60495800

CONTROL DATA

CYBER RECORD MANAGER BASIC ACCESS METHODS VERSION 1.5 USER'S GUIDE

CDC® OPERATING SYSTEMS:
NOS 1
NOS/BE 1

ALPHABETIC LIST OF FORTRAN CALL STATEMENTS

SEQUENTIAL FILES

WORD ADDRESSABLE FILES

6049580

CONTROL DATA CORPORATION

CYBER RECORD MANAGER BASIC ACCESS METHODS VERSION 1.5 USER'S GUIDE

CDC® OPERATING SYSTEMS:
NOS 1
NOS/BE 1

REVISION RECORD

Revision	Description
A (03/31/77)	Original release.
B (06/15/79)	This revision reflects CDC® CYBER Record Manager Basic Access Methods Version 1.5 at PSR level 498. Basic Access Methods support only sequential and word addressable file organizations; all information relating to indexed sequential, direct access, and actual key file organizations has been removed. The entire manual has been reprinted.
C (04/01/81)	This revision reflects CDC CYBER Record Manager Basic Access Methods Version 1.5 at PSR level 528. All program examples have been updated to FORTRAN Version 5.1, and miscellaneous technical and editorial changes have been made.

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

©COPYRIGHT CONTROL DATA CORPORATION 1977, 1979, 1981
All Rights Reserved
Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

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
Cover	-
Inside Cover	C
Title Page	-
ii	C
iii/iv	C
v/vi	C
vii	C
viii	C
ix	C
1-1 thru 1-4	C
1-5	B
1-6	C
2-1 thru 2-12	C
3-1 thru 3-27	C
4-1 thru 4-12	C
5-1 thru 5-9	C
A-1	C
A-2	C
A-3	B
A-4	B
B-1	C
C-1	C
C-2	C
D-1 thru D-6	C
E-1	C
E-2	C
F-1 thru F-4	C
Index-1 thru -4	C
Comment Sheet	C
Mailer	-
Back Cover	-

PREFACE

CONTROL DATA® CYBER Record Manager Basic Access Methods (BAM) Version 1.5 operates under control of the following operating systems:

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

BAM handles all input/output processing of files with sequential or word addressable organization. User programs concerned with either of these file types can communicate with BAM indirectly through a compiler, using the calls supplied by the language; directly through COMPASS macros; or directly through FORTRAN calls. This guide is designed specifically for FORTRAN programmers who are processing files through direct calls to BAM; the material presented, however, can be used to advantage by programmers utilizing COMPASS or any of the languages that provide indirect access to BAM.

Programming examples, written in FORTRAN Version 5, emphasize file information table (FIT) field values; specifically, why they are set by the user and how they are interpreted by BAM. Wherever practical, information is collected and organized into tabular form to provide quick reference.

The NOS manual abstracts and the NOS/BE manual abstracts are pocket-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals are useful in determining which manuals are of greatest interest to a particular reader. The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

Related material is contained in the following publications.

The following publications are of primary interest:

Publication	Publication Number
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700
FORTRAN Version 5 Reference Manual	60481300
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS/BE Version 1 Reference Manual	60493800

The following publications are of secondary interest:

Publication	Publication Number
CYBER Record Manager Advanced Access Methods Version 2 Reference Manual	60499300
CYBER Record Manager Advanced Access Methods Version 2 User's Guide	60499400
FORM Version 1 Reference Manual	60496200
NOS Version 1 Manual Abstracts	84000420
NOS Version 1 Reference Manual, Volume 2 of 2	60445300
NOS/BE Version 1 Manual Abstracts	84000470

Software Publications Release History	60481000
8-Bit Subroutines Version 1 Reference Manual	60499500

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

This manual describes a subset of the features documented in the Basic Access Methods and FORTRAN Version 5 reference manuals. Control Data cannot be responsible for the proper functioning of any features not documented in the Basic Access Methods or FORTRAN Version 5 reference manuals.

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

INTRODUCTION TO CYBER RECORD MANAGER 1

CONTROL DATA CYBER Record Manager (CRM) is a group of routines that provide an interface between user programs and the operating system routines that read and write files on hardware devices. The file processing capabilities of CYBER Record Manager are divided into two categories: the Basic Access Methods (BAM) and the Advanced Access Methods (AAM). The term BAM refers to the CYBER Record Manager routines that process sequential and word addressable file organizations. The term AAM refers to the CYBER Record Manager routines that process indexed sequential, direct access, and actual key file organizations.

The CYBER Record Manager routines handle the opening, positioning, and closing of referenced files. The routines also perform input/output operations for the files based on information provided by the calling program. This information is conveyed to BAM or AAM through a basic communication area called a file information table (FIT). The FIT includes descriptive information that is used by BAM or AAM to formulate calls for action by the operating system. FIT entries include the method by which the file is to be accessed, descriptions of record size and type, and a variety of options that include provision for end-of-data and error exit processing.

The calling program is responsible for establishing a FIT for each file that is to be referenced by the CRM routines. This operation is usually performed by the language processor as part of the language syntax; however, CRM can be accessed directly through user-supplied routines written in FORTRAN or COMPASS. FORTRAN routines access BAM and AAM through direct calls to library routines; the call statements and their appropriate parameters are summarized in appendix B. COMPASS routines access BAM and AAM through macro calls; the macros and their appropriate parameters are detailed in the CYBER Record Manager reference manuals.

The file formats created and recognized by direct calls to the CRM routines are independent of the language or processor through which input/output calls are initiated. A file created by one source language can usually be processed by another. For example, the FORTRAN programmer with no working knowledge of the COBOL language can, by using direct calls, access a file that was created by a COBOL program. Similarly, the COMPASS programmer with no working knowledge of the FORTRAN language can access a file that was created by a FORTRAN program.

ISSUING DIRECT CALLS

The systems programmer writing execution time input/output routines for high level languages can access CRM through direct calls. The internal calls generated for the FORTRAN PRINT statement and the COBOL WRITE statement are typical examples. Languages that utilize the file management capabilities of BAM or AAM include the FORTRAN, COBOL, ALGOL, and PL/I compilers; the Sort/Merge and FORM utility packages; and the Query Update data processing language. BASIC, APL, and Update do not utilize CRM.

The systems programmer writing FORTRAN or COMPASS subroutines for use by applications programs can access CRM through direct calls. Any applications language that maintains an interface capability with the FORTRAN compiler or COMPASS assembler can call a subroutine to initiate file processing.

The FORTRAN applications programmer can exercise one of two options: access CRM directly through FORTRAN direct calls, or access CRM indirectly by utilizing the standard input/output statements provided by the FORTRAN language.

The applications programmer who is performing user label processing on magnetic tapes must access BAM through direct calls. Access must be through a COMPASS macro call.

All other applications programmers can access CRM indirectly. File processing is accomplished through the use of standard input/output statements provided by the applications language. The standard input/output statements generate internal code that establishes the FIT and initiates calls to the appropriate CRM routines for file processing. The applications programmer need not be concerned with how a file is physically structured or how logical records are physically represented. Once the file organization and file structure are supplied to CRM by a logical description within the applications program or by control statements that precede the program, a file can be accessed with standard read and write functions.

In summary, the user who issues direct calls to BAM or AAM is one of the following:

- A FORTRAN or COMPASS programmer writing execution time routines for high level languages
- A FORTRAN programmer writing applications programs that require the advantages available through direct calls
- A FORTRAN or COMPASS programmer writing input/output modules for use by applications programs
- A COMPASS programmer writing modules to handle user label processing for applications programs

The programmer directly accessing CRM is responsible for the following:

- Establishing the FIT to define the file structure and subsequent processing limits
- Establishing a working storage area in a program for passing data records between the program and the file storage device
- Issuing direct calls to CRM to open the file, to process input and output operations for the file, and to close the file

The programmer indirectly accessing CRM is responsible for the following:

- Understanding CRM terminology and concepts to ensure program efficiency
- Supplementing, as necessary, input/output processing provided by the applications language by using a FILE control statement to override some of the defaults

FILE ORGANIZATIONS AVAILABLE

BAM controls the physical representation of sequential (SQ) and word addressable (WA) files. AAM controls the physical representation of indexed sequential (IS), direct access (DA), and actual key (AK) files. AAM file organizations and processing operations are detailed in the AAM reference manual and user's guide.

Table 1-1 summarizes various file characteristics and indicates which BAM file organizations apply.

SEQUENTIAL FILES

A sequential file is a collection of records stored in the same physical order in which they were generated. A sequential file has no associated pointers or indexes; consequently, a specific record is located by reading the file sequentially from beginning-of-information until it is found.

Sequential file organization is best suited to applications that require large portions of the file to be accessed. Large sequential files are generally written on magnetic tape; small sequential files usually reside on mass storage devices.

WORD ADDRESSABLE FILES

A word addressable file is a collection of contiguous computer words stored on disk. This type of file is restricted to disk storage because records are typically accessed in random order. Each word has an ordinal, called a word address, indicating its offset from the origin of the file. By stating a word address, the contents of that word (or that word and those following) can be accessed. With word addressable file organization, you can preassign locations to records as they are written, leaving space for anticipated records not yet available.

Word addressable file organization is best suited to applications handling data items that have serial numbers or that are numbered in sequential order. The information to be associated with each item is placed in a word directly related to the sequence as the information becomes available. Information can be retrieved randomly by word address; information can also be retrieved sequentially, providing records are contiguous.

TABLE 1-1. BAM FILE ORGANIZATION CHARACTERISTICS

Characteristic	SQ	WA
Access FIT	x	x
Close files	x	x
D type records	x	
Disk storage medium	x	x
Error processing	x	x
F type records	x	x
File positioning	x	
Label processing	x	
Open files	x	x
Owncode exits	x	x
R type records	x	
Random access		x
Read complete records	x	x
Read partial records	x	
Replace records	x	x
S type records	x	
Sequential access	x	x
T type records	x	
Tape storage medium	x	
Terminal file processing	x	
U type records	x	x
Variable-length blocks	x	
Variable-length records	x	x
W type records	x	x
Write boundary marks	x	
Write complete records	x	x
Write partial records	x	
Z type records	x	

BAM LOADING

BAM routines provide all input and output between a referenced sequential or word addressable file and the operating systems that physically read or write the file on a storage device. To reduce field length, BAM is divided into functional capsules that are loaded by controlling routines at execution time. The controlling routines transfer control to the Fast Dynamic Loader (FDL), which locates and loads the capsules needed to process the FORTRAN call or COMPASS macro call. Capsules are systematically loaded when needed and remain in memory as long as they are required by any open file. Open and label processing capsules, for example, are unloaded when their operations are interrupted by an operation not associated with open processing. For optimum efficiency in loading, the open processing for all files should be completed before other processing is specified.

BAM AND FORTRAN

FORTRAN direct calls to BAM allow sequential files to be processed with features not available through the standard READ and WRITE statements or through calls to READMS and WRITMS. The features are as follows:

- Reading and writing partial, as well as complete records
- Passing control to a user subroutine when section, partition, or file boundaries are read
- Passing control to a user subroutine when a fatal or nonfatal error occurs
- Processing under user control at end-of-volume
- Writing section and partition boundaries
- Processing magnetic tape labels through a user's COMPASS-coded subroutine
- Skipping records
- Managing buffers
- Closing files prior to end-of-program execution

You must create and process word addressable files through calls to BAM. You cannot establish or process them through the FORM or Sort/Merge utilities.

When BAM is called directly, the following conditions must be met for each file:

- An array must be dimensioned as 35 words for the file information table (FIT). This array identifies the referenced file for all other BAM calls.
- The file organization must be identified by a call to one of the following subroutines:

  FILESQ Sequential organization

  FILEWA Word addressable organization

- A CALL FILExx statement must be executed before any other BAM call. Parameters in the call to FILExx establish FIT fields to guide processing.
- The file must be opened by a call to OPENM.
- All read and write operations for the file must occur through BAM calls. READ and WRITE or other standard input or output statements, including READMS and WRITMS, must not be used unless you first close the file by a call to CLOSEM.
- After all processing, the file must be closed by a call to CLOSEM to ensure file integrity.

Records are written to the file from an array or variable of any type identified by the working storage area (WSA) field of the FIT. You can change this array with each write.

Any file defined to BAM through a CALL FILExx statement must not appear on the PROGRAM statement. If such a file did appear, tables and a buffer would be allocated yet never used, and possible complications in file name use in subroutines might occur.

The FILExx routines reside on system library SYSLIB; the other BAM routines reside on BAMLIB. FILExx routines reference BAMLIB when they are executed. Therefore, if you do not use a FILExx routine, you must include one of the following to reference BAMLIB:

- A LIBRARY,BAMLIB. statement before execution
- An LDSET,LIB=BAMLIB. statement in the load sequence for the job step

Table 1-2 summarizes the FORTRAN direct call statements to BAM.

BAM AND COMPASS

COMPASS macro calls to BAM represent an alternative to the READ, WRITE, or other macros previously available that used the CPC (central program control) routine to execute. BAM and CPC cannot both be used to process a given file in one program. Both sets of macros can be used in the same program only as long as they are processing different files. Existing files created through CPC can be processed by BAM once the file and record structure are properly defined in BAM terminology.

All COMPASS macro calls to BAM reside on COMPASS system text IOTEXT, which you must specify with the S=IOTEXT parameter on the COMPASS control statement at assembly time.

When BAM is called directly, the program must meet the following conditions for each file:

- The FILE macro must appear in a nonexecutable portion of the program. The macro is an assembly time statement that results in construction of the FIT. The FIT address, rather than the file name, is used in all BAM macro references to the file.
- The file must be opened by an OPENM macro.
- All read and write operations for the file must occur through BAM macros.
- After all processing, the file must be closed by a CLOSEM macro to ensure file integrity.

Table 1-3 summarizes the COMPASS macro calls to BAM.

TABLE 1-2. SUMMARY OF FORTRAN CALLS

Function	Call Name	Applicable File Type	Action Taken	Comments
File creation and maintenance	FILExx	SQ, WA	Creates a file information table (FIT).	Must be the first call executed. Any file name defined on this statement must not appear on the PROGRAM statement.
	IFETCH	SQ, WA	Retrieves the value of a specified field in the FIT.	Can precede an OPENM call.
	STOREF	SQ, WA	Sets a value in a FIT field.	Can precede an OPENM call.
	FITDMP	SQ, WA	Dumps the contents of a FIT to the error file.	Forces the EFC FIT field to 2 or 3.
File initialization and termination	OPENM	SQ, WA	Opens a file.	Tape labels are processed providing the file is rewound.
	CLOSEM	SQ, WA	Closes a file.	IFETCH, STOREF, and FITDMP can follow a CLOSEM call.
Data transfer	GET	SQ, WA	Reads a record.	For WA files, word address is automatically incremented after the read.
	GETP	SQ	Reads a partial record.	Not valid for R type records.
	PUT	SQ, WA	Writes a record.	For WA files, writing always begins on a word boundary.
	PUTP	SQ	Writes a partial record.	Not valid for R type records.
File updating	REPLC	SQ	Replaces a record.	Valid only for mass storage files with block type C and record type F or W.
File positioning	SKIP	SQ	Skips records forward or backward.	An output file can be positioned backward only. Skipping logical records backward is not valid for D, R, T, and U type records or K and E type blocks. Skipping logical records forward is not valid for U type records.
	REWND	SQ, WA	Rewinds a file.	Unlabeled or nonstandard labeled tape files rewind to the beginning of the current volume. Mass storage or standard labeled tape files rewind to beginning-of-information.
Boundary conditions	ENDFILE	SQ	Writes an end-of-partition terminator.	End-of-partition is synonymous with operating system end-of-file.
	WEOR	SQ	Writes an end-of-section terminator.	End-of-section is synonymous with operating system end-of-record. Used to terminate an S type record being constructed by PUTP.
	WTMK	SQ	Writes a tapemark.	Recommended for user tape labels only.

TABLE 1-3. SUMMARY OF COMPASS MACRO CALLS

Function	Macro Name	Applicable File Type	Action Taken	Comments
File creation and maintenance	FILE	SQ, WA	Creates a file information table (FIT).	Must appear in a nonexecutable portion of the program.
	FETCH	SQ, WA	Retrieves the value of a specified field in the FIT.	Code expansion destroys values in some user registers. Can precede an OPENM call.
	STORE	SQ, WA	Sets a value in a FIT field.	Code expansion destroys values in some user registers. Can precede an OPENM call.
	SETFIT	SQ, WA	Sets values in fields of the FIT with values supplied through the FILE control statement.	Must precede an OPENM call. Values in all user registers are destroyed.
	FITDMP	SQ, WA	Dumps the contents of a FIT to the error file.	Forces the EFC FIT field to 2 or 3.
File initialization and termination	OPENM	SQ, WA	Opens a file.	Tape labels are processed providing the file is rewound.
	CLOSEM	SQ, WA	Closes a file.	Should be the last macro issued for a file.
Data transfer	CHECK	SQ, WA	Checks I/O completion status and places an active job in recall.	Must follow a GETWR or PUTWR before other BAM operations on that file. Because end-of-data and error exits are suppressed, the FP and ES FIT fields should be checked after control returns.
	CHECKR	SQ, WA	Checks I/O completion status and returns control to the user without placing an active job in recall.	
	GET	SQ, WA	Reads a record.	For WA files, word address is automatically incremented after the read.
	GETP	SQ	Reads a partial record.	Not valid for R type records.
	GETWR	SQ	Reads data in units of words.	Must be followed by a CHECK or CHECKR call.
	PUT	SQ, WA	Writes a record.	For WA files, writing always begins on a word boundary.
	PUTP	SQ	Writes a partial record.	Not valid for R type records.
	PUTWR	SQ	Writes data in units of words.	Must be followed by a CHECK or CHECKR call.
File updating	REPLACE	SQ	Replaces a record in a file.	Replacement record must be the same size as the record replaced.

TABLE 1-3. SUMMARY OF COMPASS MACRO CALLS (Contd)

Function	Macro Name	Applicable File Type	Action Taken	Comments
File positioning	SKIP	SQ	Repositions a file backward or forward.	An output file can be positioned backward only. Skipping logical records backward is not valid for D, R, T, and U type records or K and E type blocks. Skipping logical records forward is not valid for U type records.
	REWINDM	SQ, WA	Rewinds a file.	Unlabeled or nonstandard labeled tape files rewind to beginning of current volume. Mass storage or standard labeled tape files rewind to beginning-of-information.
Boundary conditions	ENDFILE	SQ	Writes an end-of-partition terminator.	End-of-partition is synonymous with operating system end-of-file.
	WEOR	SQ	Writes an end-of-section terminator.	End-of-section is synonymous with operating system end-of-record. Used to terminate an S type record being constructed by PUTP.
	WTMK	SQ	Writes a tapemark.	Does not flush the buffer.
User label processing	GETL	SQ	Reads the next label of a label group.	Labels are retrieved in sequential order.
	PUTL	SQ	Writes a label.	The label appropriate to the current file position is submitted to be written to the output file.
	CLOSEL	SQ	Terminates label processing.	Used to exit a label processing routine and return to the calling routine for continued processing.

All BAM files are processed in the same general manner. The file is opened, records are written or read, and the file is closed. File processing is affected by:

- File information table (FIT)

  The FIT established for the file defines the file structure and specifies other information pertinent to file processing.

- Record type

  The record type declared for the file is used in determining the format of records written to the file.

- Processing options

  Once selected, an option affects all subsequent file processing until the option selection is changed.

This section of the guide presents the basic concepts of file processing to familiarize you with the general principles involved in processing a BAM file. Subsequent sections discuss file processing in detail for each BAM file organization.

FILE INFORMATION TABLE

You must establish a file information table (FIT) for each file to be processed by BAM before the file can be opened. The contents of the fields in this table define the structure of the file and govern file processing. The FIT is a 35-word table that contains fields describing such information as record type, record size, file organization, error count, error flags, and information used by internal system routines to determine file status and position. You set some fields; BAM sets others.

FIT fields can be set when the FIT is constructed, when the file is opened, and when a processing statement is executed. When the file is opened, any field that has not been set to a specific value is set to a default value. A default value is in effect until the field is changed by a subsequent program statement. When a file processing statement is executed, BAM uses the current contents of the applicable FIT fields.

CREATING THE FIT

The FIT is constructed when a call to the FILExx subroutine is executed. The format is:

CALL FILExx(fit,field,value,. . .,field,value)

The mnemonic specified for xx determines the file organization.

- FILESQ specifies a sequential file and sets the file organization (FO) field in the FIT to SQ.
- FILEWA specifies a word addressable file and sets the file organization (FO) field in the FIT to WA.

The first parameter in the CALL FILExx statement is the name of the integer array in which the FIT is stored. The remaining parameters are FIT field mnemonics and values for the fields. When the CALL FILExx statement is executed, the FIT is established in the named array and the specified fields are set to the designated values.

All of the FIT fields required for file processing can be set by the CALL FILExx statement. FIT fields are identified by mnemonics such as RT (record type) and LFN (logical file name). Values for the fields can be program locations, positive integers, and symbolic options. Appendix D lists the FIT fields applicable to each file organization and indicates the fields that can be set by the CALL FILExx statement. More detailed discussions of the individual FIT fields can be found in the sections on the file organizations.

The first parameter in a CALL FILExx statement is a single entry that names the array to hold the FIT; all subsequent parameters in the statement are paired. The first parameter of a pair specifies the FIT field mnemonic, and the second parameter specifies the value for the field. Some examples of paired parameters are:

'LFN','NEWFILE'	Logical file name NEWFILE
'RT','F'	F type records
'FL',42	Fixed-length records of 42 characters

Pairs of parameters can appear in any order in the argument list. Figure 2-1 shows some examples of the CALL FILExx statement.

```
CALL FILESQ(SQFIT,'LFN','SQFILE','BT','C','RT',
  'F','FL',30)
```

The FIT for the sequential file is constructed in the array named SQFIT; the logical file name is SQFILE, the block type is C, the record type is F (fixed), and the record length is 30.

```
CALL FILESQ(SQ1FIT,'LFN','SQ1FILE','BT','C',
  'WSA',BFR,'DX',DATAEX)
```

The FIT for the sequential file is constructed in the array named SQ1FIT; the logical file name is SQ1FILE, the block type is C, the working storage area is BFR, and the end-of-data routine is DATAEX.

```
CALL FILEWA(WAFIT,'LFN','WAFILE','ERL',50)
```

The FIT for the word addressable file is constructed in the array named WAFIT; the logical file name is WAFILE, and the trivial error limit is 50.

Figure 2-1. CALL FILExx Statement Examples

Each field is assigned a default value when a value is not specified. You should be aware of each default value and the effect it can have on a program.

Any specified value that exceeds the maximum field size is truncated. Misspelled or otherwise unrecognizable FIT mnemonics are noted on the dayfile and ignored. In each of these cases, an informative diagnostic is issued. A mnemonic that is valid but not applicable to the specified file organization is ignored providing it produces no conflict with other mnemonics.

If the program is compiled without fatal errors and is loaded and executed, FIT field values are checked for validity and consistency. If the same field is referenced more than once, the last value specified is used.

The following fields are most frequently used in the CALL FILExx statement:

BT	Block type if file organization is sequential
RT	Record type (and additional fields required by the individual record type)
MRL	Maximum record length
LFN	Logical file name
WSA	Working storage area
ERL	Trivial error limit

USING THE FIT

Execution of a BAM file processing statement depends on the current contents of the FIT. You are responsible for setting appropriate fields with execution values. A field value is provided in one of several ways:

- Specify the field and value in the CALL FILExx statement. The value becomes part of the FIT at execution time.
- Omit the field definition and accept the default value. The default becomes part of the FIT at the time the file is opened.
- Specify the field as a parameter in the FILE control statement. The value becomes part of the FIT when the file is opened, and it overrides any previous value set in the field.
- Execute the CALL STOREF statement to store a value directly into the FIT. The field is set at the time the statement is executed.
- Set the field value in a file processing statement. A value from a CALL GET or CALL PUT statement is set in the FIT when the statement is executed.

The last value set in a FIT field governs the operation using that field. The default value remains in effect until changed.

FILE Control Statement

The FILE control statement is used at file open time to set values in FIT fields. The specified values can either set fields not previously set or override values specified in the CALL FILExx statement. Values from the FILE control statement are saved on an internal scratch file until the file is opened for the first time during execution. The FILE control statement can appear anywhere in the control statement portion of the job before the statement that calls for execution of the compiled program.

Some of the FIT fields that can be set by the CALL FILExx statement can be set by the FILE control statement. FIT fields that cannot be set by the FILE control statement are those that specify a program address. Values for the fields can be positive integers and symbolic options. Appendix D lists the FIT fields applicable to each file organization and indicates the fields that can be set by the FILE control statement.

The first parameter in the FILE control statement must be the logical file name. This is followed by one or more parameters that specify FIT field settings. A parameter is specified as a FIT field mnemonic and a value separated by an equal sign. Figure 2-2 shows some examples of the FILE control statement.

```
FILE(MYFILE,FO=SQ,MRL=80,PC=50B)
```

The sequential file has the logical file name MYFILE. The maximum record length (MRL) and padding character (PC) fields are specified for the file.

```
FILE(WAFILE,FO=WA,EFC=1,DFC=2)
```

The word addressable file has the logical file name WAFILE. The error file control (EFC) and dayfile control (DFC) fields are specified for the file.

Figure 2-2. FILE Control Statement Examples

A FILE control statement cannot be continued to a second card or line image. When all parameters cannot be specified in one 80-column statement, additional FILE control statements with the same logical file name can be included in the control statement portion of the job. If the same FIT field is referenced in more than one FILE control statement, the last value encountered defines the field. Overlapping fields are reported on the dayfile. Consider the following statements:

FILE,lfn.	Cancels preceding FILE control statements for lfn
FILE.	Cancels all preceding FILE control statements

An error in the FILE control statement causes the entire statement to be ignored. The error and the parameter in question are printed on the dayfile. Control transfers to the next EXIT.

CALL STOREF Statement

A FIT field value can be set or changed during program execution by the CALL STOREF statement. The format is:

```
CALL STOREF(fit,field,value)
```

This statement can be executed before or after the file is opened; it must be after the CALL FILExx statement. The record type (RT) field cannot be set after the file has been opened. Fields that can be set by the CALL STOREF statement are indicated for each file organization in appendix D.

Only one field in the FIT can be set by each CALL STOREF statement. The mnemonic for the field and the value to be stored in the field are specified in the statement. The value can be an integer, integer variable, or a symbolic option. Integer values are retained as integers in the FIT. Symbolic values become bit strings and are fetched as a single bit or an octal value, depending on the bit string length.

Figure 2-3 shows some examples of the CALL STOREF statement.

```
CALL STOREF(SQFILE,'RL',80)
```

Execution of this statement stores 80 in the RL (record length) field of the FIT in the array named SQFILE.

```
CALL STOREF(WAFILE,'ERL',100)
```

Execution of this statement stores 100 in the ERL (trivial error limit) field of the FIT in the array named WAFILE.

Figure 2-3. CALL STOREF Statement Examples

IFETCH Function

The value in a FIT field can be retrieved from the FIT by using the IFETCH integer function or by calling IFETCH as a subroutine. The formats are:

```
IFETCH(fit,field)

CALL IFETCH(fit,field,variable)
```

The word variable denotes an integer variable in which the FIT field value will be returned.

IFETCH can be used before or after the file is opened. Fields that can be retrieved by IFETCH are indicated for each file organization in appendix D. The format of the value returned depends on the type of field requested.

- A field that you specify as an integer is returned as a right-justified integer.
- A length field value is returned as a 6-bit character equivalent except for the buffer size (BFS) field, which is returned in words (as specified).
- A one-bit field value is returned as a positive or negative integer; the magnitude of the integer is undefined.
- A symbolic field value that requires more than one bit is returned as an integer. (Refer to appendix D for the integer values.)

Figure 2-4 shows some examples of IFETCH.

RECORD TYPES

You determine the amount of information in a record by specifying record type. Eight different record types are supported by BAM. Each record type is applicable to a specific situation. All records in a given file must be the same record type.

```
ICOUNT=IFETCH(SQFILE,'RL')
```

This statement returns the value of the RL (record length) field to the variable ICOUNT.

```
IF(IFETCH(MYFILE,'PD').EQ.3) GO TO 20
```

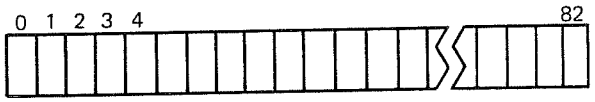
This statement causes a branch to statement 20 if the PD (processing direction) field is set to IO; the symbolic option IO is stored in PD as the bit string 011.

```
CALL IFETCH(WAFILE,'WA',IWORD)
```

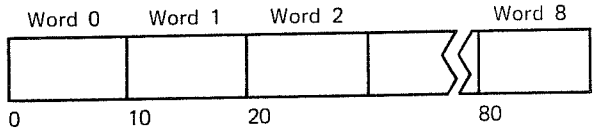
This statement returns the value of the WA (word address) field to the integer variable IWORD.

Figure 2-4. IFETCH Examples

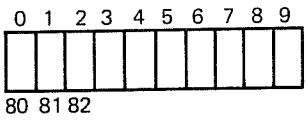
In addition to the record type (RT) field in the FIT, various other fields are required depending on the record type. FIT fields related to the length of a record or of a field within the record are specified as a number of characters, counting from 1. Fields indicating the position of a word or of a character within a word are specified as the word or character position, counting from 0. Figure 2-5 illustrates the numbering conventions used to describe a record and the position of a field in a record.



Character positions 0 through 82 contain a record with 83 characters.



Words 0 through 8 are required to hold the 83-character record.



Character positions 0, 1, and 2 in word 8 contain the last three characters of the record, which are record character positions 80, 81, and 82. Character positions 83 through 89 are undefined.

Figure 2-5. Numbering Conventions

DECIMAL CHARACTER COUNT, D TYPE RECORDS

D type records each contain a field that specifies the exact number of characters in that record. This type of record is useful when record lengths vary widely due to the varying amount of information to be recorded for a specific item. For example, a file could have D type

records when each record contains data related to telephone number activity; records vary widely in length because each telephone can be used to make a different number of calls. You can specify D type records for sequential files, but not for word addressable files.

Four FIT fields are required to describe D type records.

RT — Record type. This FIT field must specify D type records.

LP — Length field beginning character position. This FIT field must specify the record character position in which the length field begins, starting with character position 0 of the record.

LL — Length field length. This FIT field must specify the number of characters (1 through 6) in the field designating the number of characters in the record.

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for both a read and a write operation; it must be set before a record can be processed.

The length field defined by the LP and LL fields normally contains a right-justified display code value with zero or blank fill. The field can contain a binary value if the C1 field is set; the field can contain a sign overpunch value if the SB field is set.

If you set the minimum record length (MNR) field, it must include the length field and should be at least 10 characters. The default value set by BAM is the sum of the values in the LP and LL fields.

When writing a D type record, BAM uses the contents of the length field (defined by the LP and LL fields) to determine how many characters to write. Any user-specified value for the record length (RL) field is ignored. When reading a D type record, BAM again uses the length field to determine record length. At the completion of a successful read, the RL field is set to the record length.

FIXED LENGTH, F TYPE RECORDS

F type records always contain the same number of characters in every record. This is the simplest record type. It is useful in applications where all records in the file contain the same amount of information. For example, an inventory file could have F type records when each record contains the part number, normal supply value, and current supply value. You can specify F type records for sequential and word addressable files.

Two FIT fields are required to describe F type records.

RT — Record type. This FIT field must specify F type records.

FL — Fixed length. This FIT field must specify the number of characters in every record.

When writing an F type record, the number of characters specified by the FL field are always written. A longer user-supplied record is truncated; a shorter record is not recognized even if the write request specifies a record length less than the FL field value. You must add blank or zero fill if the number of significant characters in the record is less than the number specified by the FL field.

RECORD MARK, R TYPE RECORDS

R type records are terminated by a special character known as a record mark. You select the record mark character, which can be any character in the character set. The selected character can appear in the record only as the terminating character. R type records can be used to conserve storage space when record length varies disparately. For example, R type records could be used for a file containing records that list the machines an employee can operate; record lengths would vary considerably because employee skills and machine names also vary in length. You can specify R type records for sequential files, but not for word addressable files.

You must select R type records with discretion because each time a record is read or written, every character of the record must be examined in a search for the terminator. This process is very inefficient.

Three FIT fields are required to describe R type records.

RT — Record type. This FIT field must specify R type records.

RMK — Record mark character. This FIT field indicates the terminating character for each record (default is the right bracket]).

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for both a read and a write operation; it must be set before a record can be processed.

The last character of every record in the file must be the record mark you supply as part of the data record. When an R type record is read, the record mark is returned as part of the record and the record length (RL) field is set to the total record length, including the record mark. Any user-supplied value for the RL field is ignored when the record is read or written. If the minimum record length (MNR) field is set, it should be at least 10 characters.

The values for the RMK field can be specified in one of three formats that ultimately translate to the display code equivalent of the desired character.

O"nn" — The two-digit octal value of the desired record mark character is specified directly.

dd — The octal display code value is specified by the decimal value equivalent.

R"x" — The specific character is given for BAM to translate to display code; this format is not valid in a FILE control statement.

If RMK is set to zero, the default record mark character (the right bracket) terminates the record.

SYSTEM, S TYPE RECORDS

S type records have certain characteristics depending on whether the file exists on a PRU device or on an S or L tape.

- On a PRU device (disk, SI tape, I tape), each record occupies an integral number of central memory words and is terminated by a system-supplied terminating marker. Data grouping consists of one or more PRUs terminated by a short PRU or zero-length PRU. An S type record is equivalent to a logical record under NOS and a system-logical-record under NOS/BE.
- On an S or L tape, each record is a tape physical record terminated by an interrecord gap.

S type records are used by systems programmers more often than by applications programmers. This record type is used when files must interface with the operating system without the benefit of BAM. If BAM is available for reading a file, other record types are preferable. You can specify S type records for sequential files, but not for word addressable files.

S type records exist by default on the following files:

- Binary files written by compilers or other members of the product set that do not use CYBER Record Manager
- Files written by BUFFER OUT in FORTRAN
- Files written by COMPASS when BAM macros were not used and disposition was not for unit record equipment
- Files written by utility programs such as Update

Three FIT fields are required to describe S type records on PRU or S/L devices.

RT — Record type. This FIT field must specify S type records.

RL — Record length. This FIT field specifies the actual number of characters to be written. The field is required for a write operation only. After a record is read, RL contains the number of characters read.

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for a full record read; it is ignored for a partial read. If MRL is set to zero, records of any size can be written. If MRL is set to a value and that value is exceeded, an excess data error occurs.

On PRU devices, S type records always occupy an integral number of words of storage. BAM rounds upward any user value for record length to a multiple of 10 characters if necessary. The terminating marker that the operating system supplies to delineate the end of system-logical-records on PRU devices is not a part of the user record in the working storage area or buffer.

On an L tape, S type records can have any number of characters. On an S tape, S type records cannot exceed S tape block size.

TRAILER COUNT, T TYPE RECORDS

T type records each consist of a fixed-length header followed by a variable number of fixed-length trailer items. This record type is useful in situations where the amount of information known about each item is the same but where it is not known how many items a given record will have. For example, a file containing census data could have information related to the family as a whole in the fixed-length base and information related to an individual family member in each trailer item. You can specify T type records for sequential files, but not for word addressable files.

Six fields in the FIT are required to describe T type records.

RT — Record type. This FIT field must specify T type records.

HL — Header length. This FIT field specifies the number of characters in the fixed-length header portion of the record.

CP — Trailer count beginning character position. This FIT field specifies the character position (counting from 0) within the fixed-length header in which the trailer count field begins.

CL — Count field length. This FIT field specifies the number of characters (1 to 6) in the count field.

TL — Trailer length. This FIT field specifies the number of characters in one trailer item.

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for both a read and a write operation; it must be set before a record can be processed.

The value of the count field described by the CP and CL fields is assumed to be decimal, right-justified, and display code zero or blank filled. If the value of the count field is a binary integer, the COMP-1 (C1) field must be set. The sign-overpunch (SB) field must be set if the value in the count field is a sign-overpunch value.

The length specified by the HL field is the logical minimum record length; the field must be specified explicitly. The count field must be within the header portion of the record; therefore, the sum of the values in the CP and CL fields cannot be greater than the value in the HL field. The minimum record length should be at least 10 characters.

Because BAM determines record length from the HL, TL, CP, and CL fields, you should not set the record length (RL) field. BAM sets the RL field to the number of characters read or written after successfully processing a record.

UNDEFINED, U TYPE RECORDS

U type records have a format that differs from the other record types supported by BAM. This record type is most commonly specified when an existing file has records that do not correspond to any other record type supported by BAM. You can specify U type records for sequential and word addressable files.

BAM makes no assumptions about the contents of U type records. No search for record delimiters or control words is performed; if such delimiters exist, BAM assumes they are part of the data.

Three FIT fields are required to describe U type records.

RT — Record type. This FIT field must specify U type records.

RL — Record length. This FIT field specifies the number of characters to be read or written. The field is required for both read and write operations.

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for a read operation; it is not required for a write operation.

CONTROL WORD, W TYPE RECORDS

W type records are prefixed with a record control word supplied by BAM. This word indicates the user data record size and the size of the preceding record, and holds flags that BAM uses to maintain file position and check parity. The control word is written at all block boundaries. You can specify W type records for sequential and word addressable files.

Variable-length W type records are packed and, therefore, conserve space. This record type provides the fastest access for variable-length records since BAM does not search for end-of-record marks or check record count fields. Record type defaults to W for both sequential and word addressable files.

Four FIT fields are required to describe W type records.

RT — Record type. This FIT field must specify W type records (default).

RL — Record length. This FIT field specifies the number of characters to be written.

MRL — Maximum record length. This FIT field specifies the maximum number of characters to be moved to the working storage area. The field is required for both a read and a write operation.

CM — Conversion mode. This FIT field must be set to NO (default) to disallow character conversion, which would destroy control word information.

The RL field must be set for each record written. At the completion of a write operation, BAM stores the record length in the control word. The default value for RL is 0, which is an acceptable length for W type records. If you inadvertently omit the RL parameter, BAM writes a control word showing a user record length of zero characters and returns to the user program without comment.

Record length is not required for reading records because BAM has this information in the control word for each record. With the exception of a partial read request on a sequential file, any record length in the read request is ignored. At the completion of a successful read, the RL field in the FIT is set to the number of characters just read.

ZERO BYTE, Z TYPE RECORDS

Z type records are terminated by 12 bits of zero in character positions 8 and 9 of the last central memory word in the record. Because the end of a record is marked, BAM automatically discards full words of nonsignificant blanks from a record to reduce mass storage use. You can specify Z type records for sequential files, but not for word addressable files.

Two fields in the FIT are required to describe Z type records.

RT — Record type. This FIT field must specify Z type records.

FL — Full length. This FIT field specifies the maximum number of characters in any record.

When Z type records are being read, BAM sets the record length (RL) field to the number of characters read, not including blank padding. It is not necessary to specify a value for the RL field when Z type records are being written; however, by setting the RL field, you can speed processing as BAM attempts to minimize storage use and suppress blanks at the end of the user record. When a record is written, the following occurs:

- If the value of the RL field is not zero, BAM determines the end of the record by searching backward from RL for the first nonblank character. The zero byte is added in the last two character positions; binary zeros are stored between the last significant character and the first character of the zero byte.

- If the value of the RL field is zero, BAM determines the end of the record by searching backward from FL for the first nonblank character. The zero byte is added in the last two character positions; binary zeros are stored between the last significant character and the first character of the zero byte.

If the installation is using a 64-character set, you should avoid two adjacent colons in data records or ensure that the colons are not aligned as the two lower characters in a central memory word. Because the display code equivalent of these characters is also 12 bits of zero, the colons would mistakenly signal the end of the record if they occupied character positions 8 and 9 in any word within the record. If a record ends with a colon, BAM appends one blank to preserve the colon and distinguish it from binary zero fill.

SUMMARY OF RECORD TYPES

Record types are summarized in table 2-1. The listing includes required FIT fields and, where appropriate, the actual value to be entered.

TABLE 2-1. SUMMARY OF RECORD TYPES

Record Type	Description	Required FIT Fields	Restrictions
D (decimal count)	Each record contains a field that specifies the exact number of characters in that record.	RT D LP Length field character position LL Length field length MRL Maximum record length	Used with sequential files only. Illegal for backward skip. Illegal with REPLC.
F (fixed length)	Each record in the file contains the same number of characters.	RT F FL Fixed length	Backward skip requires record length to be a multiple of 10 characters when block type is C.
R (record mark)	Each record is terminated by a record mark.	RT R RMK Record mark character (default is]) MRL Maximum record length	Used with sequential files only. Illegal with GETP and PUTP. Illegal for backward skip. Illegal with REPLC.
S (system)	Each record on a PRU device is a system-logical-record; each record on an S or L tape is a tape physical record.	RT S RL Record length (write only) MRL Maximum record length (read only)	Used with sequential files only. Illegal with K or E blocking. Illegal with REPLC.
T (trailer count)	Each record consists of a fixed-length header followed by a variable number of fixed-length trailer items.	RT T HL Header length CP Trailer count CL Count field length TL Trailer length MRL Maximum record length	Used with sequential files only. Illegal for backward skip. Illegal with REPLC.
U (undefined)	Each record has a format that differs from other record types supported by BAM.	RT U RL Record length MRL Maximum record length (read only)	Illegal with REPLC. Illegal with SKIP. Illegal with GETP... 'SKIP'.
W (control word)	Each record is prefixed with a record control word.	RT W (default) RL Record length (write only) MRL Maximum record length CM NO (no character conversion)	Illegal with K or E blocking.

TABLE 2-1. SUMMARY OF RECORD TYPES (Contd)

Record Type	Description	Required FIT Fields	Restrictions
Z (zero byte)	Each record is terminated by 12 bits of zero in the low-order position of the last word in the record.	RT Z FL Full length	Used with sequential files only. Illegal with REPLC.

OWNCODE PROCESSING

Owncode processing includes several options that you can select or change at the time the file is created or at any time during subsequent file processing. Once you select one of these options, it affects all subsequent file processing until you change the option by resetting the appropriate FIT field.

End-of-data exit and error exit owncode processing apply to both sequential and word addressable file organizations. Label exit owncode processing applies only to sequential file organization. Each option is discussed in the following paragraphs.

END-OF-DATA EXIT

End-of-data exit processing applies to sequential and word addressable file organizations. You specify the option by setting the end-of-data exit (DX) field in the FIT. Defining an end-of-data exit allows you to gain control when an end-of-data condition is encountered. An end-of-data condition occurs when a sequential read on a sequential or word addressable file encounters end-of-information; when a sequential read on a sequential file encounters an end-of-section or an end-of-partition; when a forward skip on a sequential file encounters end-of-information; or when a backward skip on a sequential output file encounters beginning-of-information. The file position (FP) field in the FIT is set to 1 for a beginning-of-information condition and to 100_8 for an end-of-information condition.

The name of a user subroutine is specified for the DX field through the CALL FILExx or CALL STOREF statements. For sequential files only, you can specify an end-of-data exit in a read or partial read request.

Figure 2-6 illustrates the format of a user subroutine that processes an end-of-data condition. The subroutine must be declared EXTERNAL, and labeled or blank common should be used to pass information between the main program and the owncode subroutine. No parameters can be passed. Exit is made through a RETURN statement, which passes control to the calling program at the statement following the BAM call that resulted in the end-of-data exit.

When end-of-data exit is taken for end-of-information, continued attempts to read without repositioning the file cause an error condition. Control is then transferred to the error exit, if one is specified, instead of to the end-of-data exit. A trivial error condition is produced if continued skipping is attempted after end-of-data occurs.

End-of-data exit processing is discussed in more detail in section 5.

```
      PROGRAM DATAEX
      EXTERNAL OWNCODE
      .
      .
      .
      CALL FILESQ(SQFIT,'DX',OWNCODE, . . . )
      .
      .
      .
      END

      SUBROUTINE OWNCODE
      .
      .
      .
      RETURN
      END
```

Figure 2-6. End-of-Data User Subroutine Setup

ERROR EXIT

Error exit processing applies to sequential and word addressable file organizations. You specify the option by setting the error exit (EX) field in the FIT. Defining an error exit allows you to transfer control to a recovery subroutine after a fatal or trivial error occurs. If the error is fatal, no further input/output can be performed on that file; any such attempt causes the job to terminate.

The name of a user subroutine is specified for the EX field through the CALL FILExx or CALL STOREF statement. For word addressable files only, you can specify an error exit in a read request. For both sequential and word addressable files, you can specify an error exit in a write request.

Figure 2-7 illustrates the format of a user subroutine that processes an error condition. The subroutine must be declared EXTERNAL, and labeled or blank common should be used to pass information between the main program and the error exit subroutine. No parameters can be passed. Exit is made through a RETURN statement, which passes control to the calling program at the statement following the BAM call that resulted in the error exit.

Error processing is discussed in more detail in section 5.

LABEL EXIT

Label exit processing applies only to sequential file organization. You specify the option by setting the label exit (LX) field in the FIT. Defining a label exit allows you to transfer control to a user label subroutine during open and close processing of a tape file.

```
      PROGRAM ERROR
      EXTERNAL ERREX
      .
      .
      .
      CALL FILEWA(WAFIT,'LFN','WAFILE','EX',ERREX)
      CALL OPENM(WAFIT)
      CALL GET(WAFIT,BFR)
      .
      .
      .
      END

      SUBROUTINE ERREX
      .
      .
      .
      RETURN
      END
```

Figure 2-7. Error Exit User Subroutine Setup

The name of a user subroutine is specified for the LX field through the CALL FILESQ or CALL STOREF statement. The subroutine must be coded in COMPASS because the COMPASS macros GETL, PUTL, and CLOSEL provide for user label processing. You are responsible for returning control to BAM for continuation of open and close processing.

The user label processing (ULP) field must be set to any value other than NO. The open flag (OF) field must be set to R; this effects a file rewind on open and allows the label to be accessed. One subroutine need not account for all labels used in processing. Because the LX parameter can be reset, separate routines can be executed for various file positions. Label processing is discussed in more detail in section 3.

GENERAL FILE PROCESSING

Follow the same general procedures for all BAM files being accessed by direct calls:

1. Construct the FIT.
2. Define a working storage area.
3. Open the file.
4. Process the file.
5. Close the file.

When a BAM file processing statement is executed, parameters in the statement are placed in their respective FIT fields. Omitted parameters do not affect the current FIT values.

ESTABLISHING THE FIT

You must establish the FIT before any other references are made to the file. Detailed construction of the FIT in the proper format is performed by BAM when the CALL FILExx statement is executed. You must provide FIT field values for the logical file name (LFN) field and any fields for which the default value is not to be used. Specific fields required for file processing depend on the file organization and are discussed in the sections on file processing.

FIT field values supplied in a FILE control statement become part of the FIT when the file is opened. Fields not specified receive default settings as listed in table 2-2. Once a value for a given FIT field is changed, the default no longer applies.

TABLE 2-2. PARTIAL LIST OF FIT FIELD DEFAULT VALUES

Category	Field and Type	Default Meaning
File structure	FO=SQ	Sequential file organization
	ASCII=0	64-character display code
	BT=I	Block type I
	MBL=5120	Maximum block size 5120 characters
	MRL=0	No maximum record length
	RT=W	Record type W
File processing	CF=R	File rewind after close
	CM=NO	No code conversion for tape read/write
	OF=R	File rewind before open
	PD=INPUT	File open for read only
	VF=U	Tape volume unload after close volume
Owncode exits	DX=0	No user owncode for end-of-data processing
	EX=0	No user owncode for error processing
	LX=0	No user owncode for label processing
Error processing	DFC=0	Except for fatal errors, no dayfile messages
	EFC=0	No error file messages
	EO=T	File terminate on parity error
	ERL=0	Unlimited trivial errors

NOTE: Other default values can exist under certain circumstances.

DEFINING THE WORKING STORAGE AREA

The working storage area is a user-defined area where BAM finds or returns one record for a write or read operation. The address of this area is stored in the working storage area (WSA) field in the FIT. The working storage area is typically a one-dimensional character or integer array. If specified as a CHARACTER type data structure, WSA must be word-aligned.

Figure 2-8 shows the events that occur in response to your request to write a record. BAM moves the record from the working storage area into the file buffer in central memory. When the buffer is full, or at other times appropriate to processing, BAM makes a request to the operating system to transfer the buffer contents to a storage device.

In response to your request to read a record, BAM moves one record from the buffer into the working storage area. If the record is not in the buffer, the operating system is requested to transfer the proper block of information to the buffer so that your request can be carried out. You are aware only that records are moved from and to the working storage area upon request. All other record or block movement is internal to BAM.

You can change the working storage area anytime during execution of the program. Each record written can originate in a different area of the program. You need not move a record to a specific area for writing. Any address set in the WSA field by the write request, or stored in the WSA field before a write request, is used during execution of the write request.

BAM considers the working storage area to have a length of MRL characters. Any attempt to read more than MRL characters results in a trivial error 142_8 with MRL characters moved to WSA and the file positioned at the beginning of the next record.

A full record is returned for each read request (except in the special case of a partial record read on a sequential file). The number of characters in the record is always returned to you in the RL field in the FIT. Because the working storage area is comprised of an integral number of words, only the specific number of characters that are a part of the record can be considered valid.

OPENING THE FILE

You must open the file with a call to OPENM before any file processing can be performed. Open processing includes the following events in the order listed:

1. FIT fields specified on the CALL OPENM statement are set.
2. FILE control statement parameters are placed in FIT fields. FILE statement values can override values previously set by the CALL FILExx or CALL STOREF statement.
3. For an existing file, file descriptions are extracted from the FSTT and stored in the FIT. FSTT values can override previous FIT values.

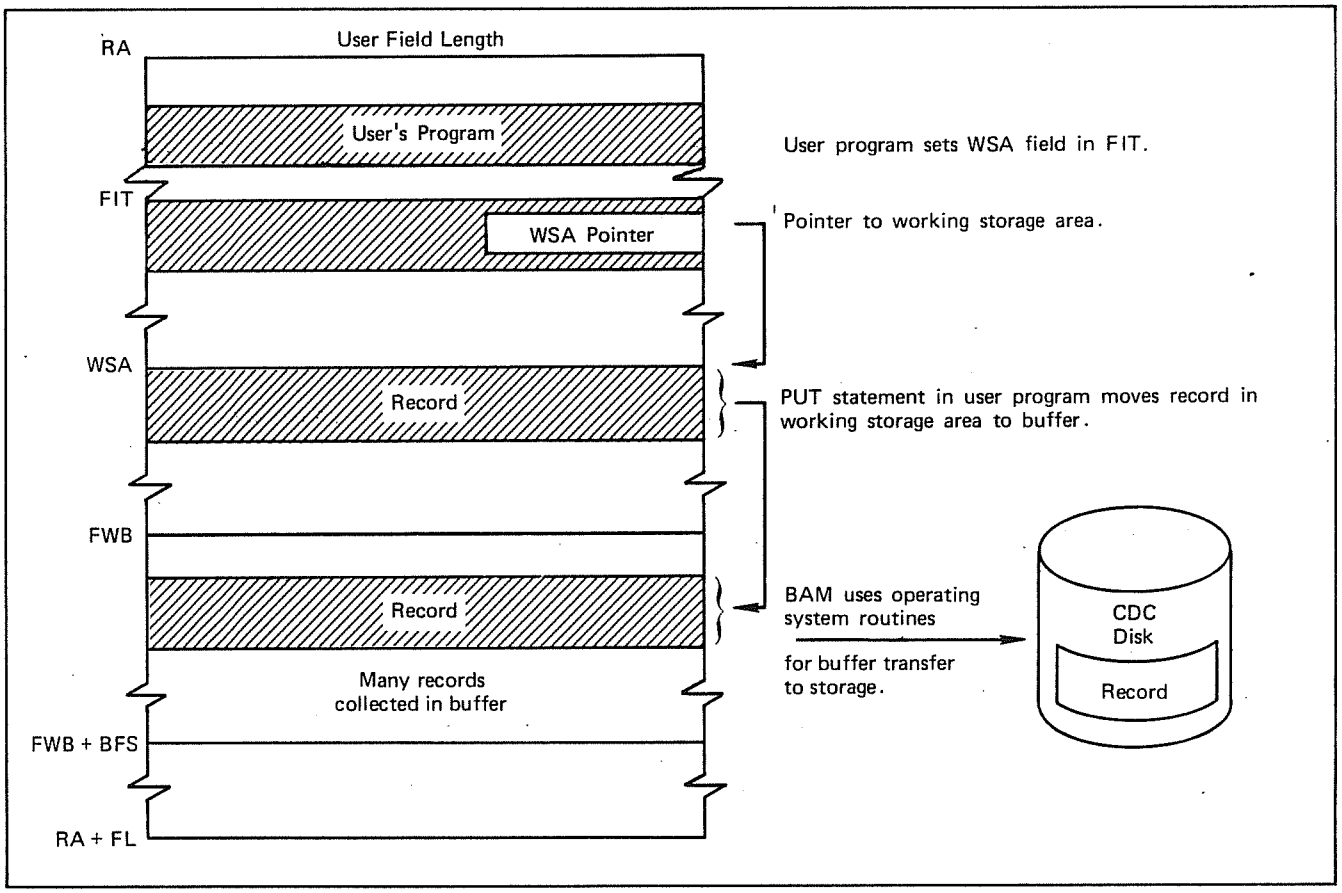


Figure 2-8. Working Storage Area Use

4. Minimum parameters required by the file organization are checked.

5. FIT fields are checked for consistency in logic.

6. Buffer space is calculated and reserved by BAM unless you have allocated the space.

7. The open/close flag (OC) field in the FIT is set to opened.

8. Any labels on magnetic tape files are processed.

If BAM detects an error in format or an omission or inconsistency in logic in the FIT fields, an appropriate diagnostic is issued. This frequently occurs when all fields required for a particular file organization have not been defined correctly or when a specified field precludes another specified field. An error detected during open processing prevents the open/close flag (OC) field from being set to opened.

PROCESSING THE FILE

Records in a BAM file can be read, written, and replaced. Sequential access is supported for both sequential and word addressable files; random access is supported for word addressable files.

The processing direction (PD) field in the FIT determines the operations that can be performed on the file. For reading records, set the PD field to either INPUT or IO; for writing records, set it to either OUTPUT or IO.

Reading Records

Sequential files are read by position (sequentially). A full read accesses the next record in logical sequence in the file. A partial read accesses a specified number of characters in logical sequence from the present position within the file. When a full read operation is performed, BAM returns the full record to the working storage area defined by the WSA field in the FIT. When a partial read operation is performed, BAM returns the specified number of characters; characters from PTL to the next word boundary are undefined.

You can read word addressable files sequentially or by word address (randomly) with full records only. A sequential read accesses the next record in logical sequence in the file. A random read accesses the record associated with the word address (WA) field of the FIT; you can set the word address value by using a CALL STOREF statement prior to the read or by including the value in the read request. When a read operation is performed, BAM returns the full record to the working storage area defined by the WSA field and resets WA to the word following the record.

At the end of the read operation, the record length (RL) field in the FIT is set by BAM. If no error occurs, the RL field reflects the number of characters returned to the working storage area; it is equal to the number of characters required by user setting or record control fields.

The number of characters transferred to the working storage area during any read operation is affected by the maximum record length (MRL) field in the FIT as well as by any other control information pertinent to the record. With the exception of S type records on disk, the value in the MRL field sets an absolute limit on data transfer as follows:

- If the MRL field is greater than zero, the specified value sets an upper limit on the number of characters that can be read, even if this limit is smaller than a given record.

- If the MRL field is zero and S type records stored on a PRU device are being read, records of any length can be read by a partial read operation. For any other type record, an MRL field of zero indicates no data can be transferred to the working storage area.

Writing New Records

A write operation causes a record to be moved from a working storage area to the buffer for the file being processed. The length of the record written is determined by the record length (RL) field in the FIT or other control information appropriate to the record type.

Sequential files are written sequentially. The record in the working storage area when the first write operation is executed becomes the first physical record in the file. The second write operation produces the second record, which is written directly following the first.

For sequential files, you can write a partial record. You must specify the number of characters to be written in the partial write request. For each request, data is transferred from the beginning of the working storage area to the file. A partial write is particularly useful for constructing a single record from information residing several places in storage. By changing the working storage area address for each call, you can gather data from several sources. You must identify the end of the record with a record length specification appropriate to the type of record under construction or with a write end-of-section request when record type is S.

You can write word addressable files randomly or sequentially with full records only. Each record begins at the address specified in the word address (WA) field of the FIT. You can write records in sequential order because BAM sets the WA field to the address of the next available word in the file after each write; when this address is accepted as the word address for the next succeeding write, records can be written sequentially. You can write records in random order by changing the word address for each write.

UPDATING THE FILE

You can update a sequential mass storage file by replacing the record previously read with a record from the working storage area. The replacement record must have the same length as the record being replaced. If a replace request is not immediately preceded by a read of the record to be replaced, a trivial error results and the request is ignored.

You can update a word addressable file by overwriting a record stored at a particular word address with a record from the working storage area. The replacement record must have the same length as the record being replaced unless space was left to add to a record or space was left by a shorter record.

CLOSING THE FILE

At the end of processing, you must close the file. Close processing includes the following events:

- User data records for an output file are written to the file storage device from the central memory buffer. This event is called flushing the buffer.
- End-of-information is written if applicable to the file organization.
- Any labels on magnetic tape files are processed.
- The open/close flag (OC) field in the FIT is set to closed.

A trivial error occurs if you attempt to close a file that has never been opened or has already been closed.

Close processing also includes file positioning. The file can be rewound, not rewound, unloaded, returned to the operating system, detached from the job, or disconnected from the terminal. If you specify rewind or no rewind, the file can be reopened by the same program. If you specify any other positioning option, the file is not available for further processing.

SEQUENTIAL FILE PROCESSING 3

A sequential file is a mass storage or magnetic tape file of records that are stored in the physical order in which they were written. No logical order exists other than the relative physical record position. To read any given record, many other records might have to be read or skipped under user program control. No key exists by which a record in a sequential file can be retrieved by key value. Sequential file organization is assumed by default if another organization is not defined.

Sequential files can reside on mass storage devices or magnetic tape. In concept, sequential files on mass storage are analogous to files on unlabeled tape; beginning-of-information is the start of the first user record, and end-of-information is the end of the last user record.

CONCEPTS OF PHYSICAL FILE STRUCTURE

Individual records in sequential files are grouped in large units called blocks. Blocking provides for efficient use of hardware on which files are stored or for construction of physical groupings in situations such as tape interchange.

On tape, the blocks become the physical records that appear between interrecord gaps. The logical block structure, therefore, is dependent on the physical structure that can be written by the routines controlling tape operations and on the routines that logically interpret the physical records. On mass storage, the logical block structure is maintained by BAM routines that interface with the operating system routines for reading and writing on a device.

Once you select an appropriate block type, BAM performs all manipulations required for block construction. You specify block type for BAM with the BT parameter. You can define four types of blocking:

- I Every block is 5120 characters long; the first word is an internal control word. Records can span blocks. Only W type records can appear in I blocks.
- C Every block is the same length. You or BAM sets the specific length appropriate to the device on which the file resides. Records can span blocks. Any type of record can appear in C blocks.
- K Every block contains the same number of records; but because all records need not be the same length, all blocks are not necessarily the same length. Any type of record except S and W can appear in a K type block. These blocks are applicable to S and L tapes only and are used frequently for file interchange with another computer system. Records cannot span blocks. You can specify padding to ensure that the minimum block size is acceptable for another computer system or to increase block size so that it exceeds noise record size on tape.
- E Block length can vary from the minimum number to the maximum number of characters defined. As many whole records as possible are placed in each block such that MBL is not exceeded. Any type of record except S and W can appear in an E type block. These blocks are applicable to S and L tapes only and are used frequently for file interchange with another computer system. Records cannot span blocks. You can specify padding to ensure that the minimum block size is acceptable for another computer system or to increase block size so that it exceeds noise record size on tape.

All sequential files processed by BAM are blocked according to one of these types. You must select block type before the file is opened for initial write operations, and you cannot change block type for the life of the file.

Table 3-1 summarizes block types.

STRUCTURE ON MASS STORAGE

A sequential file on a mass storage device can have one of the following block types:

- I Internal control word. Each block contains 5120 characters.
- C Character count. Each block is a multiple of PRU size for a particular device (640 characters for disk). Each block contains the number of characters specified by the maximum block length (MBL) field of the FIT, up to a maximum of 5120 characters.

Figure 3-1 illustrates sequential file structure for mass storage files.

STRUCTURE ON SI AND I TAPES

A sequential file on SI or I tape has a structure similar to that of a mass storage file. The PRU size on tape, however, can be a maximum of either 1280 or 5120 characters, depending on whether coded or binary data is written. You must set the conversion mode (CM) field in the FIT to YES for coded data.

Block type for binary data can be:

- I Internal control word. Each block contains 5120 characters.
- C Character count. Each block can contain a maximum of 5120 characters of binary data.

Block type for NOS/BE coded data (SI tape files only) can be:

- C Character count. Each block can contain a maximum of 1280 characters of coded data.

Figure 3-2 illustrates sequential file structure for SI and I tape files.

TABLE 3-1. SUMMARY OF BLOCK TYPES

Block Type	Allowable Record Types	Mass Storage Files	Tape Files					Description	Associated FIT Fields		Characteristics
			(NOS/BE only) SI Coded	SI Binary	(NOS only) I	S	L		Required	Optional	
I	W only	Each block contains 5120 characters. The last block in a section, partition, or file can be shorter; a short block is recorded as a short PRU.	Illegal	Each block contains 5120 characters. The last block in a section, partition, or file can be shorter. ANSI standard tape interchange not supported.				Blocks begin with a block control word, which includes a pointer to the first record in the block. Information is binary. Records can span blocks.	BT=I (default) CM=NO (default) RT=W (default)		Provide fast transfer between storage device and central memory, particularly when long records are being read.
C	Any type Although S type records are unblocked, they can be set to C blocking for SCOPE 2 compatibility.	Each block contains MBL characters, up to a maximum of 5120. MBL=0 is most efficient. For W type records, MBL=PRU size facilitates parity error recovery.	Each block can contain a maximum of 1280 characters.	Each block can contain a maximum of 5120 characters.	Each block can contain a maximum of 5120 characters.	Each block can contain a maximum of 5120 characters.	Each block can contain a maximum of BFS in characters minus 20.	Each block, except possibly the last block of a section, partition, or file is always filled. Except for S type records, records can span blocks.	BT=C MBL=max (default is 5120 for S tapes; BFS in characters minus 20 for L tapes; 0 for all other files)		Process internally with great efficiency and good transfer rate.
			For W type records, MBL=PRU size facilitates parity error recovery.			One S type record is one tape block.					
			ANSI standard tape interchange not supported.								
K	Any type except S and W	Illegal	Illegal			Each block can contain a maximum of 5120 characters. Each block contains RB records. ANSI standard tape interchange supported.	Each block can contain a maximum of BFS in characters minus 20.	Blocks are recorded as tape physical records. Block length is variable, but each block contains the same number of records. The last block of a partition or file can contain fewer than RB records. Records cannot span blocks. Blocks can be padded.	BT=K MBL=max (default is 5120) RB=n (default is 1)	MNB=min (default is 0) MUL=n (default is 2) PC=ccB (default is 76_8)	Provide good recovery for tape files encountering damage.

TABLE 3-1. SUMMARY OF BLOCK TYPES (Contd)

Block Type	Allowable Record Types	Mass Storage Files	Tape Files					Description	Associated FIT Fields		Characteristics
			(NOS/BE only) SI Coded	SI Binary	(NOS only) I	S	L		Required	Optional	
E	Any type except S and W	Illegal	Illegal			Each block can contain a maximum of 5120 characters. Each block contains as many whole records as possible within MBL. ANSI standard tape interchange supported.	Each block can contain a maximum of BFS in characters minus 20.	Block length varies from MNB to MBL. Records cannot span blocks. Blocks can be padded.	BT=E MBL=max (default is 5120)	MNB=min (default is 0) MNR=min MUL=n (default is 2) PC=ccB (default is 76_8)	Provide good recovery for tape files encountering damage.

LEGEND:

BFS Buffer size
BT Block type
CM Conversion mode
MBL Maximum block length
MNB Minimum block length
MNR Minimum record length
MUL Multiple of characters per block
PC Padding character
RB Records per block
RT Record type

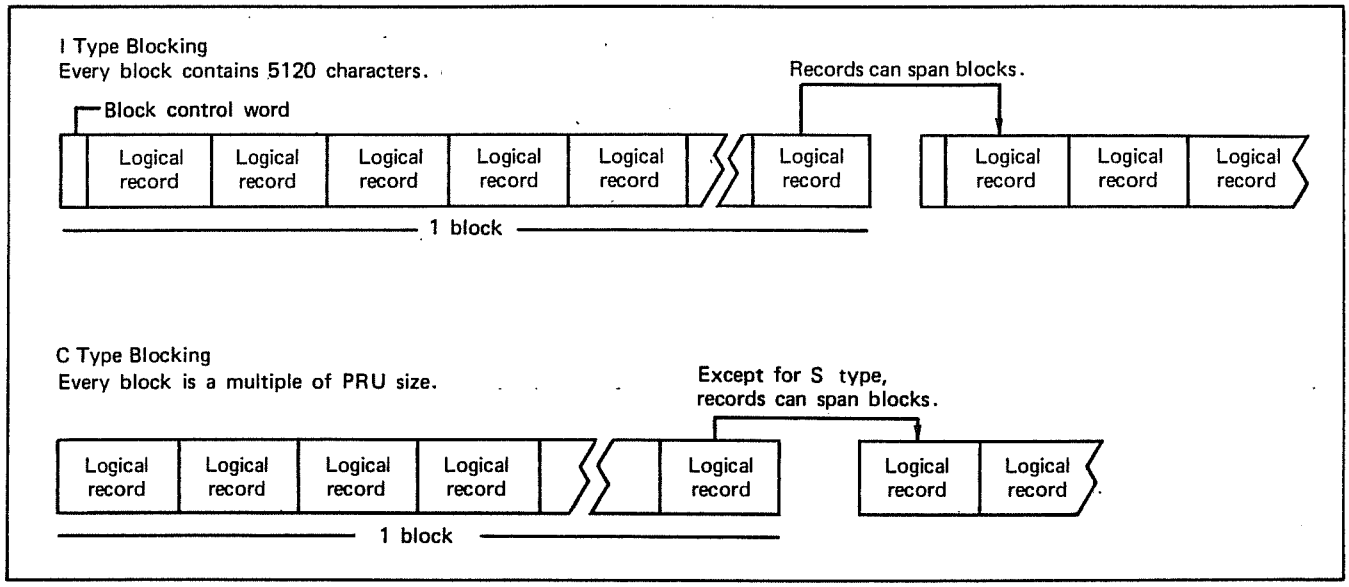


Figure 3-1. Mass Storage Sequential File Structure

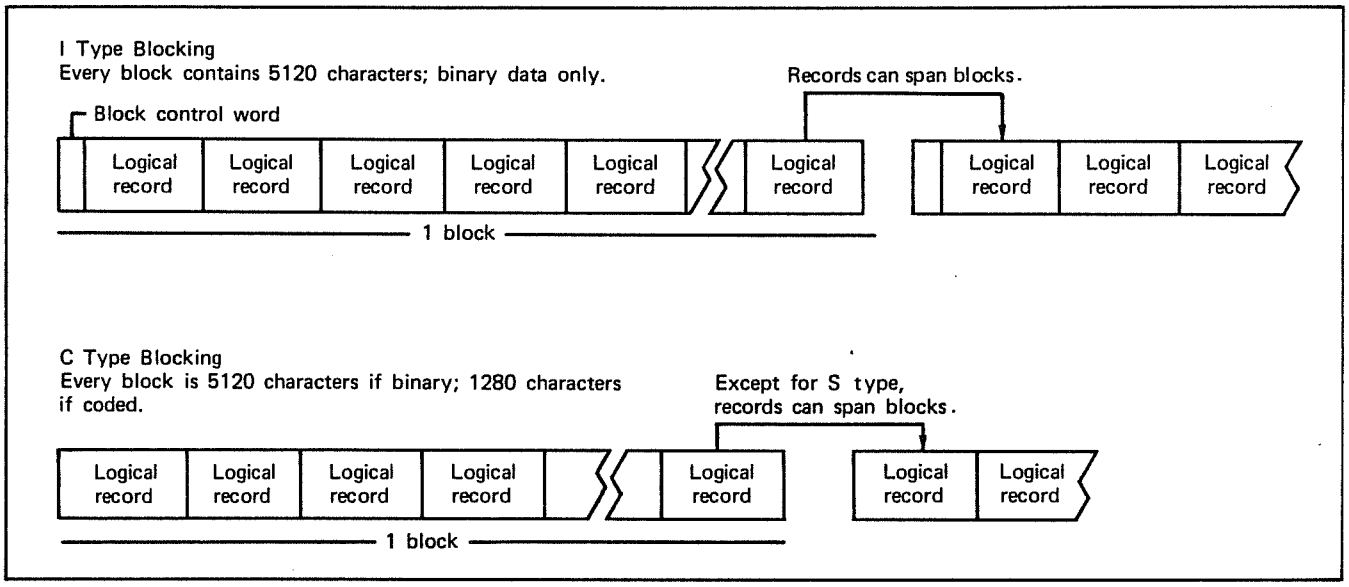


Figure 3-2. SI and I Tape File Structure

STRUCTURE ON S AND L TAPES

A sequential file on S or L tape has blocks equivalent to physical records written on the tape. S and L tapes do not define a fixed-size PRU. The length of the logical block written also defines the physical block.

Block types that can appear on S or L tapes are:

I Internal control word. Each block contains 5120 characters.

C Character count. Each block is a maximum of 5120 characters.

K Record count. Each block has the number of records defined by the blocking factor (RB).

E Exact records. Each block contains as many whole records as possible within the maximum block length declared.

Figure 3-3 illustrates sequential file structure for S and L tape files.

DESCRIBING BLOCK TYPES

Blocks are described by parameters in a FILE control statement or a CALL FILESQ statement. The block type selected affects any buffer that is defined with a BFS parameter. BFS is specified in words. If you specify BFS, it must exceed by two words the larger of one of the following:

- Block size defined or default MBL.
- Physical record unit size of the resident device for the file. For mass storage, PRU size is 64 words; for SI or I tapes, size is 512 binary words or 128 coded words. For S/L devices, PRU size is not applicable.

DESCRIBING I TYPE BLOCKS

I type blocks begin with a block control word that contains block and record identification. You cannot access this word. Contents of the control word include a pointer to the first record beginning in the block. (Records can span blocks.)

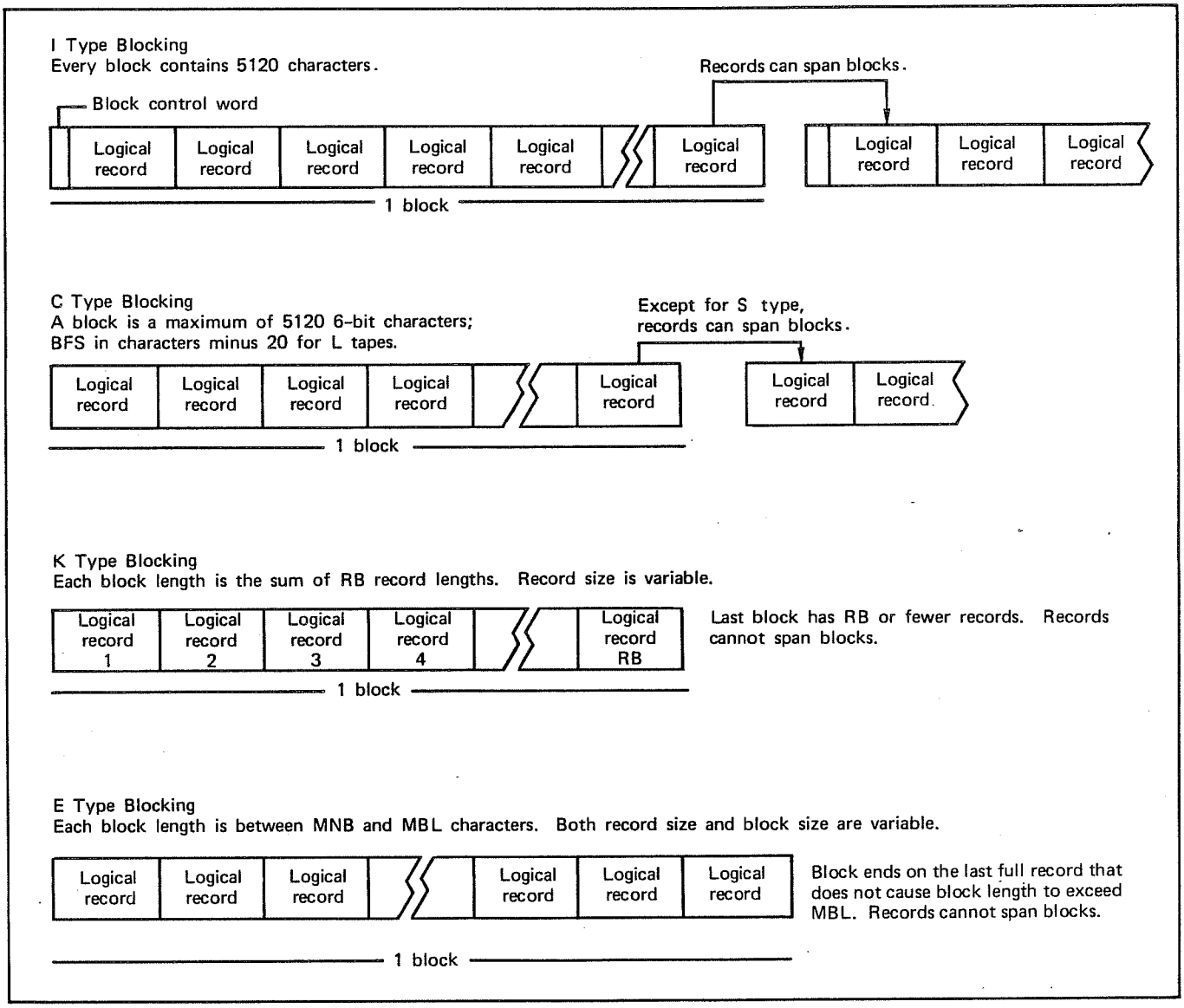


Figure 3-3. S and L Tape File Structure

I type blocks can contain only W type records. To describe an I type block, specify the following parameters for the FIT fields:

BT=I	Block type I.
RT=W	Record type in I blocks must be W.
CM=NO	No conversion of data to external code when the file is being written; CM=YES is not allowed for W type records.

If you do not specify values for the BT, RT, or CM fields, default values are as shown (I, W, and NO).

I type blocks are always 5120 characters long; therefore, any specified block length is ignored.

DESCRIBING C TYPE BLOCKS

C type blocks contain the number of characters specified by the value of the maximum block length (MBL) field. With the possible exception of the last block of a section, partition, or file, each block is always filled. Any record type except S can span blocks.

C type blocks can contain any type record. S type records cannot be blocked, although block type can be set to C for SCOPE 2 compatibility. For S type records, any user value for MBL is not changed for files on any device.

To describe a C type block, specify the following parameters for the FIT fields:

BT=C	Block type C.
MBL=max	Maximum number of characters in each block.

If you do not specify the MBL field, default values are supplied as follows:

Mass storage	MBL=0 (unblocked)
L tapes	MBL=value of the buffer size (BFS) field in characters minus 20
Other tapes	MBL=5120 binary characters

For mass storage files, the MBL field can be set to 0 or a multiple of PRU size. The most efficient value for the MBL field is 0 because fewer control words need to be checked; this is particularly true for W type records. If the value specified is not equal to 0 or a multiple of PRU size, it is rounded down to a multiple. With MBL set to PRU size with W type records, a boundary condition exists if a parity error occurs; this facilitates error recovery.

For L tapes, the MBL field can be a maximum of the value of the BFS field in characters minus 20. For all other tapes, the MBL field can be a maximum of 5120 characters.

DESCRIBING K TYPE BLOCKS

K type blocks each contain the same number of records, but the blocks can differ in length because all records need not be the same length. Records cannot span blocks.

K type blocks are restricted to S and L tapes. Any record type except S and W can appear in K type blocks.

K type blocks are often used for file interchange. You can specify padding for a minimum block size so that blocks meet requirements of other computer systems. You can also use padding to ensure that block size exceeds noise record size on tape.

To describe a K type block, specify the following parameters for the FIT fields:

BT=K	Block type K.
MBL=max	Maximum number of characters in each block. Maximum block length must not exceed 5120 characters for S tapes.
RB=n	Number of whole records to be contained in each block.

If you do not specify values for the MBL and RB fields, default values are supplied as follows:

MBL	5120 characters
RB	1

If the block is to be padded for interchange purposes, the following fields are required:

MNB=min	Minimum number of characters in each block. The value can be 0 and must not exceed MBL. The block is padded to this length if necessary. The last block on the file might contain fewer than MNB characters.
MUL=n	Multiple number of characters that must exist in the block. The value must be an even number; it can be 0, which defaults to 2, and must not exceed 62.
PC=ccB	Display code (octal) equivalent of the character to be used for padding. The value can be 0 and must not exceed 77_8.

If you do not specify values for MNB, MUL, and PC fields, default values are supplied as follows:

MNB	0
MUL	2
PC	76B, which is equivalent to ¬ in the CDC character set and to ^ in the ASCII subsets

When you specify padding, BAM ensures that at least MNB characters are written and the total number of characters is a multiple of MUL. The character specified by PC is used for padding. If record type is R, the record mark character defined by RMK must not be the same as the padding character defined by PC.

MNB is not required if padding is not desired; nevertheless, changing the default zero minimum block size to a value equivalent to the noise record defined on a system is often advisable.

DESCRIBING E TYPE BLOCKS

E type blocks contain as many whole records as possible in the maximum user-defined block size. Records cannot span blocks.

E type blocks are restricted to S and L tapes. Any record type except S and W can appear in E type blocks.

E type blocks are often used for file interchange. You can specify padding for a minimum block size so that blocks meet requirements of other computer systems. You can also use padding to ensure that block size exceeds noise record size on tape.

To describe an E type block, specify the following parameters for the FIT fields:

BT=E	Block type E.
MBL=max	Maximum number of characters in each block. Maximum block length must not exceed 5120 characters for S tapes.

If you do not specify a value for the MBL field, a default value is supplied as follows:

MBL	5120 characters

If the block is to be padded for interchange purposes, the following fields are required:

MNB=min	Minimum number of characters in each block. The value can be 0 and must not exceed MBL. The block is padded to this length if necessary. The last block on the file might contain fewer than MNB characters.
MNR=min	Minimum record length. The difference between the minimum and maximum block size must be greater than the difference between the minimum and maximum record size. The character multiple also must be less than the minimum record size. MBL minus MNB must exceed MRL minus MNR, and MNR must exceed MUL.
MUL=n	Multiple number of characters that must exist in the block. The value must be an even number; it can be 0, which defaults to 2, and must not exceed 62.
PC=ccB	Display code (octal) equivalent of the character to be used for padding. The value can be 0 and must not exceed 77_8.

If you do not specify values for MNB, MNR, MUL, and PC fields, default values are supplied as follows:

MNB	0
MNR	0
MUL	2
PC	76B, which is equivalent to ¬ in the CDC character set and to ^ in the ASCII subsets

When you specify padding, BAM ensures that at least MNB characters are written and the total number of characters is a multiple of MUL. The character specified by PC is used for padding. If record type is R, the record mark character defined by RMK must not be the same as the padding character defined by PC.

SUMMARY OF BLOCK TYPES

Figure 3-4 illustrates sequential file block types and indicates the record types that are applicable. Each FIT field that must be set, either by specification or by default, is included.

CONCEPTS OF LOGICAL FILE STRUCTURE

The logical beginning and end of a file are defined in relation to user records in that file. Beginning-of-information (BOI) is that point in a file before which no user data exists; end-of-information (EOI) is that point in a file after which no user data exists. BOI and EOI refer to one file only, even when files are stored on magnetic tape. These terms cannot encompass a multifile set. A tape trailer label, whether it is a labeled tape or internal trailer information from an unlabeled SI tape, is past EOI. An unlabeled S/L tape has no system trailer label, and EOI is undefined.

Other logical divisions that can exist between BOI and EOI are record, section, and partition.

- A record is a group of related characters. The structure and characteristics of records are defined through a record format. The beginning and ending points of a record are implicit within each format.
- A section is a group of records terminated by a special record or condition, depending on the block and record type and storage device.
- A partition is a group of one or more sections terminated by a special record or condition, depending on the block and record type and storage device.

FILE BOUNDARIES

The logical divisions of a file depend upon a physical boundary that can be written and then read as written. Depending on the particular file format and device, however, not all logical file divisions are possible.

A sequential file can have the following divisions and boundaries:

Beginning	Beginning-of-information (BOI)
Record	End is established by control information
Section	End-of-section (EOS)
Partition	End-of-partition (EOP)
End	End-of-information (EOI)

BAM terminology is not always the same as that of the operating system. Table 3-2 indicates general file terminology and the terms used by the operating system and BAM.

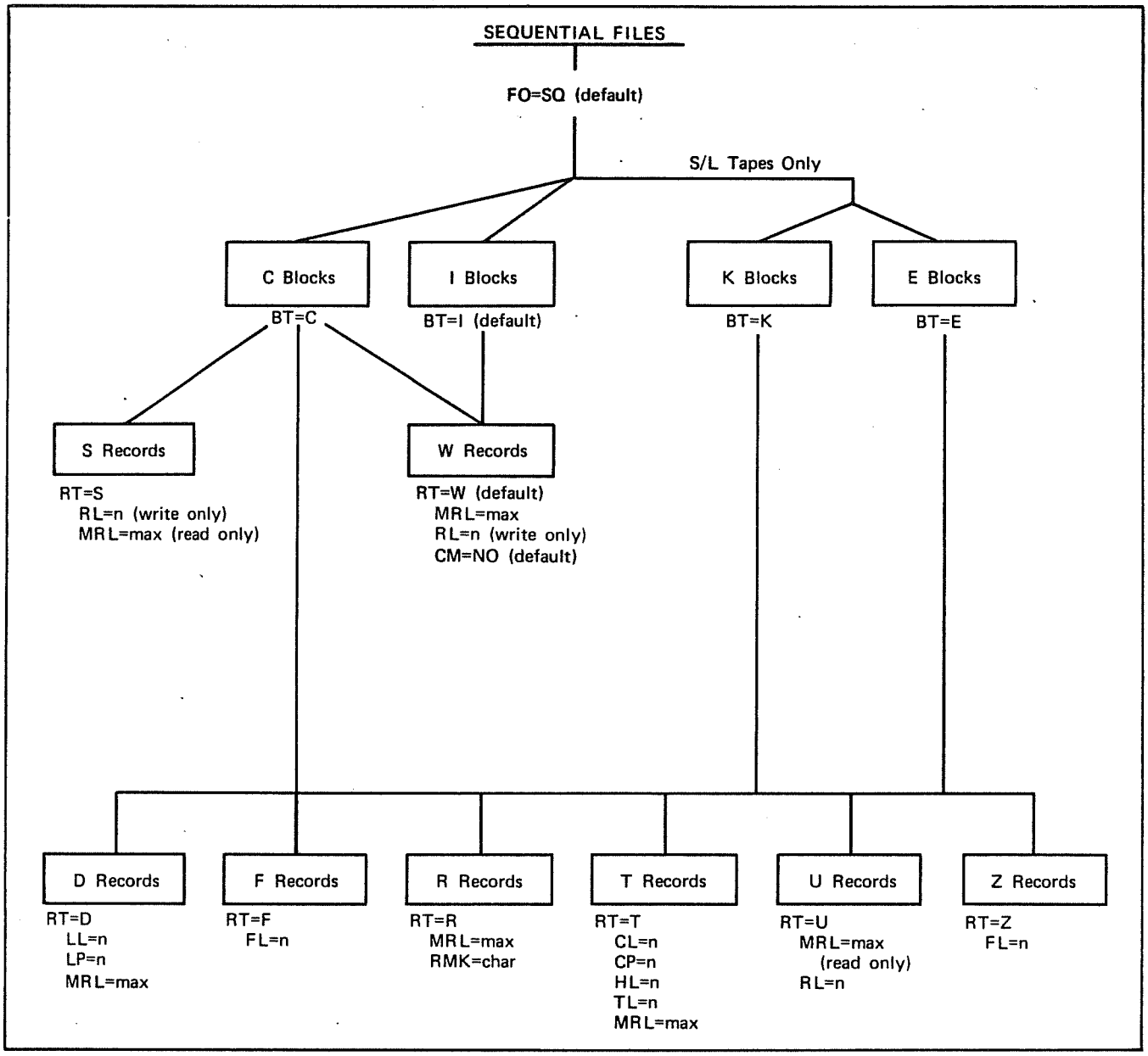


Figure 3-4. Sequential File Block and Record Type Summary

TABLE 3-2. OPERATING SYSTEM/BAM FILE TERMINOLOGY

General File Terminology	Operating System Term	BAM Term
Beginning of physical file	Beginning-of-information	Beginning-of-information
Single record	Unit record or line image	Data record
End of unit/data record	----	End-of-record
End of logical record	End-of-record	End-of-section
End of logical file	End-of-file	End-of-partition
End of physical file	End-of-information	End-of-information

Because the device on which the file resides might require a boundary condition to terminate a block or might require a boundary indicator to be written as a separate block, any block count maintained by the system might not be what you expect. The block number (BN) field of the FIT reflects actual blocks encountered and includes blocks containing only a boundary indicator, as well as blocks with user records. For magnetic tape files on NOS/BE, the job dayfile shows actual blocks encountered. A value greater than that defined by the blocking factor results from boundary blocks.

When a file is read, BAM sets the file position (FP) field of the FIT as boundary conditions are encountered. Boundary conditions are defined as follows:

- Beginning-of-information is defined for all sequential files stored on all devices.
- Sections are defined for formats shown in figure 3-5; the letter s indicates a section boundary can exist.
- Partitions are defined for formats shown in figure 3-6; the letter p indicates a partition boundary can exist.
- End-of-information is defined for all formats and devices except an unlabeled S or L tape.

For C type blocks on a PRU device with records other than type W, an EOS boundary appears before either a partition or end-of-information boundary. The EOS boundary is written as part of the response to your request for EOP or EOI. The FP field setting for a read does not distinguish an EOS written at your request from an EOS written as a result of an EOP or EOI request; consequently, you must be aware of specific boundaries existing when processing depends on particular conditions.

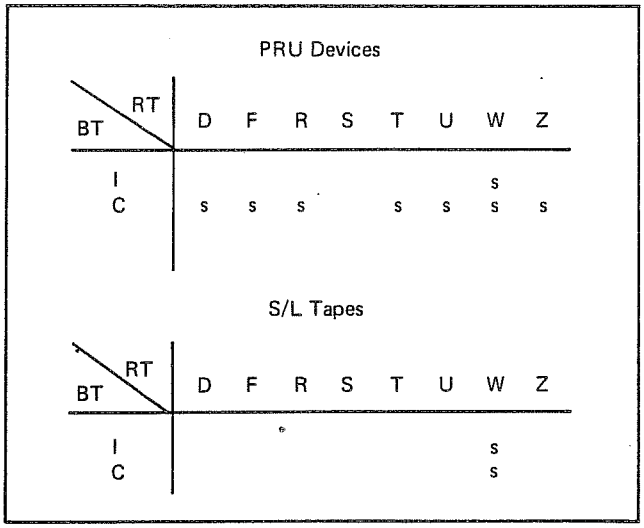


Figure 3-5. File Formats for Which Sections are Defined

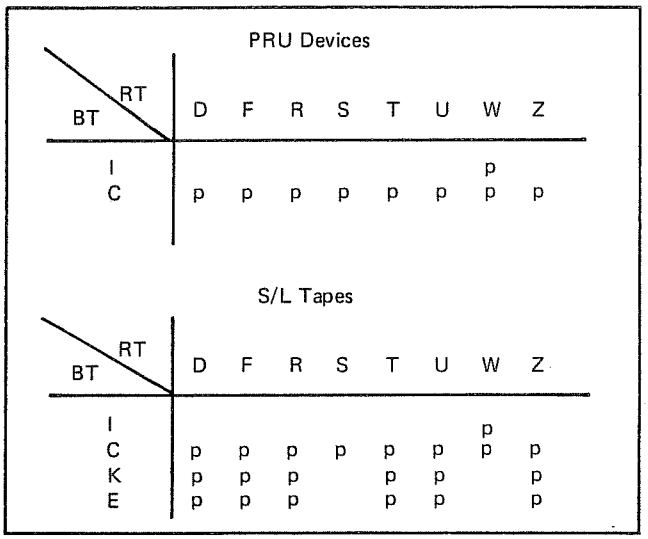


Figure 3-6. File Formats for Which Partitions are Defined

WRITING FILE BOUNDARIES

An end-of-information boundary is written by the system when the CLOSEM request is executed in a file creation run. Any information in the central memory buffer is written to the file before close occurs.

File boundaries that can be written by issuing a request within a source program are:

- End-of-section
- End-of-partition
- Tapemark

Each type of request is described in the following paragraphs. Refer to appendix F for an example on writing and processing file boundaries.

Writing an End-of-Section

An end-of-section boundary is written by executing a CALL WEOR statement. The format is:

```
CALL WEOR(fit,lvl)
```

The buffer is written to the file before the section boundary is written. This statement is also used to terminate an S type record being constructed through a series of partial write requests.

You can include a level number parameter. A value of 0 (end-of-section) or 17_8 (end-of-partition) can be written.

When terminating a section with a WEOR call, you must consider the following:

- For S type records, a read of EOS returns an EOR value to the file position (FP) field; the EOS value is never returned.
- For C, K, or E type blocks on an S/L device, an EOS cannot be detected by a GET call.
- For W type records, the file must be on a record boundary to write an EOS.

Figure 3-7 shows some examples of the CALL WEOR statement.

