FTN     5.0 528    767X I     SUBROUTINEOPT=0
           CHMOVE=         325      254  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               CHARACTER MOVE AND CONCATENATE
           CPL=            601        6  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FTN5 - COPY PARAMETER LIST
           /Q5.IO./        607      300
           FVS=           1107       15  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FCL5 - FORM VARAIBLE SUBSCRIPT.
           FCD=           1124       22  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FCL5 - FORM CHARACTER DESCRIPTOR.
           /AP.IO./       1146       17
           /FCL.C./       1165       34
           /FCL=ENT/      1221       73
           FORSYS=        1314     1256  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FORTRAN OBJECT LIBRARY UTILITIES.
           FCL=FDL        2572       63  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FCL CAPSULE LOADING
           FORUTL=        2655      165  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FCL MISC. UTILITIES.
           GETFIT=        3042      164  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               LOCATE AN FIT GIVEN A FILE NAME.
           /STP.END/      3226        1
           Q5NTRY=        3227       32  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               FCL5 - INITIALIZE FCL5 RUN TIME LIBRARY.
           SYSAID=        3261        1  SL-FTN5LIB  80/09/26 COMPASS 3.6 530               LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
```

Figure H-8. CMPR$03 Load Map

FUTURE SYSTEM MIGRATION GUIDELINES

I

This appendix contains programming practices recommended by CDC for users of the software described in this manual. When possible, application programs based on this software should be designed and coded in conformance with these recommendations.

Two forms of guidelines are given. The general guidelines minimize application program dependence on the specific characteristics of a hardware system. The feature use guidelines ensure the easiest migration of an application program to future hardware or software systems.

GENERAL GUIDELINES

Good programming techniques always include the following practices to avoid hardware dependency:

- Avoid programming with hardcoded constants. Manipulation of data should never depend on the occurrence of a type of data in a fixed multiple such as 6, 10, or 60.

- Do not manipulate data based on the binary representation of that data. Characters should be manipulated as characters, rather than as octal display-coded values or as 6-bit binary digits. Numbers should be manipulated as numeric data of a known type, rather than as binary patterns within a central memory word.

- Do not identify or classify information based on the location of a specific value within a specific set of central memory word bits.

- Avoid using COMPASS in application programs. COMPASS and other machine-dependent languages can complicate migration to future hardware or software systems. Migration is restricted by continued use of COMPASS subroutines embedded in programs using higher-level languages, and by COMPASS owncode routines used with CDC standard products. COMPASS should only be used to create part or all of an application program when the function cannot be performed in a higher-level language or when execution efficiency is more important than any other consideration.

FEATURE USE GUIDELINES

The recommendations in the remainder of this appendix ensure the easiest migration of an application program for use on future hardware or software systems. These recommendations are based on known or anticipated changes in the hardware or software system, or comply with proposed new industry standards or proposed changes to existing industry standards.

ADVANCED ACCESS METHODS

The Advanced Access Methods (AAM) offer several features within which choices must be made. The following paragraphs indicate preferred usage.

Access Methods

The recommended access methods are indexed sequential (IS), direct access (DA), and multiple index processor (MIP).

Record Types

The recommended record types are either F for fixed length records, or W for variable length records. Record length for W records is indicated in the control word; the length must be supplied by the user in the RL FIT field on a put operation and is returned to the user in RL on a get operation.

FORTRAN Usage

The following machine-independent coding practices are encouraged for a FORTRAN programmer using AAM:

- Initialize the FIT by FILExx calls or by the FILE control statement.
- Modify the FIT with STOREF calls.
- Use FORTRAN 5 CHARACTER data types when working with character fields rather than octal values of display code characters; specify lengths of fields, records, and so forth, in characters rather than words.
- Manipulate records containing character data as a CHARACTER array or variable, not as INTEGER. The working storage area (WSA), for example, should be declared as CHARACTER.

SUMMARY OF FORTRAN CALL STATEMENTS J

This appendix includes the general formats of FORTRAN calls to AAM for indexed sequential, actual key, and direct access file organizations. The following conventions are used:

- Words in uppercase must appear exactly as they are shown.
- Words in lowercase are generic terms that represent the words or symbols supplied by the programmer. In most instances, the terms are the same as actual names of FIT fields. These fields can be constants or integer variables.
- Subroutines FILExx and STOREF require specifications of field and value. The word field denotes the name of a FIT field; it must be a character string in left-justified format. The word value denotes the value to be placed in the field; it must be a character string in left-justified format for symbolic options, an integer representation for numeric options, or a program name or variable name (for example, owncode exits and working storage area).
- Function IFETCH requires a field specification. The word field denotes the name of a FIT field; it must be a character string in left-justified format. The word variable denotes an integer variable in which the value of the FIT field will be returned.
- Except for CALL FILExx, the order of parameters is fixed so that all parameters positioned to the left of a desired option must be specified. A parameter list can be truncated at any point after the fit; middle parameters cannot be defaulted. If a parameter is not applicable to a particular file organization and its position is needed in a statement, a zero must be specified as indicated in the formats. If a parameter is applicable to the file organization but not applicable to the record type, a zero must be specified to mark a needed position. Zeros should never be used for applicable fields meaning a parameter is not intended. A zero or the address of the constant zero will be used as the parameter.

INDEXED SEQUENTIAL FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEIS (fit,field,value,...,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,mkl,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit, field, variable)

CALL OPENM (fit,pd,of)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfndata,rkw,rkp,kl,0,kf,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,mkl,ex)

CALL SKIP (fit,±count)

CALL STARTM (fit,ka,kp,mkl,ex)

CALL STOREF (fit,field,value)

ACTUAL KEY FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEAK (fit,field,value,...,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,0,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit, field, variable)

CALL OPENM (fit,pd)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfndata,rkw,rkp,kl,0,kf,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,0,ex)

CALL SKIP (fit,±count)

CALL STOREF (fit,field,value)

DIRECT ACCESS FILE ORGANIZATION

CALL CLOSEM (fit,cf)

CALL DLTE (fit,ka,kp,0,ex)

CALL FILEDA (fit,field,value, . . . ,field,value)

CALL FITDMP (fit,id)

CALL GET (fit,wsa,ka,kp,0,0,ex)

CALL GETN (fit,wsa,ka,ex)

CALL GETNR (fit,wsa,ka,ex)

IFETCH (fit,field)

CALL IFETCH (fit, field, variable)

CALL OPENM (fit,pd)

CALL PUT (fit,wsa,rl,ka,kp,0,ex)

CALL REPLC (fit,wsa,rl,ka,kp,0,ex)

CALL REWND (fit)

CALL RMKDEF (lfndata,rkw,rkp,kl,0,kf,ks,kg,kc,nl,ie,ch)

CALL SEEKF (fit,ka,kp,0,ex)

CALL STOREF (fit,field,value)

CONCURRENCY AND AAM FILES

INTRODUCTION

Concurrency is a state in which two or more users or jobs can access a single AAM file at the same time.

An AAM file can be shared among any number of read-only users. Each user is unaware of the other users, and the content of the file is stable (unchanging).

Problems arise when users want to update the AAM file in addition to reading it.

The NOS and NOS/BE permanent file management systems require that no more than one job or user update a particular file at any moment. The permanent file manager prevents this from occurring by letting no more than one job have any kind of modification privileges on that file at any moment.

In addition, AAM requires that no job or user may read an AAM file while another job is updating it. If this is attempted, the jobs or users reading the file will get unpredictable outcomes. The job may work, or get incorrect results, or get AAM errors, or even abort.

CONCURRENCY: THE CORRECT SOLUTION

True multiple read/multiple update concurrency of an AAM file is reliably achieved only through a single central program that coordinates the reads and updates of many other users. Examples of such a program are TAF (Transaction Facility on NOS), CDCS (Cyber Database Control System on NOS and NOS/BE), and CVRM (Concurrent Version of Record Manager on NOS). When one of these programs is used to achieve concurrency on an AAM file, that program is the single job that reads or updates the file. All reads and updates from the multiple users of the file are performed by the central program, which in turn provides records and status information back to the user or job which had made the request.

If you are using TAF, CDCS, or CVRM to perform the reads and updates on your AAM files, then full concurrency is already available to you and you do not need to read any further.

The rest of this appendix identifies techniques that approach full concurrency without using one central program.

NOTE

We strongly urge you to use a product like TAF, CDCS, or CVRM in achieving concurrency on your AAM files. The alternative techniques described below are not 100 percent reliable. Use these techniques at your own risk.

ALTERNATIVE TECHNIQUES

The following information is provided for those programmers who find the risk of unreliable retrievals less important than the need for some kind of concurrency. We hope this information will help you to minimize your risk, and help you identify problems when they occur.

CONCURRENCY BY REPEATED ATTACH/OPEN/CLOSE/RETURN

One approach to the problem is to organize the job updating the file to ATTACH, OPEN, makes changes, CLOSE, and RETURN the file for all of its updates. The jobs that read the file would ATTACH, OPEN, read..., CLOSE, and RETURN the file for all of their accesses.

The updating job attaches the file with M=W to have exclusive access and no concurrent readers. The attaches of the file must be done in a way that either waits or tries again if the file is busy. If the updates are short enough, the readers of the file are not adversely affected.

This technique does not allow true concurrency (except by readers only), and has the danger on NOS of the updating job being locked out by the readers of the file. Another disadvantage is the expense of additional overhead. However, the results of retrieval programs are reliable since they are not accessing the file during an update. Query Update (QU/CRM) uses this style of file access to simulate concurrency.

CONCURRENCY BY PERMANENT FILE ATTACH MODES

One of the primary techniques of simulating concurrency is the use of multiple read/single update permanent file access.

On NOS, the updater attaches the file with M=M or M=U, and each reader attaches the file with M=RM or M=RU.

On NOS/BE, the updater attaches the file with RW=1, which prevents CN permission. MD, EX, and RD permissions are still needed. Each reader attaches the file with MR=1, which prevents CN, MD, and EX permissions.