Writing an End-of-Partition

An end-of-partition boundary is written by executing a CALL ENDFILE statement. The format is:

```
CALL ENDFILE(fit)
```

```
CALL WEOR(SQFIT)
```

This statement writes an end-of-section for the file described by SQFIT and terminates the block with level 00. The FP field is set to 10_8.

```
CALL WEOR(SQFIT,17B)
```

This statement writes an end-of-section for the file described by SQFIT and terminates the block with level 17_8. On an SI tape this results in a short PRU level 0 followed by a zero length PRU level 17_8.

Figure 3-7. CALL WEOR Statement Examples

The buffer is written to the file before the partition boundary is written.

When terminating a partition with an ENDFILE call, you must consider the following:

- For W type records, an ENDFILE call writes a control word with the end-of-partition flag set, and terminates the current PRU or block.
- For PRU devices when record type is not W, an ENDFILE call terminates the current system-logical-record with a short PRU level 0, and writes a zero-length PRU level 17_8.
- For S/L devices when record type is not W, an ENDFILE call terminates the current block and writes a tapemark.
- For S type records only, an ENDFILE call can be issued in midrecord.

Figure 3-8 shows an example of the CALL ENDFILE statement.

```
CALL ENDFILE(SQFIT)
```

This statement writes an end-of-partition for the file described by SQFIT. The FP field is set to 40_8.

Figure 3-8. CALL ENDFILE Statement Example

Writing a Tapemark

A tapemark is written by executing a CALL WTMK statement. The format is:

```
CALL WTMK(fit)
```

Use of a tapemark boundary should be confined to a user label processing routine for nonstandard labels in which labels are separated from data by this mark. Do not use it elsewhere.

User label processing must be performed through a COMPASS-coded routine. Refer to the BAM reference manual for details.

CREATING A SEQUENTIAL FILE

A sequential file is created by writing records to a file with a defined record type and block structure. You can declare file characteristics through a combination of FILE control statements, source language statements, and installation default values. Once the file structure is defined, records are inserted into the file by the CALL PUT statement.

FIT fields that must be defined on a sequential file creation run are as follows:

LFN	Logical file name consisting of one to seven characters and beginning with a letter
FO	File organization set to SQ (default is SQ)
BT	Block type and any fields required by the block type (default is I)
PD	Processing direction set to OUTPUT or I-O
RT	Record type and any fields required by the record type (default is W)

Other optional FIT fields that can be defined on the creation run and can be changed on a subsequent run are as follows:

DX	End-of-data exit (default is no exit)
EX	Error exit (default is no exit)
LX	Label exit (default is no exit)
ERL	Trivial error limit (default is no limit)
DFC	Dayfile control (default is only fatal errors to the dayfile)
EFC	Error file control (default is no messages to the error file)
PC	Padding character (default is 76_8)
BFS	Buffer size (default is buffer size calculated by BAM)
FWB	First word address of the buffer (default is buffer address provided by BAM)

When you create a sequential file, you must declare file characteristics by setting applicable fields in the FIT before the file is opened. Except as noted in appendix D, you can specify FIT fields in the FILE control statement, the CALL FILESQ statement, or the CALL STOREF statement. After all records have been written, you must close the file with the CALL CLOSEM statement.

On a file creation run, the only statements you can issue are those that perform the following:

- Establish the FIT.
- Open and close the file.
- Write records.
- Write boundary conditions.
- Store and fetch FIT fields.

ESTABLISHING THE FIT

The first statement referencing the sequential file must be the CALL FILESQ statement. When this statement is executed, the FIT is constructed and the specified values are stored in appropriate FIT fields. The first parameter in the CALL FILESQ statement is the name of the array to hold the FIT. The same FIT array name is the first parameter in every statement accessing the sequential file. Refer to section 2, File Processing Concepts, for a more detailed explanation of the FIT and the CALL FILESQ statement.

OPENING THE FILE

You must open the sequential file with a CALL OPENM statement before any records can be written to the file. The format is:

```
CALL OPENM(fit,pd)
```

Open processing includes storing FILE control statement values in the FIT, processing buffer parameters, supplying default values for FIT fields not set by the source program, and checking the FIT for logical consistency and required fields.

The first parameter in the open request for file creation is the name of the array that contains the FIT. The second parameter must specify I-O or OUTPUT; this sets the processing direction (PD) field in the FIT to read/write or write only.

When the following statement is executed, the file identified by the FIT in array SQFIT is opened for a read or write operation:

```
CALL OPENM(SQFIT,'I-O')
```

WRITING RECORDS

Records are written to a sequential file by executing a CALL PUT statement. The format is:

```
CALL PUT(fit,wsa,rl,0,0,0,ex)
```

Each record to be written must be established at the working storage area location.

The first parameter in the write request for file creation is the name of the array that contains the FIT. Other parameters set FIT fields that BAM uses to write the record to the file. The following FIT fields can be set by the write request:

WSA	Working storage area from which the record is written to the file.
RL	Record length, which applies to S, U, and W type records only.
EX	Error exit subroutine to be executed if an error occurs.

Fields not specified in the write request default to the current value in the FIT. After the record is written, BAM updates the following FIT fields:

RC	Record count
BN	Block number if a new block was begun
RL	Record length requested or calculated

When the following statement is executed, a record is added to the file identified by the FIT in array SQFIT:

```
CALL PUT(SQFIT,WSA,30,0,0,0,ERREXIT)
```

The array WSA is the working storage area that contains the record to be written. The record contains 30 characters. The three zeros represent parameters that are not applicable to sequential files. If an error occurs during execution of this statement, control is transferred to subroutine ERREXIT.

CLOSING THE FILE

The last program statement referencing the file must be a CALL CLOSEM statement. The format is:

```
CALL CLOSEM(fit,cf,type)
```

When the statement is executed, any data in the central memory buffer is written to the file.

The first parameter in the close request is the name of the array that contains the FIT.

The second parameter sets the close flag (CF) field, which provides for file positioning and disposition. The following values can be specified for the CF field:

- R (rewind)

  The file is rewound to beginning-of-information and the open/close flag (OC) field is set to closed; this is the default setting for FILE close.

- N (no rewind)

  The file is not rewound and the OC field is set to closed; the file remains at the current position.

- U (unload)

  The file is rewound, the OC field is cleared, the file is detached from the job, a tape is unloaded, and scratch mass storage space assigned to the file is released. This is the default setting for VOLUME close.

- RET (return)

  The file is rewound, the OC field is cleared, the file is detached from the job, and buffer space is released. A tape is unloaded and the device is returned to the system.

- DET (detach)

  The file is not rewound, the OC field is cleared, buffer space is released, and the file is disassociated from the job.

- DIS (disconnect)

  The terminal file is disconnected, the OC field is cleared, and the file is disassociated from the job. For a non-terminal file, the OC field is cleared, the file is disassociated from the job, and the file remains at current position.

A third parameter you can specify in the close request indicates the type of close that is to be performed. The following values can be specified:

- FILE

  The file is closed; this is the default setting.

- VOLUME

  The current tape reel is terminated and volumes are switched. Processing can continue on the next volume without issuing another OPENM.

A close request issued for a file that has never been opened, or that has been closed but neither unloaded nor reopened, results in a trivial error. The file is positioned as specified before the error is issued.

When the CF field is set to R or N and the file is subsequently reopened, FIT verification and FILE control statement processing are not repeated. When the CF field is set to U, RET, DET, or DIS, FIT verification and FILE control statement processing are repeated; to resume processing, the file must be reattached to the job and opened.

To ensure allocation of a new buffer when a file is reopened in a program, the file must be closed with the CF field set to DET, and the BFS field must be reset.

When the following statement is executed, the file identified by the FIT in array SQFIT is unloaded, the OC flag is cleared, and the file name is removed from the active file list.

```
CALL CLOSEM(SQFIT,'U')
```

SAMPLE CREATION PROGRAM

Program SQCREAT, shown in figure 3-9, creates a permanent sequential file through direct calls to BAM. The illustration includes the control statements used for the NOS and NOS/BE operating systems, the input file, the source listing, a printout of the contents of the input file, and an octal dump of the newly created sequential file. The program reads a ten-record input file and stores the records on file SQFILE. The input records for creating the file exist initially on INPUT.

Note the following information about the control statements:

- The DEFINE control statement effects permanent file storage of SQFILE on NOS. REQUEST and CATALOG control statements must be substituted for DEFINE when operation is under NOS/BE.

- The FILE control statement sets values in the BT, RT, and FL FIT fields. The EFC field is set to 3, which specifies errors and notes are to be written to the error file.

- The TDUMP control statement dumps the contents of the permanent file. This control statement is not available for operation under NOS/BE.

- The CRMEP control statement processes the error file. The mnemonic LO indicates all messages are to be displayed. The CRMEP control statement is detailed in section 5.

Statements in the program are defined as follows:

- DIMENSION SQFIT(35), WSA(3)

  This statement allocates a 35-word array named SQFIT for FIT construction and a 3-word working storage area named WSA.

- CALL FILESQ (SQFIT, 'LFN', 'SQFILE')

  This statement sets fields in the FIT to describe the structure of the sequential file. Two required parameters are included:

  FIT array (SQFIT)

  Logical file name (SQFILE)

  Three additional required parameters could have been included, but appear instead on the FILE control statement:

  Block type C (BT=C)

  Fixed length records (RT=F)

  Fixed record length of 30 characters (FL=30)

- CALL OPENM (SQFIT, 'I-O')

  This statement opens the file described by SQFIT for reading and writing.

- READ (*, '(3A10)', END=20) WSA
  PRINT 15, WSA

  These statements read the input file into the working storage area (WSA) and print the contents. If end-of-file is encountered, control is transferred to a statement that closes the file.

- CALL PUT (SQFIT, WSA)

  This statement writes the record from the working storage area to the BAM file.

- CALL CLOSEM (SQFIT)

  This statement writes the buffer to the file and writes end-of-information.

A printout of the ten-record input file follows the source listing. Each record contains 30 characters and, therefore, occupies three words in memory.

The file dump illustrates the internal representation of SQFILE.

CONTROL STATEMENTS

NOS Operating System

```
Job statement
USER control statement
CHARGE control statement
FTN5.
DEFINE(SQFILE/CT=PU,M=W)
FILE(SQFILE,BT=C,RT=F,FL=30,EFC=3)
LGO.
CRMEP(LO)
TDUMP,I=SQFILE.
```

NOS/BE Operating System

```
Job statement
ACCOUNT control statement
FTN5.
FILE(SQFILE,BT=C,RT=F,FL=30,EFC=3)
REQUEST(SQFILE,*PF)
LGO.
CATALOG(SQFILE,ID=BAMUG)
CRMEP(LO)
```

INPUT FILE

```
0001 TIMESHAR 04/21/78 178900
0002 ARBITRON 01/13/78 161320
0003 TICKETRO 09/01/78 147380
0004 SYNTONIC 06/14/78 000116
0005 COMMCRED 05/11/78 004672
0006 SVBUREAU 05/13/78 000197
0007 COMMACOR 02/17/78 001432
0008 LEARNCTR 03/06/78 000097
0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872
```

SOURCE LISTING

```
C     THIS PROGRAM CREATES A PERMANENT SEQUENTIAL FILE
C     (SQFILE) THROUGH DIRECT CALLS TO BAM.  THE INPUT
C     RECORDS EXIST INITIALLY ON INPUT.  THE PROGRAM READS
C     THE TEN-RECORD INPUT FILE, PRINTS EACH RECORD,
C     AND STORES THE RECORDS ON SQFILE.
C
C
      PROGRAM SQCREAT
      DIMENSION SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE')
      CALL OPENM (SQFIT, 'I-O')
   10 READ (*, '(3A10)', END = 20) WSA
      PRINT 15, WSA
   15 FORMAT (1X,3A10)
      CALL PUT (SQFIT, WSA)
      GO TO 10
   20 CALL CLOSEM (SQFIT)
      END
```

OUTPUT

```
0001 TIMESHAR 04/21/78 178900
0002 ARBITRON 01/13/78 161320
0003 TICKETRO 09/01/78 147380
0004 SYNTONIC 06/14/78 000116
0005 COMMCRED 05/11/78 004672
0006 SVBUREAU 05/13/78 000197
0007 COMMACOR 02/17/78 001432
0008 LEARNCTR 03/06/78 000097
0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872


CRMEP(LO)
```

Figure 3-9. Creating a Sequential File (Sheet 1 of 2)

```
1        - FILE DUMP -        TDUMP,I=SQFILE.

 F  1 R   1 W     0- 3333 3334 5524 1115 0523   1001 2255 3337 5035 3450   4243 5534 4243 4433 3355   3333 3335 5501 2202 1124
                     0 0  0 1    T  I M  E S    H A  R    0 4  / 2  1 /    7 8    1  7 8  9 0  0    0 0  0 2    A  R B  I T
 F  1 R   1 W     4- 2217 1655 3334 5034 3650   4243 5534 4134 3635 3355   3333 3336 5524 1103 1305   2422 1755 3344 5033 3450
                     R 0  N    0 1  / 1  3 /   7 8    1  6 1  3 2  0    0 0  0 3    T  I C  K E    T R  0    0 9  / 0  1 /
 F  1 R   1 W    10- 4243 5534 3742 3643 3355   3333 3337 5523 3116 2417   1611 0355 3341 5034 3750   4243 5533 3333 3434 4155
                     7 8    1  4 7  3 8  0    0 0  0 4    S  Y N  T O    N I  C    0 6  / 1  4 /    7 8    0  0 0  1 1  6
 F  1 R   1 W    14- 3333 3340 5503 1715 1503   2205 0455 3340 5034 3450   4243 5533 3337 4142 3555   3333 3341 5523 2602 2522
                     0 0  0 5    C  O M  M C    R E  D    0 5  / 1  1 /    7 8    0  0 4  6 7  2    0 0  0 6    S  V B  U R
 F  1 R   1 W    20- 0501 2555 3340 5034 3650   4243 5533 3333 3444 4255   3333 3342 5503 1715 1501   0317 2255 3335 5034 4250
                     E A  U    0 5  / 1  3 /   7 8    0  0 0  1 9  7    0 0  0 7    C  O M  M A    C O  R    0 2  / 1  7 /
 F  1 R   1 W    24- 4243 5533 3334 3736 3555   3333 3343 5514 0501 2216   0324 2255 3336 5033 4150   4243 5533 3333 3344 4255
                     7 8    0  0 1  4 3  2    0 0  0 8    L  E A  R N    C T  R    0 3  / 0  6 /    7 8    0  0 0  0 9  7
 F  1 R   1 W    30- 3333 3344 5503 3102 0522   2303 1055 3337 5033 3550   4243 5534 3333 4235 3755   3333 3433 5504 0124 0123
                     0 0  0 9    C  Y B  E R    S C  H    0 4  / 0  2 /    7 8    1  0 0  7 2  4    0 0  1 0    D  A T  A S
 F  1 R   1 W    34- 2603 2355 3337 5034 4250   4243 5533 3334 4342 3555
                     V C  S    0 4  / 1  7 /    7 8    0  0 1  8 7  2
 -- END OF RECORD --

 -- END OF INFORMATION --

 -- END OF DUMP --
```

Figure 3-9. Creating a Sequential File (Sheet 2 of 2)

PROCESSING A SEQUENTIAL FILE

After the file creation run, a sequential file can be attached, read, positioned, updated, and rewound. File processing is governed by many of the FIT fields set on the file creation run.

You cannot change the following FIT fields when an existing file is being processed:

FO	File organization
BT	Block type and fields required by the block type
RT	Record type and fields required by the record type
FL	Record length (F and Z type records only)

You must set the following FIT fields before the file is opened unless the default values are to be accepted:

BFS	Buffer size
FWB	First word address of the buffer

Optional fields you can set at any time before they are required by a file processing statement are as follows:

DX	End-of-data exit
EX	Error exit
LX	Label exit
ERL	Trivial error limit
DFC	Dayfile control
EFC	Error file control
PC	Padding character

ESTABLISHING THE FIT

The FIT is established for an existing file in the same manner as for a new file during the creation run. The CALL FILESQ statement is required for any program using the file. All parameters applicable to structure must be repeated in each program in which the file is processed. BAM does not make such information a part of the file.

You can use the CALL FILESQ statement to specify all FIT fields required to define minimum file structure. These values become part of the FIT during execution, but can be overwritten by FILE control statement values at the time the file is opened.

OPENING THE FILE

Before you can access any data records in an existing file, you must open the file with a CALL OPENM statement. The format is:

```
CALL OPENM(fit,pd,of)
```

Open request parameters are stored in applicable FIT fields during open processing. FILE control statement processing and FIT consistency checks are performed in the same manner as on a file creation run.

The first parameter in the open request is the name of the array that contains the FIT. Additional parameters can set the following FIT fields:

PD	Processing direction
OF	Open flag

The setting of the PD field determines the input/output statements that can be executed. You can set the PD field as follows:

INPUT	Statements that read or position the file can be executed (default).
OUTPUT	Only statements that write new records to the file can be executed.
I-O	Any file processing statement related to input/output can be executed. (If the PD field is set by any statement other than the CALL OPENM statement, the characters IO rather than I-O must be specified.)

A trivial error occurs if execution of a file processing statement is attempted and the PD field is not set to an appropriate value for that statement.

File positioning by the open request is determined by the OF field. You can set this field as follows:

R The file is rewound to beginning-of-information (default).

N The file is not rewound.

E The file is positioned at the end of current information to extend the file (applicable only to an existing file on mass storage).

During open processing, labels on a tape file are processed. Labels are not processed unless the file is rewound at open time.

When the following statement is executed, the file described by SQFIT is opened with no rewind; any file processing statement related to input/output can be executed:

```
CALL OPENM(SQFIT,'I-O','N')
```

READING RECORDS

Records in a sequential file are read in the physical order in which they were written. Records are read by executing a CALL GET statement. The format is:

```
CALL GET(fit,wsa,0,0,0,rl,dx)
```

The file must be attached and open for either input or input/output (the PD field is set to INPUT or IO).

The first parameter in the read request is the name of the array that contains the FIT. Additional parameters set the following FIT fields:

WSA Working storage area that is to receive the record.

RL Record length, which is required only for U type records.

DX End-of-data exit subroutine to be executed if a file boundary is encountered.

You can set the MRL field of the FIT to the maximum number of characters to be read before the read request is executed. MRL sets a maximum that overrides the actual record length. If control information in the record (control word or fields, such as LL or LP in D type records) indicates more data than MRL, an excess data error occurs when that record is read.

After a record is read, BAM updates the following FIT fields:

RL Record length

RC Record count

BN Block number currently in progress

When the following statement is executed, a record in the file described by SQFIT is read into working storage area WSA:

```
CALL GET(SQFIT,WSA,0,0,0,0,ENDEX)
```

The first three zeros represent parameters that are not applicable to sequential file organization; the fourth zero is not applicable to the specific record type. If end-of-data is reached during execution of this statement, control is transferred to subroutine ENDEX.

Program SQREAD, shown in figure 3-10, reads the previously created file SQFILE. The file must be attached. The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILESQ statement.
- Opens the file for input.
- Reads records through the CALL GET statement into the working storage area (WSA).
- Retrieves the value of FIT field FP.
- Tests for end-of-section, which is 10_8 or 8, before printing the contents of the buffer.
- Prints each record together with the FP field value, which is 0020_8 (end-of-record).

SKIPPING RECORDS

You can position a sequential file forward or backward a specified number of records by executing a CALL SKIP statement. The format is:

```
CALL SKIP(fit,count)
```

The type of skip allowed depends upon the record and block type. A forward skip of a tape file can cross volume boundaries; a backward skip cannot.

The first parameter in the skip request is the name of the array that contains the FIT. The second parameter is the number of records to be skipped. The skip count can be a positive number for forward skipping and a negative number for backward skipping; the skip count can also be a variable. No FIT fields are set by the skip request.

When the following statement is executed, a forward skip of 25 records in the file described by SQFIT is performed:

```
CALL SKIP(SQFIT,+25)
```

A forward skip is allowed for all record types except U.

When the following statement is executed, a backward skip of 10 records in the file described by SQFIT is performed:

```
CALL SKIP(SQFIT,-10)
```

A backward skip is allowed for F, S, W, and Z type records. For F type records in C type blocks, record length must be a multiple of 10 characters.

SOURCE LISTING

```
C     THIS PROGRAM READS THE PREVIOUSLY CREATED SQFILE,
C     RETRIEVES THE VALUE OF THE FP FIELD, AND PRINTS
C     EACH RECORD TOGETHER WITH THE FP FIELD VALUE.
C
C
      PROGRAM SQREAD
      DIMENSION SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C', 'RT', 'F', 'FL', 30,
     +            'EFC', 3)
      CALL OPENM (SQFIT)
   10 CALL GET (SQFIT, WSA)
      IFP=IFETCH (SQFIT, 'FP')
      IF (IFP .EQ. 8) GO TO 20
      PRINT 15, WSA, IFP
   15 FORMAT (1X,3A10,1X,04)
      GO TO 10
   20 CALL CLOSEM (SQFIT)
      END
```

OUTPUT

```
0001 TIMESHAR 04/21/78 178900  0020
0002 ARBITRON 01/13/78 161320  0020
0003 TICKETRO 09/01/78 147380  0020
0004 SYNTONIC 06/14/78 000116  0020
0005 COMMCRED 05/11/78 004672  0020
0006 SVBUREAU 05/13/78 000197  0020
0007 COMMACOR 02/17/78 001432  0020
0008 LEARNCTR 03/06/78 000097  0020
0009 CYBERSCH 04/02/78 100724  0020
0010 DATASVCS 04/17/78 001872  0020
```

Figure 3-10. Reading a Sequential File

A backward skip is not allowed for D, R, T, or U type records, nor for any record type with E or K type blocks.

A backward skip does not begin after a write operation until the contents of the buffer are written to the device and an end-of-information is written.

A file boundary terminates a skip in either direction. The skip continues over the boundary condition and stops. On a backward skip, the file is positioned immediately before a section or partition and after a beginning-of-information. On a forward skip, the file is positioned immediately after a section or partition, but before an end-of-information.

Crossing a section or partition boundary, or encountering beginning- or end-of-information transfers control to any user end-of-data subroutine defined in the DX field of the FIT. The FP field reflects the boundary with an octal value as follows:

0001	Beginning-of-information
0010	End-of-section
0040	End-of-partition
0100	End-of-information

Consider a file containing the following records and boundaries:

ASTER
BLAZING-STAR
COLUMBINE
section boundary
DELPHINIUM
ELDERBERRY
FUCHSIA
GERANIUM
partition boundary
HEATHER
IRIS
end-of-information

If file position is at the start of ASTER, a forward skip of 2 records positions the file at the start of COLUMBINE.

From the start of COLUMBINE, a backward skip of 2 records positions the file at the start of ASTER.

From the start of GERANIUM, a backward skip of 5 records positions the file before the section boundary. The routine indicated in the DX field of the FIT executes.

If file position is within ELDERBERRY, a forward skip of 0 records positions the file to the start of FUCHSIA. From the same position within ELDERBERRY, a backward skip of 0 records positions the file to the start of ELDERBERRY if the previous file operation was a partial read; otherwise, it produces no change in file position.

From the start of COLUMBINE, a forward skip of 100 records positions the file at the start of DELPHINIUM. The routine indicated in the DX field of the FIT executes.

Program SKIPREC, shown in figure 3-11, reads the previously created file SQFILE. The file must be attached.

```
SOURCE LISTING

C     THIS PROGRAM PERFORMS A FORWARD SKIP OF 8
C     RECORDS IN PERMANENT FILE SQFILE AND THEN
C     READS AND PRINTS THE REST OF THE FILE.
C
C
      PROGRAM SKIPREC
      DIMENSION SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C', 'RT', 'F', 'FL', 30,
     +            'EFC', 3)
      CALL OPENM (SQFIT)
      CALL SKIP (SQFIT, +8)
   10 CALL GET (SQFIT, WSA)
      IF (IFETCH (SQFIT, 'FP') .EQ. 8) GO TO 20
      PRINT 15, WSA
   15 FORMAT (1X,3A10)
      GO TO 10
   20 CALL CLOSEM (SQFIT)
      END


OUTPUT

0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872
```

Figure 3-11. Skipping Records on a Sequential File

The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILESQ statement.
- Opens the file for input.
- Performs a forward skip of 8 records before reading and printing the file.
- Closes the file when end-of-section is reached.

REPLACING A RECORD

You can update a sequential file by replacing the last record read with a record from the working storage area. The following restrictions apply:

- The file must be a mass storage file open for input/output (PD field set to IO).
- Block type must be C.
- Record type must be F or W.
- The replacement record must have the same record length as the record being replaced.
- The replace request must be preceded by a read request of the record to be replaced.

A record is replaced by executing a CALL REPLC statement. The format is:

```
CALL REPLC(fit,wsa,0,0,0,0,ex)
```

The first parameter in the replace request is the name of the array that contains the FIT. Additional parameters set the following fields in the FIT:

WSA	Working storage area that contains the replacement record.
EX	Error exit subroutine to be executed if an error occurs.

If the last operation before a replace was not a read request, a trivial error results and the replace request is ignored.

When the following statements are executed, the replacement record read from the file described by FIT2 overwrites the record previously read from the file described by FIT1:

```
CALL GET(FIT1,WSA)
CALL GET(FIT2,WSA)
CALL REPLC(FIT1,WSA,0,0,0,0,ERREX)
```

The four zeros represent parameters that are not applicable to sequential file organization. If an error occurs during execution of the statement, control is transferred to subroutine ERREX.

Program REPLACE, shown in figure 3-12, replaces the seventh record in the previously created file SQFILE. The file must be attached.

The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILESQ statement.
- Opens the file for input/output.
- Performs a forward skip of 6 records.
- Reads record 0007 into working storage area (WSA).
- Reads a new record from INPUT into the working storage area.
- Issues a replace request, which effects an overwrite of record 0007.
- Issues a rewind of SQFILE.
- Reads and prints the file to illustrate that NEWRECRD has replaced COMMACOR.

ADDING RECORDS

You can add records to an existing sequential file by executing a CALL OPENM extend followed by a CALL PUT statement. The file must be attached for either output or input/output. (The PD field is set to OUTPUT or IO.)

INPUT FILE

```
0007 NEWRECRD 05/12/78 123456
```

SOURCE LISTING

```
C     THIS PROGRAM REPLACES THE SEVENTH RECORD IN
C     PERMANENT FILE SQFILE WITH A NEW RECORD EXISTING
C     ON FILE INPUT.  IT THEN READS AND PRINTS THE
C     FILE WITH THE REPLACED RECORD.
C
C
      PROGRAM REPLACE
      DIMENSION SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C', 'RT', 'F', 'FL', 30,
     +            'EFC', 3)
      CALL OPENM (SQFIT, 'I-O')
      CALL SKIP (SQFIT, +6)
      CALL GET (SQFIT, WSA)
      READ '(3A10)', WSA
      CALL REPLC (SQFIT, WSA)
      CALL REWND (SQFIT, WSA)
   10 CALL GET (SQFIT, WSA)
      IF (IFETCH (SQFIT, 'FP') .EQ. 8) GO TO 20
      PRINT 15, WSA
   15 FORMAT (1X,3A10)
      GO TO 10
   20 CALL CLOSEM (SQFIT)
      END
```

OUTPUT

```
0001 TIMESHAR 04/21/78 178900
0002 ARBITRON 01/13/78 161320
0003 TICKETRO 09/01/78 147380
0004 SYNTONIC 06/14/78 000116
0005 COMMCRED 05/11/78 004672
0006 SVBUREAU 05/13/78 000197
0007 NEWRECRD 05/12/78 123456
0008 LEARNCTR 03/06/78 000097
0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872
```

Figure 3-12. Replacing a Record in a Sequential File

Program ADDRECS, shown in figure 3-13, adds two records to file SQFILE. The input records exist initially on INPUT. The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILESQ statement.
- Opens the file for input/output and includes the extend (E) parameter; this sets the open flag (OF) field of the FIT to indicate file positioning at end of current information.
- Reads the input file into the working storage area (WSA); rewinds the file when end-of-file is encountered.
- Reads and prints the complete file.
- Tests for end-of-section (10_8 or 8) in the FP field to bypass file boundaries and avoid reprinting the last record of a section. (Refer to appendix F, Sequential File Boundary Processing.)
- Tests for end-of-information (100_8 or 64) in the FP field to determine the end of the file.
- Closes the file when end-of-information is reached.

The printout that follows the source listing reflects the addition of the two new records 0011 and 0012. The NOS operating system automatically extends the file. When operations are under NOS/BE, an EXTEND control statement must be included after execution.

A NOS file dump is included to illustrate the internal representation of updated file SQFILE.

INPUT FILE

```
0011 ADDREC11 11/11/78 000000
0012 ADDREC12 11/11/78 000000
```

SOURCE LISTING

```
C     THIS PROGRAM ADDS TWO RECORDS TO PERMANENT
C     FILE SQFILE.  THE INPUT RECORDS EXIST INITIALLY
C     ON FILE INPUT.  THE PROGRAM THEN READS AND PRINTS
C     THE COMPLETE FILE.
C
C
      PROGRAM ADDRECS
      DIMENSION SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C', 'RT', 'F', 'FL', 30,
     +            'EFC', 3)
      CALL OPENM (SQFIT, 'I-O', 'E')
   10 READ (*, '(3A10)', END=30) WSA
   20 CALL PUT (SQFIT, WSA)
      GO TO 10
   30 CALL REWND (SQFIT)
   40 CALL GET (SQFIT, WSA)
      IF (IFETCH (SQFIT, 'FP') .EQ. 8) GO TO 40
      IF (IFETCH (SQFIT, 'FP') .EQ. 64) GO TO 50
      PRINT 45, WSA
   45 FORMAT (1X,3A10)
      GO TO 40
   50 CALL CLOSEM (SQFIT)
      END
```

OUTPUT

```
0001 TIMESHAR 04/21/78 178900
0002 ARBITRON 01/13/78 161320
0003 TICKETRO 09/01/78 147380
0004 SYNTONIC 06/14/78 000116
0005 COMMCRED 05/11/78 004672
0006 SVBUREAU 05/13/78 000197
0007 NEWRECRD 05/12/78 123456
0008 LEARNCTR 03/06/78 000097
0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872
0011 ADDREC11 11/11/78 000000
0012 ADDREC12 11/11/78 000000
```

```
1       - FILE DUMP -       TDUMP,I=SQFILE.

 F  1 R   1 W     0- 3333 3334 5524 1115 0523   1001 2255 3337 5035 3450   4243 5534 4243 4433 3355   3333 3335 5501 2202 1124
                     0 0  0 1    T  I  M  E S    H A  R    0 4  / 2  1 /    7 8    1  7 8  9 0  0     0 0  0 2     A  R B  I T
 F  1 R   1 W     4- 2217 1655 3334 5034 3650   4243 5534 4134 3635 3355   3333 3336 5524 1103 1305   2422 1755 3344 5033 3450
                     R 0  N    0 1  / 1  3 /    7 8    1  6 1  3 2  0     0 0  0 3    T  I C  K E    T R  0    0 9  / 0  1 /
 F  1 R   1 W    10- 4243 5534 3742 3643 3355   3333 3337 5523 3116 2417   1611 0355 3341 5034 3750   4243 5533 3333 3434 4155
                     7 8    1  4 7  3 8  0     0 0  0 4    S  Y N  T 0     N I  C    0 6  / 1  .4 /    7 8    0  0 0  1 1  6
 F  1 R   1 W    14- 3333 3340 5503 1715 1503   2205 0455 3340 5034 3450   4243 5533 3337 4142 3555   3333 3341 5523 2602 2522
                     0 0  0 5    C  0 M  M C    R E  D    0 5  / 1  1 /    7 8    0  0 4  6 7  2     0 0  0 6     S  V B  U R
 F  1 R   1 W    20- 0501 2555 3340 5034 3650   4243 5533 3333 3444 4255   3333 3342 5516 0527 2205   0322 0455 3340 5034 3550
                     E A  U    0 5  / 1  3 /    7 8    0  0 0  1 9  7     0 0  0 7    N  E W  R E    C R  D    0 5  / 1  2 /
 F  1 R   1 W    24- 4243 5534 3536 3740 4155   3333 3343 5514 0501 2216   0324 2255 3336 5033 4150   4243 5533 3333 3344 4255
                     7 8    1  2 3  4 5  6     0 0  0 8    L  E A  R N    C T  R    0 3  / 0  6 /    7 8    0  0 0  0 9  7
 F  1 R   1 W    30- 3333 3344 5503 3102 0522   2303 1055 3337 5033 3550   4243 5534 3333 4235 3755   3333 3433 5504 0124 0123
                     0 0  0 9    C  Y B  E R    S C  H    0 4  / 0  2 /    7 8    1  0 0  7 2  4     0 0  1 0     D  A T  A S
 F  1 R   1 W    34- 2603 2355 3337 5034 4250   4243 5533 3334 4342 3555
                     V C  S    0 4  / 1  7 /    7 8    0  0 1  8 7  2
 -- END OF RECORD --

 F  1 R   2 W     0- 3333 3434 5501 0404 2205   0334 3455 3434 5034 3450   4243 5533 3333 3333 3355   3333 3435 5501 0404 2205
                     0 0  1 1    A  D D  R E    C 1  1    1 1  / 1  1 /    7 8    0  0 0  0 0  0     0 0  1 2     A  D D  R E
 F  1 R   2 W     4- 0334 3555 3434 5034 3450   4243 5533 3333 3333 3355
                     C 1  2    1 1  / 1  1 /    7 8    0  0 0  0 0  0
 -- END OF RECORD --

 -- END OF INFORMATION --

 -- END OF DUMP --
```

Figure 3-13. Adding Records to a Sequential File

REWINDING THE FILE

You can rewind a sequential file by executing a CALL REWND statement. The format is:

```
CALL REWND(fit)
```

Positioning is determined by the type of file.

- Mass storage file

  Rewind is to beginning-of-information, which is the beginning of the first user record.

- Unlabeled tape

  Rewind is to the load point of the volume currently mounted.

- Labeled tape

  Rewind is to a point after the labels. If the current file position is not within the first volume of a multivolume file, volume boundaries are crossed as necessary to return to the file beginning; rewind is to a point after the labels at the beginning of the first file volume.

The only parameter in the rewind request is the name of the array containing the FIT.

When the following statement is executed, the file described by SQFIT is rewound to beginning-of-information:

```
CALL REWND(SQFIT)
```

If a rewind request is issued after a write request, file processing is completed before rewind. Information in the central memory buffer is written to the device, and an end-of-information boundary is written.

PARTIAL RECORD PROCESSING

You can perform partial record processing operations on sequential files with any record type other than R. Use the CALL GETP statement to read less than a record. Use the CALL PUTP statement to construct a single record from selected portions of information.

Reading Partial Records

Partial records in a sequential file are read by executing a CALL GETP statement. The format is:

```
CALL GETP(fit,wsa,ptl,'SKIP',dx)
```

The file must be open for either input or input/output (the PD field is set to INPUT or IO). The following restrictions apply:

- Record type must not be R.
- The 'SKIP' parameter must not be used with U type records.
- For D and T record types, the first GETP of a record must encompass the entire fixed base portion of these records, MNR and HL characters, respectively.

The first parameter in the partial read request is the name of the array that contains the FIT. Additional parameters set the following FIT fields:

WSA	Working storage area that is to receive the partial record.
PTL	Partial transfer length, which is the number of characters to be transferred starting at the beginning of a record or at the current position within a record. For Z type records, the value must be a multiple of 10 characters.
'SKIP'	File positioning to the beginning of the next record before the partial transfer starts. No FIT fields are set by this request.
DX	End-of-data exit subroutine to be executed if a file boundary is encountered. No data is transferred when end-of-section, end-of-partition, or end-of-information is reached.

After a record is read, BAM updates the following FIT fields:

RL	Record length
RC	Record count

A series of CALL GETP statements reads a record sequentially. The first GETP always transfers data from the beginning of a record. Processing proceeds as follows:

1. BAM sets RL to the number of characters read.
2. BAM sets the PTL field to show the number of characters actually transferred to the working storage area. The PTL returned is the PTL requested unless end-of-record is reached first, in which case PTL is less than that requested.
3. If you omit the SKIP parameter, the next GETP transfers data beginning at the next unread character in the record. If you include the SKIP parameter, the next GETP transfers data beginning at the first character position in the next record.
4. When end-of-record is encountered, BAM sets the FP field to 20_8 and stops transferring data. A subsequent GETP clears the FP field and begins data transfer from the next record.

Partial records are read into working storage. The characters from PTL to the next word boundary are undefined. If a GETP is followed by a GET, BAM skips to end-of-record and then reads a full record.

When the following partial read request for 7 characters is executed, the first 10 characters of the record in the file described by SQFIT are read into working storage area WSA; characters 8, 9, and 10 are undefined:

```
CALL GETP(SQFIT,WSA,7,0,ENDEX)
```

The zero marks the position of the 'SKIP' parameter that is not being used. If end-of-data is reached during execution of this statement, control is transferred to subroutine ENDEX.

When the following partial read request is executed, the file is positioned to the beginning of the next record, and 20 characters of the record are read into working storage area WSA:

```
CALL GETP(SQFIT,WSA,20,'SKIP')
```

S Type Record Considerations

When the length of an S type record is unknown, partial read processing is accomplished as follows:

- MRL is ignored.
- GETP requests must be issued for PTL characters, where PTL is the length of the working storage area.
- When the first GETP is executed, the FP field indicates position in the midst of a logical record. RL is incremented by the number of characters read. When some subsequent GETP completes record retrieval and the FP field value is 20_8 (end-of-record), the length of the S record format becomes known.

You must check the FP field for end-of-record to determine when the end of the S type record has been reached.

Boundary Considerations

Partial read operations do not cross boundaries. For example, consider two T type records described as follows:

CP=9 Start of count field
CL=2 Length of count field
HL=20 Length of header portion of record
TL=15 Length of trailer item
Count field value of the first record=1
Count field value of the second record=2

You issue the first statement for the first record:

```
CALL GETP(SQFIT,WSA,20)
```

BAM determines the number of trailer items in the CL field to be 1, calculates record length to be 35, sets the RL field to 20, and returns the first 20 characters to WSA.

You issue the second statement for the first record:

```
CALL GETP(SQFIT,WSA,15)
```

BAM sets the FP field to indicate end-of-record, returns the remaining 15 characters to WSA, and sets RL to 35.

You issue the first statement for the second record:

```
CALL GETP(SQFIT,WSA,20)
```

BAM clears FP, determines the number of trailer items in the CL field to be 2, calculates record length to be 50, sets RL to 20, and returns the first 20 characters to WSA.

You issue the second statement for the second record:

```
CALL GETP(SQFIT,WSA,35)
```

BAM returns only 30 characters to WSA because GETP does not cross the record boundary. BAM sets RL to 50 and PTL to 30.

Writing Partial Records

Partial records in a sequential file are written by executing a CALL PUTP statement. The format is:

```
CALL PUTP(fit,wsa,ptl,rl,ex)
```

The file must be open for either output or input/output (the PD field is set to OUTPUT or IO). The following restrictions apply:

- Record type must not be R.
- For D and T record types, control information in the first part of the record must contain length information.

The first parameter in the partial write request is the name of the array that contains the FIT. Additional parameters set the following FIT fields:

WSA Working storage area from which the partial record is written to the file.

PTL Partial transfer length, which is the number of characters in this partial write.

RL Record length, which is the total number of characters in the record under construction.

EX Error exit subroutine to be executed if an error occurs.

When a partial record is written, BAM updates the following FIT fields:

RL Record length
RC Record count

A series of CALL PUTP statements constructs a single record from several pieces of information. Each PUTP causes PTL characters to be written from the beginning of the working storage area and added to the record currently under construction. The second partial write of a record begins after the last character written.

The manner in which a record is terminated depends on the record type as follows:

F and Z The record is terminated when FL is reached, even if more than FL characters are requested.

W and U The record is terminated when RL is reached.

D and T The record is terminated when RL calculated from length information in the first part of the record is reached.

S The record is terminated when RL is set on the first PUTP and RL is reached, or when a WEOR call is executed.

When the following statement is executed, the 20 characters in WSA are written as a portion of a 40-character record under construction for the file described by NEWFIT:

```
CALL PUTP(NEWFIT,WSA,20,40)
```

Sample Partial Record Processing Program

Program PARTIAL, shown in figure 3-14, reads partial records from permanent file SQFILE and constructs new records for file PFILE. The program performs the following:

- Dimensions two arrays for FIT construction, and one character array for working storage area.
- Assigns 10 blanks to BLANK.
- Defines the file characteristics of SQFILE and PFILE through CALL FILESQ statements.
- Opens SQFILE for input and PFILE for input/output. The pd parameter is omitted in the first OPENM call because input is the default. A warning message alerts you that the next OPENM specifies additional parameters.
- Stores blanks into the three words of WSA.
- Performs a skip request to the beginning of the next record and a partial read of 14 characters from the SQFILE record.
- Performs a partial write of 14 characters for construction of a 28-character record.
- Closes SQFILE when the file position (FP) field is equal to end-of-section (10_8 or 8).

SOURCE LISTING

```
C     THIS PROGRAM READS PARTIAL RECORDS FROM PERMANENT
C     FILE SQFILE AND USES THEM TO CONSTRUCT NEW
C     RECORDS FOR FILE PFILE.  THE CONTENTS OF PFILE
C     ARE THEN PRINTED.
C
C
      PROGRAM PARTIAL
      DIMENSION SQFIT(35), PFIT(35)
      CHARACTER WSA(3)*10
      CHARACTER BLANK*10
      DATA BLANK /'          '/
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C',
     +            'RT', 'F', 'FL', 30,
     +            'EFC', 3)
      CALL FILESQ (PFIT, 'LFN', 'PFILE',
     +            'BT', 'C',
     +            'RT', 'F', 'FL', 28,
     +            'EFC', 3)
      CALL OPENM (SQFIT)
      CALL OPENM (PFIT, 'I-O')
 WARNING  *         NUMBER OF ARGUMENTS IN REFERENCE TO _OPENM IS NOT CONSISTENT
   10 DO 20 J=1,3
   20 WSA(J)=BLANK
      CALL GETP (SQFIT, WSA, 14, 'SKIP')
      IF (IFETCH (SQFIT, 'FP') .EQ. 8) GO TO 40
   30 CALL PUTP (PFIT, WSA, 14, 28)
      GO TO 10
   40 CALL CLOSEM (SQFIT)
      CALL ENDFILE (PFIT)
      CALL REWND (PFIT)
   50 CALL GET (PFIT, WSA)
      IF (IFETCH (PFIT, 'FP') .EQ. 8) GO TO 70
   60 PRINT 65, WSA
   65 FORMAT (1X,2A10,A8)
      GO TO 50
   70 CALL CLOSEM (PFIT)
      STOP
      END

        1  WARNING   ERROR IN  PARTIAL
```

OUTPUT

```
0001 TIMESHAR 0002 ARBITRON
0003 TICKETRO 0004 SYNTONIC
0005 COMMCRED 0006 SVBUREAU
0007 NEWRECRD 0008 LEARNCTR
0009 CYBERSCH 0010 DATASVCS
```

Figure 3-14. Partial Record Processing

- Writes an end-of-partition to PFILE. (For C type blocks with records other than W, BAM writes an end-of-section before writing the requested end-of-partition.)
- Rewinds and prints PFILE.

The printout that follows the source listing shows the contents of the new file. Each 28-character record in PFILE has been constructed from the first 14 characters of two records in SQFILE.

REDEFINING THE FILE

You can use an existing sequential file to create a new file. The new file can reflect a different record type.

Writing W Type Records

Program REDEFIN, shown in figure 3-15, reads the previously created SQFILE, redefines the file with W type records, and creates a new file named WRECS. The existing file must be attached and the new file must be made permanent with appropriate control statements.

The program performs the following:

- Dimensions two arrays for FIT construction, and one array for working storage area.
- Defines the characteristics of the existing file SQFILE through the CALL FILESQ statement.
- Defines the characteristics of the new file WRECS through the CALL FILESQ statement. The record type parameter could have been omitted since W is the system default.
- Opens the existing file for input and the new file for input/output. The pd parameter is omitted in the first OPENM call because input is the default. A warning message alerts you that the next OPENM specifies additional parameters.
- Reads the existing records and writes them to the new file. The record length (RL) field is required when writing W type records; the length is specified through an IFETCH function.
- Closes both files when the file position (FP) field is not equal to end-of-record.

The NOS TDUMP output lists the ten W type records. Each record is preceded by a control word.

Reading W Type Records

Program RDWRECS, shown in figure 3-16, reads file WRECS. The file must be attached.

The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the characteristics of the file through the CALL FILESQ statement.
- Opens the file for input.
- Reads and prints the records.
- Closes the file when the file position (FP) field is not equal to end-of-record.

Notice the absence of the RL parameter, which was required when the file was written. Because record length information is maintained in the control word that precedes a record, the parameter is not required for a read operation. Any record length included in the read request would be ignored.

TAPE LABELING

Magnetic tape files are identified to the operating system by a combination of VSN, REQUEST, and LABEL control statements. Specific parameters, parameter values, and positioning requirements are described in the appropriate operating system reference manual. VSN supplies tape identification; REQUEST handles tape assignments; and LABEL supplies information for standard tape label processing.

Tape formats supported by the NOS operating system are:

SI binary
I
S/L

Tape formats supported by the NOS/BE operating system are:

SI binary
SI coded
S/L

BAM handles three classes of magnetic tape files: standard labeled files, nonstandard labeled files, and unlabeled files.

STANDARD LABELED FILES

A standard labeled file has tape labels that conform to ANSI standards. The labels are defined as 80-character records that precede and follow a file or volume of user records.

The labels identify the file and the volume on which it resides. In addition, the labels provide continuity for files that extend over more than one volume, and allow positioning to a particular file when several files exist on a single volume.

An expiration date within the labels protects the tape against accidental rewriting. If a label has not expired, the tape cannot be written without explicit operator action on NOS/BE; on NOS the job aborts. The retention date parameter should be included on the operating system LABEL control statement.

Operating system control statements can be used to list standard labels. LISTLB is available for listing under NOS; LISTMF is available for listing under NOS/BE.

SOURCE LISTING

```
C     THIS PROGRAM READS PERMANENT FILE SQFILE, REDEFINES
C     THE FILE WITH W TYPE RECORDS, AND CREATES A
C     NEW FILE NAMED WRECS.
C
C
      PROGRAM REDEFIN
      DIMENSION SQFIT(35), WRECFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +             'BT', 'C',
     +             'RT', 'F', 'MRL', 30,
     +             'WSA', WSA,
     +             'EFC', 3)
      CALL FILESQ (WRECFIT, 'LFN', 'WRECS',
     +             'BT', 'C',
     +             'RT', 'W', 'MRL', 30,
     +             'WSA', WSA,
     +             'EFC', 3)
      CALL OPENM (SQFIT)
      CALL OPENM (WRECFIT, 'I-O')
 WARNING  *         NUMBER OF ARGUMENTS IN REFERENCE TO _OPENM IS NOT CONSISTENT
   10 CALL GET (SQFIT)
      IF (IFETCH (SQFIT, 'FP') .NE. 16) GO TO 20
      CALL PUT (WRECFIT, WSA, IFETCH (SQFIT, 'RL'))
      GO TO 10
   20 CALL CLOSEM (SQFIT)
      CALL CLOSEM (WRECFIT)
      STOP
      END

        1  WARNING   ERROR IN  REDEFIN
```

OUTPUT

```
1        - FILE DUMP -        TDUMP,I=WRECS.

 F  1 R   1 W     0- 4000 0000 0000 0000 0003   3333 3334 5524 1115 0523   1001 2255 3337 5035 3450   4243 5534 4243 4433 3355
                    5                     C     0 0  0 1    T  I M  E S    H A  R    0 4  / 2  1 /    7 8    1  7 8  9 0  0
 F  1 R   1 W     4- 0000 0000 0004 0000 0003   3333 3335 5501 2202 1124   2217 1655 3334 5034 3650   4243 5534 4134 3635 3355
                                D         C     0 0  0 2    A  R B  I T    R 0  N    0 1  / 1  3 /    7 8    1  6 1  3 2  0
 F  1 R   1 W    10- 0000 0000 0004 0000 0003   3333 3336 5524 1103 1305   2422 1755 3344 5033 3450   4243 5534 3742 3643 3355
                                D         C     0 0  0 3    T  I C  K E    T R  0    0 9  / 0  1 /    7 8    1  4 7  3 8  0
 F  1 R   1 W    14- 0000 0000 0004 0000 0003   3333 3337 5523 3116 2417   1611 0355 3341 5034 3750   4243 5533 3333 3434 4155
                                D         C     0 0  0 4    S  Y N  T 0    N I  C    0 6  / 1  4 /    7 8    0  0 0  1 1  6
 F  1 R   1 W    20- 0000 0000 0004 0000 0003   3333 3340 5503 1715 1503   2205 0455 3340 5034 3450   4243 5533 3337 4142 3555
                                D         C     0 0  0 5    C  O M  M C    R E  D    0 5  / 1  1 /    7 8    0  0 4  6 7  2
 F  1 R   1 W    24- 0000 0000 0004 0000 0003   3333 3341 5523 2602 2522   0501 2555 3340 5034 3650   4243 5533 3333 3444 4255
                                D         C     0 0  0 6    S  V B  U R    E A  U    0 5  / 1  3 /    7 8    0  0 0  1 9  7
 F  1 R   1 W    30- 0000 0000 0004 0000 0003   3333 3342 5516 0527 2205   0322 0455 3340 5034 3550   4243 5534 3536 3740 4155
                                D         C     0 0  0 7    N  E W  R E    C R  D    0 5  / 1  2 /    7 8    1  2 3  4 5  6
 F  1 R   1 W    34- 0000 0000 0004 0000 0003   3333 3343 5514 0501 2216   0324 2255 3336 5033 4150   4243 5533 3333 3344 4255
                                D         C     0 0  0 8    L  E A  R N    C T  R    0 3  / 0  6 /    7 8    0  0 0  0 9  7
 F  1 R   1 W    40- 0000 0000 0004 0000 0003   3333 3344 5503 3102 0522   2303 1055 3337 5033 3550   4243 5534 3333 4235 3755
                                D         C     0 0  0 9    C  Y B  E R    S C  H    0 4  / 0  2 /    7 8    1  0 0  7 2  4
 F  1 R   1 W    44- 0000 0000 0004 0000 0003   3333 3433 5504 0124 0123   2603 2355 3337 5034 4250   4243 5533 3334 4342 3555
                                D         C     0 0  1 0    D  A T  A S    V C  S    0 4  / 1  7 /    7 8    0  0 1  8 7  2
 F  1 R   1 W    50- 5000 0000 0004 0000 0000
                    /           D
 — END OF RECORD —

 — END OF INFORMATION —

 — END OF DUMP —
```

Figure 3-15. Redefining the File

```
SOURCE LISTING

C     THIS PROGRAM READS PREVIOUSLY CREATED FILE
C     WRECS AND PRINTS THE RECORDS.
C
C
      PROGRAM RDWRECS
      DIMENSION WRECFIT(35), WSA(3)
      CALL FILESQ (WRECFIT, 'LFN', 'WRECS',
     +             'BT', 'C',
     +             'RT', 'W', 'MRL', 30,
     +             'EFC', 3)
      CALL OPENM (WRECFIT)
   10 CALL GET (WRECFIT, WSA)
      IF (IFETCH (WRECFIT, 'FP') .NE. 16) GO TO 30
   20 PRINT 25, WSA
   25 FORMAT (1X,3A10)
      GO TO 10
   30 CALL CLOSEM (WRECFIT)
      STOP
      END


OUTPUT

0001 TIMESHAR 04/21/78 178900
0002 ARBITRON 01/13/78 161320
0003 TICKETRO 09/01/78 147380
0004 SYNTONIC 06/14/78 000116
0005 COMMCRED 05/11/78 004672
0006 SVBUREAU 05/13/78 000197
0007 NEWRECRD 05/12/78 123456
0008 LEARNCTR 03/06/78 000097
0009 CYBERSCH 04/02/78 100724
0010 DATASVCS 04/17/78 001872
```

Figure 3-16. Reading W Type Records

System Processing of Standard Labels

When a standard labeled file is opened, VOL1 and HDR1 labels are processed by the system. If the file is an input tape, parameters on the appropriate operating system control statement identifying the file are compared with information present on the tape. If the file is an output tape, parameters on the LABEL control statement are used to write the labels; default values are supplied for any field not specified.

When a standard labeled file is closed, the EOF1 trailer label is written by the system. If the file is continued to another reel of tape, an EOV1 rather than an EOF1 label is written at the end of the first reel; VOL1 and HDR1 labels are written on the second reel; and the file is continued. An EOF1 label for a multireel file appears at the file end; each file has only one EOF1 label. A file can have more than one HDR1 label because the first information for a file on a reel is the HDR1 label, even if that file is a continuation from a previous reel.

Two FIT fields apply to standard labeled files that are processed by the system:

LCR	Label check/creation
LT	Label type

You can specify the LCR field as a parameter on a FILE control statement, CALL FILESQ statement, or CALL STOREF statement. You can specify the following values for the LCR field:

CHK	The existing label is read and checked.
CRT	A new label is written.

You must set the LT field to a value; you can set it with a CALL FILESQ, CALL STOREF, or a FILE control statement. You must specify one of the following values for the LT field:

S	ANSI standard label
ANY	Any label type

User Processing of Standard Labels

User processing of standard labels is permitted on some label groups. All processing must be performed by a COMPASS subroutine called from within a FORTRAN program. COMPASS macros GETL, PUTL, and CLOSEL provide for user processing of standard labels. Refer to the BAM reference manual for details.

Six FIT fields apply to standard labeled files that you process:

LCR	Label check/creation
LT	Label type
ULP	User label processing
LX	Label exit
LA	User label area
LBL	Label area length

You can specify the LCR field as a parameter on a FILE control statement, CALL FILESQ statement, or CALL STOREF statement. You can specify the following values for the LCR field:

CHK	The existing label is read and checked.
CRT	A new label is written.

You must set the LT field to a value; you can set it with a CALL FILESQ, CALL STOREF, or FILE control statement. You must specify the following value for the LT field:

S	ANSI standard label

You must set the ULP field to a value; you can set it with a CALL FILESQ, CALL STOREF, or FILE control statement. You must specify one of the following values for the ULP field:

V	Volume label processing
F	File label processing
U	User header and volume labels that can follow the HDR and VOL1 labels; and user trailer labels that can follow EOF and EOV labels
VF	Combination of V and F
VU	Combination of V and U
FU	Combination of F and U
VFU	Combination of V, F, and U

You must set the LX field to a value; you can set it with a CALL FILESQ or CALL STOREF statement. The field specifies the name (address) of the label routine to receive control during open and close processing, in accordance with the ULP value.

The LA field indicates the name (address) of the array that is the label area; when the label is retrieved, it is delivered to this area. You can set the field with a CALL FILESQ or CALL STOREF statement; you can also set it with COMPASS macros GETL and PUTL.

The LBL field indicates the length in characters of the label area (1-900). You can set the field with a CALL FILESQ, CALL STOREF, or FILE control statement; you can also set it with COMPASS macros GETL and PUTL.

NONSTANDARD LABELED FILES

A nonstandard labeled file has tape labels that do not conform to ANSI standards. The labels can be either header or trailer labels or a combination of both. The delimiting and processing of nonstandard labels is the responsibility of the user. The system does not process nonstandard labels.

To process nonstandard labeled tape through BAM on NOS, you must specify the LB=KU parameter on the LABEL control statement. On NOS/BE, you must specify the NS parameter on the REQUEST control statement.

All processing must be performed by a COMPASS subroutine called from within a FORTRAN program. COMPASS macros GETL, PUTL, and CLOSEL provide for user label processing. Refer to the BAM reference manual for details.

Six FIT fields apply to nonstandard labeled files:

LCR	Label check/creation
LT	Label type
ULP	User label processing
LX	Label exit
LA	User label area
LBL	Label area length

You can specify the LCR field as a parameter on a FILE control statement, CALL FILESQ statement, or CALL STOREF statement. You can specify the following values for the LCR field:

CHK	The existing label is read and checked.
CRT	A new label is written.

You must set the LT field to a value; you can set it with a CALL FILESQ, CALL STOREF, or FILE control statement. You must specify the following value for the LT field:

NS	Nonstandard label

You must set the ULP field to a value; you can set it with a CALL FILESQ, CALL STOREF, or FILE control statement. You can specify any of the following values for the ULP field:

V
F
U
VF
VU
FU
VFU

The volume, file, and user settings are meaningless without standard labels; any can be arbitrarily selected to indicate user processing.

You must set the LX field to a value; you can set it with a CALL FILESQ or CALL STOREF statement. The field specifies the name (address) of the label routine to receive control during open and close processing.

The LA field indicates the name (address) of the array that is the label area; when the label is retrieved, it is delivered to this area. You can set the field with a CALL FILESQ or CALL STOREF statement; you can also set it with COMPASS macros GETL and PUTL.

The LBL field indicates the length in characters of the label area (1-900). You can set the field with a CALL FILESQ, CALL STOREF, or FILE control statement; you can also set it with COMPASS macros GETL and PUTL.

UNLABELED FILES

An unlabeled file has no tape labels; that is, no descriptive records appear at the beginning of the file. The first block of the file is treated as a data block.

An unlabeled file on an SI or I tape has a system-processed trailer label (EOF1) so that end-of-information can be defined. Multivolume processing is performed automatically by the operating system.

An unlabeled file on an S or L tape has no system trailer label; end-of-information is undefined. On input, a tapemark encountered after the end-of-tape reflective marker signals end-of-volume and the operating system switches volumes. When reading an unlabeled S or L tape, BAM cannot detect the end of valid data records because end-of-information is undefined. Therefore, it is possible to read past the last valid data block into the undefined area that precedes the end-of-tape marker. Reading undefined data or blank tape can produce error conditions.

One FIT field applies to unlabeled files:

LT	Label type

The LT field must reflect an unlabeled file; you can set it with a CALL FILESQ, CALL STOREF, or FILE control statement. The parameter can be set to UL (unlabeled) or eliminated to default to unlabeled.

WORD ADDRESSABLE FILE PROCESSING 4

A word addressable file is a mass storage file of records that are accessed by the number of the first word in each record. The file is simply a group of logically contiguous computer words; file boundaries are beginning-of-information and end-of-information. Blocking concepts, record keys, and file labels do not apply to this file organization.

Records are read or written beginning with a specified word address. Information can be retrieved by one direct access for an individual item or sequentially for a series of items by accessing the first item.

CONCEPTS OF PHYSICAL FILE STRUCTURE

Word addressable files must reside on mass storage. The first word in a file is considered to have the word address 1, the tenth word has the word address 10, and so forth. Each word contains 10 character positions.

No logical boundaries appear between records in a word addressable file. Each record begins in a new word. If a record has more than 10 characters, contiguous words are used as necessary. If the record length is not a multiple of 10, the rightmost character positions in the last word are undefined. Only two restrictions are imposed: writing always begins at the left on a word boundary, and the system does not check to ensure that words are not overwritten.

System efficiency is increased if all records are multiples of 64 words. Performance improvement is recognized when:

RL=n*64
WA=(m*64)+1

The letters n and m represent integers.

You can write variable length records contiguously, but you have the burden of finding the word address of the start of a record to be read. You could define and process record indexes, but so doing would defeat the single access advantage of a word addressable file.

BAM allocates storage in fixed units to a word addressable file; consequently, meaningless data usually follows the last valid record. Since record terminators do not exist, you must be familiar with the data in a file.

AVAILABLE RECORD TYPES

Three types of records can exist in word addressable files: fixed length (F), undefined (U), and control word (W).

DESCRIBING F TYPE RECORDS

F type records require a value in the fixed length (FL) field of the FIT. You must set the FL field before a read or write operation. Inclusion of the record length parameter in a read or write request is ignored when you use fixed length records.

DESCRIBING U TYPE RECORDS

U type records require a value in the record length (RL) field of the FIT for a read or write operation. You must include the RL parameter in each read or write request. You must set the maximum record length (MRL) field of the FIT before a read operation.

To illustrate a word addressable file with U type records, assume that parts for an assembly are numbered 1 through 1000 and each assembly description includes:

- A 10-character identifying name
- A 5-character assembly identification
- A 5-character number of parts on hand
- A 10-character manufacturer's code

The part number cannot be the user-defined key because a 30-character entry would overlap the next two word addresses. A key of three times the part number, however, effectively produces a table of 1000 three-word entries. (The first two words in the file are unused when this calculation is followed.)

To read an entire 30-character entry, the part number is multiplied by 3 and a read is issued for 30 characters. Any individual word in the entry could be read by multiplying the part number by 3; adding 0, 1, or 2; and issuing a read for 10 characters. If a 30-character read was issued after an addition of 1 or 2, part of the next entry would be read; consequently, you must be familiar with the data because no record terminator exists.

The file just described, which can be likened to a single record or single table, has many uses in internal system routines. An example would be when a programmer is building indexes or tree structures in which the location of data is random, and the amount of data required changes from read to read.

DESCRIBING W TYPE RECORDS

The word address of a W type record is always the address of the control word, not the first word of the user data. This applies to both read and write operations.

You must specify the record length (RL) field of the FIT for a write operation. You must set the field to the actual number of characters in the data record with no adjustment made for the control word. When W type records are written, the word address specified for the write request must be adjusted to acknowledge the control word. For instance, if all user records are 150 characters and F type records are being written, the word address for the write could be some multiple of 15. If the same records were defined as W type, the word address for the write would need to be a multiple of 16.

You must specify the maximum record length (MRL) field of the FIT before a read operation. When W type records are read, the word address specified for the read request requires no adjustment for the control word. The data record, without the control word, is returned to you.

SUMMARY OF RECORD TYPES

Figure 4-1 illustrates word addressable file record types. Each FIT field that must be set, either by specification or by default, is included.

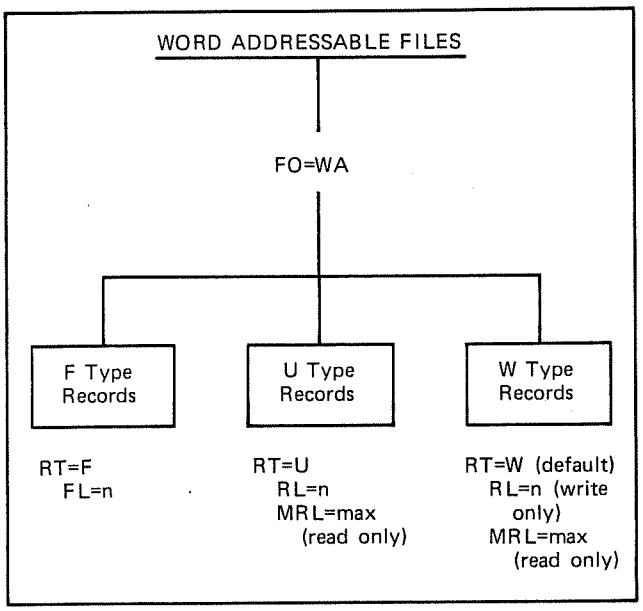


Figure 4-1. Word Addressable File Record Type Summary

CREATING A WORD ADDRESSABLE FILE

A word addressable file is created by writing records to a file with a defined record type. You can declare file characteristics through a combination of FILE control statements, source language statements, and installation default values. Once the file structure is defined, records are inserted into the file by the CALL PUT statement.

FIT fields that must be defined on a word addressable file creation run are as follows:

LFN	Logical file name consisting of one to seven characters and beginning with a letter
FO	File organization set to WA
PD	Processing direction set to OUTPUT or I-O
RT	Record type and any fields required by the record type (default is W)

Other optional FIT fields that can be defined on the creation run and can be changed on a subsequent run are as follows:

DX	End-of-data exit (default is no exit)
EX	Error exit (default is no exit)
ERL	Trivial error limit (default is no limit)
DFC	Dayfile control (default is only fatal errors to the dayfile)
EFC	Error file control (default is no messages to the error file)
BFS	Buffer size (default is buffer size calculated by BAM)
FWB	First word address of the buffer (default is buffer address provided by BAM)

When you create a word addressable file, you must define file characteristics by setting applicable fields in the FIT before the file is opened. Except as noted in appendix D, you can specify FIT fields in the FILE control statement, the CALL FILEWA statement, or the CALL STOREF statement. After all records have been written, you must close the file with the CALL CLOSEM statement.

On a file creation run, the only statements you can issue are those that perform the following:

- Establish the FIT.
- Open and close the file.
- Write records.
- Store and fetch FIT fields.

ESTABLISHING THE FIT

The first statement referencing the word addressable file must be the CALL FILEWA statement. When this statement is executed, the FIT is constructed and the specified values are stored in appropriate FIT fields. The first parameter in the CALL FILEWA statement is the name of the array to hold the FIT. The same FIT array name is the first parameter in every statement accessing the word addressable file. Refer to section 2, File Processing Concepts, for a more detailed explanation of the FIT and the CALL FILEWA statement.

OPENING THE FILE

You must open the word addressable file with a CALL OPENM statement before any records can be written to the file. The format is:

```
CALL OPENM(fit,pd)
```

Open processing includes storing FILE control statement values in the FIT, processing buffer parameters, supplying default values for FIT fields not set by the source program, and checking the FIT for logical consistency and required fields.

The first parameter in the open request for file creation is the name of the array that contains the FIT. The second parameter must specify I-O or OUTPUT; this sets the processing direction (PD) field in the FIT to read/write or write only.

When the following statement is executed, the file identified by the FIT in array WAFIT is opened for a read or write operation:

```
CALL OPENM(WAFIT,'I-O')
```

WRITING RECORDS

Records are written to a word addressable file by executing a CALL PUT statement. The format is:

```
CALL PUT(fit,wsa,rl,wa,0,0,ex)
```

Each record to be written must be established at the working storage area location.

The first parameter in the write request for file creation is the name of the array that contains the FIT. Other parameters set FIT fields that BAM uses to write the record to the file. The following FIT fields can be set by the write request:

WSA — Working storage area from which the record is written to the file.

RL — Record length, which is required for U and W type records.

WA — Word address, which is the relative address at which the record is to be written counting the first word in the file as 1. The parameter can be omitted for a sequential write.

EX — Error exit subroutine to be executed if an error occurs.

Fields not specified in the write request default to the current value in the FIT. After the record is written, BAM updates the following FIT fields:

RL — Record length

WA — Word address

The record need not be a multiple of 10 characters. The contents of the unused character positions in the last word of the record cannot be guaranteed, however, and they can change as a record is moved from the working storage area through the buffer to a mass storage device. Writing always begins on a word boundary; unused character positions from a previous write are not filled. The WA field is set to the word after the last word written.

Writing always begins at the start of the word indicated by WA. If the specified word address is beyond the end of the current file, the file is extended as necessary to include that address.

When the following statement is executed, a record is added to the file identified by the FIT in array WAFIT:

```
CALL PUT(WAFIT,WSA,30,64,0,0,ERREXIT)
```

The array WSA is the working storage area that contains the record to be written. The record contains 30 characters and is to be written to word address 100_8 or 64. The two zeros represent parameters that are not applicable to word addressable files. If an error occurs during execution of this statement, control is transferred to subroutine ERREXIT. The WA field is set to 103_8.

CLOSING THE FILE

The last program statement referencing the file must be a CALL CLOSEM statement. The format is:

```
CALL CLOSEM(fit,cf)
```

When the statement is executed, any data in the central memory buffer is written to the file.

The first parameter in the close request is the name of the array that contains the FIT.

The second parameter sets the close flag (CF) field, which provides for file positioning and disposition. The following values can be specified for the CF field:

- R (rewind)

  The file is rewound to beginning-of-information and the open/close flag (OC) field is set to closed. This is the default setting.

- N (no rewind)

  The file is not rewound and the OC field is set to closed; the file remains at the current position.

- U (unload)

  The file is rewound, the OC field is cleared, the file is detached from the job, and scratch mass storage space assigned to the file is released.

- RET (return)

  The file is rewound, the OC field is cleared, the file is detached from the job, and buffer space is released.

- DET (detach)

  The file is not rewound, the OC field is cleared, buffer space is released, and the file is disassociated from the job.

A close request issued for a file that has never been opened, or that has been closed but neither released nor reopened, results in a trivial error. The file is positioned as specified before the error is issued.

When the CF field is set to R or N and the file is subsequently reopened, FIT verification and FILE control statement processing are not repeated. When the CF field is set to U, RET, or DET, FIT verification and FILE control statement processing are repeated; to resume processing, the file must be reattached to the job and opened.

To ensure allocation of a new buffer when a file is reopened in a program, the file must be closed with the CF field set to DET, and the BFS field must be reset.

When the following statement is executed, the file identified by the FIT in array WAFIT is rewound, the OC flag is cleared, buffer space is released, and the file is disassociated from the job:

```
CALL CLOSEM(WAFIT,'RET')
```

SAMPLE CREATION PROGRAM

Program WACREAT, shown in figure 4-2, creates a permanent word addressable file through direct calls to BAM. The illustration includes the control statements used for the NOS and NOS/BE operating systems, the input file, the source listing, a printout of the contents of the input file, and an octal dump of the newly created word addresable file. The program reads a ten-record input file and stores the records on file WAFILE. The input records for creating the file exist initially on INPUT.

CONTROL STATEMENTS

NOS Operating System

```
Job statement
USER control statement
CHARGE control statement
FTN5.
DEFINE(WAFILE/CT=PU,M=W)
LGO.
TDUMP,I=WAFILE.
CRMEP(LO)
```

NOS/BE Operating System

```
Job statement
ACCOUNT control statement
FTN5.
REQUEST(WAFILE,*PF)
LGO.
CATALOG(WAFILE,ID=BAMUG)
CRMEP(LO)
```

INPUT FILE

```
ABROPS RAPID CITY SD    421.00
AKOSBC AKRON      OH  10468.00
BLTSYN BALTIMORE  MD    536.00
CHESRC CHEYENNE   WY   7000.00
DASSBC DALLAS     TX  98760.00
OKHMPI TULSA      OK   1000.00
ORLSYN ORLANDO    FL   4921.00
PRTSBC PORTLAND   OR   9274.00
PTBCPO PITTSBURGH PA    793.00
RDUSYN RALEIGH    NC    762.00
```

SOURCE LISTING

```
C     THIS PROGRAM CREATES A PERMANENT WORD ADDRESSABLE
C     FILE (WAFILE) THROUGH DIRECT CALLS TO BAM.  THE
C     INPUT RECORDS EXIST INITIALLY ON INPUT.  THE PROGRAM
C     READS THE TEN-RECORD INPUT FILE, PRINTS EACH RECORD,
C     AND STORES THE RECORDS ON FILE WAFILE.
C
C
      PROGRAM WACREAT
      DIMENSION WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +             'RT', 'F', 'FL', 30,
     +             'EFC', 3)
      CALL OPENM (WAFIT, 'I-O')
   10 READ (*, '(3A10)', END=30) WSA
      PRINT 20, WSA
   20 FORMAT (1X,3A10)
      CALL PUT (WAFIT, WSA)
      GO TO 10
   30 CALL CLOSEM (WAFIT)
      STOP
      END
```

OUTPUT

```
ABROPS RAPID CITY SD    421.00
AKOSBC AKRON      OH  10468.00
BLTSYN BALTIMORE  MD    536.00
CHESRC CHEYENNE   WY   7000.00
DASSBC DALLAS     TX  98760.00
OKHMPI TULSA      OK   1000.00
ORLSYN ORLANDO    FL   4921.00
PRTSBC PORTLAND   OR   9274.00
PTBCPO PITTSBURGH PA    793.00
RDUSYN RALEIGH    NC    762.00
```

```
CRMEP(LO)
```

Figure 4-2. Creating a Word Addressable File (Sheet 1 of 2)

```
1        - FILE DUMP -        TDUMP,I=WAFILE.

 F  1 R   1 W     0- 0102 2217 2023 5522 0120   1104 5503 1124 3155 2304   5555 5555 3735 3457 3333   0113 1723 0203 5501 1322
                        A B  R O  P S    R  A P    I D    C  I T  Y   S D             4 2  1 .  0 0    A K  O S  B C    A  K R
 F  1 R   1 W     4- 1716 5555 5555 5555 1710   5555 3433 3741 4357 3333   0214 2423 3116 5502 0114   2411 1517 2205 5555 1504
                        O N                O H        1 0  4 6  8 .  0 0    B L  T S  Y N    B  A L    T I  M O  R E       M D
 F  1 R   1 W    10- 5555 5555 4036 4157 3333   0310 0523 2203 5503 1005   3105 1616 0555 5555 2731   5555 5542 3333 3357 3333
                             5 3  6 .  0 0    C H  E S  R C    C  H E    Y E  N N  E         W Y          7  0 0  0 .  0 0
 F  1 R   1 W    14- 0401 2323 0203 5504 0114   1401 2355 5555 5555 2430   5555 4443 4241 3357 3333   1713 1015 2011 5524 2514
                        D A  S S  B C    D  A L    L A S              T X         9 8  7 6  0 .  0 0    O K  H M  P I    T  U L
 F  1 R   1 W    20- 2301 5555 5555 5555 1713   5555 5534 3333 3357 3333   1722 1423 3116 5517 2214   0116 0417 5555 5555 0614
                        S A                O K        1  0 0  0 .  0 0    O R  L S  Y N    O  R L    A N  D O             F L
 F  1 R   1 W    24- 5555 5537 4435 3457 3333   2022 2423 0203 5520 1722   2414 0116 0455 5555 1722   5555 5544 3542 3757 3333
                             4  9 2  1 .  0 0    P R  T S  B C    P  O R    T L  A N  D         O R          9  2 7  4 .  0 0
 F  1 R   1 W    30- 2024 0203 2017 5520 1124   2423 0225 2207 1055 2001   5555 5555 4244 3657 3333   2204 2523 3116 5522 0114
                        P T  B C  P O    P  I T    T S  B U  R G  H    P A             7 9  3 .  0 0    R D  U S  Y N    R  A L
 F  1 R   1 W    34- 0511 0710 5555 5555 1603   5555 5555 4241 3557 3333   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000
                        E I  G H           N C        7 6  2 .  0 0
 F  1 R   1 W    40- 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000

 -- ABOVE LINE REPEATED --

 F  1 R   1 W    74- 0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000

 -- END OF INFORMATION --

 -- END OF DUMP --
```

Figure 4-2. Creating a Word Addressable File (Sheet 2 of 2)

Note the following information about the control statements:

- The DEFINE control statement effects permanent file storage of WAFILE on NOS. REQUEST and CATALOG control statements must be substituted for DEFINE when operation is under NOS/BE.
- The TDUMP control statement dumps the contents of the permanent file. This control statement is not available for operation under NOS/BE.
- The CRMEP control statement processes the error file. The mnemonic LO indicates all messages are to be displayed. The CRMEP control statement is detailed in section 5.

Statements in the program are defined as follows:

- DIMENSION WAFIT(35), WSA(3)

  This statement allocates a 35-word array named WAFIT for FIT construction and a 3-word working storage area named WSA.

- CALL FILEWA (WAFIT, 'LFN', 'WAFILE', . . .)

  This statement sets fields in the FIT to describe the structure of the word addressable file. Four required parameters are included:

  FIT array (WAFIT)

  Logical file name (WAFILE)

  Record type (fixed)

  Fixed record length (30 characters)

  One optional parameter is included:

  Error file control (3, errors and notes are to be written to the error file)

- CALL OPENM (WAFIT, 'I-O')

  This statement opens the file described by WAFIT for reading and writing.

- READ (*, '(3A10)', END=30) WSA
  PRINT 20, WSA

  These statements read the input file into the working storage area (WSA) and print the contents. If end-of-file is encountered, control is transferred to a statement that closes the file.

- CALL PUT (WAFIT, WSA)

  This statement writes the record from the working storage area to the BAM file. Since the program is performing a sequential write, the WA parameter is not required.

- CALL CLOSEM (WAFIT)

  This statement writes the buffer to the file and writes end-of-information.

A printout of the ten-record input file follows the source listing. Each record contains 30 characters and, therefore, occupies three words in memory. When this file is subsequently processed, record word addresses will be calculated in multiples of three.

The file dump illustrates the internal representation of WAFILE. Notice the last valid record ends in file storage word 35 and no record terminator exists. Meaningless data fills the remaining storage allocation for the file.

PROCESSING A WORD ADDRESSABLE FILE

After the file creation run, a word addressable file can be attached, read, updated, and rewound. File processing is governed by many of the FIT fields set on the file creation run.

You cannot change the following FIT fields when an existing file is being processed:

FO	File organization
RT	Record type
FL	Fixed record length (F type records only)

You must set the following FIT fields before the file is opened unless the default values are to be accepted:

BFS	Buffer size
FWB	First word address of the buffer

Optional fields that you can set at any time before they are required by a file processing statement are as follows:

DX	End-of-data exit
EX	Error exit
ERL	Trivial error limit
DFC	Dayfile control
EFC	Error file control

ESTABLISHING THE FIT

The FIT is established for an existing file in the same manner as for a new file during the creation run. The CALL FILEWA statement is required for any program using the file. All parameters applicable to structure must be repeated in each program in which the file is processed. BAM does not make such information a part of the file.

You can use the CALL FILEWA statement to specify all FIT fields required to define minimum file structure. These values become part of the FIT during execution, but can be overwritten by FILE control statement values at the time the file is opened.

OPENING THE FILE

Before you can access any data records in an existing file, you must open the file with a CALL OPENM statement. The format is:

```
CALL OPENM(fit,pd,of)
```

Open request parameters are stored in applicable FIT fields during open processing. FILE control statement processing and FIT consistency checking are performed in the same manner as on a file creation run.

The first parameter in the open request is the name of the array that contains the FIT. Additional parameters can set the following FIT fields:

PD	Processing direction
OF	Open flag

The setting of the PD field determines the input/output statements that can be executed. You can set the PD field as follows:

INPUT	Statements that read the file can be executed (default).
OUTPUT	Only statements that write new records to the file can be executed.
I-O	Any file processing statement related to input/output can be executed. (If the PD field is set by any statement other than the OPEN statement, the characters IO rather than I-O must be specified.)

A trivial error occurs if execution of a file processing statement is attempted and the PD field is not set to an appropriate value for that statement.

The setting of the OF field determines file positioning. Since word addressable files can be accessed randomly according to the WA field setting, file positioning parameters on an open request are generally not applicable. The OF field, however, can be set as follows:

R	The file is rewound to beginning-of-information (default).
N	The file is not rewound.

When the following statement is executed, the file described by WAFIT is opened for input:

```
CALL OPENM(WAFIT)
```

READING RECORDS

Records in a word addressable file can be read sequentially or randomly by executing a CALL GET statement. The format is:

```
CALL GET(fit,wsa,wa,0,0,rl,ex)
```

The file must be attached and open for either input or input/output (the PD field is set to INPUT or IO).

The first parameter in the read request is the name of the array that contains the FIT. Additional parameters set the following FIT fields:

WSA	Working storage area that is to receive the record.
WA	Word address, which is the relative address of the record to be read, counting the first word in the file as 1. The parameter can be omitted for a sequential read.
RL	Record length, which is required only for U type records.
EX	Error exit subroutine to be executed if an error occurs.

Fields not specified in the read request default to the current value in the FIT. After the record is read, BAM updates the following FIT fields:

RL Record length

WA Word address

Beginning at the word address specified by the WA field of the FIT, the number of characters specified by the RL or FL field are transferred to the working storage area. When the read is complete, BAM sets the WA field to the next unread word. This allows you to read an entire file sequentially without the necessity of resetting WA once the initial file position is established.

A sequential read presumes that the file contains contiguous records. Meaningless data can be returned to the working storage area if data is not contiguous. If the file contains W type records, errors are reported if the word at WA cannot be interpreted as a correctly formatted W type control word.

For U and W type records, the MRL field of the FIT sets a maximum that overrides any other length specification for the data transfer. If the record length for a read request is greater than MRL, only MRL characters are transferred to the working storage area; error code 142_8 is returned to indicate excess data. If the record length is greater than the record limit indicated by a W type control word, error code 143_8 is returned to indicate insufficient data is returned; in this instance, fewer than requested characters are returned.

When the following statement is executed, the record starting at word address 10_8 or 8 in the file described by WAFIT is read into working storage area WSA:

```
CALL GET(WAFIT,WSA,8,0,0,30,ERREXIT)
```

The two zeros represent parameters that are not applicable to word addressable file organization. The record to be read contains 30 characters. If an error occurs during execution of this statement, control is transferred to subroutine ERREXIT.

Program WAREAD, shown in figure 4-3, reads the previously created file WAFILE. The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILEWA statement.
- Opens the file for input and begins a sequential read.
- Stores a 1 in the WA field to indicate reading is to begin with the record at word address 1.
- Moves the contents of the WA field to FORTRAN variable IWA. The WA field is automatically incremented by BAM after each read.
- Tests IWA for a value greater than 34_8 or 28. You calculate this value known to be the address of the last valid record.
- Reads and prints each record in the file.
- Begins a random read.
- Reads the record starting at word address 20_8 or 16 by including the word address in the CALL GET statement. A warning message is issued to inform you that the previous GET call specified fewer parameters.
- Reads the record starting at word address 7 by storing the word address in the WA field before issuing the CALL GET statement.

The printout that follows the source listing includes the word address for the sequential read. Since each record contains 30 characters and therefore occupies 3 words in memory, each address is a multiple of 3.

ADDING RECORDS

You can add records to an existing word addressable file sequentially or randomly by executing a CALL PUT statement. The file must be attached and open for either output or input/output. (The PD field is set to OUTPUT or IO.)

You can add a single record by specifying the word address where it is to be stored. You can add several records sequentially by specifying the starting word address for the first record; as subsequent records are written, BAM increments the word address and stores them accordingly.

Program ADDRECS, shown in figure 4-4, adds five records to existing file WAFILE. The first input file (TAPE1) contains four records for a sequential write; the second input file (TAPE2) contains one record for a random write. The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILEWA statement.
- Opens the file for input/output and begins a sequential write.
- Stores a 42_8 or 34 in the WA field to indicate writing is to begin at word address 42_8.
- Reads the records in TAPE1 into the working storage area (WSA) and prints the contents.
- Begins a random write.
- Stores a 37_8 or 31 in the WA field to indicate writing at word address 37_8.
- Reads the record in TAPE2 into the working storage area and prints the contents.
- Stores a 1 in the WA field and begins a sequential read.
- Tests IWA for a value greater than 53_8 or 43. You calculate this value known to be the address of the last valid record.
- Reads and prints the file.

The printout that follows the source listing shows the four new input records SYRACUSE, DAYTON, WARWICK, and MEMPHIS were written sequentially to word addresses 42_8 through 53_8. Input record RICHMOND was written randomly to word address 37_8.

A file dump is included to illustrate the internal representation of updated file WAFILE.

SOURCE LISTING

```
C     THIS PROGRAM ILLUSTRATES A SEQUENTIAL AND RANDOM
C     READ OF PREVIOUSLY CREATED FILE WAFILE.
C
C
      PROGRAM WAREAD
      DIMENSION WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +            'RT', 'F', 'FL', 30,
     +            'EFC', 3)
C
C     DO A SEQUENTIAL READ OF WAFILE BEGINNING AT WORD
C     ADDRESS 1 AND PRINT EACH RECORD WITH ITS WORD ADDRESS.
C
      CALL OPENM (WAFIT)
      CALL STOREF (WAFIT, 'WA', 1)
      PRINT 5
    5 FORMAT (' ',/,' SEQUENTIAL READ',/)
   10 IWA=IFETCH (WAFIT, 'WA')
      IF (IWA .GT. 28) GO TO 20
      CALL GET (WAFIT, WSA)
      PRINT 15, WSA, IWA
   15 FORMAT (1X,3A10,1X,04)
      GO TO 10
C
C     DO A RANDOM READ OF THE RECORDS AT WORD ADDRESSES
C     16 AND 7 AND PRINT EACH RECORD.
C
   20 PRINT 25
   25 FORMAT (' ',/,' RANDOM READ',/)
      CALL GET (WAFIT, WSA, 16)
 WARNING  *          NUMBER OF ARGUMENTS IN REFERENCE TO _GET IS NOT CONSISTENT
      PRINT 30, WSA
   30 FORMAT (1X,3A10)
      CALL STOREF (WAFIT, 'WA', 7)
      CALL GET (WAFIT, WSA)
      PRINT 30, WSA
      CALL CLOSEM (WAFIT)
      STOP
      END

        1  WARNING   ERROR IN  WAREAD
```

OUTPUT

```
SEQUENTIAL READ

ABROPS RAPID CITY SD    421.00 0001
AKOSBC AKRON      OH  10468.00 0004
BLTSYN BALTIMORE  MD    536.00 0007
CHESRC CHEYENNE   WY   7000.00 0012
DASSBC DALLAS     TX  98760.00 0015
OKHMPI TULSA      OK   1000.00 0020
ORLSYN ORLANDO    FL   4921.00 0023
PRTSBC PORTLAND   OR   9274.00 0026
PTBCPO PITTSBURGH PA    793.00 0031
RDUSYN RALEIGH    NC    762.00 0034

RANDOM READ

OKHMPI TULSA      OK   1000.00
BLTSYN BALTIMORE  MD    536.00
```

Figure 4-3. Reading a Word Addressable File

INPUT FILES

```
SYRSYN SYRACUSE   NY   4320 50  }
DAYFAC DAYTON     OH   4130.00 }      TAPE1
WARSYN WARWICK    RI    700.95 }
MEPSBC MEMPHIS    TN   4800.00 }

RIHSBC RICHMOND   VA   7830.45 }      TAPE2
```

SOURCE LISTING

```
C     THIS PROGRAM ADDS FIVE RECORDS TO EXISTING FILE
C     WAFILE.  THE FIRST INPUT FILE (TAPE1) CONTAINS FOUR
C     RECORDS FOR A SEQUENTIAL WRITE.  THE SECOND INPUT
C     FILE (TAPE2) CONTAINS ONE RECORD FOR A RANDOM WRITE.
C
      PROGRAM ADDRECS
      DIMENSION WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +            'RT', 'F', 'FL', 30,
     +            'EFC', 3)
C
C     READ AND PRINT THE RECORDS IN TAPE1 AND WRITE THEM
C     SEQUENTIALLY TO WAFILE BEGINNING AT WORD ADDRESS 34.
C
      CALL OPENM (WAFIT, 'I-O')
      CALL STOREF (WAFIT, 'WA', 34)
      PRINT 5
    5 FORMAT (' ',/,' SEQUENTIAL WRITE',/)
   10 READ(1, '(3A10)', END=20) WSA
      PRINT 15, WSA
   15 FORMAT (1X,3A10)
      CALL PUT (WAFIT, WSA)
      GO TO 10
C
C     READ AND PRINT THE RECORD IN TAPE2 AND WRITE IT
C     RANDOMLY TO WAFILE AT WORD ADDRESS 31.
C
   20 CALL STOREF (WAFIT, 'WA', 31)
      PRINT 25
   25 FORMAT (' ',/,' RANDOM WRITE',/)
   30 READ (2, '(3A10)', END=40) WSA
      PRINT 35, WSA
   35 FORMAT (1X,3A10)
      CALL PUT (WAFIT, WSA)
      GO TO 30
C
C     DO A SEQUENTIAL READ OF WAFILE BEGINNING AT WORD
C     ADDRESS 1 AND PRINT THE FILE.
C
   40 CALL STOREF (WAFIT, 'WA', 1)
      PRINT 45
   45 FORMAT (' ',/,' SEQUENTIAL READ',/)
   50 IWA=IFETCH(WAFIT, 'WA')
      IF (IWA .GT. 43) GO TO 60
      CALL GET (WAFIT, WSA)
      PRINT 55, WSA, IWA
   55 FORMAT (1X,3A10,1X,04)
      GO TO 50
   60 CALL CLOSEM (WAFIT)
      END
```

Figure 4-4. Adding Records to a Word Addressable File (Sheet 1 of 2)

OUTPUT

```
SEQUENTIAL WRITE

SYRSYN SYRACUSE    NY    4320 50
DAYFAC DAYTON      OH    4130.00
WARSYN WARWICK     RI     700.95
MEPSBC MEMPHIS     TN    4800.00

RANDOM WRITE

RIHSBC RICHMOND    VA    7830.45

SEQUENTIAL READ

ABROPS RAPID CITY  SD     421.00 0001
AKOSBC AKRON       OH   10468.00 0004
BLTSYN BALTIMORE   MD     536.00 0007
CHESRC CHEYENNE    WY    7000.00 0012
DASSBC DALLAS      TX   98760.00 0015
OKHMPI TULSA       OK    1000.00 0020
ORLSYN ORLANDO     FL    4921.00 0023
PRTSBC PORTLAND    OR    9274.00 0026
PTBCPO PITTSBURGH  PA     793.00 0031
RDUSYN RALEIGH     NC     762.00 0034
RIHSBC RICHMOND    VA    7830.45 0037
SYRSYN SYRACUSE    NY    4320 50 0042
DAYFAC DAYTON      OH    4130.00 0045
WARSYN WARWICK     RI     700.95 0050
MEPSBC MEMPHIS     TN    4800.00 0053


1       - FILE DUMP -       TDUMP,I=WAFILE.

F 1 R   1 W     0- 0102 2217 2023 5522 0120   1104 5503 1124 3155 2304   5555 5555 3735 3457 3333   0113 1723 0203 5501 1322
                     A B  R O  P S    R  A P    I D    C  I T  Y    S D          4 2  1 .  0 0    A K  O S  B C    A  K R
F 1 R   1 W     4- 1716 5555 5555 5555 1710   5555 3433 3741 4357 3333   0214 2423 3116 5502 0114   2411 1517 2205 5555 1504
                     0 N                 0 H        1 0  4 6  8 .  0 0    B L  T S  Y N    B  A L    T I  M O  R E       M D
F 1 R   1 W     10- 5555 5555 4036 4157 3333  0310 0523 2203 5503 1005   3105 1616 0555 5555 2731   5555 5542 3333 3357 3333
                          5 3  6 .  0 0    C H  E S  R C    C  H E    Y E  N N  E         W Y          7  0 0  0 .  0 0
F 1 R   1 W     14- 0401 2323 0203 5504 0114  1401 2355 5555 5555 2430   5555 4443 4241 3357 3333   1713 1015 2011 5524 2514
                     D A  S S  B C    D  A L    L A  S              T X        9 8  7 6  0 .  0 0    O K  H M  P I    T  U L
F 1 R   1 W     20- 2301 5555 5555 5555 1713   5555 5534 3333 3357 3333   1722 1423 3116 5517 2214   0116 0417 5555 5555 0614
                     S A                 0 K        1  0 0  0 .  0 0    O R  L S  Y N    O  R L    A N  D O             F L
F 1 R   1 W     24- 5555 5537 4435 3457 3333  2022 2423 0203 5520 1722   2414 0116 0455 5555 1722   5555 5544 3542 3757 3333
                          4  9 2  1 .  0 0    P R  T S  B C    P  O R    T L  A N  D         O R        9  2 7  4 .  0 0
F 1 R   1 W     30- 2024 0203 2017 5520 1124  2423 0225 2207 1055 2001   5555 5555 4244 3657 3333   2204 2523 3116 5522 0114
                     P T  B C  P O    P  I T    T S  B U  R G  H    P A          7 9  3 .  0 0    R D  U S  Y N    R  A L
F 1 R   1 W     34- 0511 0710 5555 5555 1603   5555 5555 4241 3557 3333   2211 1023 0203 5522 1103   1015 1716 0455 5555 2601
                     E I  G H            N C             7  6 2 .  0 0    R I  H S  B C    R  I C    H M  O N  D         V A
F 1 R   1 W     40- 5555 5542 4336 3357 3740  2331 2223 3116 5523 3122   0103 2523 0555 5555 1631   5555 5537 3635 3355 4033
                          7  8 3  0 .  4 5    S Y  R S  Y N    S  Y R    A C  U S  E         N Y          4  3 2  0    5 0
F 1 R   1 W     44- 0401 3106 0103 5504 0131  2417 1655 5555 5555 1710   5555 5537 3436 3357 3333   2701 2223 3116 5527 0122
                     D A  Y F  A C    D  A Y    T O  N               O H        4  1 3  0 .  0 0    W A  R S  Y N    W  A R
F 1 R   1 W     50- 2711 0313 5555 5555 2211   5555 5555 4233 3357 4440   1505 2023 0203 5515 0515   2010 1123 5555 5555 2416
                     W I  C K            R I             7  0 0 .  9 5    M E  P S  B C    M  E M    P H  I S             T N
F 1 R   1 W     54- 5555 5537 4333 3357 3333  0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000
                          4  8 0  0 .  0 0
F 1 R   1 W     60- 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000

-- ABOVE LINE REPEATED --

F 1 R   1 W     74- 0000 0000 0000 0000 0000  0000 0000 0000 0000 0000   0000 0000 0000 0000 0000   0000 0000 0000 0000 0000

-- END OF INFORMATION --

-- END OF DUMP --
```

Figure 4-4. Adding Records to a Word Addressable File (Sheet 2 of 2)

REPLACING A RECORD

You can update a word addressable file by replacing a record stored at a particular word address. The following restrictions apply:

- The file must be attached and open for input/output (PD field set to IO).
- The replacement record must have the same record length as the record being replaced unless space was left to add to a record or space was left by a shorter record. If the replacement record is longer than the original record, part of the next record will be overwritten.

A record stored in a particular word address is replaced by an overwrite that is accomplished by executing a CALL PUT statement for the same address. The WA field specifies the address for the overwrite.

Program REPLACE, shown in figure 4-5, replaces the record stored at word address 31_8 or 25. The program performs the following:

- Dimensions the FIT and working storage area.
- Defines the file characteristics through the CALL FILEWA statement.
- Opens the file for input/output.
- Reads the input file into the working storage area (WSA) and prints the single record.
- Issues a write request for the 30-character record in WSA to word address 31_8 or 25.
- Stores a 1 in the WA field and begins a sequential read.
- Tests IWA for a value greater than 53_8 or 43. You calculate this value known to be the address of the last valid record.
- Reads and prints the file.

The printout that follows the source listing shows the new input record ALTOONA has replaced PITTSBURGH at word address 31_8.

REWINDING THE FILE

You can rewind a word addressable file by executing a CALL REWND statement. The format is:

```
CALL REWND(fit)
```

Since word addressable files can be accessed randomly according to the WA field setting, a rewind is simply an alternative to storing a 1 in the WA field.

The only parameter in the rewind request is the name of the array containing the FIT. If a rewind request is issued after a write request, file processing is completed before rewind; information in the central memory buffer is written to the device.

When the following statement is executed, the file described by WAFIT is rewound to beginning-of-information:

```
CALL REWND(WAFIT)
```

INPUT FILE

```
AOOSYN ALTOONA     PA  00000.00
```

SOURCE LISTING

```
C     THIS PROGRAM REPLACES THE RECORD AT WORD ADDRESS
C     31 IN WAFILE WITH A RECORD EXISTING ON FILE INPUT.
C
C
      PROGRAM REPLACE
      DIMENSION WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +             'RT', 'F', 'FL', 30,
     +             'EFC', 3)
C
C     READ AND PRINT REPLACEMENT RECORD
C
      CALL OPENM (WAFIT, 'I-O')
      PRINT 5
    5 FORMAT (' ',/,' THE REPLACEMENT RECORD',/)
   10 READ (*, '(3A10)', END=20) WSA
      PRINT 15, WSA
   15 FORMAT (1X,3A10)
C
C     WRITE REPLACEMENT RECORD IN WORD ADDRESS 31
C
      CALL PUT (WAFIT, WSA, 30, 25)
      GO TO 10
C
C     READ AND PRINT CONTENTS OF WAFILE
C
   20 CALL STOREF (WAFIT, 'WA', 1)
      PRINT 25
   25 FORMAT (' ',/,' THE UPDATED FILE',/)
   30 IWA=IFETCH (WAFIT, 'WA')
      IF (IWA .GT. 43) GO TO 40
      CALL GET (WAFIT, WSA)
      PRINT 35, WSA, IWA
   35 FORMAT (1X,3A10,1X,04)
      GO TO 30
   40 CALL CLOSEM (WAFIT)
      END
```

OUTPUT

```
THE REPLACEMENT RECORD

AOOSYN ALTOONA     PA  00000.00

THE UPDATED FILE

ABROPS RAPID CITY SD    421.00 0001
AKOSBC AKRON      OH  10468.00 0004
BLTSYN BALTIMORE  MD    536.00 0007
CHESRC CHEYENNE   WY   7000.00 0012
DASSBC DALLAS     TX  98760.00 0015
OKHMPI TULSA      OK   1000.00 0020
ORLSYN ORLANDO    FL   4921.00 0023
PRTSBC PORTLAND   OR   9274.00 0026
AOOSYN ALTOONA    PA  00000.00 0031
RDUSYN RALEIGH    NC    762.00 0034
RIHSBC RICHMOND   VA   7830.45 0037
SYRSYN SYRACUSE   NY   4320 50 0042
DAYFAC DAYTON     OH   4130.00 0045
WARSYN WARWICK    RI    700.95 0050
MEPSBC MEMPHIS    TN   4800.00 0053
```

Figure 4-5. Replacing a Record in a Word Addressable File

BAM performs various checks to ensure proper file processing and maintains information in a number of FIT fields related to error processing. You can set several of these FIT fields; you can interrogate others that BAM sets.

Error messages can be directed to the dayfile, error file, or both. The error file is processed through the CRMEP control statement. A variety of options are available for error file printing and disposition.

The contents of FIT fields can be captured at various points during processing and recorded on the error file. This capability is available through the CALL FITDMP statement, which can appear anywhere within the source program.

FIT FIELDS UNDER USER CONTROL

FIT fields that you can set fall into three general categories:

Error message control	DFC, EFC, ERL
Error options for parity errors	EO
Owncode exit processing	DX, EX

These fields are summarized in table 5-1 and described in the following paragraphs.

TABLE 5-1. ERROR PROCESSING FIT FIELDS UNDER USER CONTROL

FIT Field	Definition	Set By	Values	IFETCH Return
DFC	Dayfile control	FILExx STOREF FILE control statement	0 fatal messages only (default) 1 error messages 2 notes 3 error messages and notes	N/A
EFC	Error file control	FILExx STOREF FILE control statement	0 no entries (default) 1 error messages 2 notes 3 error messages and notes	N/A
ERL	Trivial error limit	FILExx STOREF FILE control statement	0 no limit (default) n error limit where n=1-511	N/A
EO	Error option for parity error processing	FILExx STOREF FILE control statement	T terminate file access (default) D drop bad data A accept bad data TD terminate and display block with error on error file† DD drop bad data and display block with error on error file† AD accept bad data and display block with error on error file†	0 1 2 4 5 6
DX	End-of-data exit	FILExx STOREF GET (SQ only) GETP (SQ only)	0 no routine (default) name routine name	N/A
EX	Error exit	FILExx STOREF GET (WA only) PUT PUTP (SQ only) REPLC (SQ only)	0 no routine (default) name routine name	N/A

†EFC field must be set to 3

DAYFILE CONTROL, DFC

The DFC field controls the listing of error messages on the dayfile. The field can be set by the CALL FILExx, CALL STOREF, or FILE control statement.

Fatal error messages are always written to the dayfile. The messages written to the dayfile depend on the setting of the DFC field as follows:

- 0 Fatal messages only (default)
- 1 All error messages to the dayfile
- 2 Notes to the dayfile
- 3 Error messages and notes to the dayfile

ERROR FILE CONTROL, EFC

The EFC field controls the listing of error messages on the error file. The field can be set by the CALL FILExx, CALL STOREF, or FILE control statement.

The error file is a special file created with the logical file name ZZZZZEG. The messages written to the error file depend on the setting of the EFC field as follows:

- 0 No messages to the error file (default)
- 1 Error messages to the error file
- 2 Notes to the error file
- 3 Error messages and notes to the error file

TRIVIAL ERROR LIMIT, ERL

Trivial error conditions can interfere with a particular operation but do not deny further file access. You should not ignore trivial errors on the assumption they are unimportant. Trivial errors might reveal that the correct file is not being processed. Trivial errors might also reveal that an entire job completed normally, but the file was not created because the file was never opened.

The ERL field places a limit on the number of trivial errors allowed. When the limit is reached, a fatal error occurs. The field can be set by the CALL FILExx, CALL STOREF, or FILE control statement. The default is no limit on trivial error messages.

ERROR OPTION FOR PARITY ERRORS, EO

The EO field provides several alternatives when a parity error occurs. The field can be set by the CALL FILExx, CALL STOREF, or FILE control statement.

The EO field can be set as follows:

- T Terminate file processing with a fatal error (default).
- D Drop bad data, seek the next good record, and pass control to error exit with no useful information in WSA.
- A Accept bad data and pass control to error exit at end of bad data record.

When the EFC field is set to 3, the EO field can be set as follows:

- TD Terminate file processing and display block with error on error file.
- DD Drop bad data, display block with error on error file, and position to next good record.
- AD Accept bad data and display block with error on error file. Pass control to error exit at end of bad data record.

END-OF-DATA EXIT, DX

The DX field enables owncode exit processing when various end-of-data conditions arise while reading a file. The field can be set by the CALL FILExx or CALL STOREF statement. For sequential files only, the field can also be set by the CALL GET and CALL GETP statements.

Action taken for an end-of-data condition depends on the setting of the DX field as follows:

- 0 No end-of-data exit; processing continues (default).
- name Subroutine identified by variable name is to be executed.

The subroutine must be declared EXTERNAL.

On return from an end-of-data exit subroutine, processing resumes at the line after the BAM call that resulted in the DX condition. Table 5-2 lists the conditions that cause end-of-data exit processing. No end-of-data exit is taken for sequential files if the suppress buffer flag (SBF) field in the FIT is set to YES.

When an end-of-data exit subroutine receives control, the contents of the working storage area are undefined. A CALL GET statement that causes an end-of-data exit does not transfer information to the working storage area.

Within the user subroutine, processing allowed at end-of-information depends on the processing direction specified when the file was opened.

- If PD was INPUT, the following calls can be issued:

  CLOSEM
  REWND
  SKIP

- If PD was IO, the following calls can be issued:

  CLOSEM
  ENDFILE
  PUT
  PUTP
  REWND
  SKIP

You must acknowledge physical boundary conditions existing because of device restrictions, as well as boundaries written by specific request.

TABLE 5-2. CONDITIONS CAUSING END-OF-DATA EXIT

Condition	FP Field Setting	Comments
Beginning-of-information	0001	For standard labeled tapes, beginning-of-information is start of user data records.
	0002	For unlabeled or nonstandard labeled tapes, beginning-of-information is load point.
End-of-section	0010	End-of-section exists as a result of WEOR execution. It is equivalent to a level 0 7/8/9 card.
End-of-partition	0040	End-of-partition exists as a result of ENDFILE or WTMK execution. It is equivalent to a tapemark or a level 17_8 7/8/9 card.
End-of-information	0100 (file close)	End-of-information exists as a result of CLOSEM FILE execution. On mass storage, end-of-information is equivalent to a 6/7/8/9 card.

Program DXEXAMP, shown in figure 5-1, uses an end-of-data exit. The main program performs the following:

- Identifies end-of-data exit subroutine DEXIT.
- Allocates an array for FIT construction (SQFIT) and working storage area (WSA) in blank common so that they can be accessed in a subroutine as well as in the main program.
- Defines the file characteristics through the CALL FILESQ statement. The DX field is set to the name of the end-of-data exit subroutine.
- Opens the file for input.
- Reads the file.

Each time an end-of-data condition is recognized, subroutine DEXIT is called to print the contents of the FP field in the FIT.

The printout that follows the source listing indicates end-of-data was encountered three times: end-of-section twice, end-of-information once. Figure 3-13 in section 3 includes a file dump of SQFILE, which was the actual file read by program DXEXAMP. An examination of that file dump indicates the following steps occurred:

1. Records 1 through 10 were read.
2. End-of-record was encountered and control was given to end-of-data subroutine DEXIT. DEXIT returned FP=10_8 (end-of-section). Note that end-of-section is BAM terminology and corresponds to an operating system end-of-record.
3. Records 11 and 12 were read.
4. End-of-record was encountered and DEXIT again returned FP=10_8.
5. End-of-information was encountered and DEXIT returned FP=100_8.

The loop was executed 15 times and control was then passed to the CALL CLOSEM statement. Each time the CALL GET statement encountered an end-of-data condition, control transferred to DEXIT. The RETURN from DEXIT returned control as if the CALL GET statement had just executed; however, the working storage area did not contain a new record. Therefore, the CALL GET statement had to be executed 15 times to read the 12 records.

ERROR EXIT, EX

The EX field enables owncode exit processing when a fatal or trivial error occurs during file processing. The field can be set by the CALL FILExx or CALL STOREF statement. The field can also be set by the following processing calls:

- GET (WA only)
- PUT
- PUTP (SQ only)
- REPLC (SQ only)

Action taken for an error exit depends on the setting of the EX field as follows:

0	No error exit; processing continues (default).
name	Subroutine identified by variable name is to be executed.

The subroutine must be declared EXTERNAL.

When an error exit subroutine receives control, the contents of the working storage area depend on the type of call and error. When a return is executed in the EX subroutine, processing resumes at the line after the BAM call that resulted in the EX condition.

For sequential files, no error exit is taken for a GET or PUT if the suppress buffer flag (SBF) field in the FIT is set to YES.

SOURCE LISTING

```
C     THIS PROGRAM ILLUSTRATES USE OF AN END-OF-DATA EXIT.
C     THE DX FIELD IDENTIFIES END-OF-DATA EXIT SUBROUTINE
C     DEXIT, WHICH IS CALLED TO PRINT THE CONTENTS
C     OF THE FP FIELD EACH TIME AND END-OF-DATA
C     CONDITION IS ENCOUNTERED ON PERMANENT FILE SQFILE.
C
C
      PROGRAM DXEXAMP
      EXTERNAL DEXIT
      COMMON SQFIT(35), WSA(3)
      CALL FILESQ (SQFIT, 'LFN', 'SQFILE',
     +            'BT', 'C', 'RT', 'F', 'FL', 30,
     +            'EFC', 3, 'DX', DEXIT)
      CALL OPENM (SQFIT)
      DO 10 J=1,15
   10 CALL GET (SQFIT,WSA)
      CALL CLOSEM (SQFIT)
      END


      SUBROUTINE DEXIT
      COMMON SQFIT(35), WSA(3)
      N=IFETCH (SQFIT, 'FP')
      PRINT 60,N
   60 FORMAT(' FIT FIELD FP = ',03)
      RETURN
      END
```

OUTPUT

```
FIT FIELD FP = 010
FIT FIELD FP = 010
FIT FIELD FP = 100
```

Figure 5-1. Using an End-of-Data Exit

Program EXEXAMP, shown in figure 5-2, uses an error exit. The main program performs the following:

- Identifies error exit subroutine ERROR.
- Allocates an array for FIT construction (WAFIT) and working storage area (WSA) in blank common so that they can be accessed in a subroutine as well as in the main program.
- Defines the file characteristics through the CALL FILEWA statement. The EX field is set to the name of the error exit subroutine.
- Opens the file for input.
- Issues a read request for the record stored at word address 300_8 or 192.

If an error condition is recognized, subroutine ERROR is to print the contents of the error status (ES) field in the FIT.

Word addressable file WAFILE, which was created in section 4, is known to have 15 records with the last record stored at word address 53_8. The message that immediately follows the source listing indicates the invalid word address was recognized. The ES field contains error message code 120_8, which signifies an invalid word address.

Notice the error message that appears in the job's dayfile. The DFC parameter in the CALL FILEWA statement was set to 3 to indicate error messages and notes were to be transmitted to the dayfile.

FIT FIELDS UNDER SYSTEM CONTROL

BAM sets the following FIT fields that you can interrogate:

Trivial error count	ECT
Error status	ES
Fatal/nonfatal flag	FNF
Parity error flag	PEF
System parity error severity	SES

These fields are summarized in table 5-3 and described in the following paragraphs.

TRIVIAL ERROR COUNT, ECT

The ECT field holds the trivial error count. When the trivial error limit (ERL) field is set to a value greater than zero, the ECT field is incremented by BAM whenever a trivial error occurs. As long as the value of the ECT field is less than the value of the ERL field, the trivial error causes control to pass to the error exit, if specified, or to the in-line code. When the error count is the same as the trivial error limit (ECT=ERL), a fatal error occurs.

SOURCE LISTING

```
C     THIS PROGRAM IDENTIFIES AN ERROR EXIT SUBROUTINE
C     BY SETTING THE EX FIELD.  A READ REQUEST IS
C     ISSUED FOR THE RECORD STORED AT WORD ADDRESS
C     192 IN WAFILE.  THIS INVALID WORD ADDRESS CAUSES AN ERROR
C     CONDITION, AND SUBROUTINE ERROR IS CALLED TO
C     PRINT THE CONTENTS OF THE ES FIELD IN THE FIT.
C
C
      PROGRAM EXEXAMP
      EXTERNAL ERROR
      COMMON WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +            'RT', 'F', 'DFC', 3, 'FL', 30,
     +            'EX', ERROR)
      CALL OPENM (WAFIT)
      CALL GET (WAFIT, WSA, 192)
      CALL CLOSEM (WAFIT)
      END


      SUBROUTINE ERROR
      COMMON WAFIT (35), WSA (3)
      N=IFETCH(WAFIT, 'ES')
      PRINT 10,N
   10 FORMAT (' ERROR CODE IS ', O3)
      RETURN
      END
```

OUTPUT

```
ERROR CODE IS 120
```

NOS DAYFILE

```
14.58.01.USER
14.58.02.CHARGE
14.58.04.FTN5,LO=-A.
14.58.04.      54400 SCM STORAGE USED.
14.58.04.      0.017 CP SECONDS COMPILATION TIME.
14.58.04.ATTACH,WAFILE.
14.58.04.LGO.
14.58.06. RM ERROR 0120 ON LFN WAFILE
14.58.06.      END  EXEXAMP
14.58.06.      17500 MAXIMUM EXECUTION FL.
14.58.06.       .009 CP SECONDS EXECUTION TIME.
```

Figure 5-2. Using an Error Exit

ERROR STATUS, ES

The ES field holds a three-digit octal error code. When a fatal or trivial error occurs, BAM sets the ES field to the appropriate code. Refer to Program EXEXAMP in figure 5-2.

FATAL/NONFATAL FLAG, FNF

The FNF field holds a value that indicates an error is fatal or nonfatal (trivial). BAM sets the ES field to 1 to indicate a fatal error, and to 0 to indicate a nonfatal error. When FNF is set to 1, the file cannot be processed further; if an attempt is made to process the file, the job is aborted. The file can still be closed, however, after most fatal errors. Refer to the discussion of the SES field for information on error severity.

PARITY ERROR FLAG, PEF

The PEF field holds a value that indicates whether or not a parity error has occurred. BAM sets the PEF field to 1 to indicate a parity error, and to 0 to indicate no parity error. When PEF is set to 1, further action is dependent upon the value of the SES field.

TABLE 5-3. ERROR PROCESSING FIT FIELDS UNDER SYSTEM CONTROL

FIT Field	Definition	IFETCH Return Value
ECT	Trivial error count	0 through 511
ES	Error status	Error code (001 through nnn)
FNF	Fatal/nonfatal flag	0 nonfatal 1 fatal
PEF	Parity error flag	0 no error 1 parity error
SES	System parity error severity	1 read parity error level 1 2 read parity error level 2 3 read parity error level 3 4 read parity error level 4 5 write parity error level 1 6 write parity error level 2

SYSTEM PARITY ERROR SEVERITY, SES

The SES field holds a value that indicates the severity of a system parity error. When a system parity error occurs, BAM sets the SES field to a value ranging from 1 through 6. Values 1 through 4 indicate read parity errors; values 5 and 6 indicate write parity errors.

Severity is indicated by the returned value as follows:

1 BAM can recover to a record boundary. The number of bad records and blocks is known.

2 BAM can recover to a record boundary. The number of bad blocks is known, but not the number of lost records.

3 BAM can recover to a record boundary. The number of bad blocks and records is not known.

4 BAM cannot recover; the error is fatal.

5 BAM cannot recover; CALL CLOSEM VOLUME is recommended.

6 BAM cannot recover; the error is fatal.

PROCESSING THE ERROR FILE

The error file is a local mass storage file that disappears at job termination. To read the information stored on the error file, the post error processor must be called by the CRMEP control statement.

The error file buffer is always flushed when the job terminates abnormally. At the normal completion of a job step, however, the buffer is flushed only if all files are closed. Any messages in the buffer are lost if the buffer is not flushed.

Parameters in the CRMEP control statement specify the output file to be used and select the error file information to be listed on the output file. If no parameters are specified, all fatal and data manager error messages are listed on the system file OUTPUT. Data manager error messages are transmitted to the ES field by the CYBER Database Control System (CDCS) component of the DMS-170 data management system. These messages comprise the 600_8 category and can appear when the CDCS interface applies.

Parameters are specified in two ways: the mnemonic alone or the mnemonic followed by an equal sign and one or more options. Table 5-4 lists the various parameters for the CRMEP control statement and the possible settings for each parameter. Figure 5-3 shows some examples of the CRMEP control statement.

```
CRMEP(LO,SF=WAFILE)

    All messages for the file WAFILE are to be listed on
    the file OUTPUT.


CRMEP(LO=-D,ON,L=ERRFILE)

    All messages except data manager messages and those
    with error codes 142 and 143 are written to the
    output file ERRFILE.


CRMEP(LO=N,SN=1000)

    Only notes with the number 1000, which is the
    number for the FIT dump, are to be listed.
```

Figure 5-3. CRMEP Control Statement Examples

DUMPING THE FIT

The contents of the FIT can be dumped to the error file as a note by executing a CALL FITDMP statement. The format is:

```
CALL FITDMP(fit,id)
```

If the error file control (EFC) field is set to 0, it is forced to 2; if the field is set to 1, it is forced to 3.

The first parameter in the FIT dump request is the name of the array that contains the FIT. When more than one FIT is being dumped to the error file, a 10-character identifier can be associated with each FIT. The identifier is specified as the second parameter.

When the following statement is executed, FIT WAFIT is dumped to the error file as note 1000:

```
CALL FITDMP(WAFIT)
```

TABLE 5-4. CRMEP CONTROL STATEMENT PARAMETERS

Mnemonic	Omitted	Mnemonic Only	Mnemonic and Option
LO	Fatal and data manager error messages are listed.	All messages in the error file are listed.	Select (N) or omit (-N) notes. Select (F) or omit (-F) fatal messages. Select (D) or omit (-D) data manager messages. Select (T) or omit (-T) trivial messages.
SF	Select messages for all files.	Select messages for all files.	Select messages only for the specified files.
OF	Omit messages for no files.	Omit messages for no files.	Omit messages only for the specified files.
SN	Select all message numbers.	Select hardware and parity error messages.	Select only messages with the specified numbers.
ON	Omit no message numbers.	Omit only error numbers 142 and 143.	Omit messages with the specified numbers.
L	Output file is OUTPUT.	Output file is LIST.	Output file is the specified file.
RU	Error file remains at EOI after processing.	Error file is returned/unloaded after processing.	Not applicable.
PW	Page width for CRMEP output file is 72 characters for connected file, 132 for unconnected file.	Page width for CRMEP output file is 72 characters for connected file, 132 for unconnected file.	Page width can be between 40 and 160 characters.

When the following statement is executed, FIT WAFIT is dumped to the error file as note 1000 and identified by the name indicated in variable M:

```
CALL FITDMP(WAFIT,M)
```

Two FIT dumps for the same file in different parts of the program can be easily differentiated by using this convention.

Program EXEXAMP, shown again in figure 5-4, includes a CALL FITDMP statement to dump the FIT to the error file, and a CRMEP control statement to print the contents of the error file. The EFC field is omitted, defaults to 0, and is forced to 2.

The error file contains note 1000, which is the FIT dump. The contents of each FIT field reflect the value of the field at the time subroutine ERROR was entered. Applicable fields and their octal values are as follows:

BFS=101	Buffer length
DFC=3	Error messages and notes to the dayfile
EFC=2	Notes to the error file
EOIWA=101	Word address at EOI
ES=120	Error code 120 - invalid word address
EX=175	Address of error exit routine ERROR
FL=36	Fixed length records of 30 characters
FO=1	Word addressable file organization
FWB=12610	First word address of user buffer
LFN=WAFILE	Logical file name

LOP=1	Last operation OPENM (GET did not transfer data)	PD=1	Processing direction of input
LT=2	Unlabeled file	RT=1	F type records
MRL=36	Maximum record length of 30 characters	WA=300	Current word address set by GET (the read was not successful and WA was not incremented)
OC=1	File opened	WSA=11677	Working storage area address

SOURCE LISTING

```
C     THIS PROGRAM USES A CALL FITDMP STATEMENT TO
C     DUMP THE FIT TO THE ERROR FILE.  THE FIT IS
C     IDENTIFIED BY THE 10-CHARACTER NAME 'ERROR EXIT'
C     INCLUDED IN THE FITDMP STATEMENT.  A CRMEP CONTROL
C     STATEMENT PRINTS THE CONTENTS OF THE ERROR FILE.
C
C
      PROGRAM EXEXAMP
      EXTERNAL ERROR
      COMMON WAFIT(35), WSA(3)
      CALL FILEWA (WAFIT, 'LFN', 'WAFILE',
     +            'RT', 'F', 'DFC', 3, 'FL', 30,
     +            'EX', ERROR)
      CALL OPENM (WAFIT)
      CALL GET (WAFIT, WSA, 192)
      CALL CLOSEM (WAFIT)
      END

      SUBROUTINE ERROR
      COMMON WAFIT (35), WSA (3)
      CALL FITDMP (WAFIT, 'ERROR EXIT')
      N=IFETCH(WAFIT, 'ES')
      PRINT 10,N
   10 FORMAT (' ERROR CODE IS ', 03)
      RETURN
      END
```

OUTPUT

```
ERROR CODE IS 120
```

```
1        CRMEP(LO)
 RM NOTE  1000 ON LFN WAFILE   FIT DUMP ERROR EXIT  (FIT AT 011634)
                0 ASCII                    0 EO              00000000 LBL                0 PM
                1 BAL           0000000101 EOIWA                0 LCR         00000000 PNO
                0 BBH                  000 ERL     27010611140500 LFN               00 POS
                0 BCK                  120 ES              000300 LGX         00000000 PTL
                0 BFF               000175 EX                00 LL                0000 RB
           000101 BFS                    0 EXD               00 LNG         0000000000 RC
       0000000300 BN                     0 FF                01 LOP                0 RDR
                0 BT               00000036 FL                01 LOP5               0 REL
           000000 BZF          0000000000 FLM           00000000 LP                00 RKP
                0 B8F                    0 FNF                2 LT                0000 RKW
           000000 CDT                    1 FO                00 LVL         00000000 RL
                0 CF                   000 FP              000000 LX                00 RMK
               00 CL                     0 FPB         00000000 MBL                01 RT
                0 CM                    36 FTS     000000000000 MFN                 0 SB
                1 CMPLT             012610 FWB              000 MKL                 0 SBF
                0 CNF                    0 FWI         00000000 MNB                 1 SDS
         00000000 CP                     0 HB          00000000 MNR                00 SES
           000000 CPA            00000000 HL          00000036 MRL                 0 SOL
                0 C1             00000000 HMB               00 MUL                 0 SPR
               00 DC                000000 HRL                0 NDX         00000000 TL
           000000 DCA            00000000 IBL               00 NL                  00 TRC
           000000 DCT                  000 IP                0 NOFCP                0 ULP
                3 DFC                  120 IRS               1 OC                   0 VF
                0 DFLG              000000 KA                0 OF                  00 VNO
                0 DKI                  000 KL                0 ON          0000000300 WA
              000 DP                     0 KNE               0 ORG                 0 WPN
             0412 DVT                   00 KP                0 OVF         00011677 WSA
           000000 DX                  0000 KR               00 PC              000000 XBS
              000 ECT                    0 KT                1 PD    000000000000000 XN
                2 EFC              000000 LA                0 PEF
                0 EMK                  00 LAC           000000 PKA
```

Figure 5-4. Dumping the FIT (Sheet 1 of 2)

```
NOS DAYFILE

14.58.02.USER
14.58.02.CHARGE
14.58.02.FTN5,LO=-A.
14.58.03.      54400 SCM STORAGE USED.
14.58.03.      0.017 CP SECONDS COMPILATION TIME.
14.58.03.ATTACH,WAFILE.
14.58.03.LGO.
14.58.05. RM ERROR 0120 ON LFN WAFILE
14.58.05. RM NOTE  1000 ON LFN WAFILE
14.58.06.      END  EXEXAMP
14.58.06.      20200 MAXIMUM EXECUTION FL.
14.58.06.       .011 CP SECONDS EXECUTION TIME.
14.58.06.CRMEP(LO)
```

Figure 5-4. Dumping the FIT (Sheet 2 of 2)

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 is specified when the operating system is 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 throughout 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; 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	∨	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.

SUMMARY OF FORTRAN CALL STATEMENTS

This appendix includes the general formats of FORTRAN calls to BAM for sequential and word addressable 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.
- 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 or variable name (for example, owncode exit 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 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.

SEQUENTIAL FILE ORGANIZATION

CALL CLOSEM(fit,cf,type)

CALL ENDFILE(fit)

CALL FILESQ(fit,field,value,...,field,value)

CALL FITDMP(fit,id)

CALL GET(fit,wsa,0,0,0,rl,dx)

CALL GETP(fit,wsa,ptl,'SKIP',dx)

IFETCH(fit,field)

CALL IFETCH(fit,field,variable)

CALL OPENM(fit,pd,of)

CALL PUT(fit,wsa,rl,0,0,0,ex)

CALL PUTP(fit,wsa,ptl,rl,ex)

CALL REPLC(fit,wsa,0,0,0,0,ex)

CALL REWND(fit)

CALL SKIP(fit,count)

CALL STOREF(fit,field,value)

CALL WEOR(fit,lvl)

CALL WTMK(fit)

WORD ADDRESSABLE FILE ORGANIZATION

CALL CLOSEM(fit,cf)

CALL FILEWA(fit,field,value,...,field,value)

CALL FITDMP(fit,id)

CALL GET(fit,wsa,wa,0,0,rl,ex)

IFETCH(fit,field)

CALL IFETCH(fit,field,variable)

CALL OPENM(fit,pd,of)

CALL PUT(fit,wsa,rl,wa,0,0,ex)

CALL REWND(fit)

CALL STOREF(fit,field,value)

GLOSSARY C

Advanced Access Methods (AAM) -
A file manager that processes indexed sequential, direct access, and actual key file organizations and supports the Multiple-Index Processor.

Basic Access Methods (BAM) -
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. System information, such as tape labels of sequential files, can appear before the beginning-of-information.

Block -
A logical or physical grouping of records to make more efficient use of hardware. Word addressable files are not blocked. When file organization is sequential, one of the following block types must be specified by the programmer: C, I, K, or E.

Boundary -
A file boundary is a physical indication that marks a logical division within a sequential file. BOI and individual user records are always recognized; other boundaries are affected by the record and blocking type and the file storage device. A word boundary is the first character position in a central memory word.

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.

Close -
A set of terminating operations performed on a file when input and output operations are complete. All files processed by BAM must be closed.

Creation Run -
All processing of a file, from open to close, the first time the file is written or made into a BAM file.

CRMEP Control Statement -
A control statement that processes the BAM error file.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM.

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.

End-of-Information (EOI) -
The end of the last user record in a file. Trailer labels are considered to be past the end-of-information. End-of-information is undefined for unlabeled S or L tapes.

Error File -
A special file created with the logical file name ZZZZZEG to hold BAM error messages; the file is processed by the CRMEP control statement.

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 or tapes allocated for a job and on which user 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. 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 BAM. For direct processing through BAM, a user must initiate establishment of this table. All file processing executes on the basis of information in this table. You 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 you.

I Tape (Internal) -
A magnetic tape with recording format of physical records containing the contents of 0 to 512 central memory words of binary information. I tapes are only supported under the NOS operating system.

Key -
Information used to identify a record.

Level Number -
An octal number from 0 through 17_8 that is recorded in a short physical record unit or zero-length physical record unit marker; the number is used to form system-logical-record groups within files. Level number 17_8 indicates a logical end-of-partition. Level number 16_8 is used by checkpoint/restart and should not otherwise be specified by you. The system creates system-logical-records with a level number of 0 for mass storage files and SI tapes when you do not specify otherwise.

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.

L Tape (Long Stranger) -
A 7-track or 9-track, labeled or unlabeled magnetic tape with blocks containing more than 5120 characters. Normally written by or for other than CYBER 170-compatible systems.

Macro -
A single instruction which when assembled into machine code generates several machine code instructions.

Mass Storage -
A disk pack that can be accessed randomly. Extended memory is not considered mass storage.

Open -
A set of preparatory operations performed on a file before input and output can take place; required for all BAM files.

Owncode -
A routine you can write to process certain conditions. Control passes automatically to user owncode routines defined in the FIT for:

DX	End-of-data condition
EX	Error condition
LX	Tape label processing

Partition -
A division internal to a sequential file. A group of sections beginning with the first record after the end of the preceding partition and ending with a special record or condition, dependent on the block and record type and storage device. Generally, a partition is greater than a section and less than a file, but it can be equal to either or both.

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 -
On magnetic tape, information between interrecord gaps. It need not contain a fixed amount of data.

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, and SI and I tapes; the concept does not apply to S/L tapes.

PRU Device -
An SI or I format tape or 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. Applies only to mass storage files with an organization other than sequential. Contrast with Sequential Access.

Record -
The largest collection of information passed between BAM and a user program in a single read or write operation. You define 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.

Rewind -
An operation that positions a file at beginning-of-information.

SCOPE 2 -
An operating system on the CONTROL DATA CYBER 70 Model 76 and 7600 Computer Systems; 7000 Record Manager runs under SCOPE 2.

Section -
A division internal to a sequential file. Recognition of a section boundary is affected by block type, record type, and file residence. A section is a group of records beginning with the first record after the end of the preceding section and ending with a special record or condition, dependent on the block and record type and storage device. Generally, a section is greater than a record and less than a partition, but it can be equal to either or both. Sections are not defined on K and E type blocks.

Sequential Access -
Access method by which only the record located at the current file position can be accessed. Contrast with Random Access.

Sequential (SQ) File -
A file with records in the physical order in which they were written. No logical order exists other than the relative physical record position.

S Tape (Stranger) -
A magnetic tape with recording format of physical records containing the contents of 512 or fewer central memory words of information.

SI Tape (System Internal) -
A magnetic tape with recording format of physical record units containing the contents of 0 to 512 central memory words of binary information or 0 to 128 words of coded information. Coded SI tapes are not supported under the NOS operating system.

Volume -
A reel of magnetic tape. A given file can encompass more than one volume.

Word Address -
The relative location of the first word of a record in a word addressable file. Specified as the WA field of the file information table on a call for a read or write operation.

Word Addressable (WA) File -
A mass storage file containing continuous data or space for data. Words within word addressable files are numbered from 1 to n, each word containing 10 characters. Retrieving or writing of data at any given word within the file is specified by the word number, called the word address.

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.

SUMMARY OF FIT FIELDS

D

This appendix summarizes the FIT fields that are applicable to the file organizations supported by BAM. Sequential file FIT fields are listed in table D-1. Word addressable file FIT fields are listed in table D-2.

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO SEQUENTIAL FILES

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILESQ Call	STOREF Call	IFETCH Function
ASCII	ASCII character set	0,1,2	0	Yes	Used for terminal files on NOS/BE only	X	X	X	X
BBH	Buffer below highest high address	YES, NO	NO	Yes		X	X	X	X
BFS	Buffer size in words	1 thru 131071	Provided by BAM	Yes	Must be greater than MBL and PRU size	X	X	X	X
BN	Block number								X
BT	Block type	I,C,K,E	I	No		X	X	X	X
CF	Close flag	R, N, U, RET, DET, DIS	R		Set by CLOSEM	X	X	X	X
CL	Trailer count field length	1 thru 6	0	No	RT=T only	X	X	X	X
CM	Conversion mode	YES, NO	NO	No	Must be NO for RT=W	X	X	X	X
CNF	Connect file flag	YES, NO	NO	No	Used for terminal files	X	X	X	X
CP	Trailer count field start	0 thru 1310710	0	No	RT=T only	X	X	X	X
C1	Binary length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
DFC	Dayfile control	0,1,2,3	0	Yes		X	X	X	X
DX	End-of-data exit	Routine name	0	Yes			X	X	X
ECT	Trivial error count								X
EFC	Error file control	0,1,2,3	0	Yes		X	X	X	X
EO	Error option for parity errors	T, D, A, TD, DD, AD	T	Yes	IFETCH return: 0=T 4=TD 1=D 5=DD 2=A 6=AD	X	X	X	X
ERL	Trivial error limit	0 thru 511	0	Yes	0 allows indefinite number of errors	X	X	X	X
ES	Error status				IFETCH return: a 3-digit octal error code				X
EX	Error exit	Routine name	0	Yes			X	X	X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILESQ Call	STOREF Call	IFETCH Function
FF	File flush	YES, NO	NO	Yes	Buffer flush on abnormal termination	X	X	X	X
FL	Record length	1 thru 1310710	0	No	RT=F/Z only	X	X	X	X
FNF	Fatal/nonfatal flag				IFETCH return: 0 = nonfatal error 1 = fatal error				X
FO	File organization	SQ	SQ	No	IFETCH return: 0	X	X	X	X
FP	File position				IFETCH return (octal): 1 = end of labels 2 = BOI 4 = EOV 10 = EOS 20 = EOR 40 = EOP 100 = EOI				X
FWB	First word address of buffer	Buffer address	Provided by BAM	Yes			X	X	X
HL	Header length	1 thru 1310710	0	No	RT=T only	X	X	X	X
LA	User label area	Label area address	0	Yes			X	X	X
LBL	Label area length	1 thru 900	0	Yes		X	X	X	X
LCR	Label check/creation	CHK, CRT	CRT	Yes		X	X	X	
LFN	Logical file name	1 to 7 letters or digits	Required	Yes	Must start with a letter	X	X	X	X
LL	Length field length	1 thru 6	0	No	RT=D only	X	X	X	X
LOP	Last operation code				IFETCH returns (octal): 01 = OPENM 02 = CLOSEM 03 = GET/GETP 43 = PUT/PUTP 56 = REPLC 05 = SKIP 47 = WEOR 10 = REWND 63 = WTMK 74 = ENDFILE				X
LP	Length field start	0 thru MNR	0	No	RT=D only	X	X	X	X
LT	Label type	UL, S, NS, ANY	UL	No		X	X	X	X
LX	Label exit	Label routine address	0	Yes			X	X	X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILESQ Call	STOREF Call	IFETCH Function
MBL	Maximum block length	1 thru 1310710	5120; BFS in characters minus 20	No	Required for BT=K/E	X	X	X	X
MFN	Multifile name	1 to 6 letters or digits			Corresponds to the SI or M parameter on the NOS LABEL statement, and to the M parameter on the NOS/BE LABEL statement. Must start with a letter.	X			
MNB	Minimum block length	1 thru MBL	0	No	BT=K/E only	X	X	X	X
MNR	Minimum record length	0 thru MRL	0	No		X	X	X	X
MRL	Maximum record length	0 thru 1310710	0	Yes	RT=D/R/S/T/U/W only	X	X	X	X
MUL	Multiple of characters per block	0 thru 62	2	No	BT=K/E only; must be an even number	X	X	X	X
NOFCP	No FILE control statement processing	YES, NO	NO	Yes			X	X	X
OC	Open/close status				IFETCH return (binary): 00 = never opened 01 = opened 10 = closed				X
OF	Open flag	R,N,E	R	Yes	Set by OPENM	X	X	X	X
PC	Padding character	Any character	76	Yes	Specify in octal display code	X	X	X	X
PD	Processing direction	INPUT, OUTPUT, IO	INPUT	Yes	Set to I-O by OPENM; IFETCH return: 0 or 1 = INPUT 2 = OUTPUT 3 = IO	X	X	X	X
PEF	Parity error flag				IFETCH return: 0 = no error 1 = parity error				X
PNO	Multifile position				Corresponds to LABEL control statement	X			
PTL	Partial transfer length				Set by GETP or PUTP				X
RB	Records per block	1 thru 4095	1	No	BT=K only	X	X	X	X
RC	Record count				Set by GET and PUT				X
RL	Record length	1 thru MRL	0	Yes	Set by GET and PUT			X	X
RMK	Record mark character	Any character	62	No	RT=R only; must not be the same as padding character	X	X	X	X

TABLE D-1. SUMMARY OF FIT FIELDS APPLICABLE TO SEQUENTIAL FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILESQ Call	STOREF Call	IFETCH Function
RT	Record type	D,F,R,S, T,U,W,Z	W	No		X	X	X	X
SB	Sign overpunch length field	YES, NO	NO	No	RT=D/T only	X	X	X	X
SBF	Suppress buffer flag	YES, NO	NO	Yes	SBF=YES suppresses allocation of buffers and circular buffering.	X	X	X	X
SES	System parity error severity				IFETCH return: 1 = read level 1 2 = read level 2 3 = read level 3 4 = read level 4 5 = write level 1 6 = write level 2				X
SPR	Suppress read ahead	YES, NO	NO	Yes	Reset to NO at the end of processing	X	X	X	X
TL	Trailer length	1 thru 131071	0	No		X	X	X	X
ULP	User label processing	NO, V, F, VF, U, VU, FU, VFU	NO	Yes	IFETCH return (binary): 0 = NO 001 = V 010 = F 011 = VF 100 = U 101 = VU 110 = FU 111 = VFU	X	X	X	X
VF	Volume close flag	U,R,N	U			X	X	X	X
VNO	Volume number								X
WSA	Working storage area	Memory location	Required	Yes			X	X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO WORD ADDRESSABLE FILES

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILEWA Call	STOREF Call	IFETCH Function
BBH	Buffer below highest high address	YES, NO	NO	Yes		X	X	X	X
BFS	Buffer size in words	1 thru 131071	Provided by BAM	Yes		X	X	X	X
CF	Close flag	R, N, U, RET, DET, DIS	R		Set by CLOSEM	X	X	X	X
DFC	Dayfile control	0,1,2,3	0	Yes		X	X	X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO WORD ADDRESSABLE FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILEWA Call	STOREF Call	IFETCH Function
DX	End-of-data exit	Routine name	0	Yes			X	X	X
ECT	Trivial error count								X
EFC	Error file control	0,1,2,3	0	Yes		X	X	X	X
EO	Error option for parity errors	T, D, A, TD, DD, AD	T	Yes	IFETCH return: 0=T 4=TD 1=D 5=DD 2=A 6=AD	X	X	X	X
EOIWA	End-of-information word address								X
ERL	Trivial error limit	0 thru 511	0	Yes	0 allows indefinite number of errors	X	X	X	X
ES	Error status				IFETCH return: A 3-digit octal error code				X
EX	Error exit	Routine name	0	Yes			X	X	X
FF	File flush	YES, NO	NO	Yes	Buffer flush on abnormal termination	X	X	X	X
FL	Record length	1 thru 1310710	0	No	RT=F only	X	X	X	X
FNF	Fatal/nonfatal flag				IFETCH return: 0 = nonfatal error 1 = fatal error				X
FO	File organization	WA	Required	No	IFETCH return: 1	X	X	X	X
FP	File position				IFETCH return (octal): 1 = BOI 20 = EOR 100 = EOI				X
FWB	First word address of buffer	Buffer address	Provided by BAM	Yes			X	X	X
LFN	Logical file name	1 to 7 letters or digits	Required	Yes	Must start with a letter	X	X	X	X
LOP	Last operation code				IFETCH return (octal): 01 = OPENM 02 = CLOSEM 03 = GET 43 = PUT 10 = REWND				X
MRL	Maximum record length	0 thru 1310710	0	Yes	Read protection only	X	X	X	X
NOFCP	No FILE control statement processing	YES, NO	NO	Yes			X	X	X

TABLE D-2. SUMMARY OF FIT FIELDS APPLICABLE TO WORD ADDRESSABLE FILES (Contd)

FIT Field Mnemonic	Meaning	Allowable Values	Release Default If Any	Can Change After Creation	Notes	FILE Control Statement	FILEWA Call	STOREF Call	IFETCH Function
OC	Open/close status				IFETCH return (binary): 00 = never opened 01 = opened 10 = closed				X
OF	Open flag	R,N	R	Yes	Set by OPENM	X	X	X	X
PD	Processing direction	INPUT, OUTPUT, IO	INPUT	Yes	Set to I-O by OPENM; IFETCH return: 0 or 1 = INPUT 2 = OUTPUT 3 = IO	X	X	X	X
RL	Record length	1 thru MRL	0	Yes	Set by GET and PUT			X	X
RT	Record type	F,U,W	W	No		X	X	X	X
SBF	Suppress buffer flag	YES, NO	NO	Yes	SBF=YES suppresses allocation of buffers and circular buffering	X	X	X	X
WA	Word address	Memory location						X	X
WSA	Working storage area	Memory location	Required	Yes			X	X	X

As long as a file can be described adequately to CYBER Record Manager by file information table values, a COBOL, COMPASS, FORTRAN, ALGOL, PL/I, FORM, or Sort/Merge program running under NOS or NOS/BE can be written to read the following:

- Any file created by CYBER Record Manager.
- Files written by COBOL and COMPASS prior to availability of CYBER Record Manager.
- Files written by FORTRAN, except random mass storage files created through WRITMS.
- Any file written by other product set members with two exceptions: random files with a format such as that produced by the WRITOUT macro of CPC, and BASIC binary files.
- Most files produced on other machines.

You can also use CYBER Record Manager to write a file to be read by other product set members or other machines. When IBM System/360/370 files are involved, it is necessary to use the FORM utility or 8-bit subroutines to convert between the 6-bit codes of CDC and 8-bit codes. (See the FORM and 8-bit Subroutines reference manuals.)

The remainder of this appendix describes several types of file interchange:

- File format produced by other products.
- Suggestions for reading tape with unknown or partially known structure.

CDC FILE FORMATS

The CYBER Record Manager file and record types resulting from statements in various products are shown in the following paragraphs.

FORTRAN 5.1

A formatted list directed NAMELIST WRITE produces RT=Z, BT=C.

An unformatted (binary) WRITE produces RT=W, BT=I.

BUFFER IN and BUFFER OUT produce RT=S, except that files named INPUT, OUTPUT, or PUNCH produce RT=Z. CM is set to YES for a formatted statement or to NO for an unformatted statement.

I/O USING CPC MACROS

RFILE or RFILEC created files have no counterpart in CYBER Record Manager.

Name/number index files (WRITOUT calling IORW) cannot be processed by CYBER Record Manager.

WRITOUT calling IOWRITE produces RT=Z, BT=C.

Other macros have no direct counterpart.

COBOL 5.3

For sequential files on any device other than S/L tapes, block type is always C. For S/L tapes, block type depends on the BLOCK CONTAINS clause, as shown in table E-1.

TABLE E-1. BLOCK TYPES FOR S/L TAPES

BLOCK CONTAINS Clause	Block Type
Omitted	K
BLOCK CONTAINS integer RECORDS	K
BLOCK CONTAINS integer TO integer RECORDS	E
BLOCK CONTAINS integer CHARACTERS	E
BLOCK CONTAINS integer TO integer CHARACTERS	E

For sequential, indexed sequential, direct access, and actual key files, record type depends on the RECORD clause, as shown in table E-2.

For word addressable files, RT is always set to U. For relative files, RT is always set to F.

FORM 1.1

Any file type except word addressable can be created by FORM. CYBER Record Manager FIT field default values are:

FO=SQ
RT=Z
BT=C

ALGOL 5.1

CYBER Record Manager FIT field default values for each file type are as follows:

- Coded sequential files produce:

  FO=SQ, RT=Z, BT=C, FL=137 (paged),
  FL=80 (unpaged)

- Binary sequential files produce:

  FO=SQ, RT=S, BT=C, MRL=5120

- Word addressable files produce:

  FO=WA, RT=U, MRL=5120

TABLE E-2. RECORD TYPES RESULTING FROM COBOL RECORDS CLAUSE

RECORD Clause† (FD entry)	Record Description Entry			
	01 Entries of same length	01 Entries of different lengths	Entry with OCCURS/ DEPENDING ON data-name in record	Entry with OCCURS/ DEPENDING ON data-name not in record
Clause omitted	F	W	T	W
RECORD CONTAINS integer CHARACTERS	F††	F	F	F
RECORD CONTAINS integer-1 to integer-2 CHARACTERS	W	W	T	W
RECORD CONTAINS integer-1 to integer-2 CHARACTERS DEPENDING ON data-name in record	D	D	D	D
RECORD CONTAINS integer-1 to integer-2 CHARACTERS DEPENDING ON data-name outside record	W	W	W	W

†For each RECORD CONTAINS format, an equivalent RECORD VARYING format exists, giving the same respective record type.

††Record type is Z if file name is INPUT, OUTPUT, or PUNCH.

READING AN UNKNOWN STRUCTURE

A tape with an unknown structure often can be read through the following outlined procedure:

- CYBER Record Manager returns information to the FIT when a read is executed. Use this information to process the file.
- Select a file description, depending on the anticipated size of the physical block. If the tape is an SI tape or S format tape, maximum block size is 5120 characters.
- To read a tape and transfer all data to the working storage area, describe the tape as RT=S. The first GET sets the RL field of the FIT to one of the following:

  Number of characters in the tape block read if the tape is an S or L tape

  Size of system-logical-record if the tape is an SI or I format tape
- To read a specific number of characters from a tape when the blocks are known to be larger, describe the tape as RT=U, BT=K, RB=1. For the first GET, set RL to the desired number of characters.

SEQUENTIAL FILE BOUNDARY PROCESSING

Program BOUNDRY, shown in figure F-1, illustrates file boundary processing concepts. The program creates a sequential file and writes section and partition boundaries on the file. Refer to section 3 for information on logical file structure and a description of the statements used to write file boundaries.

Input records for BOUNDRY exist initially as three sets of 80-character records on three files. AFILE contains data A through E, FFILE contains data F through J, and KFILE contains data K through M. The files are copied to file TAPE1 with the following operating system copy utilities:

```
COPYBF,AFILE,TAPE1.
COPYBF,FFILE,TAPE1.
COPYBF,KFILE,TAPE1.
```

After these statements are executed, the three sets of data records exist on TAPE1 and are separated by operating system end-of-file markers. You must rewind TAPE1 to beginning-of-information before executing BOUNDRY.

The program creates sequential file NEWONE from the records in TAPE1 and writes file boundaries between the sets of data records. It then reopens NEWONE and reads the records. When a file boundary is read, control transfers to end-of-data subroutine OWNEND.

Statements in the main program that apply to file boundary processing are described as follows:

- EXTERNAL OWNEND

  This statement identifies end-of-data subroutine OWNEND, which is specified by subsequently setting the DX field in the FIT.

- COMMON SQFIT(35), WSA(8)

  This statement allocates an array for FIT construction (SQFIT) and working storage area (WSA) in blank common so they can be accessed in a subprogram, as well as in the main program.

- READ (1, '(8A10)', END=15) WSA

  This statement reads the first set of data records from TAPE1 into the working storage area and then transfers control to a statement that writes a file boundary to NEWONE.

- CALL WEOR (SQFIT)

  This statement writes a section boundary to NEWONE after the first set of data is read.

- CLOSE(1)
  OPEN(1)
  READ (1, '(8A10)', END=25) WSA

  These statements read the second set of data records from TAPE1 into the working storage area and then transfer control to a statement that writes a file boundary. The previous READ encountered an end-of-file on TAPE1; therefore, a CLOSE and an OPEN statement must precede this READ so that the records following the end-of-file can be read.

- CALL ENDFILE (SQFIT)

  This statement writes an end-of-partition to NEWONE after the second set of data is read.

- CLOSE(1)
  OPEN(1)
  READ (1, '(8A10)', END=35) WSA

  These statements read the third set of data records from TAPE1 and transfer control to a statement that closes file NEWONE.

- CALL CLOSEM (SQFIT)

  This statement writes the buffer to file NEWONE and writes an end-of-information.

- CALL OPENM (SQFIT, 'INPUT', 'R')

  This statement opens file NEWONE for reading and rewinds it to the beginning of the first record. The warning message that follows this statement tells you the previous OPENM call specified fewer parameters.

- CALL STOREF (SQFIT, 'DX', OWNEND)

  This statement sets the DX field of the FIT so that subroutine OWNEND is executed each time a file boundary is read.

The end-of-data subroutine OWNEND has access to the FIT and the data record. The subroutine tests the file position (FP) field of the FIT to determine the type of boundary and then prints an acknowledgment.

Statements that apply to the subroutine are defined as follows:

- M=IFETCH (SQFIT, 'FP')

  This statement puts the value of the file position field into the variable M.

- IF (M .EQ. 8) GO TO 10
  IF (M .EQ. 32) GO TO 20
  IF (M. EQ. 64) GO TO 30

  These statements test for 10_8 or 8 (end-of-section), 40_8 or 32 (end-of-partition), and 100_8 or 64 (end-of-information).

Output from the program follows the source listing. All messages were generated by the program to monitor execution.

The last record of a section (E, J, M) and the last record of a partition (J) are duplicated on OUTPUT. This duplication occurs because a GET call that detects a file boundary transfers control to OWNEND but does not transfer information to the working storage area. The RETURN from OWNEND returns control as if the GET call has just executed, but the working storage area does not contain a new record. Therefore, the PRINT statement executed immediately after return from OWNEND reprints the contents of the working storage area, which have not changed. You can add more statements to the program to prevent these records from being processed twice. Refer to the example on adding records to a sequential file in section 3.

INPUT FILE

```
                  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
                  BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
         AFILE    CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
                  DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
                  EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
                  FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
TAPE1             GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG
         FFILE    HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
                  IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII
                  JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
                  KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK
         KFILE    LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL
                  MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
```

SOURCE LISTING

```
C     THIS PROGRAM ILLUSTRATES FILE BOUNDARY PROCESSING CONCEPTS.
C     IT CREATES A SEQUENTIAL FILE AND WRITES SECTION AND PARTITION
C     BOUNDARIES ON THE FILE.
C
      PROGRAM BOUNDRY
      IMPLICIT INTEGER (A-Z)
      EXTERNAL OWNEND
      COMMON SQFIT(35), WSA(8)
      CALL FILESQ (SQFIT, 'LFN', 'NEWONE',
     +             'WSA', WSA,
     +             'RT', 'F', 'BT', 'C', 'FL', 80)
      CALL OPENM (SQFIT, 'I-O')
C
C     READ FIRST SET OF RECORDS AND WRITE EOS.
C
   10 READ (1, '(8A10)', END=15) WSA
      CALL PUT (SQFIT)
      GO TO 10
   15 CALL WEOR (SQFIT)
      PRINT *, ' WEOR WRITTEN FOR END-OF-SECTION'
```

Figure F-1. Processing File Boundaries (Sheet 1 of 3)

```
C
C     READ SECOND SET OF RECORDS AND WRITE EOP.
C
      CLOSE(1)
      OPEN(1)
   20 READ (1, '(8A10)', END=25) WSA
      CALL PUT (SQFIT)
      GO TO 20
   25 CALL ENDFILE (SQFIT)
      PRINT *, ' ENDFILE WRITTEN FOR END-OF-PARTITION'

C
C     READ THIRD SET OF RECORDS AND EXECUTE CLOSE FOR EOI.
C
      CLOSE(1)
      OPEN(1)
   30 READ (1, '(8A10)', END=35) WSA
      CALL PUT (SQFIT)
      GO TO 30
   35 CALL CLOSEM (SQFIT)
      PRINT *, ' CLOSEM EXECUTED FOR END-OF-INFORMATION'
C
C     OPEN THE FILE, SET UP END-OF-DATA EXIT (OWNEND), AND READ THE FILE.
C     WHEN EOS, EOP, EOI ARE ENCOUNTERED, OWNEND IS TO PRINT ACKNOWLEDGMENT.
C
      CALL OPENM (SQFIT, 'INPUT', 'R')
 WARNING* NUMBER OF ARGUMENTS IN REFERENCE TO _OPENM IS NOT CONSISTENT
      CALL STOREF (SQFIT, 'DX', OWNEND)
   40 PRINT *, ' '
      CALL GET (SQFIT)
      IF (IFETCH (SQFIT, 'FP') .EQ. 64) GO TO 45
      PRINT 901, WSA
      GO TO 40
   45 CALL CLOSEM (SQFIT)
      STOP
  901 FORMAT (8A10)
      END

        1  WARNING   ERROR IN  BOUNDRY


      SUBROUTINE OWNEND
      IMPLICIT INTEGER (A-Z)
      COMMON SQFIT(35), WSA(8)
      M=IFETCH (SQFIT, 'FP')
      IF (M .EQ. 8) GO TO 10
      IF (M .EQ. 32) GO TO 20
      IF (M .EQ. 64) GO TO 30
      CALL ABORT
   10 PRINT *, ' EOS ENCOUNTERED '
      RETURN
   20 PRINT *, ' EOP ENCOUNTERED '
      RETURN
   30 PRINT *, ' EOI ENCOUNTERED '
      RETURN
      END
```

Figure F-1. Processing File Boundaries (Sheet 2 of 3)

```
OUTPUT

  WEOR WRITTEN FOR END-OF-SECTION
  ENDFILE WRITTEN FOR END-OF-PARTITION
  CLOSEM EXECUTED FOR END-OF-INFORMATION

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
  EOS ENCOUNTERED
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG

HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH

IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII

JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
  EOS ENCOUNTERED
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ

  EOP ENCOUNTERED
JJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ

KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK

LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM
  EOS ENCOUNTERED
MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

  EOI ENCOUNTERED
```

Figure F-1. Processing File Boundaries (Sheet 3 of 3)

INDEX

COMMENT SHEET

MANUAL TITLE: CYBER Record Manager Basic Access Methods
Version 1 User's Guide

PUBLICATION NO.: 60495800 REVISION: C

NAME:

COMPANY:

STREET ADDRESS:

CITY: STATE: ZIP CODE:

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

___ Please reply ___ No reply necessary

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

OLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Publications and Graphics Division

215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

OLD

FOLD

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

LITHO IN U.S.A.

CONTROL DATA