In this appendix, an M=RM user refers to a reader of the file, and an M=M user refers to an updater of the file.

An AAM file is a complex structure of different types of blocks in a disk file. The physical order of blocks in the file is not critical. Often, the block containing the next sequence of records is not contiguous with the current block. Adding records at the logical end of the file can alter blocks in the middle, at the beginning, or at the end of the file. The effect is unpredictable. Even changing a single record can affect many blocks in the file.

For efficiency, AAM does not immediately flush these changes to disk, but keeps the blocks in memory for awhile. At these times, the current contents of the file is a combination of blocks in memory and blocks on disk, with the altered memory blocks taking priority over their disk counterparts.

During the time altered blocks exist in memory, AAM ensures that bits are set in the FSTT, both in memory and on disk, to indicate that the file is in a state of change. These bits are cleared when the file is flushed or closed and the disk copy is completely up to date.

If another control point tries to read this file using M=RM, it can look only at the disk blocks, which generally contain some of the changed blocks, but rarely all of them. This can cause the file to appear to be in a ruined state because the set of blocks on disk is inconsistent. AAM cannot distinguish this situation from that of a truly ruined file.

The symptoms of this problem are hard to predict. An M=RM user of the file could open the file at a time when the FSTT indicates there is nothing wrong with the file. Later, AAM could read in some altered blocks which could cause various AAM internal errors. At this time, the FSTT on disk would have some bits set to indicate the file was being updated. The AAM errors correspond to the different combinations of reading some blocks that are current and some blocks that are not current. The blocks that are not current have been changed in memory for the job updating the file, but have not yet been written to disk.

To minimize the possibility of errors, set FIT field FWI to YES. This tells AAM to force-write all altered blocks to disk as soon as they are changed. This minimizes the amount of time the file is on disk in an inconsistent state. However, it increases the job overhead of processing an AAM update request.

This still does not eliminate the problem. A single AAM update can affect many blocks, and each of these must be flushed to disk before the operation is considered complete. An M=RM job that accesses the file for very short periods may execute correctly. However, it can still read blocks from the file while the updated blocks are being written to the file. An M=RM job that accesses the file for long periods of time runs the risk that blocks it holds in memory may have been changed on disk, and pointers to other blocks may no longer be valid. In both of these cases, an M=RM reader of the file can receive a variety of fatal or non-fatal AAM errors.

AVOIDING ERROR 202 IN RETRIEVAL PROGRAMS

AAM sets flags in the FSTT whenever a file is in an indeterminate state, and clears those flags whenever the file is closed or flushed. AAM recognizes a potentially corrupt file by the presence of these flags in the FSTT.

If a job opens an AAM file that has one of these flags set, the job is informed of the potentially corrupt nature of the file by an error status such as 202 (File Ruined). COBOL and FORM jobs will terminate upon receiving this error status, but FORTRAN users and others that get control after the open can recognize the error and try again later.

If one job reads an AAM file while another job is updating it, it is likely that the reader will find the file in an apparently corrupted state. Note that the AAM job reading the file cannot distinguish between a corrupted file and one actively being updated by another job.

A new flag has been defined in the FIT. If this flag is set before the open, AAM bypasses one of the checks that issues error 202, allowing the file to be opened. Set this bit at your risk, because it directs AAM to read a file otherwise considered unusable.

You can try using this bit to salvage information from a damaged file by copying records to a new file and checking them carefully. Do not set this bit for a job that updates the file because you might cause a corrupted but salvageable file to become completely destroyed and useless.

This risky bit is unnamed, and is in the sign bit (leftmost bit) of the 30th word of the FIT (i.e., FIT+29). Setting the bit can cause the job to read incorrect information, receive unpredictable errors, or aborts. However, if the possibility of salvaging records from a partially corrupted file is more important to you than the associated risk, set this bit at your own risk, and never set it if you are going to update the file.

Files read with this risky bit set can get other AAM errors, such as 547 or 546, or get incorrect results. Setting the bit merely bypasses an AAM error trap. The bit does not make the results of the retrieval any more reliable.

SUMMARY

True concurrent access to CRM files under NOS and NOS/BE can be accomplished by having a single program coordinate all the updates and reads on the file. CDCS is a good example of such a program. Attempts to achieve concurrent access through the use of permanent file access modes such as M=M and M=RM, often appear to work, but eventually fail with incorrect results, AAM errors, or, in rare instances, mode errors.

We discourage the use of M=M and M=RM to achieve concurrency on AAM files. For similar reasons, we also discourage the user of M=U/M=RU and M=A/M=RA.

INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Record Manager Advanced Access Methods Version 2 Reference Manual

PUBLICATION NO.: 60499300

REVISION: F

This form is not intended to be used as an order blank. Control Data Corporati welcomes your evaluation of this manual. Please indicate any errors, suggest additions or deletions, or general comments on the back (please include page numb references).

_____ Please reply _____ No reply necessary

FOLD

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Technology and Publications Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

CUT ALONG LINE

FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAP

ORPORATE HEADQUARTERS, P.O. BOX O, MINNEAPOLIS, MINN 55440
ALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA