pdp11

RMS–11 MACRO–11 Reference Manual

Order No. AA-H683A-TC

digital

March 1979

This manual describes the use of RMS-11 facilities in MACRO-11 programs.

RMS-11 MACRO-11
Reference Manual

Order No. AA-H683A-TC

SOFTWARE VERSION: RMS-11 V1.8
RMS-11K V1.8

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, March 1979

The postage-paid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	FOCAL
DECnet	IAS
DECsystem-10	MASSBUS
DECSYSTEM-20	PDP
DECtape	RSX
DECUS	RSTS
DIBOL	UNIBUS
DIGITAL	VAX
	VMS

8/80-15

TABLE OF CONTENTS

TABLES

PREFACE

Record Management Services for the PDP-11 (RMS-11) provides powerful data management capabilities. The RMS-11 User's Guide tells you about those capabilities and how to use them. This manual is designed as a reference for MACRO-11 programmers.

RMS-11 is a set of software routines that transfers data between a running program, which uses data in a logical form called records, and the file processor portion of an operating system, which maintains the physical structure of the data on a storage device.

NOTE

You can use RMS-11 Indexed files only if you have purchased the RMS-11K software product. Your system manager can tell you if your system includes this capability.

PREREQUISITES

To read this manual, you should be a MACRO-11 programmer who has read and understood the RMS-11 User's Guide.

STRUCTURE OF THE MANUAL

Chapter 1 tells you how to use RMS-11 routines in a MACRO-11 program, including the minimum requirements and the order in which you must use them.

Chapter 2 covers the means of providing buffer space for RMS-11 operations.

Chapter 3 describes the fields of the File Access Block.

Chapter 4 describes the fields of the Record Access Block.

Chapter 5 describes the fields of the different types of Extended Attribute Blocks.

Chapter 6 describes the fields of the Name Block.

Chapter 7 describes the field access macros.

Chapter 8 describes the file and record operation macros.

Chapter 9 describes the use of Block I/O.

Appendixes provide supplementary information, such as:

- RMS-11 success and error completion codes
- Sample RMS-11 programs
- Routine to convert 64-bit date-time information to ASCII

ASSOCIATED DOCUMENTS

The RMS-11 documentation set contains the following manuals:

- RMS-11 User's Guide
- RMS-11 MACRO-11 Reference Manual
- RMS-11 Installation Guide

You must also use operating system documentation. See the Documentation Directory for your operating system.

DOCUMENTATION CONVENTIONS

RMS-11 operates similarly on the supporting operating systems (see Appendix A of the RMS-11 User's Guide). Therefore, it should be possible to produce a single manual describing that operation. However, the differences among operating systems present a barrier to that unification. The following conventions are designed to enable you to hurdle that fence.

DIFFERENCES IN OPERATING SYSTEM FUNCTIONS

Details that are common to the operating systems are printed in black, and the details that are specific to one operating system or another are printed in color as follows:

- RSTS/E-specific information is printed in red.
- IAS-, RSX-11M-, and VAX-specific information is printed in blue.

SYNTAX DESCRIPTIONS

In the descriptions of macro usage, this manual includes a general form for each macro, using the following conventions:

REQUIRED	You must include all uppercase characters as shown.
user-specific	You substitute for lowercase characters information specific to your usage.
[may be used]	You can, but do not have to include characters in brackets. The convention ...] means the series can continue until it exhausts logical possibilities.
Punctuation	You must use punctuation as shown.

The descriptions of user control block fields that contain numeric values include minimum and maximum specifications. These are logical values and normally, are not equal to the physical minimum and maximum for the field.

Additionally, a portion of a field, whether byte or word, described as "low order" or "least significant" is the portion with the lower address.

CHAPTER 1

USING RMS-11 IN A MACRO-11 PROGRAM

Record Management Services for PDP-11 operating systems (RMS-11) is a set of routines that enable programs to process files and records within files. The RMS-11 User's Guide describes the features of RMS-11. All capabilities supported by an operating system are available to a MACRO-11 program.

To obtain RMS-11 services at run time, your program must contain RMS-11 processing macros and user control blocks. The processing macros are expanded at assembly time. The resulting code is executed at run time to perform the specified operation. Each macro represents a program request for a file- or record-related service.

With every request for a service, information is exchanged between your program and the RMS-11 routines via user control blocks. These blocks are:

Block Name	Function
File Access Block (FAB)	Describes a file and contains file-related information.
Record Access Block (RAB)	Describes a Record Access Stream and the records being accessed by that stream.
Extended Attribute Block (XAB)	Contains file attribute information beyond that in the FAB.
Name Block (NAM)	Describes a location in memory containing system-related information about a file.

Prior to issuing a request for an RMS-11 service, your program must place information detailing the request in the associated control block.

Example A request to open a file must be accompanied by the name or ID of the file, information on how the file will be accessed, and details on how the file is to be shared.

Example A program request to read a record from a file must specify an access mode and if appropriate, a key value identifying the desired record.

After a request for service has been processed, RMS-11 uses the same control block to return information to your program. When a file has been successfully opened, RMS-11 provides attribute information such as the organization of the file and the format of the records in the file. After successfully retrieving a record from a file, RMS-11 provides your program with the location in memory of the record and the length of the record.

The amount of information exchanged between RMS-11 and your program varies with the nature of the request and the attributes of the file being processed.

To use RMS-11 routines in a MACRO-11 program, you must understand how to:

1. Declare the RMS-11 facilities that your program requires.

2. Allocate and initialize user control blocks designed to communicate with RMS-11.

3. Access fields in user control blocks at run time.

4. Perform file and record operations.

5. Assemble your program modules and task build them with the RMS-11 routines.

1.1 DECLARING RMS-11 FACILITIES

Every program that processes RMS-11 files must contain directives and special-purpose macros that declare the RMS-11 facilities required at assembly and run time. To declare RMS-11 facilities that are used by your program, you must do the following:

1. List RMS-11 macros in .MCALL directives.

2. Declare the processing environment.

3. Declare buffer pool requirements and techniques.

4. Issue a $INIT or $INITIF macro.

NOTE

Unless otherwise noted, RMS-11 macros use decimal radix for numeric values.

1.1.1 Listing Names of Required Macro Definitions

All macros used in your program must be listed as arguments in a .MCALL directive. Listing the macros in this way allows the actual code of each macro to be read in from the RMS-11 source macro library (RMSMAC.MLB) during assembly.

General form:

```
.MCALL arg[,arg]..]
```

where arg is a symbolic name of a macro required in the assembly of your program. The macro names can be listed in any order.

The number of .MCALL directives can be minimized because some macro definitions contain .MCALL directives. Table 1-1 contains the minimum .MCALL arguments that provide .MCALL directives for all RMS-11 macros that can be used in a program.

Table 1-1: Minimum Set of .MCALL Arguments

.MCALL Arguments	Embedded .MCALL Arguments
ORG$	None.
POOL$B	Central buffer pool declaration macros.
$INIT or $INITIF	None.
$GNCAL	Run-time field access macros (described in Chapter 7) and completion routine macros (described in this chapter).
FAB$B	File Access Block allocation and initialization macros (described in this chapter), field offset macros (described in Chapter 3), and error code macros (described in Appendix A).
RAB$B	Record Access Block allocation and initialization macros (described in this chapter), field offset macros (described in Chapter 4), and error code macros (described in Appendix A).
XAB$B	Extended Attribute Block allocation and initialization macros (described in this chapter), field offset macros (described in Chapter 5), and error code macros (described in Appendix A).
NAM$B	Name Block allocation and initialization (described in this chapter), field offset macros (described in Chapter 6), and error code macros (described in Appendix A).
$FBCAL	File operation macros (described in Chapter 9).
$RBCAL	Record operation macros (described in Chapter 9).

As shown in Table 1-1, you can ensure that all RMS-11 macros used in a program appear as arguments in .MCALL directives by coding the following sequence of .MCALL directives and macro execution in that program:

```
.MCALL  ORG$,POOL$B,$INIT
.MCALL  $GNCAL,FAB$B,RAB$B,XAB$B,NAM$B
.MCALL  $FBCAL,$RBCAL
$GNCAL
$FBCAL
$RBCAL
```

By issuing the $GNCAL, $FBCAL, and $RBCAL macros, you cause the embedded .MCALL directives to take effect.

You can omit any macro names from .MCALL directives that do not apply to a particular program.

Example If the program does not use Name or Extended Attribute Blocks, do not include the NAM$B or XAB$B macros.

You may also omit the $RBCAL or $FBCAL macros and list separately each file and record operation macros used by your program.

NOTE

If you are allocating control blocks in one module, but using field offset and/or error code macros in other modules, in those modules you must:

1. include one or more of the following arguments in .MCALL directives

2. execute the specified macros, in the form:

```
macnam RMS$L
```

except $RMSTAT, which requires no argument when executed

.MCALL Arguments	Embedded Macro Definitions
FABOF$	FAB field offsets
RABOF$	RAB field offsets
XABOF$	XAB field offsets
NAMOF$	NAM field offsets
$RMSTAT	Error codes

1.1.2 Declaring the Processing Environment

You must include one or more ORG$ macros within the set of modules that you link together with the Task Builder to produce an executable task. All ORG$ macros must be in modules that are part of the root of your task. An ORG$ macro for a particular file organization must be present even if no record operations are performed when such a file is opened.

The use of ORG$ macros in your source modules enables the Task Builder to select for linking only those portions of RMS-11 required by your program*. Each ORG$ macro declares a unique combination of file organization and record operations.

General form:

```
ORG$    org[,<recop[,recop...]>]
```

where org is the type of file organization as one of the following symbolic values:

```
IDX = Indexed file organization
REL = Relative file organization
SEQ = Sequential file organization
```

recop is a symbolic value identifying a type of record operation that will be performed on a file of the specified organization. If a single value is included, the angle brackets are not needed. If multiple values are specified, you must enclose them in angle brackets and use commas to separate each value from the preceding value.

One or more of the following symbolic values may be specified in any order:

```
CRE = file of specified organization may be created
DEL = delete operation
FIN = find operation
GET = get operation
PUT = put operation
UPD = update operation
```

Example The following code declares that one or more Sequential files will be created and put operation performed by the program. Additionally, one or more Indexed files will be opened and find, get, and update operations will be performed on those files. Finally, one or more Relative files will be opened, but no record operations will be performed: file operations may be performed.

```
ORG$    SEQ,<CRE,PUT>
ORG$    IDX,<GET,UPD,FIN>
ORG$    REL
```

---

*When you task build RMS-11 nonoverlaid. If you use an RMS-11 overlay structure, the ODL file specifies which RMS-11 modules are linked to your program. However, the ORG$ macro still determines which of those routines your program can use.

1.1.3 Declaring Buffer Pool Requirements

RMS-11 requires a collection of I/O buffers and internal control structures to support file processing at run time. The area in your program occupied by these buffers and control structures is known as the buffer pool.

The major portion of the buffer pool is composed of I/O buffers. To your program, record processing under RMS-11 appears as the movement of records directly between a file and the program itself. However, RMS-11 actually uses I/O buffers as intermediate storage during data transfers:

- When your program finds or gets a record, RMS-11 moves the block or bucket containing the record from the file to an I/O buffer. Then, on a get, RMS-11 moves the record from the buffer to your program.

- When your program puts, updates, or deletes a record, RMS-11 moves the specified data from your program to the I/O buffer. Then, for Relative and Indexed files, normally RMS-11 moves the bucket in the buffer out to the file immediately. However, for Sequential files and for Relative and Indexed files with Deferred Write specified, RMS-11 doesn't write out the buffer until it has to be used for another operation.

The size of I/O buffers depends on the organizations of the files being processed, the number of files open simultaneously, and the number of simultaneously active Record Access Streams. In providing the information needed to calculate the size requirements for the I/O buffers portion of the buffer pool, you have three choices:

1. A centralized buffer pool controlled by RMS-11.

2. Private I/O buffers for one or more files plus a centralized pool controlled by RMS-11 for other requirements.

3. A centralized buffer pool controlled by a routine you provide.

When RMS-11 controls a centralized buffer pool, RMS-11 allocates I/O buffers as well as the internal control structures required for file processing from a single area in your program. Normally, this space is inaccessible to your program: RMS-11 totally manages the space within the pool and allocates portions, as needed, for buffer space and control structures for open files.

You can also allocate private I/O buffers on a per-file basis by specifying the address and total size of each buffer in fields of the File Access Block associated with a file. When the file is open, this buffer space is completely managed by RMS-11 and your program must not access it. However, when the file is closed, the private I/O buffer space is available for use by your program.

The major advantage of private I/O buffers is avoidance of fragmentation in a centralized buffer pool. Since particular files have varying I/O buffer requirements based on their organization, a centralized buffer pool can reach the point where there is sufficient total space available for the opening of an additional file, but the space is not

contiguous. When such a situation arises, the specified file cannot be opened.

Whether you let RMS-11 control a centralized buffer pool or specify private I/O buffers, RMS-11 always requires certain internal control structures that must be allocated in the buffer pool to support file processing. The number of internal control structures required by RMS-11 in the buffer pool is based on the organizations of the files being processed, the maximum number of files open simultaneously, and the maximum number of simultaneously connected Record Access Streams. Once again, your program must provide, at assembly time, the information needed to determine the size requirements of the internal control structures that must be allocated in the centralized buffer pool.

The presence in your source modules of the macros listed in Table 1-2 allows RMS-11 to determine the size requirements for your program's buffer pool. The macros are described in Chapter 2. If you want private I/O buffers for one or more files, you can allocate these on a file-by-file basis, either statically (at assembly time) or dynamically (at run time). Refer to the descriptions of the BPA (buffer pool address) and BPS (buffer pool size) fields in Chapter 3.

Finally, you can assume total control over buffer space, allocating it when RMS-11 requires it, taking it back when RMS-11 no longer needs it. This facility, called Get Space Address (GSA), is complex and should not be used by most programmers. Chapter 2 also describes the requirements for the GSA routine.

Table 1-2: Space Pool Declaration Macros

Macro	Description	Required
POOL$B	Beginning of space pool declaration	Yes
P$BDB	Number of Buffer Descriptor Blocks	Yes
P$FAB	Number of files open simultaneously	Yes
P$RAB	Nonindexed streams connected simultaneously	If Sequential or Relative files
P$RABX	Indexed streams connected simultaneously	If Indexed files
P$IDX	Number of defined keys	If Indexed files
P$BUF	Input/output buffer requirements	If no BPA or GSA
POOL$E	End of space pool declaration	Yes

1.1.4 Initializing the Processing Environment

Your program must initialize the RMS-11 processing environment at run time before the program attempts an RMS-11 operation. You accomplish this with the $INIT or $INITIF macro.

General form:

```
$INIT     or     $INITIF
```

The code generated by the $INIT macro unconditionally initializes RMS-11 internal control structures at run time. The $INITIF code, however, initializes the internal structures only if they have not been initialized during the current execution of the task. You use the $INITIF macro in program modules that can be the first to use RMS-11, but are not always run first.

RMS-11 clears the Processor Status Word 0-Bit to indicate that initialization was successful. Therefore, normally after $INIT, the 0-Bit is cleared; if the 0-Bit is set, an RMS-11 file was open when the macro was initiated and no initialization occurred. Normally after $INITIF, the 0-Bit is set; if the 0-Bit is cleared, the processing environment was previously initialized.

If your program initiates any RMS-11 file or record operation before the environment is initialized, RMS-11 returns ER$INI error code.

1.2 USER CONTROL BLOCKS

User control blocks are formatted buffers in your program's address space. Each control block consists of data fields that are used to exchange information between your program and the RMS-11 routines.

You must allocate space for control blocks in your program at assembly time. You can also establish initial values for many of the data fields in these blocks at assembly time; or you can defer setting the contents of control block fields until run time. RMS-11 provides special macros that perform the functions needed to support control block allocation, assembly-time field initialization, and run-time field access.

1.2.1 File Access Block (FAB)

A File Access Block, abbreviated FAB, represents a file during the execution of file operation macros:

```
$CLOSE
$CREATE
$DISPLAY
$ERASE
$EXTEND
$OPEN
```

FAB fields (listed in Table 1-3) must contain the proper values before the macro is initiated, but they may be changed after the operation is complete because RMS-11 has transferred the pertinent information about the file to an internal (nonvisible) structure called an Internal File Access Block (IFAB).

Therefore, one FAB may be used to represent any number of files as long as the FAB is changed appropriately before each file operation is initiated. As a minimum you must ensure that the block is a valid FAB (BID and BLN fields contain the proper values) and that the FAB IFI field contains the value returned by RMS-11 when the file was created or opened. This value is the pointer to the IFAB. You should also set the FAB's fields to the values appropriate to the file operation you are preparing for: do not assume that the fields contain the values you set before the last use.

1.2.1.1 Allocation

- Each FAB must be allocated at assembly time. The minimum syntax is:

```
        .EVEN
label:  FAB$B
        FAB$E
```

where label is the name of the FAB and not necessarily the name of the file associated with the FAB.

The FAB$B macro allocates space for the FAB, and the FAB$E macro

stores values in the individual FAB fields and terminates the definition of the block.

Table 1-3: File Access Block Fields

Field Name	Field Size	Default	Description
ALQ	2W	0	Size of file
BID	1B	N/A	FAB identifier
*BKS	1B	1 record	Bucket size
BLN	1B	N/A	RAB length
*BLS	1W	0	Block size
BPA	1W	0	Location of private I/O buffer
BPS	1W	0	Size of private I/O buffer
CTX	1W	0	Available to user
*DEQ	1W	0	Automatic extension quantity
DEV	1B	N/A	Device characteristics
DNA	1W	0	Location of filespec defaults
DNS	1B	0	Size of filespec defaults
FAC	1B	FB$GET	Types of record operations
FNA	1W	0	Location of filespec
FNS	1B	0	Size of filespec
FOP	1W	0	File processing options
*FSZ	1B	0	Size of fixed area for VFC
IFI	1W	N/A	Pointer to IFAB
LCH	1B	0	Logical channel
*MRN	2W	0	Maximum Record Number
*MRS	1W	0	Maximum Record Size
NAM	1W	0	Pointer to NAM Block
*ORG	1B	FB$SEQ	File organization
*RAT	1B	0	Record attributes
*RFM	1B	FB$VAR	Record format
RTV	1B	0	Window size
*RTV	1B	0	Clustersize
SHR	1B	0	File sharing
STS	1W	N/A	Completion code
STV	1W	N/A	More error information
XAB	1W	0	Pointer to first XAB

*File attribute

1.2.1.2 Initialization - The value stored in a FAB field by the FAB$E macro is determined by an initialization macro, or in its absence, an RMS-11 default for the field (see "Initialization and Default" in the individual sections on the fields in Chapter 3).

If you want a field to contain a value other than its default value, you must specify its initialization macro between the FAB$B and FAB$E macros*. Initialization macros have the general form:

```
F$fnm arg
```

where fnm is the three-letter field name, such as ALQ, MRN, and so on, and

arg is one or more arguments specifying the value(s) to be entered in the indicated field; arg can be a:

- symbolic value, such as FB$PUT (representing put operations), in the form FB$nam; symbolic values are joined together with exclamation points (!).
- label, that is, the MACRO-11 term for the name of an address in the program, such as the start of a buffer.
- numeric value, such as maximum record size, specifying the number of bytes, characters, blocks, and so on.

Cautions:

- Because initialization macros operate at assembly time, you cannot use global symbols or labels as arguments. All symbols and labels must be defined locally, that is, in the same module with the macros.
- The default radix for numeric values in initialization macros is decimal.

1.2.2 Record Access Block (RAB)

A Record Access Block, abbreviated RAB, represents a Record Access Stream during the execution of record operation macros:

```
$CONNECT
$DELETE
$DISCONNECT
$FIND
$GET
$PUT
$UPDATE
$REWIND
$TRUNCATE
$FLUSH
$NXTVOL
```

The RAB not only completely describes the format of the records involved, but also all other aspects of record operations.

---

*The value of any field can also be set at run time by the field access macros $SET and $STORE; see Chapter 7.

RAB fields (listed in Table 1-4) must contain the proper values before the stream is set up (via the $CONNECT macro) and before record operations are initiated, but they can be changed after each operation is complete because RMS-11 has transferred the pertinent information about the stream to an internal (nonvisible) structure called an Internal Record Access Block (IRAB).

Therefore, one RAB may be used to represent any number of streams as long as the RAB is changed appropriately before each record operation is initiated. As a minimum you must ensure that the block is a valid RAB (BID and BLN fields contain the proper values) and that the RAB ISI field contains the value returned by RMS-11 when the stream was connected to a file. This value is the pointer to the IRAB. You should also set the RAB's fields to the values appropriate to the record operation you are preparing for: do not assume that the fields contain the values you set before the last use.

1.2.2.1 Allocation - Each RAB must be allocated at assembly time. The minimum syntax is:

```
        .EVEN
label:  RAB$B [type]
        RAB$E
```

where label is the name of the RAB, and

type indicates if the program will attempt asynchronous I/O operations using the RAB (see Chapter 1), with one of the following values:

SYN means synchronous record operations only

ASYN means both synchronous and asynchronous record operations

The RAB$B macro allocates space for the RAB, and the RAB$E macro stores values in the individual RAB fields and terminates the definition of the block.

1.2.2.2 Initialization - The value stored in a RAB field by the RAB$E macro is determined by an initialization macro, or in its absence, an RMS-11 default for the field (see "Initialization and Default" in the individual sections on the fields).

If you want a field to contain a value other than its default value, you must specify its initialization macro between the RAB$B and RAB$E macros*. Initialization macros have the general form:

---

*The value of any field can also be set at run time by the field access macros $SET and $STORE; see Chapter 7.

R$fnm arg

where fnm is the three-letter field name, such as KBF, RAC, and so on

arg is one or more arguments specifying the value(s) to be entered in the indicated field; arg can be a:

- symbolic value, such as RB$KEY, specifying a random record operation, in the form RB$nam; two or more symbolic values are joined together with exclamation points (!).
- label, that is, the MACRO-11 term for the name of an address in the program, such as the start of a buffer.
- numeric value, such as maximum record size, specifying the number of bytes, characters, blocks, and so on.

Cautions:

- Because initialization macros operate at assembly-time, you cannot use global symbols or labels as arguments. All symbols and labels must defined locally, that is, in the same module with the macros.
- The default radix for numeric values in initialization macros is decimal.

Table 1-4: Record Access Block Fields

Field Name	Field Size	Default	Description
BID	1B	N/A	RAB identifier
BKT	2W	None	Relative record number or VBN
BLN	1B	N/A	RAB length
CTX	1W	0	User area
FAB	1W	0	FAB address
ISI	1W	N/A	Pointer to IRAB
KBF	1W	0	Key buffer address
KRF	1B	0	Key of reference
KSZ	1B	0	Key buffer size
MBC	1B	0	Multiblock count
MBF	1B	0	Multibuffer count
RAC	1B	0	Record Access Mode
RBF	1W	0	Address of output record
RFA	3W	0	Record's File Address
RHB	1W	0	Fixed control area buffer
ROP	1W	0	Record processing options
RSZ	1W	0	Size of output record
STS	1W	N/A	Completion status code
STV	1W	N/A	Status value
UBF	1W	0	Input record buffer
USZ	1W	0	Input record buffer size

1.2.3 Extended Attribute Block (XAB)

An Extended Attribute Block, abbreviated XAB, is an extension of the File Access Block for an RMS-11 file. An XAB represents one of the following during file operations:

Area (Indexed only)
Key (Indexed only)
Date-time information
Protection information
Summary information (Indexed only)

XABs are generally required only when a file is created, particularly if the file is Indexed, or when the $DISPLAY macro is used to retrieve information about a file.

XAB fields (listed by XAB type in Chapter 5) must contain the proper values before the operation is initiated, but they may be changed after each operation is complete because RMS-11 has transferred the pertinent information about the file to the appropriate IFAB.

1.2.3.1 Allocation

Each XAB must be allocated at assembly time. The minimum syntax is:

```
        .EVEN
label:  XAB$B type
        XAB$E
```

where label is the name of the XAB, and

type indicates the type of information contained in the XAB and therefore its structure and fields, with one of the following values:

XB$ALL	the XAB defines a file area
XB$DAT	the XAB contains date-time information
XB$KEY	the XAB defines a key for an Indexed file
XB$PRO	the XAB specifies file protection information
XB$SUM	the XAB contains summary information about an Indexed file

A type must be specified, or the MACRO assembler will generate an error.

The XAB$B macro allocates space for the XAB, and the XAB$E macro stores values in the individual XAB fields, including the type symbolic value in the COD field, and terminates the definition of the block.

1.2.3.2 Initialization - The value stored in an XAB field by the XAB$E macro is determined by an initialization macro, or in its absence, an RMS-11 default for the field (see "Initialization and Default" in the individual sections on the fields).

If you want a field to contain a value other than its default value, you must specify its initialization macro between the XAB$B and XAB$E macros*. Initialization macros have the general form:

```
X$fnm arg
```

where fnm is the three-letter field name, such as IAN, RDT, and so on

arg is one or more arguments specifying the value(s) to be entered in the indicated field; arg can be a:

- symbolic value, such as XB$NUL, representing null key specification, in the form XB$nam; two or more symbolic values are joined together with exclamation points (!).
- label, that is, the MACRO-11 term for the name of an address in the program, such as the start of a buffer.
- numeric value, such as fill number, specifying the number of bytes, characters, blocks, and so on.

Cautions:

- Because initialization macros operate at assembly-time, you cannot use global symbols or labels as arguments. All symbols and labels must be defined locally, that is, in the same module with the macros.
- The default radix for numeric values in initialization macros is decimal.

1.2.3.3 Linking and Ordering XABs - Whenever you want to include Extended Attribute Blocks in a file operation, you must link them with the File Access Block and with each other. This linking is done with pointers, addresses stored in the FAB XAB field and then in each of the XAB NXT fields; the end of the chain is indicated by a zero NXT field.

1.2.3.3.1 Ordering by Type of XAB - Within a chain of XABs, RMS-11 does not require any ordering by type, that is, the contents of the COD field.

---

*The value of any field can also be set at run time by the field access macros $SET or $STORE; see Chapter 7.

Example To determine the attributes of a single-key Indexed file, allocate, at assembly time, one each:

Type	Label
Date XAB	DATXAB
Key XAB	KEYXAB
Protection XAB	PROXAB
Summary XAB	SUMXAB

You can link these blocks together in several ways; two of them are:

1. At assembly time, with initialization macros:

```
        .EVEN
DSPFAB: FAB$B
          .
        X$XAB   DATXAB  ;  POINT TO FIRST XAB
          .
        FAB$E
DATXAB: XAB$B   XB$DAT
          .
        X$NXT   KEYXAB  ;  POINT TO NEXT XAB
          .
        XAB$E
KEYXAB: XAB$B   XB$KEY
          .
        X$NXT   PROXAB  ;  POINT TO NEXT XAB
          .
        XAB$E
PROXAB: XAB$B   XB$PRO
          .
        X$NXT   SUMXAB  ;  POINT TO NEXT XAB
          .
        XAB$E
SUMXAB: XAB$B   XB$SUM
          .
          .
        XAB$E
```

2. At run time, with the $STORE macro:

```
MOV     #DSPFAB,R4        ;  PUT FAB ADDRESS IN R4, AS
                             REQUIRED
$STORE  #SUMXAB,XAB,R4    ;  MAKE DIFF XAB 1ST IN
                                  CHAIN
MOV     #SUMXAB,R4        ;  PUT XAB ADDRESS IN R4
$STORE  #KEYXAB,NXT,R4    ;  LINK IN NEXT XAB
MOV     #KEYXAB,R4        ;  PUT XAB ADDRESS IN R4
$STORE  #DATXAB,NXT,R4    ;  LINK IN NEXT XAB
MOV     #DATXAB,R4        ;  PUT XAB ADDRESS IN R4
$STORE  #PROXAB,NXT,R4    ;  LINK IN NEXT XAB
MOV     #PROXAB,R4        ;  PUT XAB ADDRESS IN R4
$STORE  #0,NXT,R4         ;  LINK IN NEXT XAB
```

Finally, you issue a $DISPLAY macro (see Chapter 8) and

RMS-11 fills the XAB chain with attribute information, setting fields in accordance with the COD field of each block.

1.2.3.3.2 Ordering Within XAB Type - Multiple instances of Key and Allocation XABs must be linked:

- In ascending order by contents of a numbering field, that is, the REF and AID fields respectively
- Logically contiguous, that is, there can be no XABs of other types within a series of Key or Allocation XABs
- Densely for the $CREATE operation, that is, the XABs must be numbered 1, 2, 3, and so on; by contrast, the numbering for $DISPLAY, $EXTEND, and $OPEN does not have to be dense; that is, you can select only certain keys or areas whose attributes you want set

1.2.4 Name Block

A Name Control Block, abbreviated NAM, contains system-specific information about a file, including:

Full File Specification
: An ASCII string representing RMS-11's merger of:
  - the primary file name string described by the FAB FNA and FNS fields
  - the default name string described by the FAB DNA and DNS fields
  - the system defaults

File ID
: An index that the file processor can use to locate a file without consulting directories; must be used with device ID.

Device ID
: An indicator for the device containing the file; must be used with file ID.

RMS-11 provides this information in NAM Block fields (listed in Table 1-5) during create and open operations and uses this information as input during erase and open operations.

You indicate the existence of a NAM Block for these services by setting the FAB NAM field to the address of a properly allocated NAM Block. NAM Block fields must contain the proper values before the operation is initiated, but they may be changed after the operation is complete.

1.2.4.1 Allocation - Each NAM Block must be allocated at assembly time. The minimum syntax is:

```
        .EVEN
label:  NAM$B
        NAM$E
```

where label is the name of the NAM Block.

The NAM$B macro allocates space for the NAM, and the NAM$E macro stores values in the individual NAM fields and terminates the definition of the block.

1.2.4.2 Initialization - The value stored in a NAM field by the NAM$E macro is determined by an initialization macro, or in its absence, an RMS-11 default for the field (see "Initialization and Default" in the individual sections on the fields).

If you want a field to contain a value other than its default value, you must specify its initialization macro between the NAM$B and NAM$E macros*. Initialization macros have the general form:

```
N$fnm arg
```

where fnm is the three-letter field name, such as ESA, ESL, and so on, and

arg is one or more arguments specifying the value(s) to be entered in the indicated field; arg can be a:

- label, that is, the MACRO-11 term for the name of an address in the program, such as the start of a buffer.
- numeric value, such as expanded string size, specifying the number of bytes.

Cautions:

- Because initialization macros operate at assembly time, you cannot use global symbols or labels as arguments. All symbols and labels must be defined locally, that is, in the same module with the macros.
- The default radix for numeric values in initialization macros is decimal.

---

*The value of any field can also be set at run time by the field access macros $SET and $STORE; see Chapter 7.

Table 1-5: NAM Block Fields

Field Name	Field Size	Default	Description
DVI	1W	0	Device ID
ESA	1W	0	Expanded string address
ESL	1B	0	Expanded string length
ESS	1B	0	Expanded string size
FID	1B	0	File ID

1.3 CONTROL BLOCK FIELD ACCESS AT RUN TIME

RMS-11 field access macro retrieve, modify, and test the contents of fields in the RMS-11 control blocks, FABs, RABs, and XABs, at run time. These macros enable you to treat the control block fields as logical entities, without regard for the placement of the fields within the control blocks and to a large degree, for the sizes of the fields.

Table 1-6 contains a summary of the available macros. Each macro is also described in a separate section of Chapter 7.

Table 1-6: RMS-11 Field Access Macros

Macro Name	Field Size	Function
$COMPARE	1 byte or 1 word	Compares the contents of a field with a value you specify.
$FETCH	Any size	Copies the contents of a field into a location you specify.
$OFF	1 byte or 1 word	Resets one or more bits within a bit string field.
$SET	1 byte or 1 word	Sets one or more bits within a bit string field.
$STORE	Any size	Copies the contents of a location you specify into a field.
$TESTBITS	1 byte or 1 word	Tests one or more bits within a bit string field.

NOTE

RMS-11 assumes octal radix for all numeric values used as operands for the field access macros. You indicate decimal radix with an explicit decimal point following a numeric value.

1.4 FILE AND RECORD OPERATIONS

You use the RMS-11 file and record operation macros to access and manipulate files and records within files. These macros combine with the control blocks (Chapters 3 through 6) to form your program's run-time interface with RMS-11.

Before executing one of these macros, your program sets values in a control block's fields, then specifies the block as an argument to the macro. The macro initiates all processing involved with the indicated RMS-11 operation. During the operation, RMS-11 returns information about the processing in fields of the associated control block.

If you do not initialize or set fields, relying on defaults, RMS-11 changes the fields to contain the default or minimum values as output from the operation.

See the RMS-11 User's Guide for a description of the operations themselves.

While differing in function, the file and record operation macros use the same general format and calling sequence. Within the calling sequence, you identify the control block associated with the operation and optional completion routines.

Before your program initiates a file or record operation macro, it must:

1. Ensure that the appropriate values are set in the fields used by RMS-11 during the operation. You do this with either initialization macros (see Chapters 3, 4, and 5) or field access macros (see Chapter 7).

2. Execute the correct calling sequence.

1.4.1 Completion Routines

You can write subroutines that RMS-11 executes as an extension of a file or record operation macro. These completion routines can be used after the successful completion or after error termination of an operation.

RMS-11 invokes a completion routine if you supply an address at the appropriate point in the calling sequence for the operation. At the end of the completion routine, RMS-11 restores the stack and other parameters and returns control to your program at the point after the macro was initiated.

The use of completion routines is always optional. However, if you do not use completion routines, your program should check the value of the associated block's STS field after the macro has been executed. If the status code is negative, an error occurred during the operation. If the status code is positive, the operation was successful, although an STS value greater than one indicates qualified success.

See Appendix A for a description of the completion codes. The STS field should never contain zero after an RMS-11 operation.

When using completion routines, you must be aware of conventions in the following areas:

Register Usage
: General register R5 contains the address of the same argument list or a copy of the argument list, that was part of the calling sequence to the RMS-11 operation (see Section 8.0.2). Therefore, you can use the control block address at 2(R5) to access fields in the control block.

RMS-11 Operations within Completion Routines
: A completion routine can execute file and record operation macros. Each operation is an extension of the original operation that caused the completion routine to be used. See Appendix A of the RMS-11 User's Guide for restrictions on this capability in the RMS-11 Asynchronous Environment.}

To return control from a completion routine to RMS-11, your program must do the following:

1. Restore the stack pointer (SP) to its value at the beginning of the completion routine. Your program must not attempt to cause control flow changes by modifying the stack.

2. Execute a $RETURN macro, in the form:

```
$RETURN
```

   This macro requires no arguments and denotes the end of a completion routine.

1.4.2 Calling Sequence

Each file and record operation macro requires a word-aligned formatted argument list. Your program can construct this list or allow RMS-11 to build it. Generally, your program generates less code and use less stack to build the list than does RMS-11.

1.4.2.1 Your Program Supplies the Argument List - If your program constructs the argument list, it executes RMS-11 file and record operations with the following sequence:

1. The program constructs the argument list in the following form, with the arguments arranged in the order shown:

Argument	Size	RMS-11 Interpretation
Undefined	1 byte	Not used.
Argument Count	1 byte	Binary value from 1 through 3 representing the number of arguments to be used from the argument list. This field equals 1 if you do not supply completion routine addresses.
Block Address	1 word	Address of a FAB for file operations or a RAB for record operations.
Error Address	1 word	Address of a completion routine you want called if the operation fails.
Success Address	1 word	Address of a completion routine you want called if the operation completes successfully. Not used by file operation macros.

2. Store the address of the argument list in general register R5.

3. Execute the file or record operation macro without arguments, in the form:

```
$macnam
```

4. If completion routines were specified, continue processing because success and/or failure has been tested and handled. Otherwise, check the STS field of the associated control block.

Example The following code constructs an argument list and uses it to execute a get operation:

```
        MOV     #LIST,R5        ;ADDRESS OF ARGUMENT LIST
        $GET                    ;READ A RECORD FROM THE FILE
         .
         .

LIST:   .WORD   3               ;NUMBER OF ARGUMENTS
        .WORD   INRAB           ;RECORD ACCESS BLOCK ADDRESS
        .WORD   ERR1            ;ERROR ROUTINE ADDRESS
        .WORD   SUCC1           ;SUCCESS ROUTINE ADDRESS
```

Example The following code constructs an argument list specifying a success completion routine, but no error completion routine:

```
LIST:   .WORD   3
        .WORD   INRAB
        .WORD   -1              ;NO ERROR ROUTINE
        .WORD   SUCC1
```

Example The following code constructs an argument list specifying error and success routines, but the program determines before the operation is initiated that neither routine applies:

```
        MOV     #1,LIST          ;SHRINK SIZE OF LIST
        MOV     #LIST,R5         ;ADDRESS OF ARGUMENT LIST
        $GET                     ;READ A RECORD FROM THE FILE
        $COMPARE #0,STS,2(R5)    ;LOOK AT STATUS CODE
        BGT     NXTSTP           ;SUCCESS
GETERR:  .
         .
         .

LIST:   .WORD   3        ;NUMBER OF ARGUMENTS
        .WORD   INRAB    ;RECORD ACCESS BLOCK ADDRESS
        .WORD   ERR1     ;ERROR ROUTINE ADDRESS
        .WORD   SUCC1    ;SUCCESS ROUTINE ADDRESS
```

1.4.2.2 RMS-11 Generates the Argument List - Your program can execute RMS-11 file and record operation macros with the form:

```
$macnam block[,error[,success]]
```

where block is the address of a FAB for file operations or a RAB for record operations.

error is the address of a completion routine you want called if the operation fails.

success is the address of a completion routine you want called if the operation completes successfully. Not used by file operation macros.

The macro builds an argument list on your program's stack from the information provided. Then it initiates the processing appropriate to the indicated operation. After the macro is executed, your program should check the STS field of the associated block unless completion routines were specified.

1.4.3 File Operation Macros

A file operation macro causes RMS-11 to perform an action related to an entire file. The macro name indicates the type of operation performed. The fields of the FAB associated with the macro in the calling sequence identifies the file and qualifies the operation.

Table 1-7 summarizes the RMS-11 file operation macros.

Table 1-7: RMS-11 File Operation Macros

Macro Name	Description
$CLOSE	Closes an open RMS-11 file so that your program can no longer access its contents.
$CREATE	RMS-11 creates and opens an RMS-11 file as described by the associated FAB and XABs, if any.
$DISPLAY	Stores attributes of an existing RMS-11 file in FAB and XAB fields.
$ERASE	Deletes an existing RMS-11 file and removes its entry(s) from a directory.
$EXTEND	Increases the number of blocks allocated to an RMS-11 file.
$OPEN	Opens an existing RMS-11 file, making its contents available for processing.

1.4.4 Record Operation Macros

After it has created or opened an RMS-11 file, your program can perform record operations on it. These operations involve the following concepts that are explained in the RMS-11 User's Guide and Chapter 1 of this manual.

Record Access Streams
File sharing
Context, Current Record, and Next Record
Synchronous and asynchronous record operations

Table 1-8 summarizes the RMS-11 record operation macros.

1.5 CREATING THE TASK

After you have written a MACRO-11 program as described in this chapter, you must assemble each module with the following reference in your command string:

```
LB:[1,1]RMSMAC.MLB/ML
```

After you have assembled all modules in your program successfully, you must link the modules with the RMS-11 routines using the Task Builder utility supplied with your operating system. You can link the RMS-11 routines in your task without overlays or with disk-resident or memory-resident overlays. See the RMS-11 User's Guide for a discussion of these options.

NOTE

Do not use the /SQ switch with the Task Builder. RMS-11 requires PSECTS to be in alphabetical order.

Table 1-8: RMS-11 Record Operation Macros

Macro Name	Description
$CONNECT	Establishes a Record Access Stream.
$DELETE	Deletes a record from an RMS-11 Relative or Indexed file.
$DISCONNECT	Terminates a Record Access Stream.
$FIND	Locates a record in an RMS-11 file.
$FLUSH	Moves all data in unwritten I/O buffers to disk.
$FREE	Unlocks a bucket locked by a Record Access Stream.
$GET	Moves a record from an RMS-11 file into your program's user buffer.
$NXTVOL	Continues processing with the next volume of magnetic tape multivolume set.
$PUT	Moves a record from your program's user buffer to an RMS-11 file.
$REWIND	Resets a Record Access Stream's context to the logical beginning of an RMS-11 file.
$TRUNCATE	Deletes record at the end of an RMS-11 Sequential file.
$UPDATE	Replaces a record in an RMS-11 file with a record from your program's user buffer.
$WAIT	Suspends processing until an RMS-11 asynchronous record operation completes.

CHAPTER 2

PROVIDING BUFFER SPACE

2.1 CENTRAL BUFFER POOL

The central buffer pool must be allocated at assembly time with the series of macros and arguments described in this section and Table 2-1. The macro series must start with the POOL$B macro and end with the POOL$E macro.

You can use multiple buffer pool allocations among the program modules that you link together with the Task Builder utility. The Task Builder sums the size requirements indicated by all buffer pool declarations.

Table 2-1: Buffer Pool Declaration Macros

Macro	Description	Required
POOL$B	Beginning of buffer pool declaration	Yes
P$BDB	Number of Buffer Descriptor Blocks	Yes
P$FAB	Number of files open simultaneously	Yes
P$RAB	Number of nonIndexed streams connected simultaneously	If Sequential or Relative files
P$RABX	Number of Indexed streams connected simultaneously	If Indexed files
P$IDX	Number of defined keys	If Indexed files
P$BUF	Input/output buffer requirements	If no BPA or GSA
POOL$E	End of buffer pool declaration	Yes

REQUIRED

```
************
*          *
*   P$BDB  *
*          *
************
```

2.1.1 P$BDB

The P$BDB macro ensures that the buffer pool contains sufficient space for internal RMS-11 control structures known as Buffer Descriptor Blocks (BDBs).

General form:

```
P$BDB nbrbdbs
```

where nbrbdbs is a numeric value or symbol representing the number of Buffer Descriptor Blocks required to support the file processing performed by your program.

To determine this value, use the following equation:

```
nbrbdbs = maxbuf + maxrel + (2 * maxidx)
```

where maxbuf is the count of the I/O buffers that can be used simultaneously, that is, the maximum number of I/O buffers ever in use for simultaneously open files.

You calculate this value by totaling the multibuffer counts in the MBF fields of RABs for all combinations of simultaneously connected Record Access Streams. The maximum value among all such combinations is the desired maxbuf value. RMS-11 must allocate one BDB for each I/O buffer being used at one time.

maxrel is the maximum number of Record Access Streams ever connected simultaneously for put operations to Relative files (whether or not an actual put operation is performed). RMS-11 allocates one BDB for each stream connected to a Relative file.

maxidx is the maximum number of Record Access Streams ever active simultaneously for put operations to Indexed files (whether or not an actual put operation is performed). RMS-11 allocated one BDB for each stream connected to an Indexed file.

REQUIRED

****
*   *
* P$FAB *
*   *
****

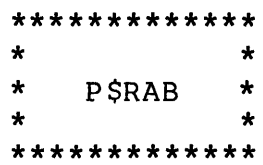
2.1.2 P$FAB

The P$FAB macro ensures that the buffer pool contains sufficient space for internal RMS-11 control structures related to File Access Blocks (FABs).

General form:

```
P$FAB number
```

where number is a numeric value or symbol representing the maximum number of files that are open simultaneously at run time.



2.1.3 P$RAB

The P$RAB macro ensures that the buffer pool contains sufficient space for internal RMS-11 control structures related to Record Access Blocks (RABs) for Sequential and Relative files. You can omit this pool macro if your program does not perform record operations on Sequential or Relative files.

General form:

```
R$RAB number
```

where number is a numeric value or symbol representing the maximum number of RABs that your program connects simultaneously to Sequential and Relative files.

```
************
*          *
*  P$RABX  *
*          *
************
```

2.1.4 P$RABX

The P$RABX macro ensures that the buffer pool contains sufficient space for internal RMS-11 control structures related to Record Access Blocks (RABs) for Indexed files. You can omit this pool macro if your program does not perform record operations on Indexed files.

General form:

```
P$RABX rabs,keysiz,nbrkeys
```

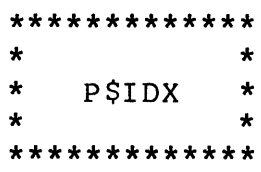
where rabs is a numeric value or symbol representing the maximum number of Record Access Streams that your program connects to Indexed files simultaneously.

keysiz is a numeric value or symbol representing the size, in bytes, of the largest key field that can be accessed by one of the streams.

nbrkeys is a numeric value or symbol representing the number of keys that can change values during an update operation on an Indexed file. You must specify this value whenever your program creates or opens an Indexed file with FB$UPD set in the FAB FAC field.

Example The following code indicates that there will be at most one stream connected to an Indexed file at any point during program execution. The largest key field in any such file is 32 bytes and no keys can change during update operations.

```
POOL$B          ;BEGIN POOL DECLARATION
  .
  .
P$RABX  1,32    ;ONE FILE AND BIGGEST KEY, NO CHANGES
  .
  .
POOL$E          ;END POOL DECLARATION
```



2.1.5 P$IDX

The P$IDX macro ensures that the buffer pool contains sufficient space for internal RMS-11 control structures containing summary information about an Indexed file's keys. You can omit this pool macro if your program does not open Indexed files.

General form:

```
P$IDX number
```

where number is a numeric value or symbol representing the total number of keys defined for all Indexed files open simultaneously. Include all keys even if they are never used for find or get operations.

```
************
*          *
*   P$BUF  *
*          *
************
```

2.1.6 P$BUF

The P$BUF macro ensures that the buffer pool contains sufficient space for the I/O buffers required by your program. You can omit this macro only if you are providing private buffers for all files or a GSA routine.

General form:

```
P$BUF iosiz
```

where iosiz is a numeric value or symbol representing the total bytes required for I/O buffers by your program. To calculate this number, use the following equation and round the result up to a multiple of four.

```
iosiz = strmsz1 + strmsz2 + ... + strmszn
```

where iosiz is the total I/O buffer requirement, and

strmsz1 strmszn are the I/O buffer space requirements (in bytes) for the Record Access Streams associated with the file that are active simultaneously.

You calculate the requirements for each stream as follows:

- For Sequential files on disk: STRMSZ = 512*MBC

- For Sequential files on magnetic tape: STRMSZ = BLS

  where BLS is the size, in characters, of each physical block of the magtape file, that is, the value contained in the BLS field of the FAB, and

  MBC is the value contained in the MBC field of the RAB associated with the stream.

- For Relative files: STRMSZ = BKS*512

  where BKS is the number of blocks in a bucket of the file (from the FAB BKS fields).

- For Indexed files: STRMSZ = BKS*MBF*512

  where BKS is the number of blocks in the largest bucket of the file (selected from the FAB BKS and the Allocation XAB BKZ fields), and

  MBF is the value contained in the MBF field of the RAB associated with the stream.

2.2 GET SPACE ROUTINE

A Get Space routine gives you complete control over the allocation of buffer space in your task. A single GSA routine serves all files used by the program.

RMS-11 uses the GSA routine when it requires or releases space. When RMS-11 requests space, it expects one of two outputs from your routine:

- the low-byte address of a contiguous block of bytes at least as big as that requested.
- an error indication that space is not available. RMS-11 issues an ER$DME message and returns control to the program.

When RMS-11 releases space, it expects no output from your routine; it assumes that the release of space was successful if the routine terminates successfully.

NOTES

- Your routine must start on a word boundary: precede the label defining the routine's starting address with a .EVEN directive.
- RMS-11 trusts your routine: it performs no parameter checking when the routine finishes and therefore can be lead into an error or even fatal situation if the routine did not allocate space properly.

2.2.1 Specifying a Routine

You can specify the starting address of your GSA routine at assembly time or at run time. Either way, your routine is used only when the internal address maintained by RMS-11 is not zero. Your program can also query RMS-11 at run time for the Get Space Address it is using.

2.2.1.1 Specifying a Get Space Address at Assembly Time -

General form:

```
GSA$    label
```

where label is the name of your GSA routine as specified in the label field of a source line and followed by a colon.

Example

```
GSARTN:   .
          .
          RTS     PC
;
          GSA$    GSARTN
```

2.2.1.2 Specifying a Get Space Address at Run Time -

General form:

```
$SETGSA argument
```

where argument is suitable as an operand for a MOV instruction. Where you use label with the $GSA macro, you use #label with the $SETGSA macro. You cannot use R6 as an argument; the assembler returns the error message:

```
.ERROR  ;  R6 MUST BE R0 - R5;
```

Example

```
GSARTN:   .
          .
          RTS     PC
;
          $SETGSA #GSARTN
```

Example

```
$SETGSA R5
```

Example

```
$SETGSA 10(SP)
```

Example

```
$SETGSA 10(R3)
```

2.2.1.3 Retrieving a Get Space Address at Run Time -

General form:

```
$GETGSA
```

RMS-11 sets general register R0 equal to the Get Space Address it has stored internally. If you have not executed a GSA$ or $SETGSA macro, this value is 0.

2.2.2 Interfaces to Routines

The following subsections describe the interfaces between RMS-11 and your GSA routine as well as the format of RMS-11 pool block headers.

RMS-11 sets a value in general register 2 (R2) to indicate whether:

- space should be allocated for an operation (R2 = 0)
- space is being returned after an operation (R2 <> 0)

The GSA routine is analogous to the $RQCB/$RLCB request and release core blocks routine as far as RMS-11 is concerned; the interfaces are identical. See also "Comments" below.

2.2.2.1 RMS-11 Request For Space - RMS-11 calls the routine with the following inputs:

R0 = address of RMS-11 Pool List Head

R1 = amount of space requested (in bytes)

R2 = 0

RMS-11 expects the following outputs from the user routine:

If C-bit = 0, all bytes requested have been allocated and
R0 = address of allocated block

If C-bit = 1, space was not available

General registers R3, R4, R5, and R6 and the stack must contain the same values when the user routine finishes as they did when it was called.

2.2.2.2 RMS-11 Release Of Space - RMS-11 calls the routine with the following inputs:

R0 = address of RMS-11 Pool List Head Block

R1 = size in bytes of block being released

R2 = address of block being released

No outputs are expected: the user routine must complete the release; however, general registers R3, R4, R5, and R6 and the stack must contain the same values when the routine finishes as they did when it was called.

2.2.2.3 RMS-11 Pool Block Header Formats - The RMS-11 buffer pool is actually subdivided into pools for each of the different sized internal structures that RMS-11 must maintain:

Buffer Descriptor Blocks
Key Descriptor Blocks
Internal FABs and RABs
I/O Buffers
Key Buffers

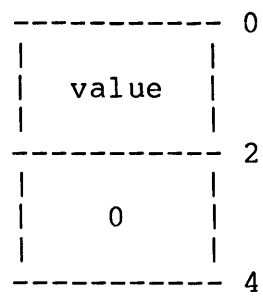
Each of these pools is described with a Pool List Head that points to the first free contiguous block of bytes within the pool.

Within all pools, space is divided into contiguous sections of bytes called blocks. Each block starts with a block header containing a pointer to the next free block in the pool and its own size in bytes.

Each pool is therefore accessed via the Pool List Head (which points to the first free block), then via a linked series of block headers

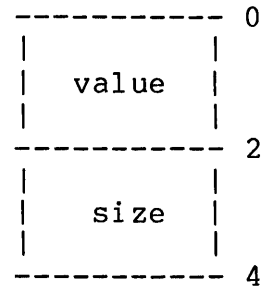
(which point to the next free block as well as tell how many bytes are in the block).

RMS-11 Pool List Head Block:



- if value = 0, there is no available space left in the pool described by this List Head
- if value <> 0, value points to the first available pool block header

RMS-11 Pool Block Header:



where value is either zero (0) or not:

- if value = 0, this is the last available block in this pool
- if value <> 0, value points to the next available pool block

size is the size in bytes of this pool block including the four-byte header

2.2.3 Comments

- You must write the GSA routine and include it in the program. The referenced routine is NOT part of RMS-11.
- Your routine does not have to use the RMS-11 Pool List Head Block, but is responsible for its upkeep if it does.

- If your routine does not allocate enough space for the impending operation, but signals RMS-11 that it has, the operation will fail in an unpredictable manner.

- Since your own routine could fail to allocate enough space and signal such failure to RMS-11, causing a dynamic memory error, your program should check for the ER$DME error code after every file and connect operation.

- Your routine could utilize the system routines $RQCB and $RLCB to accomplish the required functions, as follows:

```
GSARTN: TST     R2      ;  CHECK RMS-11 REQUEST
        BNE     2$
        JMP     $RQCB   ;  RMS-11 WANTS MORE SPACE
    2$: JMP     $RLCB   ;  RMS-11 WANTS TO RETURN SPACE
```

  The JMP instruction allows the system routine to return directly to RMS-11.

- Your routine, when requested to allocate space, could use $RQCB as a subroutine (via JSR) to check if enough space is currently available:

  -- if it is, $RQCB allocates it, returning with the C-bit = 0

  -- if it is not ($RQCB fails; C-bit = 1), your routine could:

     * extend the task
     * cannibalize other pools

     then allocate space out of the new room, set R0 as a pointer to the allocated buffer, and return to RMS-11.

- The space allocation and release procedures used by your routine must be symmetrical. For instance, the system routines $RQCB and $RLCB round all sizes up to a multiple of four bytes; these routines are symmetrical in that respect. If your routine employs one of these procedures, but not the other, the substitute must conform to this rounding standard.

CHAPTER 3

FILE ACCESS BLOCK

The initialization macros (this chapter) and the field access macros (Chapter 7) are provided so that you do not have to know the specific position and to a large extent, the size of each field in a FAB. You can, therefore, treat the fields as logical entities. It is also possible that the positions and sizes of the fields will change from release to release of RMS-11.

However, you can determine the position of any field in a FAB as an offset from the FAB's starting address. RMS-11 represents these offset values with symbols in one of the following forms:

- O$fnm

  where fnm is the three-letter name of a one-byte or one-word field; fnm is the name used to reference the field in the initialization and field access macros.

  Example O$STS for the status code field

- O$fnmx

  where fnm is the name of a multiword field; fnm is the name used to reference the field in the initialization and field access macros.

  x is a number associated with an individual word in the field, from 0 through the end of the field.

  Example O$ALQ0 for less significant word and Q$ALQ1 for the more significant word in the ALQ field

The values of these symbols can be found in the symbol table of an assembly listing file for any module containing the FAB.

Table 3-1: File Access Block Fields

Field Name	Field Size	Default	Description
ALQ	2W	0	Size of file
BID	1B	N/A	FAB identifier
*BKS	1B	1 record	Bucket size
BLN	1B	N/A	RAB length
*BLS	1W	0	Block size
BPA	1W	0	Location of private I/O buffer
BPS	1W	0	Size of private I/O buffer
CTX	1W	0	Available to user
*DEQ	1W	0	Automatic extension quantity
DEV	1B	N/A	Device characteristics
DNA	1W	0	Location of filespec defaults
DNS	1B	0	Size of filespec defaults
FAC	1B	FB$GET	Types of record operations
FNA	1W	0	Location of filespec
FNS	1B	0	Size of filespec
FOP	1W	0	File processing options
*FSZ	1B	0	Size of fixed area for VFC
IFI	1W	N/A	Pointer to IFAB
LCH	1B	0	Logical channel
*MRN	2W	0	Maximum Record Number
*MRS	1W	0	Maximum Record Size
NAM	1W	0	Pointer to NAM Block
*ORG	1B	FB$SEQ	File organization
*RAT	1B	0	Record attributes
*RFM	1B	FB$VAR	Record format
RTV	1B	0	Window size
*RTV	1B	0	Clustersize
SHR	1B	0	File sharing
STS	1W	N/A	Completion code
STV	1W	N/A	More error information
XAB	1W	0	Pointer to first XAB

*File attribute

```
**************
*            *
*    ALQ     *
*            *
**************
```

3.1 ALQ

The two-word Allocation Quantity (ALQ) field contains the size (allocation quantity) of a disk file, in blocks. RMS-11 ignores the field for magnetic tape files.

3.1.1 Use

Input to:

$CREATE
: You set the ALQ field equal to the number of blocks to be allocated in the initial extent of the file.

$EXTEND
: You set the ALQ field equal to the number of blocks to be added to the file.

Output from:

$OPEN
: RMS-11 sets the ALQ field equal to the virtual block number of the last block in the existing file; this is also the number of blocks in the file. RMS-11 obtains this information from the file attributes. RMS-11 updates attributes during $CREATE and an implicit or explicit file extension. RMS-11 uses only the attributes for this information: any blocks allocated to the file by non-RMS-11 tasks is essentially invisible to RMS-11; however, when RMS-11 extends the file, those blocks quickly satisfy the request.

$EXTEND
: RMS-11 sets the ALQ field equal to the number of blocks that were added to the file.

3.1.2 Input Values

$CREATE

MINIMUM = 0, meaning that the actual size of the file, when it is created, depends on the file organization:

- Sequential files are created with four blocks.

- Relative and Indexed files are created with an allocation four times bucket size.

MAXIMUM = number of free blocks on the device containing the file; specifying a larger value results in error code ER$FUL

$EXTEND

MINIMUM = 1, meaning that one block should be added to the file (although a zero extent is possible, it is not logical)

MAXIMUM = number of blocks on the device containing the file; specifying a larger value results in error code ER$FUL

For Relative and Indexed files, RMS-11 rounds values up to multiple of bucket size.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.1.3 Initialization and Default

Macro takes the form:

F$ALQ numeric

where numeric is a number of blocks as discussed under "Values."

If there is no initialization macro, ALQ = 0.

3.1.4 Comments

If Allocation XABs are linked to the FAB, the create and extend operations ignore the FAB ALQ field and obtain allocation quantities from the XABs; see Chapter 8. However, the open operation works as described.

```
************
*          *
*    BID   *
*          *
************
```

3.2 BID

The FAB$B macro sets the one-byte Block Identifier (BID) field to the File Access Block identifier, with the symbolic value of FB$BID.

CAUTION

DO NOT CHANGE THE BID FIELD.

3.2.1 Use

Before RMS-11 uses a FAB during a file operation, it verifies that the block is a valid FAB; one of the checks examines the BID field. If this field does not contain the proper code, RMS-11 aborts the operation with an ER$FAB error code.

FILE ATTRIBUTE

****
*   *
* BKS   *
*   *
****

3.3 BKS

The one-byte Bucket Size (BKS) field contains the size of a bucket, in disk blocks, for a Relative or Indexed file. The field has no meaning for Sequential files.

3.3.1 Use

Input to:

$CREATE
You set the BKS field equal to the number of disk blocks in a bucket for the file to be created, if the ORG field contains either FB$REL or FB$IDX*.

Output from:

$OPEN
RMS-11 sets the BKS field equal to the bucket size established for the file when it was created, if the ORG field contains either FB$REL or FB$IDX*. RMS-11 obtains the information from the file attributes.

3.3.2 Input Values

MINIMUM = 0, meaning that RMS-11 calculates a size so that a bucket contains at least one record.

MAXIMUM = number of blocks allowed by the operating system (specifying a larger value results in error code ER$BKS):

IAS = 32 blocks
RSTS/E = 15 blocks
RSX-11M = 32 blocks

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

---

*If the file is Sequential, RMS-11 ignores the field during $CREATE and sets it to zero during $OPEN.

3.3.3 Initialization and Default

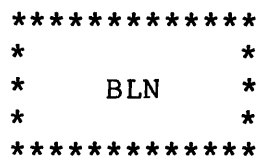
Macro takes the form:

```
F$BKS numeric
```

If there is no initialization macro, BKS = minimum number of blocks to contain one record.

3.3.4 Comments

- If Allocation XABs are linked to the FAB, the create and extend operations ignore the FAB BKS field and obtain bucket size(s) from the XAB(s); see Chapters 5 and 8.
- See the RMS-11 User's Guide for a discussion on bucket sizes.
- Records may not span bucket boundaries.



3.4 BLN

The FAB$B macro sets the one-byte Block Length (BLN) field to the File Access Block length, with the symbolic value of FB$BLN.

CAUTION

DO NOT CHANGE THE BLN FIELD.

3.4.1 Use

Before RMS-11 uses a FAB during a file operation, it verifies that the block is a valid FAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.

FILE ATTRIBUTE

****
*   *
* BLS   *
*   *
****

3.5 BLS

The one-word Block Size (BLS) field contains the physical block size* in characters for a Sequential file stored on MAGNETIC TAPE only.

3.5.1 Use

Input to:

$CREATE
: You set the BLS field equal to the number of characters per physical block in the magtape file to be created. RMS-11 rounds this number up to the next multiple of four before it creates the file and sets up an I/O buffer.

Output from:

$OPEN
: RMS-11 sets the BLS field equal to the size of the physical blocks in the file existing on magtape. RMS-11 obtains the information from the file label on tape. If the information is not in the label, RMS-11 uses the default block size of the device.

3.5.2 Input Values

If you intend to use magtape to transfer data to a non-PDP-11 computer system, note the following:

- If the destination is another DIGITAL system, block size should be less than or equal to 512 characters.
- If the destination is a non-DIGITAL system, block size should be less than or equal to 2048 characters.

---

*A block on magnetic tape is the data between interblock gaps. block size is expressed in characters; the representation of a character on tape is determined by the tape formatting standard.

MINIMUM = 0, at run-time RMS-11 interprets a value of 0 as the operating system default: in all systems covered by this manual, the default is 512 characters.

= 18, the smallest valid block size allowed by magtape device driver software

MAXIMUM = 8192 characters

A value can be set in the field with the initialization macro shown be or with the $STORE field access macro (see Chapter 7).

3.5.3 Initialization and Default

Macro takes the form:

```
F$BLS numeric
```

If there is no initialization macro, BLS = 0.

```
************
*          *
*    BPA   *
*          *
************
```

3.6 BPA

The one-word Buffer Pool Address (BPA) field points to an area in your program set aside for use as an I/O buffer for all Record Access Streams connected to the file represented by the FAB. If the address is zero, you are allocating the I/O buffer in some other way (see Chapters 1 and 2).

3.6.1 Use

Input to:

$CREATE/$OPEN

- If you want to specify a private buffer for the current file access (open to close), store the buffer's address in the BPA field before you open the file; also store the size of the buffer in the BPS field.
- If you are not using a private buffer, ensure that the BPA field is zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the buffer indicated: the results are unpredictable.

Output from:

$CLOSE
RMS-11 sets the BPA field equal to the address of the private buffer pool it is returning for your use.

3.6.2 Input Values

BPA must contain either:

- zero (0) = there is no private buffer for this file; all space is allocated by RMS-11 from the central pool or by your GSA routine
- address of the first byte of a private buffer for the file; the buffer must begin on a word boundary

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.6.3 Initialization and Default

Macro takes the form:

```
F$BPA label
```

If there is no initialization macro, BPA = 0.

3.6.4 Comments

- If you are using a private buffer for the file's I/O buffer, you still have to allocate space in the central buffer pool for the internal structures (BDBs, IFAB, and so on) associated with the file; see Chapter 2.
- Once you have assigned a private buffer to a file in a FAB and opened that file, you cannot use that space until the file is closed.

```
************
*          *
*    BPS   *
*          *
************
```

3.7 BPS

The one-word Buffer Pool Size (BPS) field contains the size, in bytes, of the private buffer indicated by the BPA field. RMS-11 uses this value to calculate the ending address of the private buffer.

CAUTION

RMS-11 does no further checking of the memory area defined by the FAB BPA and BPS fields. RMS-11 uses all the space allotted if it needs it.

3.7.1 Use

Input to:

$CREATE/$OPEN
: If BPA is equal to zero, RMS-11 ignores the BPS field; if BPA is not equal to zero, that is, you are providing a private buffer, you must set the BPS field equal to size of that buffer in bytes.

Output from:

$CLOSE
: RMS-11 sets the BPS field equal to the size of the private buffer it is returning for your use.

3.7.2 Input Values

BPS must contain either:

- any value* if the BPA field is zero
- size of the private buffer for the file, in bytes, as a multiple of two

  To calculate the size of the private buffer, use the equation:

  BPS = strmsz1 + strmsz2 + ... + strmszn

---

*Recommended value is the default, zero.

where BPS is the size of the private buffer,

strmsz1
strmsz2
.
.
.
strmszn

are the I/O buffer space requirements (in bytes) for the Record Access Streams associated with the file that are active simultaneously. You calculate the requirements for each stream as follows:

-- For Sequential files on disk: strmsz = 512*MBC

-- For Sequential files on magnetic tape: strmsz = BLS

where BLS is the size, in characters, of each physical block of the magtape file, that is, the value contained in the BLS field of the FAB, and

MBC is the value contained in the MBC field of the RAB associated with the stream.

-- For Relative files: strmsz = BKS*512

where BKS is the number of blocks in a bucket of the file (from the FAB BKS fields).

-- For Indexed files: strmsz = BKS*MBF*512

where BKS is the number of blocks in the largest bucket of the file (selected from the FAB BKS and the Allocation XAB BKZ fields), and

MBF is the value contained in the MBF field of the RAB associated with the stream.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.7.3 Initialization and Default

Macro takes the form:

```
F$BPS numeric
```

If there is no initialization macro, BPS = 0.

3.7.4 Examples

```
        .EVEN
INBUF:  .BLKB   4096.
          .
          .
MASTER: FAB$B
          .
          .
        F$BPA   INBUF
        F$BPS   4096
          .
          .
        FAB$E
```

```
************
*          *
*    CTX   *
*          *
************
```

3.8 CTX

The one-word User Context (CTX) field is not used by RMS-11, but is made available to you to contain any information you want associated with the file during run time.

3.8.1 Use

Anything you want. For example, you might use the CTX field to communicate with a common completion routine.

3.8.2 Input Values

Anything you want, as long as it fits in 16 bits.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.8.3 Initialization and Default

Macro takes the form:

```
F$CTX argument
```

where argument is any value you want.

If there is no initialization macro, CTX = 0.

3.9 DEQ

The one-word Default Extension Quantity (DEQ) field contains the number of blocks RMS-11 requests whenever it automatically extends the file (opposed to the explicit extend operation). RMS-11 requests additional blocks from the operating system* whenever a task attempts a put or update operation and there is not enough room in the file represented by the FAB to complete the operation. For Relative and Indexed files, RMS-11 rounds DEQ up to the nearest multiple of bucket size before requesting the extension.

3.9.1 Use

Input to:

$CREATE
: You set the DEQ field equal to the number of blocks that should be requested in each automatic extension of the file to be created.

$OPEN
: You set the DEQ field equal to the number of blocks to be used by RMS-11 as a temporary default extension quantity until the file is closed; the file attribute is not changed.

Output from:

$OPEN
: If the DEQ field is zero, RMS-11 sets it equal to the number of blocks established as the default extension quantity when the file was created.

3.9.2 Input Values

Default extension quantity should be a multiple of bucket size for Relative and Indexed files.

MINIMUM = 0, meaning that whenever RMS-11 has to extend the file, it requests five blocks for Sequential files and four times

---

*Automatic extension can fail; see "Comments."

bucket size for Relative and Indexed files.

MAXIMUM = 65,535 blocks

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.9.3 Initialization and Default

Macro takes the form:

```
F$DEQ numeric
```

If there is no initialization macro, DEQ = 0.

3.9.4 Comments

- If Allocation XABs are linked to the FAB, RMS-11 uses the FAB DEQ value as a file default extension quantity, substituting it whenever an area has to be extended automatically and the XAB DEQ field for that area is zero.

- Automatic extension can fail for at least the following reasons; there might be others:

  -- Operating system is RSTS/E and file is:

  ```
  contiguous;  error code = ER$PRV
  shared;  error code = ER$PRV
  ```

  -- No more room is available on the storage device; error code = ER$FUL

- On the RSTS/E operating system, file allocation is done in clusters; see the RMS-11 User's Guide for a discussion of the interaction between clusters and default extension quantity.

- See the RMS-11 User's Guide for a discussion of default extension quantity and its optimization.

```
************
*          *
*    DEV   *
*          *
************
```

3.10 DEV

The one-byte Device Characteristics (DEV) field contains a bit string indicating the generic characteristics of the device containing the file represented by the FAB.

3.10.1 Use

Output from:

$CREATE/$OPEN
: RMS-11 sets the field equal to the appropriate device indicator. You can use field access macros (see Chapter 7) to examine the field.

3.10.2 Output Values

The DEV field contains one or more of the following symbolic values:

FB$CCL
: Carriage control device, such as printer or terminal

FB$MDI
: Multiple-directory-structured device, such as a disk with a Master File Directory and at least one User File Directory

FB$REC
: Unit record device, such as a terminal or line printer; all unit record devices are considered to be Sequential in nature

FB$SDI
: Single-directory device, such as a disk with a Master File Directory, but no User File Directories present

FB$SQD
: Sequential, block-oriented device, that is, magnetic tape

FB$TRM
: Terminal device with keyboard and hard- or soft-copy output

```
************
*          *
*    DNA   *
*          *
************
```

3.11 DNA

The one-word Default Name String (DNA) field points to an ASCII string containing the program-specific defaults for file specifications. During the file open process, if the file name string indicated by the FNA field is not a complete file specification, RMS-11 examines the DNA field. If it is not zero, RMS-11 uses the string indicated to supply the missing components. If DNA is zero, or not all the defaults are supplied by the DNA string, RMS-11 uses the system defaults.

3.11.1 Use

Input to:

$CREATE/$OPEN

- If you want to specify a default name string for a file operation, store the string's address in the DNA field before you initiate the operation; also store the size of the string in the DNS field.
- If you are not using a default name string, ensure that the DNA field is zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the string indicated: the results are unpredictable.

3.11.2 Input Values

The DNA field must contain either:

- zero (0) = there is no default name string
- address of the default name string. This string may contain values for one or more of the following file specification components:

  device
  account
  file name
  file type
  file extension
  version number
  protection code

The string must be written in ASCII notation, that is, with the .ASCII/.ASCIZ MACRO-11 directive.

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

3.11.3 Initialization and Default

Macro takes the form:

```
F$DNA label
```

If there is no initialization macro, DNA = 0.

3.11.4 Examples

To set up a default device specification:

```
          .
          .
ANYFAB: FAB$B
          .
          .
        F$DNA   DEFDEV
        F$DNS   3
          .
          .
        FAB$E

DEFDEV: .ASCII /SY:/
```

3.11.5 Comments

- The default name string may include logical names.
- RMS-11 checks the DNA string validity during each create and open operation. RMS-11 returns an error if it finds a bad component even if the FNA string supplies a valid value for that component.
- See also FNA discussion about file specifications, parsing sequence on RSTS/E, and so on.

```
*************
*           *
*    DNS    *
*           *
*************
```

3.12 DNS

The one-byte Default Name String Size (DNS) field contains the size, in bytes, of the default name string indicated by the DNA field.

3.12.1 Use

Input to:

$CREATE/$OPEN
: If the DNA field is zero, RMS-11 ignores the DNA field; if the DNA field is not zero, that is, you are providing a default name string, you must set the DNS field equal to the size of that string in bytes.

3.12.2 Input Values

The DNS field must contain either:

- any value*, if the DNA field is zero
- the size of the default name string, in bytes, with a maximum of 255

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.12.3 Initialization and Default

Macro takes the form:

```
F$DNS numeric
```

If there is no initialization macro, DNS = 0.

---

*Recommended value is the default, zero.

```
*************
*           *
*    FAC    *
*           *
*************
```

3.13 FAC

The one-byte File Access (FAC) field contains a bit string indicating the types of record operations that may be performed on the file represented by the FAB. This declaration enables RMS-11 to set up transfer vectors to the appropriate RMS-11 routines. RMS-11 rejects any record operation that was not specified in the FAC field when the file is opened (error code ER$FAC).

3.13.1 Use

Input to:

$CREATE
: You set the FAC field to indicate all operations your program performs on the file during the current access (open to close).

NOTE

The FAC field must contain at least FB$PUT.

$OPEN
: You set the FAC field to indicate all operations your program performs on the existing file during the current access (open to close).

3.13.2 Input Values

FAC may contain one or more of the following symbolic values:

FB$DEL $DELETE operations

FB$GET $GET and/or $FIND operations

FB$PUT $PUT operations

FB$REA $READ Block I/O operations (see Chapter 9)

FB$TRN $TRUNCATE operations: valid for Sequential files only

FB$UPD $UPDATE operations

FB$WRT $WRITE Block I/O operations (see Chapter 9)

If you want to specify more than one operation, you must concatenate the values using an exclamation point (!).

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

3.13.3 Initialization and Default

Macro takes the form:

```
F$FAC symbolic[!symbolic...]
```

If there is no initialization macro, FAC = FB$GET.

3.13.4 Comments

- If the file is an ANSI magtape file, RMS-11 automatically positions the tape at the end of the file when it is opened, unless the FOP field contains FB$NEF.
- Since the FAC field is a bit string, values cannot be added to the field with the $STORE field access macro. You should use the $SET field access macro (see Chapter 7). However, you use the $STORE macro to (re)set all bits in the field.
- If you specify FB$DEL, FB$TRN, and/or FB$UPD, but do not include FB$GET, RMS-11 activates that capability anyway. Delete, truncate, and update operations must be preceded by a successful find or get operation.

```
************
*          *
*    FNA   *
*          *
************
```

3.14 FNA

The one-word File Name String Address (FNA) field points to the ASCII string that is the file specification of the file represented by the FAB. RMS-11 examines the string indicated, if FNA is not zero. If any file specification components are missing from the string, or FNA is zero, RMS-11 checks the DNA field. If DNA is not zero, RMS-11 uses the string indicated to fill in the filespec; if DNA is zero, or the string indicated is not sufficient, RMS-11 inserts system default values into the file specification. Then it passes the complete filespec to the operating system to either open or create, then open, a file by that name.

3.14.1 Use

Input to:

$CREATE/$OPEN

- If you want to specify a file specification for a file operation, store the specification's address in the FNA field before you initiate the operation; also store the size of the specification in the FNS field.
- If you are not using a file specification*, ensure that the FNA field is zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the string indicated: the results are unpredictable.

3.14.2 Input Values

FNA must contain either:

- zero (0) = there is no file specification
- address of file specification string for the file

The string must be written in ASCII notation, that is, with the .ASCII/.ASCIZ MACRO-11 directive.

---

*If you are using all defaults, program and/or system, to define the file specification, or file ID to identify the file without a file specification (see Chapter 8).

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.14.3 Initialization and Default

Macro takes the form:

```
F$FNA label
```

If there is no initialization macro, FNA = 0.

3.14.4 Comments

- File specifications contained in the FNA-indicated buffer must conform to operating system standards. See the RMS-11 User's Guide, Appendix A, for RMS-11 restrictions on the operating systems and operating system restrictions on RMS-11.
- On RSTS/E, RMS-11 parses file specifications in the following sequence. RMS-11:

  1. Places the string described by the DNA and DNS fields in the FIRQB.
  2. Issues a .FSS call on the string described by the FNA and FNS fields. In the process, the monitor merges the FNA and DNA strings in the FIRQB, overriding only those components already existing in the FNA string. The monitor also translates logical names.
  3. Checks if the FSS processing encountered an RMS-11 restriction on RSTS/E file specifications, such as switches or an equal sign (=).
  4. Examines the resultant file specification for device and account components. If these components are missing, RMS-11 obtains the system defaults for those fields and puts them into the file specification.
  5. Examines the resultant file specification. If a component is missing, returns the appropriate error code; if there are too many components, returns the error code ER$XTR.

```
************
*          *
*    FNS   *
*          *
************
```

3.15 FNS

The one-byte File Name String Size (FNS) field contains the size, in bytes, of the file specification indicated by the FNA field.

3.15.1 Use

Input to:

$CREATE/$OPEN
: If the FNA field is zero, RMS-11 ignores the FNA field; if FNA is not zero, that is, you are providing a file specification, you should set the FNA field equal to the size of that string in bytes.

3.15.2 Input Values

The FNA field must contain either:

- any value* if FNA is zero
- size of the file specification string, in bytes, with a maximum of 255

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.15.3 Initialization and Default

Macro takes the form:

```
F$FNS numeric
```

If there is no initialization macro, FNS = 0.

---

*Recommended value is the default, zero.

```
*************
*           *
*    FOP    *
*           *
*************
```

3.16 FOP

The one-word File Processing Options (FOP) field contains a bit string indicating the file processing options that you have selected for the file represented by the FAB.

3.16.1 Use

Input to:

$CREATE
: You set the FOP field to indicate the processing option(s) you want applied to the new file until it is closed.

$OPEN
: You set the FOP field to indicate the processing options you want applied to the existing file until it is closed.

$EXTEND
: You set the FOP field equal to FB$CTG if you want the file extended contiguously, that is, the new blocks allocated in such a manner that they are contiguous with the present extent of the file.

$CLOSE
: You set the FOP field equal to FB$RWC if you want a magnetic tape file rewound when the file is closed, but did not include the option when the file was opened.

Output from:

$OPEN
: If the file is contiguous, RMS-11 sets the FOP field equal to FB$CTG; otherwise, the field is zero. You can test for this value with field access macros (see Chapter 7).

3.16.2 Input Values

The FOP field may contain one or more of the following symbolic values:

FB$CTG
: RMS-11 is to allocate contiguously the amount of space specified in the FAB ALQ field. Used on $CREATE and $EXTEND only.

FB$DFW directs RMS-11 to defer writing the I/O buffer out to a file after a $DELETE, $UPDATE, or $PUT operation. Under default conditions, RMS-11 causes a physical transfer of data from your program to the file as part of any of the above record operations--for Relative and Indexed files only. However, if you specify FB$DFW in the RAB when you initiate one of these operations, RMS-11 does not automatically write out the buffer. Instead, it defers the disk write until it requires the buffer for some other purpose, such as reading a different bucket into memory or the buffer is full. Since this is similar to the default condition for Sequential files, specifying Deferred Write does not affect record operations on Sequential files.

FB$DLK RMS-11 unlocks the file so that it is available for access if this program does not close it in a normal manner. Used on $CREATE and $OPEN only. See "Comments."

FB$FID RMS-11 uses the value in the NAM FID field to open or erase the file (see Chapter 8). Used on $OPEN and $ERASE only.

FB$MKD RMS-11 creates this file, then deletes it when the file is closed. Used on $CREATE only.

FB$NEF RMS-11 does not position the ANSI magnetic tape to the end of the file even though the FAB FAC field contains FB$PUT. Used on $OPEN only.

FB$POS RMS-11 positions the magtape to the point immediately after the most recently closed file before it creates the file specified by this FAB; all subsequent files on the tape are logically deleted. If FB$POS is not specified, RMS-11 positions the magtape at the end of the last file on the volume. However, the FB$RWO option overrides the FB$POS option. Used on $CREATE only.

FB$RWC RMS-11 requests the operating system to rewind the magtape when the file is closed. If FB$RWC is set when the file is opened ($CREATE or $OPEN), RMS-11 does not notice its absence when the file is closed; that is, you can't undo this once specified. However, if FB$RWC was not specified for the open, you can include it for the $CLOSE and RMS-11 rewinds the tape.

FB$RWO RMS-11 requests the operating system to rewind the magtape before the file represented by the FAB is created and/or opened. Used on $CREATE and $OPEN only.

FB$SUP RMS-11 supersedes an existing file if the file specification indicated by this FAB contains an explicit version number. Used on $CREATE only.

FB$TMD RMS-11 creates this file as a temporary file and deletes it when the file is closed. Used on $CREATE only.

FB$TMP RMS-11 creates this file as a temporary file and does not delete it; that is, RMS-11 retains the file, when it is closed, even though it is not entered into the directory. The FB$TMD option overrides the FB$TMP option. Used on $CREATE only.

If you want to specify more than one option, you must concatenate the values using an exclamation point (!).

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

3.16.3 Initialization and Default

Macro takes the form:

```
F$FOP symbolic[!symbolic...]
```

If there is no initialization macro, FOP = 0, meaning:

- The file is created or extended noncontiguously.
- If file is not closed normally, it remains locked (see "Comments").
- Open operation accesses file via filespec.
- RMS-11 positions magtape to end of the file when it opens the file.
- RMS-11 positions magtape to end of the last file on volume before it creates the specified file.
- RMS-11 does not request the operating system to rewind magtape when the file is closed.
- RMS-11 fails with error code ER$FEX if the filespec contains an explicit version number and the file already exists.
- RMS-11 creates a permanent file.

3.16.4 Comments

- The IAS and RSX-11M operating systems provide protection to any file that is not closed properly* by a task that creates or opens it: the file is locked and not accessible by any user task until a utility (PIP) unlocks it.

  This protection is particularly valuable for Sequential files. When RMS-11 opens a Sequential file, it reads the file's attributes, including end-of-file location from the file header into memory; however, RMS-11 does not update the file header until the

file is closed. Until the file's attributes are revised on disk, any extension of the file while it was opened (and any data added to it) is not permanently recorded. Therefore, if a Sequential file is not closed properly*, all data added while it was open could be lost if another task does access the file and make its own revision of the file header.

These considerations do not apply to Relative and Indexed files. Therefore, they are the only files for which the FB$DLK value should be specified.

- Since the FOP field is a bit string, values cannot be added to the field with the $STORE field access macro. You should use the $SET field access macro (see Chapter 7). However, you use the $STORE macro to (re)set all bits in the field.

---

*Caused by an improperly designed task, operating system collapse, or hardware malfunction.

FILE ATTRIBUTE

****
*   *
* FSZ   *
*   *
****

3.17 FSZ

The one-byte Fixed Control Area Size (FSZ) field contains the size, in bytes, of the fixed control area for Variable-with-Fixed-Control (VFC) records in the file represented by the FAB.

3.17.1 Use

The FSZ field is only used when a file contains (or will contain) VFC records. It therefore applies only to Sequential and Relative files.

Input to:

$CREATE
: You set the FSZ field equal to the number of bytes in the fixed control area of the VFC records that will be written into the file to be created.

Output from:

$OPEN
: RMS-11 sets the FSZ field to the size of the fixed control area established when the file was created.

3.17.2 Input Values

MINIMUM = 0, meaning that RMS-11 sets the control area size to two bytes*

= 1, the smallest fixed control area allowed

MAXIMUM = 255

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.17.3 Initialization and Default

Macro takes the form:

```
F$FSZ numeric
```

---

*Compatible with FCS.

If there is no initialization macro, FSZ = 0.

3.17.4 Comments

RMS-11 uses this number as the size of the buffer indicated by the RAB RHB field.

```
************
*          *
*    IFI   *
*          *
************
```

3.18 IFI

The $OPEN macro sets the one-word Internal File Identifier (IFI) field to an address that links the FAB to the IFAB RMS-11 creates when a file is opened. The $CLOSE macro clears the IFI field.

CAUTION

DO NOT CHANGE THE IFI FIELD.

3.18.1 Use

RMS-11 uses the address in the IFI field to transfer pertinent data from the FAB to the IFAB.

```
************
*          *
*    LCH   *
*          *
************
```

3.19 LCH

The one-byte Logical Channel (LCH) field contains the number of the logical channel through which all I/O operations between the file and the task are performed.

3.19.1 Use

Input to:

$CREATE/$OPEN
: You set the LCH field to the number of the channel the file should be linked with until it is closed. Each file opened must have a unique LCH (error code ER$LBY).

3.19.2 Input Values

MINIMUM is system-dependent

MAXIMUM is system-dependent

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.19.3 Initialization and Default

Macro takes the form:

```
F$LCH numeric
```

If there is no initialization macro, LCH = 0, meaning the field must be set with the $STORE field access macro before the file open is attempted.

3.19.4 Comments

- Multiple Record Access Streams is a method of using one logical channel to carry multiple functions.

- Once a logical channel is assigned to a device, no operation, including $CLOSE, automatically deassigns the channel. The channel remains assigned to the device until you reassign it.

3.20 MRN

The two-word Maximum Record Number (MRN) field contains the maximu number of records that can be written to the Relative file associate with the FAB. RMS-11 checks the relative record number used with eac put operation against this maximum and rejects (error code ER$MRN) an operation with too high a number.

3.20.1 Use

The MRN field is only meaningful for Relative files.

Input to:

$CREATE
: You set the MRN field to the highest relative number of an record that will (or should) be written into the file*. If you want no checks on relative record number, set MRN equal to zero.

Output from:

$OPEN
: RMS-11 sets the MRN field equal to the value established when the file* was created.

3.20.2 Input Values

MINIMUM = 0, meaning that RMS-11 writes the MRN value as 2,147,483,647** during the create operation, returns this number during an open operation, and makes no checks on relative record numbers during put operations.

MAXIMUM = number of records that will fit on the device containing the file

---

*Unless the FAB ORG field contains FB$REL, RMS-11 ignores the MRN field during $CREATE and sets it to zero during $OPEN.
**The largest possible positive value for the 32-bit field.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.20.3 Initialization and Default

Macro takes the form:

```
F$MRN numeric
```

If there is no initialization macro, MRN = 0.

3.21 MRS

The one-word Maximum Record Size (MRS) field contains the maximum size, in bytes, of any record in the file associated with the FAB. RMS-11 checks all records written to the file against this maximum and rejects those that are too large (error code ER$RSZ).

3.21.1 Use

Input to:

$CREATE
You set the MRS field equal to the number of bytes in:

- all fixed-length records in the file
- the largest variable-length or stream record that can be written to the file.
- The variable area in the largest VFC record that can be written to the file, that is, the number does not include the fixed control area.

Output from:

$OPEN
RMS-11 sets the MRS field equal to the value established when the file was created.

3.21.2 Input Values

MINIMUM = 0, meaning that RMS-11 makes no checks on the length of records written to the file via either $PUT or $UPDATE. However, there are restrictions on the use of zero in the MRS field; see "Comments."

MAXIMUM = 32767, the largest number that can be stored in the signed two-byte field. However, Maximum Record Size is further limited by bucket size in Relative and Indexed files and by task buffer space for all files.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.21.3 Initialization and Default

Macro takes the form:

```
F$MRS numeric
```

If there is no initialization macro, MRS = 0.

3.21.4 Comments

A nonzero MRS field is required during the $CREATE operation in the following situations:

- Fixed-length records (FAB RFM field = RB$FIX)
- Relative file organization (FAB ORG field = FB$REL)

```
*************
*           *
*    NAM    *
*           *
*************
```

3.22 NAM

The one-word Name Block Address (NAM) field points to a separately allocated Name Control Block (NAM) and thereby activates RMS-11 to use the NAM Block when the file is opened. See Chapter 6.

3.22.1 Use

Input to:

$CLOSE

If you have specified a valid NAM Block, RMS-11 clears the first word of the FID field, making the field invalid for any subsequent operations.

$CREATE

- If you have allocated a NAM Block and want RMS-11 to set the fields when it opens the file, you set the NAM field equal to the address of the block.
- If you have not allocated a NAM Block and/or you do not want RMS-11 to fill it in during the file open procedure, set the NAM field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as a NAM Block: if that area is not a valid NAM Block, RMS-11 returns the error code ER$NAM.

$ERASE

- If you want RMS-11 to use the device and file IDs in NAM DVI and FID fields to identify, then erase a file. The file ID must be valid and probably was supplied by RMS-11 during a create or open operation. You must also set FB$FID in the FAB FOP field.
- If you have not allocated a NAM Block and/or you do not want RMS-11 to use it in during the erase operation, set the NAM field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as a NAM Block: if that area is not a valid NAM Block, RMS-11 returns the error code ER$NAM.

$OPEN

- If you have allocated a NAM Block and want RMS-11 to fill in the fields when it opens the file, you set the NAM field equal to the address of the block.

- If you want RMS-11 to use the device and file IDs in NAM DVI and FID fields to identify, then open a file. The file ID must be valid and probably was supplied by RMS-11 during a create or open operation. You must also set FB$FID in the FAB FOP field.

- If you have not allocated a NAM Block and/or you do not want RMS-11 to fill it in during the open operation, set the NAM field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as a NAM Block: if that area is not a valid NAM Block, RMS-11 returns the error code ER$NAM.

3.22.2 Input Values

The NAM field must contain either:

- zero (0) = there is no NAM Block associated with this FAB
- address of a Name Control Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.22.3 Initialization and Default

Macro takes the form:

```
F$NAM label
```

If there is no initialization macro, NAM = 0.

FILE ATTRIBUTE

****
*   *
* ORG   *
*   *
****

3.23 ORG

The one-byte File Organization (ORG) field contains an indicator for the organization of the file associated with the FAB.

3.23.1 Use

Input to:

$CREATE
: You set the ORG field to indicate the organization you want established for the file to be created.

Output from:

$OPEN
: RMS-11 stores in the ORG field the indicator for the organization of the existing file. You can test this value with field access macros (see Chapter 7).

3.23.2 Input Values

The ORG field may contain one of the following symbolic values:

FB$SEQ Sequential file organization

FB$REL Relative file organization

FB$IDX Indexed file organization

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.23.3 Initialization and Default

Macro takes the form:

F$ORG symbolic

If there is no initialization macro, ORG = FB$SEQ.

FILE ATTRIBUTE

```
**********
*        *
*  RAT   *
*        *
**********
```

3.24 RAT

The one-byte Record Attributes (RAT) field contains a bit string describing the attributes of the records in the file associated with the FAB.

3.24.1 Use

Input to:

$CREATE
: You set the RAT field to indicate the characteristics of the records that will be stored in the file to be created.

Output from:

$OPEN
: RMS-11 stores in the RAT field indicators of the record attributes established for the file when it was created. You can test for these values using field access macros (see Chapter 7).

3.24.2 Input Values

The RAT field may contain one or more of the following symbolic values:

FB$BLK
: records in the file do not cross block boundaries; valid only for Sequential files. This attribute restricts record size to 512 bytes.

FB$CR
: when RMS-11 writes a record from this file to a unit record device, that is, a terminal or printer opened as a Sequential file, it precedes the record with a line feed character and follows it with a carriage return character. RMS-11 may or may not actually add the characters, depending on the operating system.

FB$FTN
: the first byte of each record in the file contains a FORTRAN forms control character

If you want to specify more than one attribute, you must concatenate the values using an exclamation point (!).

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

3.24.3 Initialization and Default

Macro takes the form:

```
F$RAT symbolic[!symbolic...]
```

If there is no initialization macro, RAT = 0, meaning:

- Records span block boundaries
- No carriage or FORTRAN control

3.24.4 Comments

- If the file associated with the FAB is a unit record device, such as a terminal or printer, you should put at least FB$CR in the RAT field.
- By default, records in all files are allowed to cross block boundaries. Although this attribute can be restricted for Sequential files, there seems to be little reason to do so.
- Since the RAT field is a bit string, values cannot be added to the field with the $STORE field access macro. You should use the $SET field access macro (see Chapter 7). However, you use the $STORE macro to (re)set all bits in the field.

FILE ATTRIBUTE

```
************
*          *
*    RFM   *
*          *
************
```

3.25 RFM

The one-byte Record Format (RFM) field contains an indicator for the format of the records in the file associated with the FAB.

3.25.1 Use

Input to:

$CREATE
: You set the RFM field to indicate the record format you want established for the file to be created.

Output from:

$OPEN
: RMS-11 stores in the RFM field the indicator for the record format of the existing file. You can test this value with field access macros (see Chapter 7).

3.25.2 Input Values

The RFM field may contain one of the following symbolic values:

FB$FIX
: fixed-length records

FB$STM
: ASCII stream record format; valid only for disk Sequential files. The RMS-11 User's Guide explains how RMS-11 handles stream records.

FB$UDF
: no record format is defined for the file; valid only for Sequential files during create operations. When accessing a file with undefined record format, your program must use block I/O (see Chapter 9).

FB$VAR
: variable-length records

FB$VFC
: variable-with-fixed-control records; valid only for disk Sequential and Relative files

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.25.3 Initialization and Default

Macro takes the form:

```
F$RFM symbolic
```

If there is no initialization macro, RFM = FB$VAR.

```
************
*          *
*    RTV   *
*          *
************
```

The meaning of the RTV field depends on your operating system. This section defines the field for:

IAS/RSX-11M

3.26 RTV

The one-byte Retrieval Window Size (RTV) field contains the number of retrieval pointers kept in memory* for the file represented by the FAB. Retrieval pointers map virtual block numbers to logical block numbers. See Chapter 8 of the RMS-11 User's Guide for more information.

3.26.1 Use

Input to:

$CREATE/$OPEN
: You set the RTV field equal to the size of window you want maintained for the file until it is closed.

3.26.2 Input Values

The RTV field must contain one of the following:

- -1 = as much of the file as possible is mapped with one window
- 0 = RMS-11 uses the current default value
- a nonzero positive value, specifying the number of retrieval pointers to be maintained in memory:

  MINIMUM = 1
  MAXIMUM = 127

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

---

*This set of pointers is called a "window."

3.26.3 Initialization and Default

Macro takes the form:

```
F$RTV numeric
```

If there is no initialization macro, RTV = 0.

The meaning of the RTV field depends on your operating system. This section defines the field for:

RSTS/E

3.27 RTV

The one-byte Clustersize (RTV) field contains the number of blocks in each cluster of the file represented by the FAB.

3.27.1 Use

Input to:

$CREATE
: You set the RTV field to the clustersize you want established for the file to be created.

Output from:

$OPEN
: RMS-11 sets the RTV field to the clustersize established for the file when it was created.

3.27.2 Input Values

The value in the RTV field must be:

- a positive integer
- a power of two; else ER$CRE
- greater than or equal to volume* clustersize

MINIMUM = 0, meaning that RMS-11 uses the volume* clustersize

MAXIMUM = 255, meaning that RMS-11 sets clustersize to 256 blocks

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

---

*"Volume" means the disk pack as opposed to the disk device, which has its own characteristic clustersize.

3.27.3 Initialization and Default

Macro takes the form:

```
F$RTV numeric
```

If there is no initialization macro, RTV = 0.

```
************
*          *
*    SHR   *
*          *
************
```

3.28 SHR

The one-byte File Sharing (SHR) field contains a bit string indicating the operations on the file represented by the FAB you are willing to share with other tasks. This sharing criterion is applied whether or not this is the first task to access the file: if one or more other tasks have already opened the file to write and the SHR field indicates no write sharing, the open operation fails; however, if this task is the first to open the file, specifying no write sharing, all other tasks that indicate write operations* are denied access.

See also Chapter 2 of the RMS-11 User's Guide.

3.28.1 Use

Input to:

$CREATE/$OPEN
: You set the SHR field to indicate the type of operations you allow other tasks accessing the file.

3.28.2 Input Values

The SHR field may contain one of the following:

- zero (0), meaning no write sharing; however, other tasks may access the file for reading only (FB$GET only in the FAB ROP field)
- FB$WRI, meaning that other tasks may access the file for write operations

NOTE

For Sequential files, FB$WRI can be used only when the FAC field does NOT contain FB$PUT or FB$UPD. If you want write access, you cannot allow others to have write access.

A value can be set in the field with the initialization macro shown

---

*FB$DEL, FB$PUT, and/or FB$UPD in FAB ROP field.

below or with the $SET field access macro (see Chapter 7).

3.28.3 Initialization and Default

Macro takes the form:

```
F$SHR FB$WRI
```

If there is no initialization macro, SHR = 0.

```
************
*          *
*    STS   *
*          *
************
```

3.29 STS

The one-word Completion Status Code (STS) field contains a code indicating the success or failure (and type of failure) of a file operation. See Appendix A for symbolic and octal codes for both success and failure of operations.

3.29.1 Use

Output from:

$CLOSE/$CREATE/$DISPLAY/$ERASE/$EXTEND/$OPEN
: RMS-11 sets the STS field equal to a completion code. You should test this field ($COMPARE field access macro) for a negative value after each file operation, or you can use an error completion routine that checks the value (see Chapter 8).

3.29.2 Input Values

No input values; see Appendix A for output values.

```
************
*          *
*    STV   *
*          *
************
```

3.30 STV

The one-word Status Value (STV) field contains additional information about some error codes returned in the STS field.

3.30.1 Use

Output from:

$CLOSE/$CREATE/$DISPLAY/$ERASE/$EXTEND/$OPEN

RMS-11 sets the STV field whenever it has additional information to communicate about an error. See Appendix A for the specific errors on which the STV field is used. You should check the STV field for a nonzero value whenever you detect an error (negative value in STS field).

3.30.2 Input Values

No input values; see Appendix A for output values.

****
*   *
* XAB   *
*   *
****

3.31 XAB

The one-word Extended Attribute Block Address (XAB) field, if not zero, points to a separately allocated Extended Attribute Block (XAB) and thereby indicates to RMS-11 that there is a chain of at least one XAB that must be examined for input to and filled with output from a file operation. RMS-11 finds the other XABs in the chain, if there are any, via the NXT field in the XAB indicated by the FAB XAB field.

3.31.1 Use

Input to:

$CREATE/$DISPLAY/$EXTEND/$OPEN

- If you have allocated one or more XABs and want RMS-11 to use the chain, you set the XAB field equal to the address of the first XAB in the chain.
- If you have not allocated an XAB and/or you do not want RMS-11 to use it during the file open procedure, set the XAB field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns error code ER$XAB.

3.31.2 Input Values

The XAB field must contain either:

- zero (0) = there is no XAB associated with this FAB
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

3.31.3 Initialization and Default

Macro takes the form:

F$XAB label

If there is no initialization macro, XAB = 0.

3.31.4 Comments

- See Chapter 5 for details on XABs.
- An Indexed file requires at least one XAB to define the Primary Key.

CHAPTER 4

RECORD ACCESS BLOCK

The initialization macros (this chapter) and the field access macros (Chapter 7) are provided so that you do not have to know the specific position and to a large extent, the size of each field in a RAB. You can, therefore, treat the fields as logical entities. It is also possible that the positions and sizes of the fields will change from release to release of RMS-11.

However, you can determine the position of any field in a RAB as an offset from the RAB's starting address. RMS-11 represents these offset values with symbols in one of the following forms:

- O$fnm

  where fnm is the three-letter name of a one-byte or one-word field; fnm is the name used to reference the field in the initialization and field access macros.

  Example O$STS for the status code field

- O$fnmx

  where fnm is the name of a multiword field; fnm is the name used to reference the field in the initialization and field access macros.

  x is a number associated with an individual word in the field, from 0 through the end of the field.

  Example O$BKT0 for less significant word and Q$BKT1 for the more significant word in the BKT field

The values of these symbols can be found in the symbol table of an assembly listing file for any module containing the RAB.

Table 4-1: Record Access Block Fields

Field Name	Field Size	Default	Description
BID	1B	N/A	RAB identifier
BKT	2W	None	Relative record number or VBN
BLN	1B	N/A	RAB length
CTX	1W	0	User area
FAB	1W	0	FAB address
ISI	1W	N/A	Pointer to IRAB
KBF	1W	0	Key buffer address
KRF	1B	0	Key of reference
KSZ	1B	0	Key buffer size
MBC	1B	0	Multiblock count
MBF	1B	0	Multibuffer count
RAC	1B	0	Record Access Mode
RBF	1W	0	Address of output record
RFA	3W	0	Record's File Address
RHB	1W	0	Fixed control area buffer
ROP	1W	0	Record processing options
RSZ	1W	0	Size of output record
STS	1W	N/A	Completion status code
STV	1W	N/A	Status value
UBF	1W	0	Input record buffer
USZ	1W	0	Input record buffer size

```
**********
*        *
*  BID   *
*        *
**********
```

4.1 BID

The RAB$B macro sets the one-byte Block Identifier (BID) field to the Record Access Block identifier, with the symbolic value of RB$BID.

CAUTION

DO NOT CHANGE THE BID FIELD.

4.1.1 Use

Before RMS-11 uses a RAB during a record operation, it verifies that the block is a valid RAB; one of the checks examines the BID field. If this field does not contain the proper code, RMS-11 terminates the operation with a ER$RAB error code.

```
*************
*           *
*    BKT    *
*           *
*************
```

4.2 BKT

The one-byte Bucket (BKT) field contains:

- the relative record number of the record accessed by the Record Access Stream during sequential access operations on a Relative file
- the Virtual Block Number when the Record Access Stream is performing Block I/O operations

4.2.1 Use

Input to Block I/O Operations:

$READ/$WRITE
: You set the BKT field equal to the number of the virtual block you want read or written.

Output from Relative File record operations:

$FIND/$GET/$PUT
: If you have specified sequential access (RB$SEQ in the RAB RAC field), RMS-11 sets the BKT field equal to the relative record number of the record read or written by the operation.

4.2.2 Input Values (Block I/O only)

MINIMUM = 0

MAXIMUM = value of FAB ALQ field

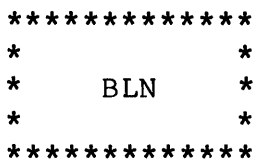
A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.2.3 Initialization and Default

Logical for Block I/O operations only, the macro takes the form:

```
R$BKT numeric
```

If there is no initialization macro, BKT = 0.



4.3 BLN

The RAB$B macro sets the one-byte Block Length (BLN) field to the Record Access Block length, with one of the following symbolic values:

RB$BLN = synchronous RAB

RB$BLL = asynchronous RAB

CAUTION

DO NOT CHANGE THE BLN FIELD.

4.3.1 Use

Before RMS-11 uses a RAB during a record operation, it verifies that the blocks is a valid RAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.

```
*************
*           *
*    CTX    *
*           *
*************
```

4.4 CTX

The one-word User Context (CTX) field is not used by RMS-11, but is made available to you to contain any information you want associated with the Record Access Stream during run time.

4.4.1 Use

Anything you want. For example, you might use the CTX field to communicate with a common completion routine.

4.4.2 Input Values

Anything you want, as long as it fits in 16 bits.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.4.3 Initialization and Default

Macro takes the form:

```
R$CTX argument
```

where argument is any value you want.

If there is no initialization macro, CTX = 0.

```
**********
*        *
*  FAB   *
*        *
**********
```

4.5 FAB

The one-word File Access Block (FAB) field points to the File Access Block connected to this RAB to form a Record Access Stream (RAS). The FAB must be associated with an open file at the time the $CONNECT macro is used. If an invalid address is stored in the FAB field, RMS-11 returns error code ER$IFI.

4.5.1 Use

Input to:

$CONNECT
: You set the FAB field equal to the address of the FAB you want associated with this RAB to make a Record Access Stream.

4.5.2 Input Values

Before the $CONNECT macro is executed, you must put a valid FAB address in the FAB field.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.5.3 Initialization and Default

Macro takes the form:

```
R$FAB label
```

If there is no initialization macro, FAB = 0, meaning you must set the field with the $STORE field access macro before initiating $CONNECT.

4.5.4 Comments

If you are using the same FAB for more than one file, you should set up all Record Access Streams associated with one file before you modify the FAB in preparation for another $OPEN or $CREATE.

4.6 ISI

The $CONNECT macro sets the one-word Internal Stream Identifier (ISI) field to an address that links the RAB to the IRAB RMS-11 has created.

CAUTION

DO NOT CHANGE THE ISI FIELD.

4.6.1 Use

RMS-11 uses the address in the ISI field to transfer pertinent data from the RAB to the IRAB.

```
************
*          *
*    KBF   *
*          *
************
```

4.7 KBF

The one-word Key Buffer (KBF) field points to the key that RMS-11 uses to locate a record during a random record operation:

- For Relative files, RMS-11 uses the value as the relative record number for $FIND, $GET, and $PUT operations.
- For Indexed files, RMS-11 uses the buffer contents as the key value during its index search for $FIND and $GET operations. The RAB KRF field determines the specific index searched and the contents of the RAB KSZ and ROP fields decide the match criteria used.

4.7.1 Use

Input to:

$FIND/$GET
: You set the KBF field to the address of the buffer in your program that contains the key for the next random record operation (RB$KEY in RAB RAC field). You must also set the KSZ and ROP fields appropriately.

$PUT
: For Relative files, you set the KBF field to indicate the relative record number of the cell where RMS-11 should write the specified record.

4.7.2 Input Values

The KBF field must contain a valid address before any random record operation macro is executed.

The address must indicate a buffer in the current program that contains either:

- a relative record number as a two-word integer that is positive, at least 1, and no greater than the Maxium Record Number (MRN) set for the file when it was created. The address must point to the least significant byte of the record number.
- a key value for an index search; this value can be any of the valid Indexed key types.

The buffer does not have to be word-aligned.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.7.3 Initialization and Default

Macro takes the form:

```
R$KBF label
```

If there is no initialization macro, KBF = 0, meaning you must set the field with the $STORE field access macro before any random record operation.

```
************
*          *
*    KRF   *
*          *
************
```

4.8 KRF

The one-byte Key of Reference (KRF) field indicates the index to be referenced in the current operation.

4.8.1 Use

The KRF field is only used for record operations on Indexed files.

Input to:

$FIND/$GET
: You set the KRF field to indicate the index that should be searched for the value defined by the contents of the KBF and KSZ fields; match criterion is also specified by the ROP field.

$CONNECT/$REWIND
: You set the KRF field to indicate the index that is used to set the context of the Record Access Stream, that is, the logical first record in the file.

4.8.2 Input Values

MINIMUM = 0, meaning the Primary Key

MAXIMUM = 254

All values beginning with 1 indicate Alternate Keys, first, second, and so on.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.8.3 Initialization and Default

Macro takes the form:

```
R$KRF numeric
```

If there is no initialization macro, KRF = 0.

```
************
*          *
*    KSZ   *
*          *
************
```

4.9 KSZ

The one-byte Key Size (KSZ) field contains the size, in bytes, of the key buffer indicated by the KBF field. RMS-11 requires specific sizes for a relative record number and all Indexed key types except string (see "Values"). However, for string keys, RMS-11 uses the number of characters specified in KSZ as a length criterion in the index search required by the record operation. Therefore, before a search can be successful, RMS-11 must find a record with a key value that matches the value indicated by KBF to the number of characters indicated by KSZ. If KSZ equals the length of the key (established when the file was created), the search requires an exact match, whereas when KSZ is less than the key length, only an approximate match is needed.

The value of the ROP field also affects match criteria.

4.9.1 Use

Input to:

$FIND/$GET
: For Relative files, you must set the KSZ field equal to 0 or 4.

  For Indexed files, you set the KSZ field equal to:

  - specific number of bytes for all key types except string
  - the number of characters in a key field that must match the KBF value before the search is successful

$PUT
: For Relative files, you must set the KSZ field equal to 0 or 4.

4.9.2 Input Values

The values of the KSZ field depend on the type of key indicated by the KBF field:

- Relative Record Number (FB$REL in FAB ORG field), KSZ = 4 (0 defaults to 4)

- String Key Type (FB$IDX in FAB ORG field):

  MINIMUM = 1

  MAXIMUM = size of key indicated by the KRF field

- 15-bit Signed Integer Key Type, KSZ = 2 (0 defaults to 2)
- 31-bit Signed Integer Key Type, KSZ = 4 (0 defaults to 4)
- 16-bit Unsigned Binary Key Type, KSZ = 2 (0 defaults to 2)
- 32-bit Unsigned Binary Key Type, KSZ = 4 (0 defaults to 4)

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.9.3 Initialization and Default

Macro takes the form:

```
R$KSZ numeric
```

If there is no initialization macro, KSZ = 0, meaning you must set the field with the $STORE field access macro before any Indexed random record operation.

4.9.4 Comments

RMS-11 uses the KSZ field only for Indexed random record operations with string keys.

```
************
*          *
*    MBC   *
*          *
************
```

4.10 MBC

The one-byte Multi-Block Count (MBC) field contains number of blocks from a disk Sequential file that RMS-11 reads or writes as a single I/O unit while the Record Access Stream is set up (between $CONNECT and $DISCONNECT).

4.10.1 Use

The MBC field should be set for disk Sequential files only; take the default of one for all other files. See also the RMS-11 User's Guide.

Input to:

$CONNECT
: You set the MBC field equal to the number of blocks you want read from or written to the file in each I/O access operation.

4.10.2 Input Values

MINIMUM = 0, meaning one block is the I/O unit

MAXIMUM = 63.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.10.3 Initialization and Default

Macro takes the form:

```
R$MBC numeric
```

If there is no initialization macro, MBC = 0.

4.10.4 Comments

- See the RMS-11 User's Guide for a discussion of the use of MBC in optimization.

- The value in the MBC field determines the size of the I/O buffer used for the file represented by the FAB indicated by the RAB FAB field. See Chapters 1 and 2.

```
************
*          *
*    MBF   *
*          *
************
```

4.11 MBF

The one-byte Multi-Buffer (MBF) field contains the number of buffers* RMS-11 allocates for I/O operations when the Record Access Stream is set up ($CONNECT).

4.11.1 Use

The MBF field should be set only for Indexed files, although RMS-11 uses it for Relative files as well: take the default value of one for Relative files.

Input to:

$CONNECT
: If you want to cache index Root buckets, set the MBF field to a number greater than two to make buffers available (see "Values").

4.11.2 Input Values

MINIMUM = 0, meaning RMS-11 allocates the minimum number of buffers required by the file organization. Sequential and Relative files require one buffer; Indexed files require two buffers.

MAXIMUM = 255

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.11.3 Initialization and Default

Macro takes the form:

```
R$MBF numeric
```

If there is no initialization macro, MBF = 0.

---

*A buffer is one bucket long.

4.11.4 Comments

- The value in the MBF field affects the size of the buffer pool for the file. See Chapters 1 and 2.

- Although RMS-11 does respond to the MBF field for non-Indexed files, the use of a value greater than the minimum (other than the default) is a waste of address space and the use of a macro to set the value.

- See the RMS-11 User's Guide for a discussion of the use of MBF in optimization.

- The value in the MBF field should incorporate the minimum requirements for the Indexed file: add the number of Roots you want cached to the minimum requirements to get the value for the MBF field.

- If the value is:

  -- less than minimum, RMS-11 allocates the minimum number of buffers

  -- more than two for Sequential files, RMS-11 allocates only two buffers

  -- more than is available, RMS-11 allocates as many buffers as possible

```
*************
*           *
*    RAC    *
*           *
*************
```

4.12 RAC

The one-byte Record Access (RAC) field contains an indicator for the access method to be used in a record operation.

4.12.1 Use

Input to:

$FIND/$GET/$PUT
: You set the RAC field to indicate the access method RMS-11 should use to locate the next record position.

4.12.2 Input Values

The RAC field must contain one of the following symbolic values:

RM$KEY
: RMS-11 performs a random record operation, using specifications from the KBF, KSZ, and if necessary KRF and ROP fields. Random access applies only to Relative and Indexed files.

RB$RFA
: RMS-11 performs a record operation using the Record's File Address specified in the RFA field to locate the record position. RFA access applies only to disk files.

RB$SEQ
: RMS-11 performs a sequential record operation.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.12.3 Initialization and Default

Macro takes the form:

R$RAC symbolic

If there is no initialization macro, RAC = RB$SEQ.

```
*************
*           *
*    RBF    *
*           *
*************
```

4.13 RBF

The one-word Record Buffer (RBF) field points to the buffer in the program that contains the
record moved by the record operation.

4.13.1 Use

Input to:

$PUT/$UPDATE
: Regardless of the Record Transfer Mode, you must set the RBF field equal to the address of the first byte of the record you want written to the file. You must also set the RSZ field.

Output from:

$GET
: Regardless of Record Transfer Mode, RMS-11 sets the RBF field equal to the address of the first byte of the record read from the file. The RSZ field is also set.

$PUT
: When your program is writing records to a Sequential file in the Locate Record Transfer Mode, RMS-11 returns an address in the RBF field after each $PUT operation. This address is the location where your program can build the next record. If your program uses this location, it must set the RSZ field to describe the record that begins at the location. However, if your program does not use the address, it must reset both the RBF and RSZ fields.

4.13.2 Input Values

The RBF field must contain a valid address.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.13.3 Initialization and Default

Macro takes the form:

```
R$RBF label
```

If there is no initialization macro, RBF = 0, meaning you must set the field with $STORE before a put or update operation is initiated.

4.13.4 Comments

During the simplest of processes, that is, in Move Record Transfer Mode and the same records are being read and written/updated, the RBF/RSZ and UBF/USZ fields describe the same buffer within your program:

- RBF/RSZ describe the record that RMS-11 moves out of your program to the file.
- UBF/USZ describe the buffer where RMS-11 places each record read from the file.

In anticipation of this common usage, RMS-11 resets RBF after a successful get operation so that you don't have to set it before initiating a $UPDATE macro: when RBF and UBF point to the same buffer, RBF is not changed.

```
*************
*           *
*    RFA    *
*           *
*************
```

4.14 RFA

The three-word Record's File Address (RFA) field contains the file address of the target record for or from a get or find operation or of the record written by a put operation.

4.14.1 Use

Input to:

$FIND/$GET
: If you have set the RAC field to RB$RFA, you set the RFA field equal to the Record's File Address of the record you want found or read.

Output from:

$FIND/$GET
: If the RAC field does not contain RB$RFA, RMS-11 sets the RFA field equal to the Record's File Address of the record found or read.

$PUT/$UPDATE
: RMS-11 sets the RFA field equal to the Record's File Address where it wrote the record specified.

4.14.2 Input Values

If the RAC field contains RB$RFA, the RFA must contain a valid Record's File Address for the file; otherwise, error code ER$RFA.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.14.3 Initialization and Default

Macro takes the form:

```
R$RFA numeric
```

If there is no initialization macro, RFA = 0.

```
************
*          *
*    RHB   *
*          *
************
```

4.15 RHB

The one-word Record Header Buffer (RHB) field points to the buffer in the program containing the fixed control area of Variable-with-Fixed-Control (VFC) records. When RMS-11 gets a record, it separates the fixed control area from the variable portion of the record; it places the fixed data in the buffer indicated by the RHB field and the variable data in the buffer indicated by the UBF field. For put operations, RMS-11 prefixes the data from the RHB buffer to the data in the RBF buffer before it moves the entire record from the program to the I/O buffer and then to the file.

If no RHB buffer is indicated (the RHB field is zero), but the file contains VFC records (FB$VFC in the FAB RFM field), RMS-11 does the following:

- ignores the fixed control area of records read ($GET) from the file
- fills the fixed control area of records written ($PUT) to the file with zeros
- does not change the fixed control area of records during update operations

4.15.1 Use

Input to:

$GET/$PUT/$UPDATE

- If you want the fixed control area of each record read and written properly, set the RHB field equal to the address of a buffer in the program.
- If you do not want the fixed control area of each record read or written, ensure that the RHB field is zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as a buffer: the results are unpredictable.

4.15.2 Input Values

The RHB field must contain either:

- zero (0) = there is no buffer for the fixed control area

- address of the fixed control area buffer. This buffer must be at least as long as the value* in the FAB FSZ field indicates.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.15.3 Initialization and Default

Macro takes the form:

```
R$RHB label
```

If there is no initialization macro, RHB = 0.

4.15.4 Comments

During Locate Mode operations, the variable area data is accessed in the I/O buffer, but the data in the fixed control area is moved to and from the buffer described by the RAB RHB and FAB FSZ fields.

---

*This value is a file attribute established when the file was created and set in the FAB FSZ field by the file open process.

****
*   *
* ROP   *
*   *
****

4.16 ROP

The one-word Record Processing Options (ROP) field contains a bit string indicating optional functions that you want RMS-11 to perform while this Record Access Stream is set up (between $CONNECT and $DISCONNECT).

4.16.1 Use

Input to:

$CONNECT/$DELETE/$FIND/$GET/$PUT
: You set the ROP field to indicate the optional functions you want RMS-11 to perform.

4.16.2 Input Values

The ROP field may contain one or more of the following symbolic values:

RB$ASY
: RMS-11 performs an asynchronous record operation, possibly returning control to your program before the operation is completed. The RAB must have been defined as asynchronous type by the RAB$B macro (see "Allocation" at the beginning of this chapter).

RB$EOF
: RMS-11 sets the context of the stram at the end of the file when the $CONNECT macro is finished; valid for disk Sequential files only. This feature enables you to optimize buffer space: when you are adding records to a Sequential file, you can:

  - disconnect the Record Access Stream when you are not using the file. This operation releases buffer space for use by other operations.
  - connect the Record Access Stream at end-of-file, ready to put another record.

RB$FDL
: RMS-11 uses the Fast Delete procedure; valid for Indexed files only. During $DELETE execution, RMS-11 flags the user data record in the file as being deleted, but it updates Alternate indexes depending on whether duplicates are allowed:

- If duplicates are allowed, the SIDR array entry for the deleted record is NOT flagged as deleted. Thereafter, a find or get operation using that key value for that Alternate index will go to the Primary Level 0 bucket before it detects that the record has been deleted.

- If duplicates are not allowed, the SIDR is removed from the Alternate Level 0 bucket.

RB$KGE RMS-11's search during a find or get operation depends on file organization:

- In Relative files, RMS-11 checks the cell specified by the relative record number in the RAB KBF field; if the cell contains a valid record, the search ends. However, if the cell is empty or contains a deleted record, RMS-11 continues searching sequentially through the cells until it finds a valid record, encounters end-of-file, or exceeds the Maximum Record Number.

- In Indexed files, RMS-11 searches for the first record containing a value in the key specified by the KRF field that is greater than or equal to the value described by the KBF and KSZ fields.

RB$KGT RMS-11's search during a find or get operation depends on file organization:

- In Relative files, RMS-11 starts at the cell with a relative record number one higher than the number in the RAB KBF field and searches sequentially through the cells until it finds a valid record, encounters end-of-file, or exceeds the Maximum Record Number.

- In Indexed files, RMS-11 searches for the first record containing a value in the key specified by the KRF field that is greater than the value described by the KBF and KSZ fields.

RB$MAS indicates to RMS-11 that you intend to insert a series of records at the logical end of an Indexed file. These records must be sorted in ascending order by the Primary Key value, and the lowest Primary Key value in the series, that of the first record, must be higher than any Primary Key value already in the file. To implement Mass Insert, RMS-11 keeps the data bucket locked after the $PUT operation ends and maintains a pointer into the index bucket above the data bucket in the index. This effort enables RMS-11 to rapidly insert records into the file, splitting buckets economically if necessary. The presence of Alternate Keys degrades this performance. Activating Deferred Write (FB$DFW) enhances it.

RB$LOA RMS-11 obeys bucket fill numbers when it is inserting records; valid for Indexed files only. See Allocation XAB DFL and IFL fields and the RMS-11 User's Guide.

RB$LOC RMS-11 uses the Locate Record Transfer Mode for get operations on all file organizations and for put operations for Sequential files only. See the RMS-11 User's Guide.

RB$UIF If RMS-11 encounters a record in the target cell during a put operation, it updates the record rather than returning error code ER$REX; valid for Relative files only.

If you want to specify more than one option, you must concatenate the values using an exclamation point (!).

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

4.16.3 Initialization and Default

Macro takes the form:

```
R$ROP symbolic[!symbolic...]
```

If there is no initialization macro, ROP = 0, meaning:

- Synchronous record operation
- Context in a disk Sequential file after $CONNECT is the first record in the file.
- RMS-11 removes SIDR duplicate array entries from Alternate indexes during delete operations on Indexed files.
- RMS-11 requires that a record's key value be equal to the value specified by the KBF and KSZ fields before it returns the record to the task.
- RMS-11 completely fills Indexed file buckets.
- Move Record Transfer Mode
- If RMS-11 encounters a record in the target cell during a put operation on a Relative file, it returns error code ER$REX.

4.16.4 Comments

Since the ROP field is a bit string, values cannot be added to the field with the $STORE field access macro. You should use the $SET field access macro (see Chapter 7). However, you use the $STORE macro to (re)set all bits in the field.

```
*************
*           *
*    RSZ    *
*           *
*************
```

4.17 RSZ

The one-word Record Size (RSZ) field contains the size, in bytes, of the record indicated by the RBF field.

4.17.1 Use

Input to:

$PUT/$UPDATE
: You set the RSZ field equal to the size of the record in the record buffer indicated by the RBF field. RMS-11 moves only the number of bytes specified.

Output from:

$GET
: RMS-11 sets the RSZ field equal to the length of the record it placed in the buffer indicated by the UBF field.

4.17.2 Input Values

MINIMUM = 1

MAXIMUM = 65535, but maximum record size is also affected by file structure, file organization, and task address space. See the RMS-11 User's Guide.

A value of 0 is valid for only Sequential and Relative files with stream or variable-length records.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.17.3 Initialization and Default

Macro takes the form:

```
R$RSZ numeric
```

If there is no initialization macro, RSZ = 0, meaning you must set the field with the $STORE field access macro before a $PUT operation* is initiated.

NOTE

RSZ is also input to the $UPDATE macro; however, an update operation must be preceded by a get (or find): that operation sets the RSZ field so you don't have to set it before an update operation unless the record size changes.

4.18 STS

The one-word Completion Status Code (STS) field contains a code indicating the success or failure (and type of failure) of the last record operation. See Appendix A for symbolic and octal codes for both success and failure of operations.

4.18.1 Use

Output from:

$CONNECT/$DELETE/$DISCONNECT/$FIND/$GET/$PUT/$UPDATE/$REWIND/$TRUNCATE/$FLUSH/$NXTVOL

RMS-11 sets the STS field equal to a completion status code. You should test this field ($COMPARE field access macro) for a negative value after each record operation, or you can use an error routine that checks the value (see Chapter 8).

4.18.2 Input Values

No input values; see Appendix A for output values.

```
************
*          *
*    STV   *
*          *
************
```

4.19 STV

The one-word Status Value (STV) field contains additional information about some error codes returned in the STS field.

4.19.1 Use

Output from:

$CONNECT/$DELETE/$DISCONNECT/$FIND/$GET/$PUT/$UPDATE/$REWIND/$TRUNCATE/$FLUSH/$NXTVOL

RMS-11 sets the STV field whenever it had additional information to communicate about an error. See Appendix A for the specific errors on which the STV field is used. You should check the STV field for a nonzero value whenever your program detects an error (negative value in STS field).

4.19.2 Input Values

No input values; see Appendix A for output values.

```
*************
*           *
*    UBF    *
*           *
*************
```

4.20 UBF

The one-word User Buffer (UBF) field points to a buffer in your program where RMS-11 should place a record read from the file or that RMS-11 can use during Locate Mode if necessary.

4.20.1 Use

Input to:

$CONNECT
: RMS-11 transfers the address in the UBF field to the RBF field so that your program does not have to treat the first put in Locate Mode on a Sequential file as a special case.

$GET
: Regardless of Record Transfer Mode (RTM), you set the UBF field equal to the address of the first byte of the record buffer you have provided for RMS-11 to store the record it reads from the file:

  - In Move RTM, RMS-11 automatically places the retrieved record in the buffer indicated by UBF.
  - In Locate RTM, RMS-11 normally leaves the retrieved record in the stream's I/O buffer and then sets the RBF field to its address. However, RMS-11 can determine that it cannot perform the record operation in Locate mode (see UG): it then moves the record into the buffer indicated by UBF.

  In either circumstance, if the buffer described by UBF is too small to contain the retrieved record, RMS-11 moves as much of the record as possible into the buffer (the number of bytes in the USZ field), updates the context of the stream, and returns error code ER$RTB

$PUT
: During Locate Mode operations on Sequential files only, UBF must continue to point to a valid record buffer within your program. Normally after a put operation, RMS-11 sets the RBF field to the address of a location in the stream's I/O buffer where your program can build the next record. However, if there is not enough room in the buffer to contain another record (based on the value in the FAB MRS field), RMS-11 changes RBF to UBF, thereby pointing your

program to the user buffer to build the next record. Then, during the next put, RMS-11 fills out the current I/O buffer from the UBF buffer, writes the I/O buffer to the file, and then starts a new I/O buffer with the data left in the UBF buffer, setting RBF back into the I/O buffer.

4.20.2 Input Values

The UBF field must contain the address of a buffer in your program. The buffer should be large enough to contain the largest record in the file.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.20.3 Initialization and Default

Macro takes the form:

```
R$UBF label
```

If there is no initialization macro, UBF = 0.

```
************
*          *
*    USZ   *
*          *
************
```

4.21 USZ

The one-word User Buffer Size (USZ) field contains the size, in bytes, of the buffer indicated by the UBF field.

4.21.1 Use

Input to:

$GET
: You set the USZ field equal to the size of the buffer you are providing to hold a record. RMS-11 reads in only the number of bytes specified.

$PUT
: During Locate Mode operations on Sequential files only. See discussion under "Use" for the UBF field.

4.21.2 Input Values

MINIMUM = 1

MAXIMUM = 65535, but maximum record size is also affected by file structure, file organization, and task address space. See the RMS-11 User's Guide.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

4.21.3 Initialization and Default

Macro takes the form:

```
R$USZ numeric
```

If there is no initialization macro, USZ = 0.

CHAPTER 5

EXTENDED ATTRIBUTE BLOCKS

The initialization macros (this chapter) and the field access macros (Chapter 7) are provided so that you do not have to know the specific position and to a large extent, the size of each field in a XAB. You can, therefore, treat the fields as logical entities. It is also possible that the positions and sizes of the fields will change from release to release of RMS-11.

However, you can determine the position of any field in a XAB as an offset from the XAB's starting address. RMS-11 represents these offset values with symbols in one of the following forms:

- O$fnm

  where fnm is the three-letter name of a one-byte or one-word field; fnm is the name used to reference the field in the initialization and field access macros.

  Example O$STS for the status code field

- O$fnmx

  where fnm is the name of a multiword field; fnm is the name used to reference the field in the initialization and field access macros.

  x is a number associated with an individual word in the field, from 0 through the end of the field.

  Example O$SIZ0 for the size of the first segment in a key as the most significant word

The values of these symbols can be found in the symbol table of an assembly listing file for any module containing the XAB.

5.1 Allocation Extended Attribute Block

Each Allocation XAB describes an area of a file. An area is a portion of an RMS-11 file that is treated as a separate entity for purposes of:

Initial allocation
Extension
Placement
Bucket size

See also the RMS-11 User's Guide.

Sequential and Relative files are always composed of a single area (Area 0), while Indexed files should contain more than one area.

Allocation XABs enable you to control:

- The physical placement of an RMS-11 file of any organization on a disk volume:

  You provide one (for any type of file) or more (for Indexed files only) Allocation XABs linked to the FAB you use to create the file. Within each XAB are two fields:

  Alignment Block Type (ALN)
  Allocation Starting Point (LOC)

  If the ALN field is not zero when the $CREATE macro is executed, RMS-11 uses the values in the two fields to direct the placement of the first block of the area on the device specified; the XAB ALQ field controls the size of the initial allocation.

  NOTE

  On RSTS/E, the File Processor allows RMS-11 to place only Area 0. RMS-11 allocates all other areas as extents of Area 0. See also comments for the AOP field in this section.

- The internal structure of an Indexed file:

  Whether or not you use placement control, you can use Allocation XABs to define multiple areas within an Indexed file, assign the index and data levels of the various indexes to particular areas, and vary the size of buckets for each area.

Allocation XABs are completely optional. If you do not require either placement control or an optimized Indexed file, do not include Allocation XABs in your XAB chain; RMS-11 uses the following defaults:

- RMS-11 creates the file as a single area, using the value in the FAB ALQ for the initial allocation quantity. The default extension quantity is derived from the FAB DEQ field. The operating system

completely controls the physical placement of the file on the disk volume.

- For Indexed files, RMS-11 sets up a single bucket size and stores buckets of all types (index, SIDR, and user data record) in the single area of the file.

If you do use Allocation XABs, they are used as:

Input to $CREATE and $EXTEND
: Before the initiation of a $CREATE macro, all Allocation XABs must be set up and included in the XAB chain linked in dense ascending order by AID field: the XAB for Area 0 must be the first Allocation XAB in the chain; the XAB for Area 1 must be next; and so on. The NXT field for Area 0's XAB must point to Area 1's XAB, Area 1's XAB NXT field must point to Area 2's XAB, and so on.

Output from $DISPLAY, $EXTEND, and $OPEN
: If you want one or more area definitions, link a sufficient number of Allocation XABs. RMS-11 sets the appropriate fields to the values established when the file was created. The Allocation XABs must be adjoining, linked in ascending order, but they do not have to be densely ordered. Therefore, you can link Allocation XABs for Areas 1 and 3 only, for example, if descriptions of those areas are the only ones you want.

NOTE

Since RMS-11 creates a file with areas by allocating Area 0 first and then extending the file for each additional area, you can't have a contiguous file with multiple areas on RSTS/E. However, after you create the file with areas, you can use PIP to copy it into a contiguous space.

Table 5-1: Allocation Extended Attribute Block Fields

Field Name	Field Size	Default	Description
AID	1B	0	Area id number
ALN	1B	0	Alignment block type
ALQ	2W	0	Allocation quantity
AOP	1B	0	Allocation options
BKZ	1B	1 record	Area bucket size
BLN	1B	N/A	All XAB length
COD	1B	N/A	XAB type
DEQ	1W	0	Area extension quantity
LOC	2W	0	Number of block specified by ALN
NXT	1W	0	Next XAB
VOL	1W	0	Not used

```
*************
*           *
*    AID    *
*           *
*************
```

5.1.1 AID

The one-byte Area Identification Number (AID) field identifies the area described by the XAB.

5.1.1.1 Use

Input to:

$CREATE/$EXTEND
: You set the AID field to tell RMS-11 the area defined by the XAB.

$DISPLAY/$OPEN
: You set the AID field to tell RMS-11 the area whose description it should set into the fields of the XAB.

Additionally, RMS-11 uses the AID field to check the sequencing of the Allocation XABs in the chain before continuing with the file operation; if the XABs are out of sequence, RMS-11 terminates the operation.

5.1.1.2 Input Values

MINIMUM = 0*

MAXIMUM = 254

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.1.3 Initialization and Default - Macro takes the form:

```
X$AID numeric
```

If there is no initialization macro, AID = 0.

---

*Zero is the only value acceptable for Sequential and Relative files.

```
************
*          *
*    ALN   *
*          *
************
```

5.1.2 ALN

The one-byte Alignment Block Type (ALN) field determines whether placement control is used in the allocation of blocks in this area by the file operation.

5.1.2.1 Use

Input to:

$CREATE

- If you want placement control over the initial allocation of the area, you set the ALN field to indicate that the number in the LOC field is one of the following:

  Device Cluster Number
  Logical Block Number
  Virtual Block Number
  Cylinder number on VAX

  The AOP field also affects placement control.

- If you do not want placement control, you set the ALN field to zero.

$EXTEND

- If you want placement control over the explicit extension of the area, you set the ALN field to indicate that the block number in the LOC field is either a Logical Block Number or a Virtual Block Number. The AOP field also affects placement control.

- If you do not want placement control, you set the ALN field to zero.}

Output from:

$DISPLAY/$OPEN
RMS-11 sets the ALN field equal to the value specified when the file was created.

5.1.2.2 Input Values - The ALN field must contain one of the following:

zero — no placement control

XB$LBN — RMS-11 requests the operating system to place the first block of the specified allocation on or near the logical block / device cluster indicated by the number in the LOC field.

XB$VBN — RMS-11 requests the operating system to place the first block of the specified allocation near the virtual block of the file indicated by the LOC field; not valid during $CREATE when the AID field is zero, that is, for Area 0.

XB$CYL — RMS-11 requests the operating system to place the first block of the specified allocation at the first block of a cylinder as indicated by the LOC field. VAX only

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.2.3 Initialization and Default - Macro takes the form:

```
X$ALN symbolic
```

If there is no initialization macro, ALN = 0.

5.1.2.4 Comments - On RSTS/E, you can specify location information for Area 0 only. If RMS-11 detects nonzero values in the ALN and LOC fields in other Allocation XABs, it returns the error code ER$ALQ.

```
*************
*           *
*    ALQ    *
*           *
*************
```

5.1.3 ALQ

The two-word Allocation Quantity (ALQ) field contains the size of the area represented by the XAB, in blocks.

5.1.3.1 Use

Input to:

$CREATE
: You set the ALQ field equal to the number of blocks to be allocated in the initial extent of the area.

$EXTEND
: You set the ALQ field equal to the number of blocks to be added to the area.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the ALQ field equal to the number of blocks unused in the current extent of the area (see "Comments"). RMS-11 obtains this information from the file Prologue.

5.1.3.2 Input Values

$CREATE

MINIMUM = 0, meaning that this area will only be as long as necessary; that is, Area 0 will contain the file Prologue and all other areas will be allocated with no blocks.

MAXIMUM = number of free blocks on the device containing the file.

$EXTEND

MINIMUM = 1, meaning that one block should be added to the area (although a zero extent is possible, it is not logical)

MAXIMUM = number of free blocks on the device containing the area.

RMS-11 rounds values up to a multiple of area bucket size.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.3.3 Initialization and Default - Macro takes the form:

```
X$ALQ numeric
```

If there is no initialization macro, ALQ = 0.

5.1.3.4 Comments

- When Allocation XAB(s) are linked to a FAB, create and extend operations ignore the FAB ALQ field, using the XAB ALQ field(s) for the particular area(s). If you include only one Allocation XAB, for Area 0, RMS-11 uses the ALQ field in the XAB instead of the FAB for the size of Area 0, which is essentially the whole file.
- The current extent of an area is the number of blocks that were last added to the area, via either automatic or explicit extension. It is, therefore, not necessarily the total size of the area.

```
************
*          *
*    AOP   *
*          *
************
```

5.1.4 AOP

The one-byte Allocation Options (AOP) field contains a bit string indicating the type of allocation RMS-11 makes during the file operation.

5.1.4.1 Use

Input to $CREATE/$EXTEND:
You set the AOP field to indicate how exact you require the actual alignment of the area's allocation with the value specified in the LOC field.

Output from $DISPLAY/$OPEN:
RMS-11 sets the AOP field equal to the value specified when the file was created.

5.1.4.2 Input Values - The AOP field may contain one or both of the following symbolic values:

XB$HRD The operation fails unless the allocation starts exactly at the Device Cluster /Logical Block Number specified by the LOC field; valid only if the ALN field contains XB$LBN.

XB$CTG The operation fails unless the allocation is contiguous within itself.

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

5.1.4.3 Initialization and Default - Macro takes the form:

```
X$AOP symbolic[!symbolic]
```

If there is no initialization macro, AOP = 0, meaning:

- Allocation is made as close to the requested block as possible.
- Allocation is not necessarily contiguous with the current extent of the area.

```
************
*          *
*    BKZ   *
*          *
************
```

5.1.5 BKZ

The one-byte Bucket Size (BKZ) field contains the size of a bucket, in disk blocks, in the area represented by the XAB.

5.1.5.1 Use - RMS-11 uses the BKZ field for Relative and Indexed files.

Input to:

$CREATE
: You set the BKZ field equal to the number of disk blocks in a bucket for the area to be created.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the BKZ field equal to the bucket size established for the area when the file was created. RMS-11 obtains the information from the file Prologue.

5.1.5.2 Input Values -

MINIMUM = 0, meaning that RMS-11 calculates a size so that a bucket contains at least one record.

MAXIMUM = number of blocks allowed by the operating system (specifying a larger value results in error code ER$BKZ):

```
    IAS = 32 blocks
RSTS/E = 15 blocks
RSX-11M = 32 blocks
```

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

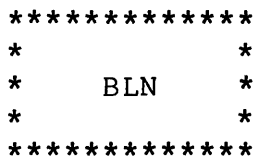
5.1.5.3 Initialization and Default - Macro takes the form:

```
X$BKZ numeric
```

If there is no initialization macro, BKZ = minimum number of blocks to contain one record.

5.1.5.4 Comments -

- When Allocation XABs are linked to a FAB, RMS-11 ignores the FAB BKZ field, using the XAB BKZ fields for the particular areas.
- You gain no particular advantage from different bucket sizes for different areas.



5.1.6 BLN

The XAB$B macro sets the one-byte Block Length (BLN) field to the Extended Attribute Block length.

CAUTION

DO NOT CHANGE THE BLN FIELD.

5.1.6.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.

****
*   *
* COD   *
*   *
****

5.1.7 COD

The XAB$B macro sets the one-byte Code (COD) field to the Extended Attribute Block type, with the symbolic value of XB$ALL.

CAUTION

DO NOT CHANGE THE COD FIELD.

5.1.7.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the COD field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$COD error code.

```
************
*          *
*    DEQ   *
*          *
************
```

5.1.8 DEQ

The one-word Default Extension Quantity (DEQ) field contains the number of blocks RMS-11 requests whenever it automatically extends the area (opposed to an explicit extend operation). RMS-11 requests additional blocks from the operating system* whenever a task attempts a put or update operation and there is not enough room in the area represented by the XAB to complete the operation.

5.1.8.1 Use - RMS-11 uses the DEQ field for Indexed files only.

Input to:

$CREATE
: You set the DEQ field equal to the number of blocks that should be requested in each automatic extension of the area to be created.

$OPEN
: You set the DEQ field equal to the number of blocks to be used by RMS-11 as a temporary default extension quantity until the area is closed; the area attribute is not changed.

Output from:

$OPEN
: If the DEQ field is zero, RMS-11 sets it equal to the number of blocks established as the default extension quantity when the area was created.

5.1.8.2 Input Values - Default extension quantity should be a multiple of bucket size; RMS-11 rounds it up to the nearest multiple anyway before it sends the extension request to the operating system.

MINIMUM = 0, meaning that whenever RMS-11 has to extend the area, it will request the minimum blocks necessary to complete the current operation, that is, one bucket.

MAXIMUM = 65,535 blocks

---

*Automatic extension can fail;see "Comments."

RMS-11 rounds value up to a multiple of area bucket size.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.8.3 Initialization and Default - Macro takes the form:

```
X$DEQ numeric
```

If there is no initialization macro, DEQ = 0.

5.1.8.4 Comments

- When Allocation XABs are linked to a FAB, RMS-11 uses the XAB DEQ field whenever an area has to be extended automatically, unless that field is zero; in that case, RMS-11 uses the value in the FAB DEQ field.

- Automatic extension can fail for at least the following reasons; there might be others:

  -- Operating system is RSTS/E and file is:

  ```
  contiguous;  error code = ER$PRV
  shared;  error code = ER$PRV
  ```

  -- No more room is available on the storage device; error code = ER$FUL

- On the RSTS/E operating system, area allocation is done in clusters; see the RMS-11 User's Guide for a discussion of the interaction between clusters and default extension quantity.

- See the RMS-11 User's Guide for a discussion of default extension quantity and its optimization.

```
************
*          *
*    LOC   *
*          *
************
```

5.1.9 LOC

The two-word Allocation Starting Point (LOC) field contains a number interpreted by RMS-11 as the Device Cluster, Logical Block, Virtual Block, or cylinder number (depending on the value of the ALN field) where the allocation specified by the operation should start. The ALQ field specifies the allocation quantity and the AOP field indicates how definite the placement is.

5.1.9.1 Use

Input to:

$CREATE/$EXTEND
: If you specify a nonzero value for the ALN field, you must set the LOC field equal to the appropriate number (depending on the value in the ALN field) where you want the specified allocation to begin.

5.1.9.2 Input Values - The LOC field must contain one of the following:

- any value* if the ALN field is zero
- numeric value within the range:

  MINIMUM = 0; if the ALN field contains XB$VBN, RMS-11 requests placement near the current end of the file, unless AID is zero.
  MAXIMUM = number of free blocks / clusters on the device containing the file, if the ALN field contains XB$LBN

  = number of cylinders on the device containing the file, if the ALN field contains XB$CYL

  = number of blocks currently in the file, if the ALN field contains XB$VBN

- the numeric value -1 if the ALN field contains XB$CYL; RMS-11 requests placement at the first block of any cylinder

---

*Recommended value is the default, zero.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.9.3 Initialization and Default - Macro takes the form:

```
X$LOC numeric
```

If there is no initialization macro, LOC = 0.

```
*************
*           *
*    NXT    *
*           *
*************
```

5.1.10 NXT

The one-word Next XAB (NXT) field, if not zero, points to another separately allocated XAB and thereby indicates to RMS-11 that there is at least one more XAB in the chain connected to the FAB via the FAB XAB field. See "Linking and Ordering XABs" at the beginning of this chapter.

5.1.10.1 Use

Input to:

$CREATE/$DELETE/$EXTEND/$OPEN

- If you have allocated another XAB and want RMS-11 to use it, you set the NXT field equal to the address of that XAB.
- If you have not allocated an XAB and/or you want the XAB chain to end here, set the NXT field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns the error code ER$XAB.

5.1.10.2 Input Values - The NXT field must contain either:

- zero (0) = there are no more XABs in this chain
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.1.10.3 Initialization and Default - Macro takes the form:

X$NXT label

If there is no initialization macro, NXT = 0.

5.1.11 VOL

The one-word Relative Volume Number (VOL) field is currently not used by RMS-11.

5.1.11.1 Use - NONE

5.1.11.2 Input Values - NONE

5.1.11.3 Initialization and Default - NONE

5.1.11.4 Examples - NONE

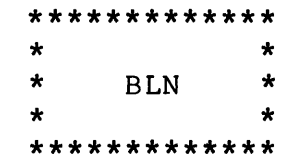
5.1.11.5 Comments - NONE

5.2 Date Extended Attribute Block

The Date XAB contains date and time information about the creation and revision of the file, as well as an RMS-11 revision number. The information is output only by RMS-11 during a $DISPLAY or $OPEN operation; RMS-11 obtains the data from the file attributes.

Table 5-2: Date Extended Attribute Block Fields

Field Name	Field Size	Default	Description
BLN	1B	N/A	DAT XAB length
CDT	4W	N/A	Creation date-time
COD	1B	N/A	XAB type
NXT	1W	0	Next XAB
RDT	4W	N/A	Revision date-time
RVN	1W	N/A	RMS-11 revision number



5.2.1 BLN

The XAB$B macro sets the one-byte Block Length (BLN) field to the Extended Attribute Block length.

CAUTION

DO NOT CHANGE THE BLN FIELD.

5.2.1.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.

```
*************
*           *
*    CDT    *
*           *
*************
```

5.2.2 CDT

The four-word Creation Date (CDT) field contains the date and time the file associated with the linked FAB was created.

5.2.2.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the CDT field equal to the creation date it finds in the file attributes.

5.2.2.2 Output Values - The value RMS-11 puts into the CDT field is an eight-byte binary number representing the number of hundreds of nanoseconds that elapsed between November 17, 1858*, and the time the file was created.

This method of storing dates is standard for DIGITAL operating systems, although it has not been implemented in IAS, RSTS/E, or RSX-11M yet**. RMS-11 translates the dates these operating systems do maintain into the 64-bit format before storing it in the Date XAB. The resultant date is only as accurate as the operating system date:

IAS	accurate to the second
RSTS/E	accurate to the minute
RSX-11M	accurate to the second

Appendix C contains a MACRO-11 routine for converting this 8-byte binary number to ASCII characters; this is the same routine used by RMSDSP (see the RMS-11 User's Guide).

---

*The Smithsonian Astrophysical Base Date, celebrating the day the first photographic plate was exposed at the Harvard Smithsonian Observatory.

**It has been implemented in the VAX/VMS System.

****
*   *
* COD   *
*   *
****

5.2.3 COD

The XAB$B macro sets the one-byte Code (COD) field to the Extended Attribute Block type, with the symbolic value of XB$DAT.

CAUTION

DO NOT CHANGE THE COD FIELD.

5.2.3.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the COD field. If this field does not contain the proper code, RMS-11 terminates the operation with a ER$COD error code.

5.2.4 NXT

The one-word Next XAB (NXT) field, if not zero, points to another separately allocated XAB and thereby indicates to RMS-11 that there is at least one more XAB in the chain connected to the FAB via the FAB XAB field. See "Linking and Ordering XABs" at the beginning of this chapter.

5.2.4.1 Use

Input to:

$CREATE/$DELETE/$EXTEND/$OPEN

- If you have allocated another XAB and want RMS-11 to use it, you set the NXT field equal to the address of that XAB.
- If you have not allocated an XAB and/or you want the XAB chain to end here, set the NXT field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns error code ER$XAB.

5.2.4.2 Input Values - The NXT field must contain either:

- zero (0) = there are no more XABs in this chain
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.2.4.3 Initialization and Default - Macro takes the form:

X$NXT label

If there is no initialization macro, NXT = 0.

```
************
*          *
*    RDT   *
*          *
************
```

5.2.5 RDT

The four-word Revision Date (RDT) field contains the date and time the file associated with the linked FAB was last updated, that is, the date and time the last write record operation* was executed against the file.

5.2.5.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the RDT field equal to the revision date it finds in the file header.

5.2.5.2 Output Values - The value RMS-11 puts into the CDT field is an eight-byte binary number representing the number of hundreds of nanoseconds that elapsed between November 17, 1858**, and the time the file was created.

This method of storing dates is standard for DIGITAL operating systems, although it has not been implemented in IAS, RSTS/E, or RSX-11M yet***. RMS-11 translates the dates these operating systems do maintain into the 64-bit format before storing it in the Date XAB. The resultant date is only as accurate as the operating system date:

IAS	accurate to the second
RSTS/E	accurate to the day
RSX-11M	accurate to the second

Appendix C contains a MACRO-11 routine for converting this 8-byte binary number to ASCII characters; this is the same routine used by RMSDSP (see the RMS-11 User's Guide).

---

*Includes RMS-11 delete, put, and update operations and any non-RMS-11 operations, such as FCS, BASIC+, and utilities (PIP and so on). Since higher-level language statements eventually break down to RMS-11 macros, the revision date is valid no matter the source language of the program doing the file revision.

**The Smithsonian Astrophysical Base Date, celebrating the day the first photographic plate was exposed at the Harvard Smithsonian Observatory.

***It has been implemented in the VAX/VMS System.

```
************
*          *
*    RVN   *
*          *
************
```

5.2.6 RVN

RMS-11 returns valid data to the RVN field only on IAS and RSX-11M operating systems. On RSTS/E systems, this field is always zero. The one-word Revision Number (RVN) field indicates the number of times the file has been opened for write operations by RMS-11* and non-RMS-11 tasks since it was created (that open is therefore not counted).

5.2.6.1 Use

Output from:

$DISPLAY/$OPEN
RMS-11 sets the RVN field equal to the revision number it finds in the file header.

5.2.6.2 Input Values - The value stored in the RVN field is a simple binary number, easily translated to octal or decimal (see subroutines in the sample program in Appendix B).

---

*If the FAB FAC field contains FB$DEL, FB$PUT, FB$UPD, and/or FB$WRI, the operating system increments the revision number in the file header during an open operation.

5.3 Key Extended Attribute Block

A Key XAB describes one key of an Indexed file. This includes specifications for the key field (type, position, length) as well as the index structure supporting that key. See the RMS-11 User's Guide.

Key XABs are used as:

Input to $CREATE

Before the initiation of a $CREATE macro, if the FAB ORG field contains FB$IDX, you must link at least one Key XAB to the FAB to define the Primary Key. You must also use one Key XAB to define each additional key you want the file to have. If you specify more than one Key XAB, they must be:

- linked in densely ascending order by REF field; that is, the Primary Key XAB is the first Key XAB to occur in the chain, the First Alternate Key XAB next, and so on
- adjoining in the XAB chain; that is, the Primary Key XAB NXT field points to the First Alternate Key XAB, the First Alternate Key XAB points to the Second Alternate Key XAB, and so on

Output from $DISPLAY and $OPEN:

If you want one or more key definitions, link in a sufficient number of key XABs. RMS-11 sets the appropriate fields to the values established when the file was created. If there is more than one, the Key XABs must be:

- linked in ascending order by REF field, but not necessarily densely; for example, if you only want descriptions of the Second and Fourth Alternate Keys, you must link only the XABs for those keys, in that order
- adjoining in the XAB chain; that is, the first Key XAB in the chain must point to the second one, and so on

NOTE

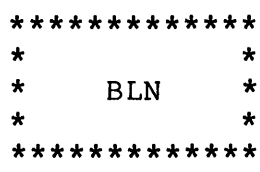
Since Key XABs describe both keys and their indexes, the terms:

"key described by the XAB"
"index described by the XAB"

are used in the following field descriptions; they both refer to the same XAB. The value in the REF field determines which key/index is described.

Table 5-3: Key Extended Attribute Block Fields

Field Name	Field Size	Default	Description
BLN	1B	N/A	KEY XAB length
COD	1B	N/A	XAB type
DAN	1B	0	Level 0 area number
DBS	1B	0	Data bucket size
DFL	1W	0	Level 0 fill number
DTP	1B	0	Key data type
DVB	4W	0	VBN of first data bucket
FLG	1B	0	Key characteristics
IAN	1B	0	Levels 2+ area number
IBS	1B	0	Index bucket size
IFL	1W	0	Levels 2+ fill number
KNM	1W	0	Pointer to key name
LAN	1B	0	Level 1 area number
LVL	1B	0	Index level of root
MRL	1B	0	Minimum length record size
NSG	1B	0	Number of key segments
NUL	1B	0	Null key value
NXT	1W	0	Next XAB
POS	8W	0	Key position(s) in record
REF	1B	0	Key described by XAB
RVB	2W	0	VBN of index root
SIZ	8W	0	Size(s) of key segments
TKS	1B	0	Total key size



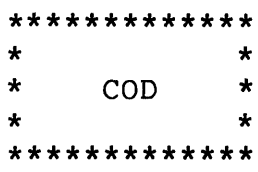
5.3.1 BLN

The XAB$B macro sets the one-byte Block Length (BLN) field to the Extended Attribute Block length.

CAUTION

DO NOT CHANGE THE BLN FIELD.

5.3.1.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.



5.3.2 COD

The XAB$B macro sets the one-byte Code (COD) field to the Extended Attribute Block type, with the symbolic value of XB$KEY.

CAUTION

DO NOT CHANGE THE COD FIELD.

5.3.2.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the COD field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$COD error code.

```
************
*          *
*    DAN   *
*          *
************
```

5.3.3 DAN

The one-byte Data Area Number (DAN) field contains the number of the file area in which Level 0* of the index described by this XAB is (or will be) stored.

5.3.3.1 Use

Input to:

$CREATE

- If you are not creating a multi-area file**, that is, no or one Allocation XAB, ensure that the DAN field is zero (the default).
- If you are creating a multi-area file, that is, there are multiple Allocation XABs also linked to the FAB, set the DAN field to the number of the area in which you want the data level of this index (set by REF field) stored. This number must be the same as the AID field of one of the Allocation XABs.

Output from:

$DISPLAY/$OPEN

RMS-11 sets the DAN field equal to the number of the area in which Level 0 of the index indicated by the REF field is stored.

5.3.3.2 Input Values

MINIMUM = 0, meaning Area 0**

MAXIMUM = 254, meaning the maximum possible area that may be allocated; however, the maximum within each program is restricted to the highest number in an Allocation XAB AID field

---

*The data level (Level 0) of the Primary index contains your data records while Level 0 of Alternate indexes contain Secondary Index Data Records that point into the Primary index Level 0. See the RMS-11 User's Guide.

**If there are no Allocation XABs, the entire file is in Area 0.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.3.3 Initialization and Default - Macro takes the form:

```
X$DAN numeric
```

If there is no initialization macro, DAN = 0.

5.3.4 DBS

The one-byte Data Bucket Size (DBS) field contains the number of blocks in a bucket in the file area that contains Level 0 of the index described by this XAB.

5.3.4.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the DBS field equal to the bucket size of the area established for Level 0 of this index when the file was created.

5.3.4.2 Output Values

MINIMUM = 1

MAXIMUM = number of blocks allowed by the operating system

```
*************
*           *
*    DFL    *
*           *
*************
```

5.3.5 DFL

The one-word Data Bucket Fill Number (DFL) contains the number of bytes in each bucket of this index's Level 0 that should be used to store data if the RAB ROP field contain RB$LOA when the Record Access Stream is set up ($CONNECT); if RB$LOA is not set, RMS-11 ignores the fill number. See the RMS-11 User's Guide for a discussion of fill numbers.

5.3.5.1 Use

Input to:

$CREATE
: You set the DFL field equal to the number of bytes you want filled in each Level 0 bucket when RMS-11 is honoring fill numbers.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the DFL field equal to fill number established for Level 0 of the index designated by the REF field when the file was created.

5.3.5.2 Input Values

MINIMUM = 0, meaning the buckets should be completely filled regardless of value in RAB ROP field

= half of the bytes in the bucket; this is a logical minimum, meaning you can specify less, but RMS-11 ignores it, supplying its own 50% figure

MAXIMUM = total number of bytes in a bucket less one record (including overhead); this is a logical maximum; the physical maximum (total number of bytes in a bucket) can be achieved by specifying zero

You can figure the number of bytes in a bucket as follows:

```
NRBKT = 512*BKS
```

where BKS is the bucket size, expressed in blocks.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.5.3 Initialization and Default - Macro takes the form:

```
X$DFL numeric
```

If there is no initialization macro, DFL = 0.

```
*************
*           *
*    DTP    *
*           *
*************
```

5.3.6 DTP

The one-word Key Data Type (DTP) field contains a bit string indicating the data type of the key described by this XAB.

CAUTION

If you are currently using RMS-11K V1.5 and want to use the DTP field, you must reassemble your program using the RMS-11K V1.8 RMSLIB.MLB.

5.3.6.1 Use

Input to:

$CREATE
: You set the DTP field to indicate the data type of the key you have defined with the POS and SIZ field.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the DTP field to indicate the data type established for the key indicated by the REF field when the file was created.

5.3.6.2 Input Values - The DTP field must contain one of the following symbolic values:

XB$BN2 Each 16-bit unsigned binary key requires two bytes.

XB$BN4 Each 32-bit unsigned binary key requires four bytes.

XB$IN2 Each 15-bit signed integer key requires two bytes.

XB$IN4 Each 31-bit signed integer key requires four bytes.

XB$PAC A packed decimal key has two decimal digits stored in each byte.

XB$STG Each character of a string key is stored as a binary value.

A value can be set in the field with the initialization macro shown

below or with the $SET field access macro (see Chapter 7).

5.3.6.3 Initialization and Default - Macro takes the form:

X$DTP symbolic

If there is no initialization macro, DTP = XB$STG.

```
*************
*           *
*    DVB    *
*           *
*************
```

5.3.7 DVB

The two-word Data Virtual Block Number (DVB) field specifies the number of the file's virtual block that contains the first block of the first* bucket in Level 0 of the index described by this XAB.

5.3.7.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the DVB field equal to the number of the virtual block containing the first data from this index in the file.

5.3.7.2 Output Values

MINIMUM = first data block in the file (after the Prologue) if the index is Primary

MAXIMUM = last block in the file

---

*Based on physical ordering from the beginning of the file.

```
**************
*            *
*     FLG    *
*            *
**************
```

5.3.8 FLG

The one-word Flags (FLG) field contains a bit string indicating characteristics of the key described by the XAB.

5.3.8.1 Use

Input to:

$CREATE
: You set the FLG field to indicate the characteristics you want associated with the key described by the REF field.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the FLG field to indicate the key characteristics established when the file was created.

5.3.8.2 Input Values - The FLG field may contain one or more of the following symbolic values:

XB$CHG
: the value in the key field described by this XAB may be changed during an update operation; valid only for Alternate Keys (REF field greater than 0) and in conjunction with XB$DUP

XB$DUP
: more than one record within the file may have the same value in the key field described by this XAB

XB$NUL
: if the string key described by this XAB is completely filled with the character in the NULL field or the integer, binary, or packed decimal key is zero, RMS-11 makes no entry in the index for the record; valid for Alternate Keys only

RMS-11 restricts the combination of XB$CHG and XB$DUP as follows:

	Combination			
Key Type	XB$CHG + XB$DUP	XB$CHG + Default	Default + XB$DUP	Default + Default
Primary	Error	Error	Allowed	Allowed
Alternate	Allowed	Error	Allowed	Allowed

A value can be set in the field with the initialization macro shown below or with the $SET field access macro (see Chapter 7).

5.3.8.3 Initialization and Default - Macro takes the form:

```
X$FLG symbolic[!symbolic[!symbolic]]
```

If there is no initialization macro, FLG = 0, meaning:

- For Primary Key, duplicate keys are not allowed.
- For Alternate Keys
  - -- duplicate keys allowed
  - -- key values may change during update operation
  - -- no null key value

5.3.8.4 Comments

- At all times, for Primary Keys:
  - -- key values cannot change during update operation
  - -- no null key values
- Since the FLG field is a bit string, values cannot be added to the field with the $STORE field access macro. You should use the $SET field access macro (see Chapter 7). However, you use the $STORE macro to (re)set all bits in the field.

```
************
*          *
*    IAN   *
*          *
************
```

5.3.9 IAN

The one-byte Index Area Number (IAN) field contains the number of the file area in which Levels 2+ of the index described by this XAB are (or will be) stored.

5.3.9.1 Use

Input to:

$CREATE

- If you are not creating a multi-area file*, that is, no or one Allocation XAB, ensure that the IAN field is zero (the default).
- If you are creating a multi-area file, that is, there are multiple Allocation XABs also linked to the FAB, set the IAN field to the number of the area in which you want the index levels two and up of this index (set by REF field) stored. This number must be the same as the AID field of one of the Allocation XABs.

Output from:

$DISPLAY/$OPEN

RMS-11 sets the IAN field equal to the area in which Levels 2+ of the index indicated by the REF field are stored.

5.3.9.2 Input Values

MINIMUM = 0, meaning Area 0*

MAXIMUM = 254, meaning the maximum possible area that may be allocated; however, the maximum within each program is restricted to the highest number in an Allocation XAB AID field

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

---

*If there are no Allocation XABs, the entire file is in Area 0.

5.3.9.3 Initialization and Default - Macro takes the form:

X$IAN numeric

If there is no initialization macro, IAN = 0.

```
*************
*           *
*    IBS    *
*           *
*************
```

5.3.10 IBS

The one-byte Index Bucket Size (IBS) field contains the number of blocks in a bucket in the one or two file areas that contain Levels 1+ of the index described by this XAB.

5.3.10.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the IBS field equal to the bucket size of the area(s) established for Levels 1+ of this index when the file was created.

5.3.10.2 Output Values

MINIMUM = 1

MAXIMUM = number of blocks allowed by the operating system

```
************
*          *
*    IFL   *
*          *
************
```

5.3.11 IFL

The one-word Index Bucket Fill Number (IFL) contains the number of bytes in each bucket of this index's Levels 1+ that should be used to store index information if the RAB ROP field contains RB$LOA when the Record Access Stream is set up ($CONNECT); if RB$LOA is not set, RMS-11 ignores the fill number. See the RMS-11 User's Guide for a discussion of fill numbers.

5.3.11.1 Use

Input to:

$CREATE
: You set the IFL field equal to the number of bytes you want filled in each Levels 1+ bucket when RMS-11 is honoring fill numbers.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the IFL field equal to fill number established for Levels 1+ of the index designated by the REF field when the file was created.

5.3.11.2 Input Values

MINIMUM = 0, meaning the buckets should be completely filled regardless of the value in the RAB ROP field

= half of the bytes in the bucket; this is a logical minimum, meaning you can specify less, but RMS-11 ignores it, supplying its own 50% figure

MAXIMUM = total number of bytes in a bucket less one record (including overhead); this is a logical maximum; the physical maximum (total number of bytes in a bucket) can be achieved by specifying zero

You can figure the number of bytes in a bucket as follows:

```
NRBKT = 512*BKS
```

where BKS is the bucket size, expressed in blocks.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.11.3 Initialization and Default - Macro takes the form:

```
X$IFL numeric
```

If there is no initialization macro, IFL = 0.

```
************
*          *
*    KNM   *
*          *
************
```

5.3.12 KNM

The one-word Key Name Address (KNM) field points to an ASCII string that is associated with the key described by this XAB. RMS-11 does not use this field for processing, but does store it in the file Prologue during a create operation and outputs it during $DISPLAY and $OPEN. The RMSDSP utility also prints out this string (see the RMS-11 User's Guide).

5.3.12.1 Use

Input to:

$CREATE
: If you want to associate a name or other identification with the key described by this XAB, set the KNM field equal to the address of that name.

$DISPLAY/$OPEN

- If you do not want to provide a 32-byte buffer to hold the key name string, you set the KNM field to zero; RMS-11 does not output the key name.
- If you are providing a buffer for the key name string, you set the KNM field equal to the buffer's address.

5.3.12.2 Input Values - The KNM field must contain either:

- zero (0) = no ASCII key name string
- address of an ASCII string; RMS-11 reads and writes 32 bytes starting with this location:
  - -- during $CREATE, be sure that you have valid data (if only blanks) for that length
  - -- during $DISPLAY and $OPEN, be sure that the indicated buffer is at least 32 bytes long or the key name will overlay other data or instructions

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.12.3 Initialization and Default - Macro takes the form:

```
X$KNM label
```

If there is no initialization macro, KNM = 0.

```
************
*          *
*    LAN   *
*          *
************
```

5.3.13 LAN

The one-byte Lowest Index Level Area Number (LAN) field contains the number of the file area in which Level 1 of the index described by this XAB is (or will be) stored.

5.3.13.1 Use

Input to:

$CREATE

- If you are not creating a multi-area file*, that is, no or one Allocation XAB, ensure that the LAN field is zero (the default).
- If you are creating a multi-area file, that is, there are multiple Allocation XABs also linked to the FAB, set the LAN field to the number of the area in which you want the index level one of this index (set by REF field) stored. This number must be the same as the AID field of one of the allocation XABs.

Output from:

$DISPLAY/$OPEN
RMS-11 sets the LAN field equal to the area in which Level 1 of the index indicated by the REF field is stored.

5.3.13.2 Input Values

MINIMUM = 0, meaning the same area as the rest of the index (Levels 2+), not necessarily Area 0

= 1, lowest area number that can be specifically identified

MAXIMUM = 254, meaning the maximum possible areas that may be allocated; however, the maximum within each program is restricted to the highest number in an Allocation XAB AID field

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.13.3 Initialization and Default - Macro takes the form:

X$LAN numeric

If there is no initialization macro, LAN = 0.

5.3.13.4 Comments - The bucket size for the area containing Level 1 of an index must be the same as the bucket size for the area containing the rest of the index (Levels 2+).

5.3.14 LVL

The one-byte Root Level (LVL) field contains the level number of the Root bucket of the index described by this XAB. This number is one less than the depth of the index.

5.3.14.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the LVL field equal to the number of the index level of the Root bucket for the index identified by the REF field.

5.3.14.2 Input Values

MINIMUM = 1, meaning that the index is the shallowest possible, with one level of data and one level of index, the Root

MAXIMUM = no logical limit

```
*************
*           *
*    MRL    *
*           *
*************
```

5.3.15 MRL

The one-word Minimum Record Length (MRL) field contains the length, in bytes, that a record must be to include the key described by the XAB.

5.3.15.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the MRL field to the number of bytes in the shortest record that can contain the key identified by the REF field.

5.3.15.2 Output Values - The value in the MRL field is based on the key's position (specified by the POS field) and the key's length (indicated by the SIZ field). In a single-segment key, the sum of the two provides the minimum record length. However, in a multi-segment key, the pertinent segment is the one farthest from the beginning of the record. The starting position of that segment, plus its length, determine the minimum length record.

5.3.15.3 Comments

- If the index described by the XAB is the Primary index, then no record smaller than the MRL value can be written to the file; RMS-11 returns error code ER$RSZ.
- If the index described by the XAB is an Alternate index, then if the record is shorter than the MRL, RMS-11 makes no entry in the index for the record.

```
*************
*           *
*    NSG    *
*           *
*************
```

5.3.16 NSG

The one-byte Number of Key Segments (NSG) field contains the number of segments specified in the key described by the XAB when the file was created. The POS and SIZ fields contain this number of values.

5.3.16.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the NSG field equal to the number of positions and lengths it finds in the POS and SIZ fields.

5.3.16.2 Input Values

MINIMUM = 1

MAXIMUM = 8

Only string keys will have NSG values greater than 1.

```
*************
*           *
*    NUL    *
*           *
*************
```

5.3.17 NUL

The one-byte Null Key Character (NUL) field contains the binary representation of the null key character for the key described by this XAB. This field is valid for string Alternate Keys only (REF field greater than 0). See the RMS-11 User's Guide for a discussion of null key values.

5.3.17.1 Use

Input to:

$CREATE
: You set the NUL field equal to the character or code you want used as the null key character for this key; you should also put the symbolic value XB$NUL in the FLG field.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the NUL field equal to the null key value established when the file was created.

5.3.17.2 Input Values - RMS-11 uses the NUL field only if the FLG field contains XB$NUL and the key is a string data type; if both are true, the NUL field must contain* a binary number whose octal value ranges from 000 to 377. You may represent an ASCII character (via the MACRO-11 single-quote (') convention) or a non-ASCII octal value.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.17.3 Initialization and Default - Macro takes the form:

```
X$NUL value
```

where value is a valid MACRO-11 representation of a one-byte value.

If there is no initialization macro, NUL = 0.

---

*If one of these conditions is false, the NUL may contain any value; the recommended value in that case is the default, zero.

5.3.17.4 Examples

- The following code specifies a null key value of Z for a string key:

```
KEYDEF: XAB$B   XB$KEY
          .
        X$DTP   XB$STG
          .
        X$FLG   XB$NUL
        X$NUL   'Z
          .
          .
        XAB$E
```

- The following code specifies the null key feature for a string key, but the null key value is the ASCII character DEL:

```
KEYDEF: XAB$B   XB$KEY
          .
        X$DTP   XB$STG
          .
        X$FLG   XB$NUL
        X$NUL   ^177
          .
          .
        XAB$E
```

- The following code specifies the null key feature for a packed decimal key; however, no null key value is necessary:

```
KEYDEF: XAB$B   XB$KEY
          .
        X$DTP   XB$PAC
          .
        X$FLG   XB$NUL
          .
          .
        XAB$E
```

```
*************
*           *
*    NXT    *
*           *
*************
```

5.3.18 NXT

The one-word Next XAB (NXT) field, if not zero, points to another separately allocated XAB and thereby indicates to RMS-11 that there is at least one more XAB in the chain connected to the FAB via the FAB XAB field. See "Linking and Ordering XABs" at the beginning of this chapter.

5.3.18.1 Use

Input to:

$CREATE/$DELETE/$EXTEND/$OPEN

- If you have allocated another XAB and want RMS-11 to use it, you set the NXT field equal to the address of that XAB.
- If you have not allocated an XAB and/or you want the XAB chain to end here, set the NXT field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns the error code ER$XAB.

5.3.18.2 Input Values - The NXT field must contain either:

- zero (0) = there are no more XABs in this chain
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.18.3 Initialization and Default - Macro takes the form:

```
X$NXT label
```

If there is no initialization macro, NXT = 0.

```
************
*          *
*    POS   *
*          *
************
```

5.3.19 POS

The eight-word Key Position (POS) field contains the starting positions within the record for all segments of the key described by this XAB. Combined with the SIZ and DTP fields, the POS field completely defines the key.

An RMS-11 key string may contain from one to eight segments; all other key types require a single segment. Each word in the POS field specifies the starting position of a segment: the first word, the first segment, the second word, the second segment, and so on. Each segment must be contiguous, but the segments do not have to be contiguous with each other. When processing records, RMS-11 combines the individual segments and treats them as a single, logically contiguous string beginning with the first segment and ending with the last. See also the RMS-11 User's Guide discussion of segmented keys.

5.3.19.1 Use

Input to:

$CREATE
: You set the POS field to include the starting position of each segment in the key described by this XAB. See "Comments" for $STORE syntax.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the POS field to include starting positions of the key segments established when the file was created.

5.3.19.2 Input Values - The POS field must contain at least one numeric value in the range:

MINIMUM = 0, meaning the segment starts in the first byte of the record

MAXIMUM = 65535, meaning the segment starts in the last byte of the largest possible record

If a string key has multiple segments, you must specify a starting position for each segment, each within the above range, using the form:

<seg0,seg1[,seg2,...,seg7]>

The values do not have to represent ascending byte positions in the record.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see "Comments" below and Chapter 7).

5.3.19.3 Initialization and Default - Macro takes the form:

```
X$POS [<]numeric[,numeric,...>]
```

If there is no initialization macro, POS = 0.

5.3.19.4 Comments

- You may store or fetch the position of an individual segment in the POS field, by using the term

  ```
  POSn
  ```

  where n is an integer in the range 0 to 7, indicating the segment number whose position you want to affect.

  Example You want to check the starting position of the third segment of a key string:

  ```
  $FETCH R0,POS2,R4
  ```

  You want to reset the starting position of the first segment of a key string:

  ```
  $STORE R0,POS0,R4
  ```

  In both cases, R0 is used as a buffer for the position number.

```
************
*          *
*    REF   *
*          *
************
```

5.3.20 REF

The one-byte Key of Reference (REF) field identifies the index and key described by this XAB.

5.3.20.1 Use

Input to:

$CREATE/$EXTEND
: You set the REF field to tell RMS-11 exactly what key and index you are describing in the XAB.

$DISPLAY/$OPEN
: You set the REF field to tell RMS-11 exactly what key and index you want RMS-11 to describe in the XAB.

5.3.20.2 Input Values

MINIMUM = 0, meaning Primary Key

MAXIMUM = 254

All values from 1 to 254 indicate Alternate Keys, first, second, and so on.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.20.3 Initialization and Default - Macro takes the form:

```
X$REF numeric
```

If there is no initialization macro, REF = 0.

```
*************
*           *
*    RVB    *
*           *
*************
```

5.3.21 RVB

The two-word Root Virtual Block Number (RVB) field specifies the number of the file's virtual block that contains the Root of the index represented by this XAB.

5.3.21.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the RVB field equal to the number of the first virtual block of the Root bucket for the index indicated by the REF field.

5.3.21.2 Output Values -

MINIMUM = first data block of the file, after the Prologue

MAXIMUM = last block of the file

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

```
************
*          *
*    SIZ   *
*          *
************
```

5.3.22 SIZ

The eight-byte Key Size (SIZ) field contains the lengths, in bytes, of all segments of the key described by this XAB. Each byte in the SIZ field specifies the size of a segment: the first byte, the first segment, second byte, the second segment, and so on. There must be a length specified for each segment given a starting position in the POS field.

5.3.22.1 Use

Input to:

$CREATE
: You set the SIZ field to include the length of each segment in the key described by this XAB. See "Comments" for $STORE syntax.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the SIZ field to include lengths of the key segments established when the file was created.

5.3.22.2 Input Values - The SIZ field must contain at least one numeric value which depends on key data type (specified by the DTP field in this XAB), as follows:

- 15-bit Signed Integer Key Type, SIZ = 0 or 2
- 31-bit Signed Integer Key Type, SIZ = 0 or 4
- 16-bit Unsigned Binary Key Type, SIZ = 0 or 2
- 32-bit Unsigned Binary Key Type, SIZ = 0 or 4
- String Key Type

  MINIMUM = 0 (null segment)

  MAXIMUM = 255

If a string key has multiple segments, you must specify a length for each segment, using the form:

```
<len0,len1[,len3,...,len7]>
```

The sum of the multiple lengths cannot exceed 255.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.3.22.3 Initialization and Default - Macro takes the form:

```
X$SIZ [<]numeric[,numeric,...>]
```

If there is no initialization macro, SIZ = 0.

5.3.22.4 Examples

- The following code specifies a single-segment key eight bytes long:

```
KEYDEF: XAB$B   XB$KEY
          .
          .
        X$SIZ   8.
          .
          .
        XAB$E
```

- The following code specifies a key with four segments, defining their sizes and positions:

```
KEYDEF: XAB$B   XB$KEY
          .
          .
        X$SIZ   <8,2,5,32>
        X$POS   <19,0,13,28>
          .
          .
        XAB$E
```

5.3.22.5 Comments - You may store or fetch the length of an individual segment in the SIZ field, by using the term

```
SIZn
```

where n is an integer in the range 0 to 7, indicating the segment number whose length you want to affect.

Example You want to check the length of the third segment of a key string:

```
$FETCH R0,SIZ2,R4
```

You want to reset the length of the first segment of a key string:

```
$STORE R0,SIZ0,R4
```

In both cases, R0 is used as a buffer for the length.

```
************
*          *
*    TKS   *
*          *
************
```

5.3.23 TKS

The one-byte Total Key Size (TKS) field specifies the number of bytes contained in all segments of the key described by the XAB.

5.3.23.1 Use

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the TKS field equal to the sum of the segment lengths established via the SIZ field for the key indicated by the REF field.

5.3.23.2 Input Values

MINIMUM = 1

MAXIMUM = 255

5.4 Protection Extended Attribute Block

The Protection XAB describes the protection code of a file, including specifications for the owner's account number. You use a Protection XAB generally when you do not want the default protection value for the operating system applied to the file.

Protection XABs are used as:

Input to $CREATE and $CLOSE
: For an RMS-11 file of any organization, you may set its protection code when it is created and change that code when it is closed.

Output from $DISPLAY and $OPEN
: RMS-11 supplies the current protection value for the file.

Table 5-4: Protection Extended Attribute Block Fields

Field Name	Field Size	Default	Description
BLN	1B	N/A	PRO XAB length
COD	1B	N/A	XAB type
NXT	1W	0	Next XAB
PRG	1W	0	Programmer part of account number
PRJ	1W	0	Project part of account number
PRO	1W	0	File protection value

****
*   *
* BLN   *
*   *
****

5.4.1 BLN

The XAB$B macro sets the one-byte Block Length (BLN) field to the Extended Attribute Block length.

CAUTION

DO NOT CHANGE THE BLN FIELD.

5.4.1.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.

```
**************
*            *
*     COD    *
*            *
**************
```

5.4.2 COD

The XAB$B macro sets the one-byte Code (COD) field to the Extended Attribute Block type, with the symbolic value of XB$PRO.

CAUTION

DO NOT CHANGE THE COD FIELD.

5.4.2.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the COD field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$COD error code.

****
*   *
* NXT   *
*   *
****

5.4.3 NXT

The one-word Next XAB (NXT) field, if not zero, points to another separately allocated XAB and thereby indicates to RMS-11 that there is at least one more XAB in the chain connected to the FAB via the FAB XAB field. See Chapter 1 at the beginning of this chapter.

5.4.3.1 Use

Input to:

$CREATE/$DELETE/$EXTEND/$OPEN

- If you have allocated another XAB and want RMS-11 to use it, you set the NXT field equal to the address of that XAB.
- If you have not allocated an XAB and/or you want the XAB chain to end here, set the NXT field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns the error code ER$XAB.

5.4.3.2 Input Values - The NXT field must contain either:

- zero (0) = there are no more XABs in this chain
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.4.3.3 Initialization and Default - Macro takes the form:

X$NXT label

If there is no initialization macro, NXT = 0.

```
*************
*           *
*    PRG    *
*           *
*************
```

5.4.4 PRG

The one-word Programmer Number (PRG) field contains the second number in the file owner's account number pair.

5.4.4.1 Use

Input to:

$CLOSE/$CREATE
: You set the PRG field to the individually unique number of the owner of the file.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the PRG field to the individually unique number of the file owner it reads from the file directory.

5.4.4.2 Input Values

MINIMUM = 0, meaning the default programmer number, based on the account in which the program is currently running

= 1, the smallest programmer number you can identify

MAXIMUM = 255

The programmer number specified in the Protection XAB must agree with the programmer number specified in the filespec indicated by the FAB FNA field or in the default name string indicated by the FAB DNA field.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.4.4.3 Initialization and Default - Macro takes the form:

```
X$PRG numeric
```

If there is no initialization macro, PRG = 0.

```
*************
*           *
*    PRJ    *
*           *
*************
```

5.4.5 PRJ

The one-word Project Number (PRJ) field contains the first number in the file owner's account number pair.

5.4.5.1 Use

Input to:

$CLOSE/$CREATE
: You set the PRJ field to the group or project number of the owner of the file.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the PRJ field to the group or project number of the file owner it reads from the file header.

5.4.5.2 Input Values

MINIMUM = 0, meaning the default project number, based on the account in which the program is currently running

= 1, the smallest project number you can identify

MAXIMUM = 255

The project number specified in the Protection XAB must agree with the project number specified in the filespec indicated by the FAB FNA field or in the default name string indicated by the FAB DNA field.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.4.5.3 Initialization and Default - Macro takes the form:

```
X$PRJ numeric
```

If there is no initialization macro, PRJ = 0.

```
************
*          *
*    PRO   *
*          *
************
```

5.4.6 PRO

The one-word System File Protection Code (PRO) field contains the system-specific code that indicates the access relationship allowed for the file represented by the FAB linked to this XAB.

5.4.6.1 Use

Input to:

$CREATE
: You set the PRO field to the protection code that controls access to the file as you want it.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the PRO field equal to the protection code found in the file directory.

5.4.6.2 Input Values - Specific protection values depend on operating system standards:

- IAS/RSX-11M

  The file protection value identifies the file access privileges of four classes of users:

  group — those accounts with the project number contained in the PRJ field of this XAB

  owner — that account with the project and programmer numbers contained in this XAB

  system — privileged accounts

  world — all accounts not in the other categories

  The PRO field assumes the following format:

```
1
5              8                  0
-----------------------------------
| WORLD | GROUP | OWNER | SYSTEM |
-----------------------------------
```

Each of the categories is allocated four bits, with the following meanings with respect to file access:

bit	3	2	1	0
	DELETE	EXTEND	WRITE	READ

If a bit is not set (zero), the respective type of access to the file is allowed; if the bit is set (one), that type of access is denied.

You should use a numeric value for that PRO field that sets the appropriate bits; this number may be decimal, octal, or binary, though using the MACRO-11 octal radix unary operator makes it easier.

- RSTS/E

The file protection value identifies the file access privileges of three classes of users:

group — those accounts with the project number contained in the PRJ field of this XAB

owner — that account with the project and programmer numbers contained in this XAB

others — all accounts not in the owner or group categories

The value of the PRO field is the numeric sum of one or more of the following codes:

1 read protect against owner

2 write protect against owner

4 read protect against everyone in group except owner

8 write protect against everyone in group except owner

16 read protect against others

32 write protect against others

See also your operating system documentation about protection codes.

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.4.6.3 Initialization and Default - Macro takes the form:

```
X$PRO numeric
```

If there is no initialization macro, PRO = 0.

5.5 Summary Extended Attribute Block

The Summary XAB generally describes an Indexed file, including the number of keys and areas established for the file when it was created. Summary XABs may also be used for Relative files if you want to determine their prologue version number.

Summary XABs are used as output only during $DISPLAY and $OPEN operations.

Example You want to open an Indexed file, but you don't know anything about its keys or areas, including their number. You therefore do not know how many Key or Allocation XABs to link to the FAB before you issue the $OPEN macro. You can, however, link a Summary XAB to the FAB and initiate an open operation. You then examine the NOA and NOK fields of the Summary XAB to determine how many of the other XABs you need for a $DISPLAY macro.

Table 5-5: Summary Extended Attribute Block Fields

Field Name	Field Size	Default	Description
BLN	1B	N/A	SUM XAB length
COD	1B	N/A	XAB type
NOA	1B	0	Number of areas
NOK	1B	0	Number of keys
NXT	1W	0	Next XAB
PVN	1W	0	RMS-11 Prologue version number

```
************
*          *
*    BLN   *
*          *
************
```

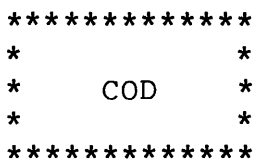
5.5.1 BLN

The XAB$B macro sets the one-byte Block Length (BLN) field to the Extended Attribute Block length.

CAUTION

DO NOT CHANGE THE BLN FIELD.

5.5.1.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the BLN field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$BLN error code.



5.5.2 COD

The XAB$B macro sets the one-byte Code (COD) field to the Extended Attribute Block type, with the symbolic value of XB$SUM.

CAUTION

DO NOT CHANGE THE COD FIELD.

5.5.2.1 Use - Before RMS-11 uses an XAB during a file operation, it verifies that the block is a valid XAB; one of the checks examines the COD field. If this field does not contain the proper code, RMS-11 aborts the operation with a ER$COD error code.

```
************
*          *
*    NOA   *
*          *
************
```

5.5.3 NOA

The one-byte Number Of Areas (NOA) field contains the number of file areas that were created in the file represented by the linked FAB.

5.5.3.1 Use - The NOA field is only applicable to Indexed files.

Output from:

$DISPLAY/$OPEN
RMS-11 sets the NOA file equal to the number of areas in the file associated with the XAB.

5.5.3.2 Output Values

MINIMUM = 1

MAXIMUM = 254

```
*************
*           *
*    NOK    *
*           *
*************
```

5.5.4 NOK

The one-byte Number Of Keys (NOK) field contains the number of keys that were created in the file represented by the linked FAB.

5.5.4.1 Use - The NOK field is only applicable to Indexed files.

Output from:

$DISPLAY/$OPEN
RMS-11 sets the NOK file equal to the number of keys in the file associated with the XAB.

5.5.4.2 Output Values

MINIMUM = 0, meaning a Relative file

1, minimum number of keys for an Indexed file

MAXIMUM = 255

```
*************
*           *
*    NXT    *
*           *
*************
```

5.5.5 NXT

The one-word Next XAB (NXT) field, if not zero, points to another separately allocated XAB and thereby indicates to RMS-11 that there is at least one more XAB in the chain connected to the FAB via the FAB XAB field. See **Chapter 1 at the beginning of this chapter.**

5.5.5.1 Use

Input to:

$CREATE/$DELETE/$EXTEND/$OPEN

- If you have allocated another XAB and want RMS-11 to use it, you set the NXT field equal to the address of that XAB.
- If you have not allocated an XAB and/or you want the XAB chain to end here, set the NXT field equal to zero; otherwise, RMS-11 interprets the value in the field as an address and tries to use the area indicated as an XAB: if that area is not a valid XAB, RMS-11 returns the error code ER$XAB.

5.5.5.2 Input Values - The NXT field must contain either:

- zero (0) = there are no more XABs in this chain
- address of an Extended Attribute Block separately allocated in the program

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

5.5.5.3 Initialization and Default - Macro takes the form:

X$NXT label

If there is no initialization macro, NXT = 0.

```
************
*          *
*    PVN   *
*          *
************
```

5.5.6 PVN

The one-word Prologue Version Number (PVN) field contains the RMS-11 Prologue version number.

5.5.6.1 Use -

The PVN field is only applicable to Relative and Indexed files.

Output from:

$DISPLAY/$OPEN
: RMS-11 sets the PVN field equal to the version number it finds in the file Prologue, if the Summary XAB is long enough to contain this field (see "Comments").

5.5.6.2 Output Values -

MINIMUM = 1, for Relative files and Indexed files with string keys

MAXIMUM = 2, for Indexed files with nonstring keys

5.5.6.3 Comments -

RMS-11 fills in the PVN field if the Summary XAB is long enough to contain it. The XAB won't be long enough if you assembled your program using RMS-11 V1.0 RMSLIB.MAC and then task built it using RMS-11 V1.5 RMSLIB.OLB.

CHAPTER 6

NAME BLOCK

The initialization macros (this chapter) and the field access macros (Chapter 7) are provided so that you do not have to know the specific position and to a large extent, the size of each field in a NAM Block. You can, therefore, treat the fields as logical entities. It is also possible that the positions and sizes of the fields will change from release to release of RMS-11.

However, you can determine the position of any field in a NAM as an offset from the NAM's starting address. RMS-11 represents these offset values with symbols in following form:

O$fnm

where fnm is the three-letter name of a NAM Block field; fnm is the name used to reference the field in the initialization and field access macros.

Example O$ESA for the expanded string address field

The values of these symbols can be found in the symbol table of an assembly listing file for any module containing the FAB.

Table 6-1: NAM Block Fields

Field Name	Field Size	Default	Description
DVI	1W	0	Device ID
ESA	1W	0	Expanded string address
ESL	1B	0	Expanded string length
ESS	1B	0	Expanded string size
FID	1B	0	File ID

```
*************
*           *
*    DVI    *
*           *
*************
```

6.1 DVI

The three-word Device ID (DVI) field contains a code that the file processor can use in conjunction with a file ID to locate a file without consulting device directories.

6.1.1 Use

Input to:

$ERASE/$OPEN
: You set the NAM DVI field equal to a device ID value RMS-11 provided during a previous create or open by file specification operation. You must also set the NAM FID field equal to the file ID returned during the same operation as well as storing FB$FID in the FAB FOP field.

Output from:

$CREATE/$OPEN by file specification
: RMS-11 sets the DVI field equal to the code the file processor provides during the open by file specification process.

6.1.2 Input Values

You should make no attempt to analyze or change the device ID provided by RMS-11. Store the contents of the NAM DVI field as a three-word quantity and supply it as such when you execute an erase or open by file ID operation.

```
*************
*           *
*    ESA    *
*           *
*************
```

6.2 ESA

The Expanded String Address (ESA) field contains the address of a buffer that you have allocated. During an open, create, or erase operation, RMS-11 places in this buffer the file specification string resulting from the application of default information (provided by the default name string of the FAB and system defaults) to the primary file name string of the FAB.

The ESA buffer must be present if the FAB NAM field is not zero.

NOTE

The NAM ESA field and the expanded string buffer are not required or used during an open or erase by file ID operation.

6.2.1 Use

Input to $CREATE/$ERASE/$OPEN:
You set the ESA field equal to the address of a buffer where you want RMS-11 to store the full file specification.

6.2.2 Input Values

ESA must contain either:

- zero (0) = there is no file specification buffer for this operation; valid only for open and erase by file ID
- address of the first byte of a buffer for the full file specification; the buffer does not have to be word-aligned

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

6.2.3 Initialization and Default

Macro takes the form:

```
N$ESA address
```

If there is no initialization macro, BKS = 0.

6.3 ESL

The one-byte Expanded String Length (ESL) field contains the actual length of the full file specification RMS-11 stored beginning with the address in the NAM ESA field.

6.3.1 Use

Output from:

$CREATE/$ERASE/$OPEN by file specification
RMS-11 sets the ESL field equal to the size in bytes of the expanded file specification resulting from the operation.

```
************
*          *
*    ESS   *
*          *
************
```

6.4 ESS

The one-byte Expanded String Size (ESS) field contains the size of the buffer whose address you stored in the NAM ESA field.

6.4.1 Use

Input to:

$CREATE/$ERASE/$OPEN
: You set the ESS field equal to the size of the buffer where you want RMS-11 to store the full file specification.

6.4.2 Input Values

MINIMUM = 1

MAXIMUM = 255

A value can be set in the field with the initialization macro shown below or with the $STORE field access macro (see Chapter 7).

6.4.3 Initialization and Default

Macro takes the form:

```
N$ESS numeric
```

If there is no initialization macro, ESS = 0.

6.5 FID

The three-word File ID (FID) field contains a code that the file processor can use in conjunction with a device ID to locate a file without consulting device directories.

6.5.1 Use

Input to:

$ERASE/$OPEN
: You set the NAM FID field equal to a file ID value RMS-11 provided during a previous create or open by file specification operation. You must also set the NAM DVI field equal to the device ID returned during the same operation as well as storing FB$FID in the FAB FOP field.

Output from:

$CREATE/$OPEN by file specification
: RMS-11 sets the FID field equal to the index the file processor provides during the open by file specification process.

6.5.2 Input Values

You should make no attempt to analyze or change the File ID provided by RMS-11. Store the contents of the NAM FID field as a three-word quantity and supply it as such when you execute an erase or open by file ID operation.

CHAPTER 7

FIELD ACCESS MACROS

RMS-11 field access macro retrieve, modify, and test the contents of fields in the RMS-11 control blocks, FABs, RABs, and XABs, at run time. These macros enable you to treat the control block fields as logical entities, without regard for the placement of the fields within the control blocks and to a large degree, for the sizes of the fields.

Table 7-1 contains a summary of the available macros. Each macro is also described in a separate section of this chapter.

RMS-11 assumes octal radix for all numeric values used as operands for the field access macros.

In all two-word fields containing numeric values, the least significant bits appear in the word with the lower address.

Table 7-1: RMS-11 Field Access Macros

Macro Name	Field Size	Function
$COMPARE	1 byte or 1 word	Compares the contents of a field with a value you specify.
$FETCH	Any size	Copies the contents of a field into a location you specify.
$OFF	1 byte or 1 word	Resets one or more bits within a bit string field.
$SET	1 byte or 1 word	Sets one or more bits within a bit string field.
$STORE	Any size	Copies the contents of a location you specify into a field.
$TESTBITS	1 byte or 1 word	Tests one or more bits within a bit string field.

```
************
*          *
* $COMPARE *
*          *
************
```

7.1 $COMPARE

The $COMPARE macro compares a one-byte or one-word control block data field with a value you specify and sets the PDP-11 condition codes.

7.1.1 General Form

```
$COMPARE   source,fnm,reg
```

where source represents a value you want compared with the contents of a control block data field. You can express this operand using any valid addressing mode. The operand must be word-aligned for comparison with one-word data fields.

fnm is the three-letter name of a one-byte or one-word field. The assembler generates an error message if the name is invalid for the block referenced by reg or if the field specified by fnm is a multiword field.

reg is a general register (R0 through R5) loaded with the address of the control block containing the specified field.

7.1.2 Effect

The $COMPARE macro makes the syntax checks as described under "General Form." Then, if the source operand is #0, the macro generates a TST or TSTB instruction, according to the size of the field. If the source operand is not #0, the macro generates a CMP or CMPB instruction appropriately, comparing the source operand with the specified field as an offset from the contents of the specified register.

At run time, condition codes are set according to the instruction generated.

7.1.3 Examples

- $COMPARE #SU$SUC,STS,R1

  The contents of the STS field in the control block pointed to by R1 are compared with the symbolic completion value SU$SUC.

- $COMPARE 2(R1),RSZ,R5

  The contents of the Record Size field in the RAB indicated by R5 is compared with the operand specified by indexed addressing mode.

- $COMPARE #0,STS,R1

```
BGT     14$     ;  CONTINUE IF SUCCESSFUL
BLT     ERRTN   ;  HANDLE ERROR IF THERE IS ONE
JMP     TRUBLE  ;  SOFTWARE IS IN A JAM
```

  General register R1 points to the block controlling the operation just completed (FAB for file operations, RAB for record operations). The program then branches according to the contents of the STS field.

```
*************
*           *
*  $FETCH   *
*           *
*************
```

7.2 $FETCH

The $FETCH macro copies the contents of a control block data field into a location you specify. This macro can be used to access a field of any size.

7.2.1 General Form

```
$FETCH     destination,fnm,reg
```

where destination is a location within your program where you want the contents of a control block field copied. The following restrictions apply to this operand:

- You cannot use immediate mode or any form of deferred addressing mode: the assembler generates an error.
- If fnm is POS or SIZ, you cannot use register mode addressing: the assembler generates an error.
- For multiword fields other than POS or SIZ, use register mode addressing carefully: the $FETCH macro uses successive registers as destination operands for successive words of the data field; it uses the register containing the control block address if the data field is long enough. The results are unpredictable.
- For single- and multiword fields, the destination location must be word-aligned.

fnm is the three-letter name of any field within the control block. The assembler generates an error if the name is invalid for the block referenced by reg.

To reference Key XAB SIZ and POS fields, you:

- specify SIZ or POS to fetch all eight words of the field. The macro copies the words into successive locations beginning with the destination you specify.
- specify SIZ or POS plus a number from 0 through 7 to fetch a single word from the field. See "Examples."

reg is a general register (R0 through R5) loaded with the address of the control block containing the source data field.

7.2.2 Effect

The $FETCH macro makes the syntax checks as described under "General Form." Then, the macro generates a MOVB or one or more MOV instructions, according to size of the field.

7.2.3 Examples

- $FETCH R2,RBF,R4

  General register R4 contains the address of a RAB. The macro copies the contents of that RAB's RBF field into general register R2.

- $FETCH 8.(R3),MRN,R1

  General register R1 contains the address of a FAB. The macro copies both words of that FAB's MRN field into successive words beginning with the specified location.

- $FETCH LSEG3,SIZ2,R5

  General register R5 contains the address of a Key XAB. The macro copies the length of the third key segment into the location labeled LSEG3.

```
************
*          *
*   $OFF   *
*          *
************
```

7.3 $OFF

The $OFF macro resets one or more bits within a one-byte or one-word bit string data field.

7.3.1 General Form

```
$OFF        value,fnm,reg
```

where value is an expression or location specifying the bits within the data field you want reset,

fnm is the three-letter name of a one-byte or one-word field. The assembler generates an error message if the name is invalid for the block referenced by reg or if the field specified by fnm is a multiword field.

reg is a general register (R0 through R5) loaded with the address of the control block containing the specified field.

7.3.2 Effect

The $OFF macro makes the syntax checks as described under "General Form." Then, the macro generates one or more BIC or BICB instructions, according to size of the field.

7.3.3 Examples

- ```
  $OFF      RB$KGE,ROP,R2
  ```

  General register R2 contains the address of a RAB. The macro operates on that RAB's ROP field, resetting the bit that specifies greater than or equal key match during a random record operation.

7.3.4 Comments

The $OFF macro resets individual bits. You should use the $STORE macro instead if you want to clear an entire bit string field or reset a value in a field not a bit string.
  ```

```
*************
*           *
*   $SET    *
*           *
*************
```

7.4 $SET

The $SET macro sets one or more bits within a one-byte or one-word bit string data field.

7.4.1 General Form

```
$SET        value,fnm,reg
```

where value is an expression or location specifying the bits within the data field you want set,

fnm is the three-letter name of a one-byte or one-word field. The assembler generates an error message if the name is invalid for the block referenced by reg or if the field specified by fnm is a multiword field.

reg is a general register (R0 through R5) loaded with the address of the control block containing the specified data field.

7.4.2 Effect

The $OFF macro makes the syntax checks as described under "General Form." Then, the macro generates one or more BIS or BISB instructions, according to size of the field.

7.4.3 Examples

- `$SET     FB$GET!FB$UPD,FAC,R4`

  General register R4 contains the address of a FAB. The macro sets bits within that FAB's FAC field indicating get and update operations will be performed on the associated file. The $SET macro is executed before a $CREATE or $OPEN operation is initiated.

- `$SET     RB$EOF,ROP,R1`

  General register R1 contains the address of a RAB. The macro sets the bit within that RAB's ROP field that specifies positioning to end-of-file during the following $CONNECT record operation.

7.4.4 Comments

The $SET macro sets individual bits. You should the $STORE macro instead if you want to:

- set only the specified values in a bit string field, ensuring that no other bits remain set
- set a value in a field not a bit string $STORE macro.

```
************
*          *
*  $STORE  *
*          *
************
```

7.5 $STORE

The $STORE macro copies values from a location you specify into a control block data field. This macro can be used to access a field of any size.

7.5.1 General Form

```
$STORE    source,fnm,reg
```

where source represents a value or series of values you want stored in a control block data field. The following restrictions apply to this operand:

- You cannot use any form of deferred addressing mode: the assembler generates an error.
- Immediate mode addressing can be used only with one-byte and one-word fields: the assembler generates an error.
- If fnm is POS or SIZ, you cannot use register mode addressing: the assembler generates an error.
- For multiword fields other than POS or SIZ, use register mode addressing carefully: the $STORE macro uses successive registers as source operands for successive words of the data field; it uses the register containing the control block address if the data field is long enough. The results are unpredictable.
- For single- and multiword fields, the source location must be word-aligned.

fnm is the three-letter name of any field within the control block. The macro changes all bytes in this field. The assembler generates an error if the name is invalid for the block referenced by reg.

To reference Key XAB SIZ and POS fields, you:

- specify SIZ or POS to change all eight words of the field. The macro copies eight words from successive locations beginning with the source location you specify.

- specify SIZ or POS plus a number from 0 through 7 to store a single word into the field. See "Examples."

reg is a general register (R0 through R5) loaded with the address of the control block containing the source data field.

7.5.2 Effect

The $STORE macro makes the syntax checks as described under "General Form." Then, the macro generates a MOVB or one or more MOV instructions, according to size of the field. For multiword fields, the macro generates a MOV instruction for each word in the field, beginning with the specified source location and using successive words after that.

7.5.3 Examples

- $STORE 0,ALQ,R3

  General register R3 contains the address of a FAB. The macro clears that FAB's two-word ALQ field.

- $STORE INPUT,FAB,R1

  General register R1 contains the address of a RAB. The macro stores the address of a FAB in that RAB's FAB field.

- $STORE 23.,SIZ2,R1

  General register R1 points to a Key XAB. The macro changes the size of the third segment of the key defined by the XAB.

```
*************
*           *
* $TESTBITS *
*           *
*************
```

7.6 $TESTBITS

The $TESTBITS macro compares one or more bits within a one-byte or one-word control block data field with a value you specify and sets the PDP-11 condition codes.

7.6.1 General Form

```
$TESTBITS  value,fnm,reg
```

where value is an expression or location specifying the bits within the data field you want tested,

fnm is the three-letter name of a one-byte or one-word field within a control block. The assembler generates an error message if the name is invalid for the block referenced by reg or if the field specified by fnm is a multiword field.

reg is a general register (R0 through R5) loaded with the address of the control block containing the specified data field.

7.6.2 Effect

The $OFF macro makes the syntax checks as described under "General Form." Then, the macro generates one or more BIT or BITB instructions, according to size of the field.

7.6.3 Examples

```
$TESTBITS        FB$UPD!FB$PUT,FAC,R3
```

General register R3 contains the address of a FAB. The macro checks that FAB's FAC field to determine if the current program can issue update or put operations. If either or both bits are set, condition code Z is not set; if neither is set, code Z is set.

CHAPTER 8

FILE AND RECORD OPERATION MACROS

8.1 FILE OPERATION MACROS

A file operation macro causes RMS-11 to perform an action related to an entire file. The macro name indicates the type of operation performed. The fields of the FAB associated with the macro in the calling sequence identifies the file and qualifies the operation.

Table 8-1 summarizes the RMS-11 file operation macros.

Table 8-1: RMS-11 File Operation Macros

Macro Name	Description
$CLOSE	Closes an open RMS-11 file so that your program can no longer access its contents.
$CREATE	RMS-11 creates and opens an RMS-11 file as described by the associated FAB and XABs, if any.
$DISPLAY	Stores attributes of an existing RMS-11 file in FAB and XAB fields.
$ERASE	Deletes an existing RMS-11 file and removes its entry(s) from a directory.
$EXTEND	Increases the number of blocks allocated to an RMS-11 file.
$OPEN	Opens an existing RMS-11 file, making its contents available for processing.

****
*   *
* $CLOSE *
*   *
****

8.1.1 $CLOSE

The $CLOSE macro closes an open RMS-11 file, making its contents unavailable to your program. The operation effectively disconnects all the file's Record Access Streams; if a stream has an outstanding I/O request, the close fails with the ER$ACT error code. However, you should shut down all streams with $DISCONNECT macros before initiating a close operation.

8.1.1.1 Buffer Requirements

One BDB and 512 bytes of I/O buffer space.

NOTE

Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.1.2 Input Fields

FAB FOP (can contain FB$RWC to rewind a magtape file)
IFI

PRO XAB All fields

8.1.1.3 Output Fields

FAB IFI (set to zero)
BPA (address of buffer area returned for your use)
BPS (size of buffer area returned for your use)
STS
STV

8.1.1.4 General Form -

```
$CLOSE     fab[,error]
```

where fab is the address of a File Access Block representing an open RMS-11 file.

error is the address of a completion routine to be called if the $CLOSE operation fails.

8.1.1.5 Comments -

- Even if RMS-11 returns the ER$WER error code in the FAB STS field, the file is closed and deaccessed.
- If you issue a record operation on a file after it is closed, RMS-11 returns error code ER$ISI.
- If your program operates as follows, be sure to detach the NAM Block before initiating the $CLOSE macro: RMS-11 destroys the NAM FID field during close.

  1. Program links NAM Block to FAB, then opens file by file specification.
  2. Program closes file.
  3. Program opens or erases file by file ID.

```
***************
*             *
*   $CREATE   *
*             *
***************
```

8.1.2 $CREATE

The $CREATE macro creates an RMS-11 file with the attributes you specify in a FAB and any XABs linked to that FAB, then makes the new files available for processing by your program.

Key and Allocation XABs must be grouped in densely ascending order (see Section 5.0.3); otherwise, RMS-11 returns the error code ER$ORD. An illogical XAB type in the chain causes the ER$COD error code.

Example Summary XAB.

Example Key XAB for a Sequential file.

If no Allocation XABs are linked to the FAB, the create operation creates the file as a single area (Area 0), using data from the FAB fields to control its processing.

However, if one or more Allocation XABs are linked to the FAB, the create operation ignores the FAB ALQ and BKS fields, creating the file according to the values in the ALL XAB(s), establishing areas as specified.

If a NAM Block is linked to the FAB, RMS-11 provides the expanded file specification of the new file.

The new file must be closed before your program terminates.

8.1.2.1 Buffer Requirements - One BDB and 512 bytes of I/O buffer space.

NOTE

> Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.2.2 Input Fields -

```
    FAB ALQ (ignored if Allocation XABs linked)
        BKS (ignored if Allocation XABs linked)
        BLS (magnetic tape files only)
        BPA
        BPS
        DEQ (used when no ALL XABs or ALL XAB DEQ field is zero)
        DNA
        DNS
        FAC (must contain at least FB$PUT)
        FNA
        FNS
        FOP
        FSZ (VFC records only)
        LCH
        MRN (Relative files only)
        MRS
        NAM
        ORG
        RAT
        RFM
        RTV
        SHR
        XAB

ALL XAB All fields

KEY XAB All fields

PRO XAB All fields
```

8.1.2.3 Output Fields -

```
    FAB DEV
        IFI
        STS
        STV

    NAM DVI
        ESA
        ESL
        FID
```

8.1.2.4 General Form -

```
     $CREATE    rab[,error]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $CREATE operation fails.

```
***************
*             *
*  $DISPLAY   *
*             *
***************
```

8.1.3 $DISPLAY

The $DISPLAY macro causes RMS-11 to store attribute information from an open file in XAB fields.

Key and Allocation XABs must be grouped in ascending order (error code ER$ORD), but they do not have to be densely ordered. RMS-11 ignores excess and illogical XABs.

8.1.3.1 Buffer Requirements - One BDB and 512 bytes of I/O buffer space.

NOTE

Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.3.2 Input Fields -

```
    FAB IFI
        XAB

ALL XAB AID
        NXT

DAT XAB NXT

KEY XAB NXT

PRO XAB NXT

SUM XAB NXT
```

8.1.3.3 Output Fields -

```
    FAB STS
        STV

ALL XAB ALN
        ALQ (number of blocks unused in current extent)
        AOP (can contain XB$CTG)
        BKZ
        DEQ
        LOC

DAT XAB All fields

KEY XAB All fields

PRO XAB All fields

SUM XAB All fields

    NAM All fields
```

8.1.3.4 General Form -

```
     $DISPLAY   fab[,error]
```

where fab is the address of a File Access Block representing an existing RMS-11 file.

error is the address of a completion routine to be called if the $DISPLAY operation fails.

8.1.3.5 Examples -

- A program that opens a file with unknown attributes cannot link in the proper Key and Allocation XABs. That program can link a Summary XAB to the FAB, then open the file. From the SUM XAB, the program determines whether XABs are appropriate, then how many of each are necessary. The program links in the proper XABs and executes a $DISPLAY macro to obtain the attribute information.

- A program has room for only one XAB. It can accumulate the information it needs on areas and keys by executing a series of $DISPLAY macros, changing the XAB fields appropriately between operations.

```
***************
*             *
*   $ERASE    *
*             *
***************
```

8.1.4 $ERASE

The $ERASE macro deletes an RMS-11 file and removes its directory entry(s). The blocks occupied by the file are made available for reuse in other files.

You can erase a file by file specification or by file ID. You indicate the file specification in a FAB. To specify a file by file ID, you must:

- link a NAM Block to the FAB
- indicate the correct ID with the NAM FID field
- set the value FB$FID in the FAB FOP field
- set appropriate values in the other input fields (see "Input Fields")

If the file is open, your program cannot erase it on the logical channel it was opened on.

Example A file is opened with the FAB LCH field set to 3. Channel 3 cannot be used for any other file operation, including erase, until that file is closed.

You cannot erase magnetic tape files or files on unit record devices.

8.1.4.1 Buffer Requirements - One BDB and 512 bytes of I/O buffer space.

NOTE

Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.4.2 Input Fields -

```
FAB BPA
    BPS
    DNA
    DNS
    FNA
    FNS
    LCH (must be different from open LCH)
    NAM

NAM DVI
    ESS
    FID
```

8.1.4.3 Output Fields -

```
FAB STS
    STV
```

8.1.4.4 General Form -

```
$ERASE     fab[,error]
```

where fab is the address of a File Access Block representing an existing RMS-11 file.

error is the address of a completion routine to be called if the $ERASE operation fails.

8.1.4.5 Comments - If your or another program has a file open and your program erases the file (on a different logical channel), the file processor does not delete the file from the directory and release its allocated blocks until all accessors close the file.

```
**************
*            *
*   $EXTEND   *
*            *
**************
```

8.1.5 $EXTEND

The $EXTEND macro requests the file processor to add blocks to a file's allocation.

If no Allocation XABs are linked to the FAB, RMS-11 extends Area 0*, using data from the FAB fields to control its processing. If the FAB FOP field contains FB$CTG, RMS-11 requests the file processor to allocate the requested blocks contiguously with those blocks already allocated to the file; if no contiguous blocks are available, RMS-11 returns an error.

However, if one or more Allocation XABs are linked to the FAB, RMS-11 extends the areas indicated by the ALL XAB AID fields by the amounts specified in the ALL XAB ALQ fields. If the XAB AOP field contains XB$CTG, RMS-11 requests the file processor to allocate the requested blocks contiguously with those blocks already allocated to the area; if no contiguous blocks are available, RMS-11 returns an error.}

8.1.5.1 Buffer Requirements - One BDB and 512 bytes of I/O buffer space.

NOTE

> Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.5.2 Input Fields -

```
    FAB ALQ (ignored if Allocation XABs linked)
        FOP (can contain FB$CTG;  ignored if Allocation XABs linked)
        IFI
        XAB

ALL XAB AID
        ALQ
        AOP (can contain XB$CTG)
```

---

*For Sequential and Relative files, Area 0 is the file.

8.1.5.3 Output Fields -

FAB ALQ (number of blocks actually added to the file)
STS
STV

ALL XAB ALQ (number of blocks actually added to the file)

8.1.5.4 General Form -

$EXTEND rab[,error]

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $EXTEND operation fails.

8.1.5.5 Comments - RMS-11 applies the following restrictions to the erase operation:

- The file must be open.
- All Record Access Streams connected to the file must be inactive; otherwise, RMS-11 returns error code ER$ACT.
- The file must have been opened with at least one of the following specified in the FAB FAC field:

  FB$DEL
  FB$PUT
  FB$UPD

- The file does not reside on magnetic tape; otherwise, RMS-11 returns error code ER$IOP.

```
***************
*             *
*    $OPEN    *
*             *
***************
```

8.1.6 $OPEN

The $OPEN macro makes an existing RMS-11 file available for processing by your program. RMS-11 also returns the basic attributes of the file in the FAB associated with the macro; if XABs are linked to the FAB, RMS-11 files in attribute information appropriately; and if a NAM Block is linked, RMS-11 provides the expanded file specification of the open file.

Key and Allocation XABs must be grouped in ascending order (error code ER$ORD), but they do not have to be densely ordered. RMS-11 ignores excess and illogical XABs.

You can open a file by file specification or by file ID. You indicate the file specification in a FAB. To specify a file by file ID, you must:

- link a NAM Block to the FAB
- indicate the correct ID with the NAM FID field
- set the value FB$FID in the FAB FOP field
- set appropriate values in the other input fields (see "Input Fields")

8.1.6.1 Buffer Requirements - One BDB and 512 bytes of I/O buffer space.

NOTE

Since any record operation requires at least one BDB and an I/O buffer a block long, the buffer requirements for this operation are typically met without specific attention on your part.

8.1.6.2 Input Fields -

```
    FAB BPA
        BPS
        DEQ (used as temporary run-time value)
        DNA
        DNS
        FAC
        FNA
        FNS
        FOP
        LCH
        NAM
        RTV
        SHR
        XAB

ALL XAB AID
        DEQ
        NXT

DAT XAB NXT

PRO XAB NXT

SUM XAB NXT

    NAM DVI
        ESS
        FID
```

8.1.6.3 Output Fields -

```
    FAB ALQ
        BKS
        DEQ (if input value = 0, set to value set at create time)
        BLS (magnetic tape files only)
        DEV
        FOP (contains FB$CTG if the file is contiguous)
        FSZ (VFC records only)
        IFI
        MRN (Relative files only)
        MRS
        ORG
        RAT
        RFM
        RTV
        STS
        STV

ALL XAB ALN
        ALQ (number of blocks unused in current extent)
        AOP  (can contain XB$CTG)
```

BKZ
DEQ
LOC

DAT XAB All fields

PRO XAB All fields

SUM XAB All fields

NAM All fields

8.1.6.4 General Form -

```
$OPEN      fab[,error]
```

where fab is the address of a File Access Block representing an existing file.

error is the address of a completion routine to be called if the $OPEN operation fails.

8.1.6.5 Comments - If you intend to extend the file while it is open, ensure that the FAB FAC field contains FB$DEL, FB$PUT, and/or FB$UPD before your program initiates the open operation.

8.2 RECORD OPERATION MACROS

After it has created or opened an RMS-11 file, your program can perform record operations on it. These operations involve the following concepts that are explained in the RMS-11 User's Guide and Chapter 1 of this manual.

Record Access Streams
File sharing
Context, Current Record, and Next Record
Synchronous and asynchronous record operations

Table 8-2: RMS-11 Record Operation Macros

Macro Name	Description
$CONNECT	Establishes a Record Access Stream.
$DELETE	Deletes a record from an RMS-11 Relative or Indexed file.
$DISCONNECT	Terminates a Record Access Stream.
$FIND	Locates a record in an RMS-11 file.
$FLUSH	Moves all data in unwritten I/O buffers to disk.
$FREE	Unlocks a bucket locked by a Record Access Stream.
$GET	Moves a record from an RMS-11 file into your program's user buffer.
$NXTVOL	Continues processing with the next volume of magnetic tape multivolume set.
$PUT	Moves a record from your program's user buffer to an RMS-11 file.
$REWIND	Resets a Record Access Stream's context to the logical beginning of an RMS-11 file.
$TRUNCATE	Deletes record at the end of an RMS-11 Sequential file.
$UPDATE	Replaces a record in an RMS-11 file with a record from your program's user buffer.
$WAIT	Suspends processing until an RMS-11 asynchronous record operation completes.

```
**************
*            *
*  $CONNECT  *
*            *
**************
```

8.2.1 $CONNECT

The $CONNECT macro establishes a Record Access Stream by associating a RAB with a FAB. RMS-11 also allocates buffer space or uses the GSA routine you provided to allocate buffer space, for:

- internal control structures from the central buffer pool
- I/O buffers from the central or private pool as you indicate in the FAB, according to file organization and the values specified in RAB MBC and MBF fields

8.2.1.1 Input RAB Fields -

BID
BLN
FAB
KRF (Indexed files only)
MBC (disk Sequential files only)
MBF (Relative and Indexed files only)
ROP
UBF (see RBF under "Output RAB Fields")

8.2.1.2 Output RAB Fields -

ISI
RBF (= UBF value for Locate Mode on Sequential files)
STS
STV

8.2.1.3 General Form -

```
$CONNECT   rab[,error[,success]]
```

where rab is the address of a Record Access Block you want connected to a FAB.

error is the address of a completion routine to be called if the $CONNECT operation fails.

success is the address of a completion routine to be called if the $CONNECT operation succeeds.

```
**************
*            *
*   $DELETE   *
*            *
**************
```

8.2.2 $DELETE

The $DELETE macro deletes an existing record from a Relative or Indexed file. If your program initiates a delete operation on a Sequential file, RMS-11 returns error code ER$IOP.

The $DELETE operation always applies to the Current Record. Therefore, your program must successfully execute a $FIND or $GET macro before initiating a $DELETE macro; otherwise, RMS-11 returns error code ER$CUR.

8.2.2.1 Input RAB Fields -

BID
BLN
ISI
ROP

8.2.2.2 Output RAB Fields -

STS
STV

8.2.2.3 General Form -

```
$DELETE    rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $DELETE operation fails.

success is the address of a completion routine to be called if the $DELETE operation succeeds.

```
**************
*            *
* $DISCONNECT *
*            *
**************
```

8.2.3 $DISCONNECT

The $DISCONNECT macro terminates a Record Access Stream, ending the association between a FAB and a RAB. RMS-11 releases all buffer space for reuse.

8.2.3.1 Input RAB Fields -

BID
BLN
ISI

8.2.3.2 Output RAB Fields -

ISI (set to zero)
STS
STV

8.2.3.3 General Form -

```
$DISCONNECT rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $DISCONNECT operation fails.

success is the address of a completion routine to be called if the $DISCONNECT operation succeeds.

8.2.3.4 Comments - The $DISCONNECT macro does not rewind magnetic tape files. If you want the file positioned to beginning-of-file for a subsequent connect operation, use the $REWIND macro before initiating the disconnect operation.

****
*   *
* $FIND *
*   *
****

8.2.4 $FIND

The $FIND macro locates a specified record in a file and returns its Record's File Address in the RAB RFA field. During the operation, RMS-11 sets the Current Record pointer and for Sequential Access Mode only, the Next Record pointer.

8.2.4.1 Input RAB Fields -

BID
BLN
ISI
KBF (Random Access Mode only)
KRF (Random Access Mode on Indexed files only)
KSZ (Random Access Mode only)
RAC
RFA (if RAC contains RB$RFA)
ROP

8.2.4.2 Output RAB Fields -

BKT (Sequential Access Mode on Relative files only)
RFA (if RAC does not contain RB$RFA)
STS
STV

8.2.4.3 General Form -

```
$FIND      rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $FIND operation fails.

success is the address of a completion routine to be called if the $FIND operation succeeds.

8.2.4.4 Comments -

- RMS-11 allows Access by RFA during find operations on disk files only.

- Normally, you would not use Access by RFA for find operations on Relative and Indexed files, since the output of the operation is the RFA used as input. However, a find by RFA returns error code ER$DEL if the specified record once existed in the file, but was subsequently deleted: a random find operation returns error code ER$RNF whether the record never existed or was deleted.

- In Sequential Access Mode, a find operation on an Indexed file uses the index of reference set by the most recent successful get, sequential find, or connect operation. If you want to use a different index for sequential processing, you should execute a rewind or a random get operation specifying the key of reference (RAB KRF field) you want.

- If the file allows duplicates in the key of reference, RMS-11 does not indicate if the record located is one of a series containing duplicates in that field.

****
*   *
* $FLUSH *
*   *
****

8.2.5 $FLUSH

The $FLUSH macro writes all modified I/O buffers associated with a Record Access Stream to disk, if they haven't been written. With this operation, you can ensure that all record activity up to a point in time is reflected in the file. For Relative and Indexed files, any bucket locked by the stream remains locked.

8.2.5.1 Input RAB Fields -

BID
BLN
ISI

8.2.5.2 Output RAB Fields -

STS
STV

8.2.5.3 General Form -

```
$FLUSH     rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $FLUSH operation fails.

success is the address of a completion routine to be called if the $FLUSH operation succeeds.

****
*   *
* $FREE *
*   *
****

8.2.6 $FREE

The $FREE macro unlocks a bucket that RMS-11 locked on behalf of a Record Access Stream. If no bucket is locked, RMS-11 returns error code ER$RNL.

See Chapter 2 of the RMS-11 User's Guide for more information on file sharing.

8.2.6.1 Input RAB Fields -

BID
BLN
ISI

8.2.6.2 Output RAB Fields -

STS
STV

8.2.6.3 General Form -

```
$FREE      rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $FREE operation fails.

success is the address of a completion routine to be called if the $FREE operation succeeds.

8.2.6.4 Comments - If you are using multiple Record Access Streams to access a file, you must not merely reissue a record operation that fails with error code ER$RLK. Since one of your program's streams can have the target bucket locked, you could place the program into an infinite loop if you continue to issue the same operation. Therefore, you should execute a $FREE macro for all other streams to the same file. Then you can safely reinitiate the original record operation until RMS-11 indicates successful completion.

```
***************
*             *
*    $GET     *
*             *
***************
```

8.2.7 $GET

The $GET macro retrieves a record from an RMS-11 file, moving the record from the I/O buffer to your program's user buffer* and returning the record's RFA in the RAB RFA field. After a successful operation, the RAB RBF and RSZ fields describe the retrieved record with the address of its first byte and its length, respectively.

8.2.7.1 Input RAB Fields -

BID
BLN
ISI
KBF (Random Access Mode only)
KRF (Random Access Mode on Indexed files only)
KSZ (Random Access Mode only)
RAC
RFA (if RAC contains RB$RFA)
RHB (VFC records only)
ROP
UBF (regardless of Record Transfer Mode)
USZ

8.2.7.2 Output RAB Fields -

BKT (Sequential Access Mode on Relative files only)
RBF
RFA (if RAC does not contain RB$RFA)
RSZ
STS
STV

8.2.7.3 General Form -

```
$GET       rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

---

*Unless you specify Locate Record Transfer Mode (RB$LOC in the RAB ROP field).

error is the address of a completion routine to be called if the $GET operation fails.

success is the address of a completion routine to be called if the $GET operation succeeds.

8.2.7.4 Comments -

- After a successful get operation from a unit record device, such as a terminal, the less significant byte of the RAB STV field contains a code representing the character that terminated the input record, as follows:

Code (in octal)	Character Represented
15	Carriage Return
33	Escape
32	CTRL/Z
0	other

Except when the code is 0, the terminating character is never in the record described by the RAB RBF and RSZ fields.

```
Example   $GET                       ;RETRIEVE A RECORD
          CMPB    #0,O$STS(R5)      ;CHECK IF TERMINATOR STILL IN RE-
          CORD
```

- RMS-11 allows Access by RFA during get operations on disk files only.

- Normally, you would not use Access by RFA for get operations on Relative and Indexed files, since the output of the operation is the RFA used as input. However, a get by RFA returns error code ER$DEL if the specified record once existed in the file, but was subsequently deleted: a random get operation returns error code ER$RNF whether the record never existed or was deleted.

- In Sequential Access Mode, a get operation not preceded by a successful find uses the index of reference set by the most recent successful get, find, or connect operation. If you want to use a different index for sequential processing, you should execute a rewind or a random get or find operation specifying the key of reference (RAB KRF field) you want.

- If the file allows duplicates in the key of reference, RMS-11 does not indicate if the record retrieved is one of a series containing duplicates in that field.

The $NXTVOL macro is not supported by RSTS/E.

****
*   *
* $NXTVOL *
*   *
****

8.2.8 $NXTVOL

The $NXTVOL macro can be used only when the Record Access Stream is accessing a multivolume file on magnetic tape. You execute this macro when you want to continue processing the current file on the next volume before the end of the current volume is reached. See also the magnetic tape handling appendix of the RMS-11 User's Guide.

Next-volume processing depends on whether your program is reading or writing data on the tape:

Input File Processing

1. RMS-11 skips all records in I/O buffers for the current file section.

2. RMS-11 requests the next tape volume from the file processor:

   - If there is no next volume, RMS-11 returns error code ER$EOF.

   - If there is another volume, the file processor rewinds the current volume, requests the system operator to mount the next volume, and reads the header label (HDR1) of the first file section on the next tape. If the tape is not the proper volume, the processor requests the operator to mount the correct tape and repeats this step.

3. The operation terminates.

Output File Processing

1. RMS-11 writes I/O buffers to the current file section in an implicit flush operation.

2. RMS-11 requests the next tape volume from the file processor:

   a. The processor closes the current file section, writing EOV1 and EOV2 labels and rewinding the volume.

   b. The processor requests the system operator to mount the next volume.

   c. The processor creates a file with the same name and the next higher section number and opens the file

for write operations. The file set identifier is identical with the volume identifier of the first volume in the volume set.

3. The operation terminates.

8.2.8.1 Input RAB Fields -

BID
BLN
ISI

8.2.8.2 Output RAB Fields -

STS
STV

8.2.8.3 General Form -

```
$NXTVOL    rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $NXTVOL operation fails.

success is the address of a completion routine to be called if the $NXTVOL operation succeeds.

```
***************
*             *
*    $PUT     *
*             *
***************
```

8.2.9 $PUT

The $PUT macro writes a new record into an RMS-11 file. The RAB RBF and RSZ fields describe the record to be written. You cannot use Access by RFA. Put operations in Random Access Mode do not change the Next Record pointer.

8.2.9.1 Input RAB Fields -

BID
BLN
ISI
KBF (Random Access Mode on Relative files only)
KSZ (Random Access Mode on Relative files only)
RAC
RBF
RHB (VFC records only)
ROP
RSZ
UBF (Locate Mode for Sequential files only)
USZ (Locate Mode for Sequential files only)

8.2.9.2 Output RAB Fields -

BKT (Sequential Access Mode on Relative files only)
RBF (Locate Mode for Sequential files)
RFA
STS
STV

8.2.9.3 General Form -

```
$PUT        rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $PUT operation fails.

success is the address of a completion routine to be called if the $PUT operation succeeds.

8.2.9.4 Comments -

- Since RMS-11 supports put operations in Sequential Access Mode only for Sequential files, you must set the value RB$SEQ in the RAB RAC field before initiating a $PUT operation on a Sequential file.

- RMS-11 supports put operations only at the end of Sequential files. Your program can reach end-of-file by one of the following methods:

  -- Set RB$EOF in the RAB ROP field before initiating the connect operation. RMS-11 sets the Current Record pointer to end-of-file.

  -- Execute $FIND (or $GET) macros until RMS-11 returns error code ER$EOF.

- RMS-11 restricts put operations on Relative files as follows:

  -- No record can be written that is longer than the Maximum Record Size specified when the file was created.

  -- The cell specified by the contents of the RAB KBF field cannot contain a valid record--unless you set the value RB$UIF in the RAB ROP field.

  -- The relative record number in the RAB KBF field cannot be greater than the Maximum Record Number specified when the file was created.

- RMS-11 restricts put operations on Indexed files as follows:

  -- No record can be written that is longer than the Maximum Record Size specified when the file was created.

  -- No record can be written that is too short to contain the complete Primary Key field. The record does not have to contain all defined Alternate Key fields.

- If a put operation on an Indexed file specifies a record that contains a key value already existing in the file, RMS-11 response depends on whether you allowed duplicate key values when you created the file:

  -- If duplicates are not allowed, RMS-11 returns error code ER$DUP.

  -- If duplicates are allowed, RMS-11 inserts the record after all records with the same key value and before a record with a higher key value. Then RMS-11 returns success code SU$DUP.

****
*   *
* $REWIND *
*   *
****

8.2.10 $REWIND

The $REWIND macro sets the context of a Record Access Stream to the logical beginning of the associated file. Following the operation, there is no Current Record, and the Next Record is the first record in the file; for Indexed files, the value of the RAB KRF sets the index that defines the first record logically.

8.2.10.1 Input RAB Fields -

BID
BLN
ISI
KRF (Indexed files only)

8.2.10.2 Output RAB Fields -

STS
STV

8.2.10.3 General Form -

```
$REWIND    rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $REWIND operation fails.

success is the address of a completion routine to be called if the $REWIND operation succeeds.

8.2.10.4 Comments - RSTS/E does not support the $REWIND macro for magnetic tape files only.

```
***************
*             *
*  $TRUNCATE  *
*             *
***************
```

8.2.11 $TRUNCATE

The $TRUNCATE macro truncates a Sequential file. If you initiate a truncate operation on a Relative or Indexed file, RMS-11 returns error code ER$IOP.

A truncate operation deletes the Current Record and all records following that record and sets Next Record to point to the end-of-file. Therefore, your program must successfully execute a $FIND or $GET macro before initiating a $UPDATE macro; otherwise, RMS-11 returns error code ER$CUR.

8.2.11.1 Input RAB Fields -

BID
BLN
ISI

8.2.11.2 Output RAB Fields -

STS
STV

8.2.11.3 General Form -

```
$TRUNCATE  rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $TRUNCATE operation fails.

success is the address of a completion routine to be called if the $TRUNCATE operation succeeds.

8.2.11.4 Comments -

Your program can follow a truncate operation immediately with one or more sequential put operations.

```
***************
*             *
*   $UPDATE   *
*             *
***************
```

8.2.12 $UPDATE

The $UPDATE macro replaces an existing record with the record described by the RAB RBF and RSZ fields.

The $UPDATE operation always applies to the Current Record. Therefore, your program must successfully execute a $FIND or $GET macro before initiating a $UPDATE macro; otherwise, RMS-11 returns error code ER$CUR.

RMS-11 error codes indicating an illegal input value in a RAB field do not affect the record being updated.

Example ER$RSZ

However, other errors can mean that the target record is lost.

Example ER$WER

8.2.12.1 Input RAB Fields -

BID
BLN
ISI
RBF
RHB (VFC records only)
RSZ

8.2.12.2 Output RAB Fields -

RFA
STS
STV

8.2.12.3 General Form -

```
$UPDATE    rab[,error[,success]]
```

where rab is the address of a Record Access Block representing a Record Access Stream.

error is the address of a completion routine to be called if the $UPDATE operation fails.

success is the address of a completion routine to be called if the $UPDATE operation succeeds.

8.2.12.4 Comments -

- RMS-11 restricts update operations on Sequential files as follows:

  -- No updates on magnetic tape files or unit record devices.

  -- No updates on disk Sequential files with stream format records.

  -- No updates that change the length of the target record: your program must not change the RAB RSZ field between the get or find operation and the update operation.

- For Relative files, the replacement record cannot be longer than the Maximum Record Size specified when the file was created.

- For an update operation on an Indexed file that allows duplicate Primary Keys, the replacement record length must equal the target record length; that is, your program must not change the RAB RSZ field between the get or find operation and the update operation.

- For an update operation on an Indexed file that does not allow duplicate Primary Keys, the record length can change, but the replacement record cannot be longer than the file's Maximum Record Size or too short to contain the complete Primary Key field.

```
***************
*             *
*    $WAIT    *
*             *
***************
```

8.2.13 $WAIT

The $WAIT macro suspends your program's operation until an outstanding asynchronous record operation completes.

8.2.13.1 Input RAB Fields -

BID
BLN
ISI

8.2.13.2 Output RAB Fields -

STS
STV
plus fields of asynchronous record operation your program is waiting for

8.2.13.3 General Form -

```
$WAIT      rab[,error[,success]]
```

where rab is the address of an asynchronous Record Access Block representing a Record Access Stream with an asynchronous record operation in progress.

error is the address of a completion routine to be called if the $WAIT operation fails.

success is the address of a completion routine to be called if the $WAIT operation succeeds.

CHAPTER 9

PERFORMING BLOCK I/O

Block I/O allows you to bypass the record processing capabilities of RMS-11. Through macros described in this chapter, you can read and write the virtual blocks of a file.

CAUTION

Many elements of the internal structure of RMS-11 files are not visible during record processing. With Block I/O, however, you can examine these elements.

Exercise extreme caution when altering the virtual blocks of RMS-11-structured Sequential, Relative, or Indexed files.

To use Block I/O, your program must do the following:

1. Allocate one BDB for each stream* connected for Block I/O. You specify BDBs with the P$BDB macro between the POOL$B and POOL$E macros (see Chapter 2).

2. Allocate an I/O buffer with at least 512 bytes (to support file operations). Use one of the techniques described in Chapters 1 and 2. If your program reads or writes more than one block at a time, allocate a buffer large enough to support all operations.

3. Set one or both of the following values in the FAB FAC field before initiating any file operation macros:

   FB$REA if you want to read blocks from an RMS-11 file

   FB$WRI if you want to write blocks to an RMS-11 file

4. Initiate one of the following file operaton macros:

   $CREATE
   You create a file that can be processed only with Block I/O operations. RMS-11 restricts such files as follows:

   - disk or magnetic tape medium

---

*RMS-11 permits a single Block I/O stream for Sequential files and multiple streams for Relative and Indexed files.

- Sequential organization (FB$SEQ in the FAB ORG field)
- Undefined record format (FB$UDF in the FAB RFM field)

$OPEN
You access a file with any RMS-11 file organization; however, you can use only Block I/O operations to access data in the file.

5. Set up a Record Access Block (see Chapter 4) and initiate a $CONNECT macro for each Record Access Stream. No stream can be connected to the file for record processing.

6. Use the macros described in this chapter to processing data in the file.

7. Disconnect each stream.

8. Close the file.

Table 9-1 summarizes the RMS-11 Block I/O macros.

Table 9-1: RMS-11 Block I/O Macros

Macro Name	Description
$READ	Retrieve a specified number of bytes from a virtual block.
$WRITE	Write a specified number of bytes to a virtual block.
$SPACE	Move a magnetic tape a specified number of blocks.

```
**************
*            *
*    $READ   *
*            *
**************
```

9.1 $READ

The $READ macro retrieves a specified number of bytes from a file beginning on a specified virtual block boundary. You must supply a word-aligned buffer for the data.

9.1.1 Input RAB Fields

BKT (VBN where read starts)
ISI
UBF (address for input buffer)
USZ (size of input buffer as a multiple of two bytes)

9.1.2 Output RAB Fields

RBF (location of data read from file)
RSZ (number of bytes read)
STS
STV

9.1.3 General Form

```
$READ      rab[,error[,success]]
```

where rab is the address of a Record Access Block containing the specification for the read operation.

error is the address of a completion routine to be called if the $READ operation fails.

success is the address of a completion routine to be called if the $READ operation succeeds.

9.1.4 Comments

- You can read multiple blocks by specifying an appropriate multiple of 512 bytes in the RAB USZ field.

- The value RMS-11 returns in the RAB RSZ field does not count the terminator character received from a unit record or terminal device. The RAB STV field reports on that character; see the $GET record operation in Chapter 8.

- If the RAB STS field contains ER$EOF, the RAB RSZ still contains the number of bytes transferred.

```
**************
*            *
*   $WRITE   *
*            *
**************
```

9.2 $WRITE

The $WRITE macro writes a specified number of bytes into a file beginning on a specified virtual block boundary.

9.2.1 Input RAB Fields

BKT (VBN where write starts)
ISI
RBF (address of first byte of one or more blocks to be written)
RSZ (size of data to be written as a multiple of two bytes)

9.2.2 Output RAB Fields

STS
STV (actual number of bytes transferred)

9.2.3 General Form

```
$WRITE     rab[,error[,success]]
```

where rab is the address of a Record Access Block containing the specification for the write operation.

error is the address of a completion routine to be called if the $WRITE operation fails.

success is the address of a completion routine to be called if the $WRITE operation succeeds.

9.2.4 Comments

- You can write multiple blocks by specifying an appropriate multiple of 512 bytes in the RAB RSZ field.
- If the RAB RSZ does not contain an even multiple of 512 bytes, the number of bytes specified are written, but the disk contents of the unwritten portion of the last block affected are undefined.

The $SPACE macro is not supported on RSTS/E

****
*   *
* $SPACE *
*   *
****

9.3 $SPACE

The $SPACE macro causes a magnetic tape file to move forward or backward. The file must have been opened for Block I/O and reside of magnetic tape; otherwise, RMS-11 returns error code ER$IOP.

9.3.1 Input RAB Fields

BKT (number of blocks to be spaced; sign indicates direction)
ISI
ROP (can contain RB$ASY; see Chapter 4)

9.3.2 Output RAB Fields

STS
STV (number of blocks spaced)

9.3.3 General Form

```
$SPACE     rab[,error[,success]]
```

where rab is the address of a Record Access Block containing the specification for the spacing operation.

error is the address of a completion routine to be called if the $SPACE operation fails.

success is the address of a completion routine to be called if the $SPACE operation succeeds.

9.3.4 Comments

RMS-11 examines only the lower addressed byte of the one-word BKT field. RMS-11 interprets this byte as a signed 15-bit integer:

- A positive integer represents the number of blocks the file is to be spaced forward

- A negative number represents the number of blocks the file is to be backspaced.

APPENDIX A

RMS-11 COMPLETION CODES

All RMS-11 file and record operations return a completion code into the STS field of the associated control block (FAB or RAB). A symbolic name is defined for each code, with one of the following forms:

Successful completion codes:

    SU$cod

Error completion codes:

    ER$cod

where cod represents the success qualifier or the reason the operation failed.

For certain error conditions, RMS-11 uses the STV field to communicate additional information to your program. The tables in this appendix list all instances where an STS code indicates the presence of further information in the STV field.

A limited number of severe error conditions cause RMS-11 to use a fatal error crash routine. Section A.3 describes these conditions and the crash routine.

A.1 SUCCESSFUL COMPLETION CODES

Table A-1 describes successful completion codes returned by RMS-11 operations.

Table A-1: RMS-11 Successful Completion Codes

Symbolic Value	Numeric Value	STV Field Value	Description
SU$SUC	1/1		Operation successful without qualification.
SU$DUP	2/2		Conditional success: A record inserted into an Indexed file by a put or update operation contains at least one key value present in another record.
SU$IDX	3/3	RMS-11 code	A put or update operation on an Indexed file ended successfully, but RMS-11 could not optimize the index structure during the operation. Therefore, RMS-11 will require extra I/O operations to retrieve the record. With this success code, RMS-11 can include an RMS-11 error code in the STV field of the control block to indicate why the index structure was not updated.
SU$RRV	4/4		No longer a valid completion code. See ER$RVU in Table A-2.

A.2 ERROR COMPLETION CODES

Table A-2 shows:

- the RMS-11 error completion codes in alphabetical order by symbolic value
- the numeric values of the codes in both octal and decimal radix
- a brief explanation of the cause of the error

When Table A-2 indicates that the STV field contains a file processor code (with the term fipcode), refer to the description of such codes in one of the following manuals:

- Error code appendix of the IAS/RSX-11 I/O Operations Reference Manual
- User recoverable error messages in an appendix of the RSTS/E Programming Manual. Note that the value returned in STV is the negative of the decimal value shown in the Programming Manual. That is, if STV contains "-20.," look up "20."

When Table A-2 indicates that the Status Value field contains an FSS directive error code (with the term fsscode), you should refer to the error code appendix of the RSTS/E System Directives Manual for a description of the codes.

Table A-2: RMS-11 Error Completion Codes

Symbolic Value	Numeric Value	STV Field Value	Description
ER$ABO	177760/-16.	ER$STK/ER$MAP	Operation aborted: Stack save area exhausted or memory resident control structures corrupted.
ER$ACC	177740/-32.	fipcode	File processor error: File processor could not access the file.
ER$ACT	177720/-48.	File activity	precludes operation. Example You attempted to close a file before an asynchronous record operation finished.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$AID	177700/-64.	XAB address	The area indicated by the XAB does not exist in the file.
ER$ALN	177660/-37.	XAB address	Illegal alignment value for Placement Control.
ER$ALQ	177640/-96.	XAB address	Illegal allocation quantity: The quantity exceeds 65K blocks during file creation on a non-Large Files RSTS/E system or equals zero during an explicit file extension operation.
ER$ANI	177620/-112.		Records in a file on ANSI-labeled magnetic tape are variable-length, but not in ANSI-D format.
ER$AOP	177600/-128.	XAB address	Invalid type of allocation.
ER$AST	177560/-144.		Invalid operation at AST level: You attempted to issue a synchronous operation from an asynchronous record operation completion routine.
ER$ATR	177540/-160.	fipcode	File processor error: Read failure on file attributes.
	177530/-168		Invalid File ID. See ER$FID.
ER$ATW	177520/-176.	fipcode	File processor error: Write failure on file attributes.
ER$BKS	177500/-192.		File bucket size exceeds maximum for operating system.
ER$BKZ	177460/-208.	XAB address	Area bucket size exceeds maximum for operating system.
ER$BLN	177440/-224.		Control block (FAB, RAB, or XAB) length is invalid.
ER$BOF	177430/-232.		Beginning of file detected during magnetic tape spacing operation.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$BPA	177420/-240.		Invalid I/O buffer: Private buffer pool not on word boundary.
ER$BPS	177400/-256.		Invalid I/O buffer: Private buffer pool size not a multiple of two bytes.
ER$BUG	177360/-272.		RMS-11 aborts your task because it detected an internal error. Contact a DIGITAL Software Specialist.
ER$CCR	177340/-288.		You attempted to connect more than one record access stream to a Sequential file.
ER$CHG	177320/-304.		During an update operation, you attempted to change a key field that does not allow changes.
ER$CHK	177300/-320.		Indexed file bucket corrupted. Do as many of the following steps as necessary: 1. Move the disk pack containing the file to another device and try the process again. If it works, the error was caused by a hardware read failure. 2. Recreate the file using the RMSIFL or RMSCNV utility. If this works, the corrupted bucket was in an index bucket not used during sequential access by Primary Key. 3. Restore the file from your last backup copy.
ER$CLS	177260/-336	fipcode	File processor error: During RMS-11 file close operation.
ER$COD	177240/-352.	XAB address	XAB type is invalid for the organization or operation.
ER$CRE	177220/-368.	fipcode	File processor error: File processor could not create file.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$CUR	177200/-384.		No Current Record: Delete, truncate, or update operation was not immediately preceded by a successful get or find.
ER$DAC	177160/-370.	fipcode	File processor error: File processor deaccess failure during RMS-11 file close operation.
ER$DAN	177140/-416.	XAB address	Invalid area number specified in Key XAB DAN field.
ER$DEL	177120/-432.		Record accessed by RFA access mode has been deleted.
ER$DEV	177100/-448.		• Syntax error in device name • No such device • Inappropriate device for operation Example You attempted to create an Indexed file on magnetic tape.
ER$DFW	177070/-456.	fipcode	File processor error: File processor could not write bucket; RMS-11 deferred the I/O operation until it needed the I/O buffer for another bucket because the user program specified Deferred Write.
ER$DIR	177060/-464.		Syntax error in filespec directory name.
ER$DME	177040/-480.		Dynamic memory exhausted: An RMS-11 buffer pool has insufficient free space.
ER$DNF	177020/-496.		Directory not found.
ER$DNR	177000/-512.		Device not ready.
ER$DPE	176770/-520.	fipcode	Device positioning error.
ER$DTP	176760/-528.	XAB address	Invalid key data type.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$DUP	176740/-544.		Invalid record operation: You attempted to insert a record that would cause duplicate values in a key field where duplicates are not allowed.
ER$ENT	176720/-560.	fipcode	File processor error: File processor could not enter filespec in directory.
ER$ENV	176700/-576.		You attempted an operation when the RMS-11 routines were not in the task: In MACRO-11, the operation or file organization was not specified in ORG$ macro; in BASIC-PLUS-2, you didn't specify correct switches with BUILD command.
ER$EOF	176660/-592.		• For record processing: End of file. • For Block I/O: Invalid VBN.
ER$ESS	176640/-608.		Expanded file-name string area in NAM Block is too short.
ER$EXP	176630/-616.		File expiration date not reached.
ER$EXT	176620/-624.	fipcode	File processor error: During RMS-11 file extension operation.
ER$FAB	176600/-640.		FAB BID field does not contain FB$BID.
ER$FAC	176560/-656.		Invalid record operation: Operation does not match access declaration made when file was created or opened.
ER$FEX	176540/-672.		You tried to create a file that exists.
ER$FID	177530/-168.		Invalid file ID.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$FLG	176520/-688.	XAB address	Invalid combination of key characteristics. Example No duplicate key values, but key values can change during update operations.
ER$FLK	176500/-704.		File locked by another user: You cannot access the file because your sharing specifications cannot be met.
ER$FND	176460/-720.	fipcode	File processor error: File processor could not find filespec in specified directory.
ER$FNF	176440/-736.		File not found during file open operation.
ER$FNM	176420/-752.		Syntax error in file-name.
ER$FOP	176400/-768.		Invalid file access option specified in FAB FOP field.
ER$FSS	176370/-776.	fsscode	RSTS/E monitor error: the File-name String routine is unable to parse the file-name string supplied by RMS-11.
ER$FUL	176360/-784.		Device full: RMS-11 cannot create or extend file.
ER$IAN	176340/-370.	XAB address	Invalid area number specified in Key XAB IAN field.
ER$IDX	176320/-816.		Specified index was not created. This code can only occur in the STV field when STS contains ER$RNF.
ER$IFI	176300/-832.		FAB IFI field contains invalid value.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$IMX	176260/-848.	XAB address	More than 254 keys and/or areas defined or multiple Summary, Protection, or Date/Time XABs present during operation.
ER$INI	176240/-864.		$INIT or $INITIF macro call never issued.
ER$IOP	176220/-880.		Illegal operation. Example You attempted to truncate a nonSequential file. Example You attempted to delete or extend a magnetic tape file. Example You issued a Block I/O operation to a stream not connected for block operations. Example You issued a record operation to a stream connected for Block I/O operations.
ER$IRC	176200/-896.		Illegal record encountered in Sequential file: Record-length field is invalid.
ER$ISI	176160/-912.		RAB ISI field is invalid. You may have altered it or failed to connect the stream.
ER$KBF	176140/-928.		No key specified during random record operation: RAB KBF field equals 0.
ER$KEY	176120/-944.		Negative Relative Record Number during random operation or bad format in packed decimal key value.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$KRF	176100/-960.		Invalid key of reference: • During random get or find operation. • During connect or rewind operation: Error code is returned for the first sequential get or find operation following the connect or rewind.
ER$KSZ	176060/-976.		Invalid key size.
ER$LAN	176040/-992.	XAB address	Invalid area number specified in Key XAB LAN field.
ER$LBL	176020/-1008.		Invalid medium: Magnetic tape is not labeled in accordance with ANSI standards.
ER$LBY	176000/-1024.		Logical channel busy: You attempted to create or open a file using a logical channel in use; that is, you already opened a file on that channel.
ER$LCH	175760/-1040.		Invalid logical channel or unit number.
ER$LEX	175750/-1048.	XAB address	You attempted to extend an area containing an unused extent.
ER$LOC	175740/-1056.	XAB address	Invalid location during Placement Control.
ER$MAP	175720/-1072.		Memory-resident data structures, such as I/O buffers, corrupted. This code can occur in the STV field when Status Code contains ER$ABO.
ER$MKD	175700/-1088.	fipcode	File processor error: File processor could not mark file for deletion.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$MRN	175660/-1104.		• Maximum Record Number field contains a negative value during creation of Relative file. • Relative Record Number for random operation to Relative file exceeds Maximum Record Number specified when file was created.
ER$MRS	175640/-1120.		Maximum Record Size is zero during file creation and one of the following is true: • Record format is fixed • File organization is Relative.
ER$NAM	175620/-1136.		Odd address in FAB NAM field on file open, creation, or erase operation.
ER$NEF	175600/-1152.		You attempted a put operation to a Sequential file when stream is not positioned to end-of-file.
ER$NID	175560/-1168.		Dynamic memory exhausted: Not enough buffer area to open an Indexed file.
ER$NPK	175540/-1184.		You attempted to create an Indexed file without defining a Primary Key.
ER$OPN	175520/-1200.	fipcode	File processor error: During RMS-11 file open operation.
ER$ORD	175500/-1216.	XAB address	XABs not ordered properly.
ER$ORG	175460/-1232.		Invalid file organization.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$PLG	175440/-1248.		A Prologue block may be corrupted. Do as many of the following steps as necessary: 1. Move the disk pack containing the file to another device and try the process again. If it works, the error was caused by a hardware read failure. 2. Recreate the file using the RMSIFL or RMSCNV utility. 3. Restore the file from your last backup copy.
ER$POS	175420/-1264.	XAB address	You specified a key position beyond the end of the record.
ER$PRM	175400/-1280.	XAB address	File directory entry contains date and time information not semantically correct. The file may be corrupted. Recreate field using RMSIFL or RMSCNV utility.
ER$PRV	175360/-1296.		Privilege violation: access to the file denied by the operating system.
ER$RAB	175340/-1312.		RAB BID field does not contain RB$BID.
ER$RAC	175320/-1328.		Invalid or illogical record access option specified in RAB RAC field.
ER$RAT	175300/-1344.		You specified both Carriage Return control and FORTRAN forms control.
ER$RBF	175260/-1360.		RAB RBF field contains an odd address (Block I/O access only).
ER$RER	175240/-1376.	fipcode	File processor error: • In record processing: Read failure on file block. • In Block I/O, VBN = 0, an illegal value.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$REX	175220/-1392.		Record exists: During a put operation in random mode to a Relative file, you tried to insert a record into a cell containing a record.
ER$RFA	175200/-1408.		Invalid RFA during RFA access.
ER$RFM	175160/-1424.		Invalid record format.
ER$RLK	175140/-1440.		Target bucket locked by another task or another stream in the same program.
ER$RMV	175120/-1456.	fipcode	File processor error: File processor could not remove filespec from directory.
ER$RNF	175100/-1472.	ER$IDX	Record specified during random get or find operation does not exist in Relative or Indexed file. For Indexed files only, STV may contain ER$IDX.
ER$RNL	175060/-1488.		You initiated a free operation, but no bucket was locked.
ER$ROP	175040/-1504.		Invalid record processing option or illogical combination of values specified in RAB ROP field.
ER$RPL	175020/-1520.	fipcode	File processor error: Read failure on file Prologue.
ER$RRV	175000/-1536.		Invalid RRV record encountered in Indexed file. File may be corrupted. Recreate field using RMSIFL or RMSCNV utility.
ER$RSA	174760/-1552.		Record stream active: In asynchronous environment, you attempted a record operation on a stream that is performing an operation.
ER$RSZ	174740/-1568.		• Record size is zero during Block I/O.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
			• Record size exceeds one of the following: -- MRS for variable-length or VFC records. -- magnetic tape block size. -- data bucket size of Indexed file. -- 510(10) bytes and block spanning not allowed. -- end of I/O buffer during Locate Mode put operation on Sequential file. • Record is too short to contain Primary Key of Indexed file. • Record size is not equal to size of Current Record for update operation on a disk Sequential file or on an Indexed file where Primary Key duplicate are allowed. • Record size does not equal MRS for fixed-length records.
ER$RTB	174720/-1584.	Bytes moved	Record too big for user buffer: RMS-11 could not move entire record retrieved by get operation to user buffer. This error does not destroy the context of the stream. Rather, the stream's context is updated as if the operation were completely successful and as much of the record as possible is moved to the user buffer.
ER$RVU	174710/-1592.		During a put or update operation, RMS-11 moved the specified record to the file successfully, but could not update one or more Record Reference Vector (RRV). The file is corrupted, but you can retrieve every record via the Primary index. Therefore, you should create a new Indexed file and populate it from the bad file using either the RMSIFL or the RMSCNV utility.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$SEQ	174700/-1600.		Invalid record operation: During a sequential put operation to an Indexed file, Primary Key of record to be written is not equal to or greater than key of previous record.
ER$SHR	174660/-1616.		You specified allow write declaration for Sequential file.
ER$SIZ	174640/-1632.	XAB address	Invalid key size. Example Key is bigger than Maximum Record Size.
ER$STK	174620/-1648.		During asynchronous record operation, RMS-11 found the stack too big to saved. This code can only occur in the STV field when Status Code contains ER$ABO.
ER$SYS	174600/-1664.	Directive or QIO status error code	The interface between RMS-11 and the operating system has changed. Report this error on an SPR to DIGITAL.
ER$TRE	174560/-1680.		Index in Indexed file is corrupted. Recreate file using RMSIFL or RMSCNV utility.
ER$TYP	174540/-1696.		Syntax error in extension, such as more than three characters specified.
ER$UBF	174520/-1712.		User buffer improperly specified: address is either zero or buffer is not word-aligned (for Block I/O access only).
ER$USZ	174500/-1728.		User buffer length equals zero.
ER$VER	174460/-1744		Syntax error in file version number.
ER$VOL	174440/-1760.	XAB address	Nonzero relative volume number.

(Continued on next page)

Table A-2 (continued)

Symbolic Value	Numeric Value	STV Field Value	Description
ER$WCD	174430/-1768.		Explicit or default file specification field contains wild card character.
ER$WER	174420/-1776.	fipcode	File processor error: Write failure on file block.
ER$WLK	174410/-1784.		File processor error: Device is write locked.
ER$WPL	174400/-1792.	fipcode	File processor error: Error while writing Prologue.
ER$XAB	174360/-1808.	XAB address	FAB XAB field or XAB NXT field contains an odd address.
ER$XTR	174340/-1824.		Explicit or default file specification contains extraneous field.

A.3 FATAL ERROR CRASH ROUTINE

RMS-11 issues a BPT instruction whenever it encounters an inconsistent internal conditions. This action is taken only when RMS-11 cannot continue processing because it might damage your files or task image.

Example When RMS-11 encounters an invalid FAB or RAB, it cannot perform a normal error routine because it has no recognizable control block to set a status code in.

The BPT instruction generated is in the R0RMSA module of RMS-11. The general registers contain the following values at the time this instruction is executed:

R0 = RMS-11 error code
R1 = Entry Stack Pointer value
R2 = Entry return Program Counter
R3 = Address of system impure area

General registers R1 and R2 are always valid if the crash routine is caused by a fatal user call error (described in Section A.3.1). When the crash routine is used because of inconsistent internal conditions (described in Section A.3.2), the contents of R1 and R2 can be meaningless if RMS-11 was executing an asynchronous record operation.

A.3.1 Fatal User Call Errors

If general register R0 contains 176600(8) (ER$FAB) or 175340(8) (ER$RAB), your program initiated an RMS-11 operation using an invalid control block: FAB for file operations and RAB for record operations. This condition can occur for any one of the following reasons:

- The address of the control block is 0.
- The address of the control block is odd.
- The control block's BID field does not contain the proper block identifier code: FB$BID for FABs and RB$BID for RABs.

Example On RSX-11M, a fatal user call error resulted in:

```
TASK "TTnn " TERMINATED
T BIT TRAP OR BPT EXECUTION
  PC=021144
  PS=170004
  R0=175340
  R1=001162
  R2=003646
  R3=045276
  R4=000000
  R5=001164
  SP=001126
```

A.3.2 RMS-11 Inconsistent Internal Conditions Errors

If general register R0 contains 177360(8) (ER$BUG) or 175720(8) (ER$MAP), RMS-11 encountered a fatal internal problem. Such conditions can be caused by your program (destroying some internal RMS-11 data base, for example), but RMS-11 can also be responsible.

If one of your tasks crashes with one of these error codes, send a Software Performance Report (SPR) to DIGITAL with the following information:

- The contents of the general registers
- The first ten words, at a minimum, or all words on the system stack
- The operation your task was performing (open, get, put, and so on)
- The organization of the file being processed
- A load map of the task (produced by the Task Builder utility)
- A post-mortem dump

APPENDIX B

SAMPLE CODE SEGMENTS

The programs and code segments included in this appendix work; that is, they run, performing the indicate RMS-11 operations, without error. They are samples of programs that do run, but you should not construe them to be examples of approved programming techniques.

B.1 COPYING SEQUENTIAL FILES

The sample code segments in this section demonstrate the copying of records from an existing Sequential file to a new Sequential file.

```
; DEMO.RMS
;
 PROGRAM TO COPY RECORDS FROM A SEQUENTIAL FILE NAMED
; FILE1.DAT  TO A NEW SEQUENTIAL FILE NAMED  FILE2.DAT
;
; STEP 1:       ACCESS THE NECESSARY RMS MACROS
        .MCALL  $INIT,ORG$,FAB$B,RAB$B,POOL$B,$CREATE,$OPEN,$CLOSE
        .MCALL  $CONNECT,$GET,$PUT,$FETCH,$STORE,$COMPARE
        ;
;STEP 2:        DEFINE CONTROL BLOCKS AND FILE NAME STRINGS
        ;
        ;THE FAB FOR THE INPUT FILE, WHICH ALREADY EXISTS, WILL BE
        ;FILLED WITH MOST OF THE ESSENTIAL INFORMATION CONCERNING
        ;FILE1.DAT WHEN THE FILE IS OPENED. THE PROGRAM NEED ONLY
        ;SPECIFY ENOUGH INFORMATION TO OPEN THE FILE.
        ;
FAB1:   FAB$B           ;FAB FOR FILE1.DAT
              F$FNA     NAME1   ;ADDRESS OF NAME STRING
              F$FNS     9       ;STRING IS 9 CHARACTERS LONG
              F$LCH     1       ;ACCESS ON CHANNEL 1
              FAB$E             ;END OF FAB1
NAME1: .ASCII  /FILE1.DAT/      ;NAME STRING FOR FAB1
       ;HERE, AND WITH FILE2, WE ASSUME THAT THE FILE EXISTS
       ;ON THE SYSTEM DISK UNDER THE ACCOUNT ON WHICH WE ARE
       ;LOGGED IN.
       .EVEN            ;(CONTROL BLOCKS MUST BE WORD ALIGNED)
       ;
       ;THE FAB FOR THE OUTPUT FILE, WHICH DOES NOT YET EXIST, MUST
       ;BE FILLED WITH THE INFORMATION NECESSARY TO CREATE IT (AS IN
       ;THE PREVIOUS CASE, SOME FIELDS SIMPLY CONTAIN DEFAULT VALUES
       ;AND ARE NOT EXPRESSED EXPLICITLY). SOME OF THIS INFORMATION
```

```
        ;WILL DEPEND ON THE CHARACTERISTICS OF FILE1.DAT, AND MUST BE
        ;FILLED IN AT RUN-TIME AFTER FILE1.DAT HAS BEEN OPENED.
FAB2:   FAB$B           ;FAB FOR FILE2.DAT
        F$FNA   NAME2   ;ADDRESS OF NAME STRING
        F$FNS   9       ;STRING IS 9 CHARACTERS LONG
        F$LCH   2       ;ACCESS ON CHANNEL 2
        F$FAC   FB$PUT  ;WRITE ACCESS REQUIRED
        FAB$E           ;END OF FAB2
NAME2:  .ASCII  /FILE2.DAT/     ;NAME STRING FOR FAB2
        .EVEN
RAB1:   RAB$B           ;RAB FOR FILE1.DAT
        R$FAB   FAB1    ;ADDRESS OF OWNER (FAB)
        R$RAC   RB$SEQ  ;SPECIFY SEQUENTIAL ACCESS
        R$UBF   RECBUF  ;ADDRESS OF RECORD BUFFER FOR $GETS
        R$USZ   500     ;SIZE OF THIS BUFFER (500. BYTES)
        R$RHB   HEDBUF  ;ADDRESS OF RECORD HEADER BUFFER
                        ;(NECESSARY FOR VFC RECORDS ONLY)
        RAB$E           ;END OF RAB1
RAB2:   RAB$B           ;RAB FOR FILE2.DAT
        R$FAB   FAB2    ;ADDRESS OF OWNER
        R$RAC   RB$SEQ  ;SPECIFY SEQUENTIAL ACCESS
        R$RBF   RECBUF  ;ADDRESS OF RECORD BUFFER FOR $PUTS
        R$RSZ   500     ;SIZE OF THIS BUFFER (500. BYTES)
        R$RHB   HEDBUF  ;ADDRESS OF RECORD HEADER BUFFER
                        ;(NECESSARY FOR VFC RECORDS ONLY)
        RAB$E           ;END OF RAB2
        ;
;STEP 3:        ALLOCATE THE BUFFERS SPECIFIED ABOVE
RECBUF: .BLKW   250.
HEDBUF: .BLKW   128.
        ;
;STEP 4:        GENERATE RMS INTERNAL SPACE POOL
        POOL$B          ;BEGIN POOL SPECIFICATION
        P$FAB   2       ;A FAB FOR EACH FILE
        P$RAB   2       ;A RAB FOR EACH FAB
        P$BDB   2       ;AN I/O BUFFER FOR EACH RAB
        P$BUF   1024    ;MINIMUM BUFFER SIZE IS 512. BYTES
        POOL$E          ;END OF POOL SPECIFICATION
        ;
;STEP 5:        DEFINE THE RMS FUNCTIONALITY REQUIRED
        ORG$    SEQ,<CRE,GET,PUT>       ;SEQUENTIAL FILES ONLY, $FIND,
                                ;$UPDATE, $DELETE NOT REQUIRED
        ;
;STEP 6:        PROVIDE A GENERAL ERROR ROUTINE TO HANDLE UNEXPECTED
        ;       ERRORS WHICH MIGHT OCCUR (WHAT IF FILE1.DAT DID NOT
        ;       EXIST, OR CONTAINED A RECORD LARGER THAN 500 BYTES?).
        ;       THIS IS AN ALTERNATIVE TO THE 'COMPLETION ROUTINE'
        ;       FUNCTION PROVIDED BY RMS.
ERROR:  ;(CODE WHICH WILL HANDLE THE ERROR, PROMPT AT THE TERMINAL FOR
        ;FURTHER INSTRUCTIONS, ETC.)
        ;
        ;
;STEP 7:        WRITE THE PROGRAM
        ;
START:  $INIT                   ;INITIALIZE RMS.
        $OPEN   #FAB1           ;OPEN FILE1.DAT,
        MOV     #FAB1,R0        ;SET UP FOR $COMPARE:
```

```
        $COMPARE #0,STS,R0      ;NEG STS VALUE MEANS OPEN FAILURE
        BGT     1$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR        ;OTHERWISE EXECUTE ERROR ROUTINE.
1$:     MOV     #FAB2,R1        ;COMPLETE INITIALIZATION OF FAB2:
        $FETCH  R2,RAT,R0       ;GET RAT FIELD FROM FAB1
        $STORE  R2,RAT,R1       ;AND MOVE IT INTO FAB2;
        $FETCH  R2,RFM,R0       ;DO THE SAME WITH THE RFM FIELD;
        $STORE  R2,RFM,R1
        $FETCH  R2,FSZ,R0       ;FSZ IS PERTINENT ONLY IF FILE1.DAT
        $STORE  R2,FSZ,R1       ;MAY CONTAIN VFC RECORDS.
        $FETCH  R2,MRN,R0       ;YOU MAY OR MAY NOT WISH TO TRANSFER
        $STORE  R2,MRN,R1       ;MRN, MRS, AND FOP. IF MRN IS COPIED,
        $FETCH  R2,MRS,R0       ;REMEMBER THAT IT IS A 2-WORD FIELD,
        $STORE  R2,MRS,R1       ;AND WILL DESTROY THE CONTENTS OF R3.
        ;
        ;INITIALIZATION OF FAB2 SHOULD NOW BE COMPLETE EXCEPT FOR ANY
        ;SPECIAL CASES (FOR EXAMPLE, IF FILE2.DAT IS ON MAGNETIC TAPE,
        ;YOU MAY WISH TO SET THE BLS FIELD).
        $FETCH  R2,ALQ,R0
        $STORE  R2,ALQ,R1
        $CREATE R1              ;NOW CREATE FILE2.DAT (R1=FAB2):
        $COMPARE #0,STS,R1      ;CHECK FOR FAILURE
        BGT     2$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR        ;OTHERWISE EXECUTE ERROR ROUTINE.
2$:     MOV     #RAB1,R0        ;CONNECT THE RABS.
        MOV     #RAB2,R1
        $CONNECT        R0
        $COMPARE #0,STS,R0
        BGT     3$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR
3$:     $CONNECT R1
        $COMPARE #0,STS,R1
        BGT     4$
        JSR     PC,ERROR
4$:     $GET    R0              ;GET A RECORD FROM FILE1.DAT.
        $COMPARE #ER$EOF,STS,R0 ;WERE WE AT END-OF-FILE?
        BEQ     DONE            ;IF SO, CLEAN UP AND EXIT.
        $COMPARE #0,STS,R0      ;SOME OTHER ERROR?
        BGT     5$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR        ;IF SO, HANDLE IT.
5$:     $FETCH  R2,RSZ,R0       ;COPY RECORD LENGTH FROM
        $STORE  R2,RSZ,R1       ;RAB1 TO RAB2.
        $PUT    R1              ;OUTPUT THE RECORD TO FILE2.DAT.
        $COMPARE #0,STS,R1
        BGT     6$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR
6$:     BR      4$              ;LOOP UNTIL DONE.
DONE:   MOV     #FAB1,R0        ;BACK TO THE FABS FOR $CLOSE.
        MOV     #FAB2,R1
        $CLOSE  R0
        $COMPARE #0,STS,R0
        BGT     1$              ;BRANCH IF SUCCESSFUL
        JSR     PC,ERROR
1$:     $CLOSE  R1
        $COMPARE #0,STS,R1
        BGT     2$
        JSR     PC,ERROR
```

```
2$:
        ;RMS IS NOW DONE:  FILE2.DAT NOW CONTAINS ALL THE DATA RECORDS
        ;OF FILE1.DAT, PLUS OTHER INFORMATION (MRS, MRN, ETC.)
        ;YOU CHOSE TO DUPLICATE. INSERT YOUR OWN EXIT CODE AND EXIT.
        .END    START
```

B.2 COUNTING WORDS IN A SEQUENTIAL FILE

This sample program reads through a sequential file named RNO.DOC, noting pages and counting words. The program uses RMS-11 for terminal I/O.

```
        .TITLE  WORDS.MAC
        .SBTTL  READ A DOCUMENTATION FILE AND COUNT THE WORDS
        .ENABL  REG                     ; SO CAN USE R0, R1, ETC.
        .NLIST ME
        .NLIST BIN
        ;
        ; PROGRAM TO READ A DOC FILE PRODUCED BY RUNOFF OR RNO AND
        ; COUNT THE WORDS,
        ; STORING ENTRIES IN AN RMS ISAM FILE
        ;
        ;
        .MCALL  EXIT$S
        .MCALL  ORG$,POOL$B,$INIT
        .MCALL  $GNCAL,FAB$B,RAB$B,XAB$B
        .MCALL  $FBCAL,$RBCAL
        $GNCAL
        $FBCAL
        $RBCAL
        ;
        ; THE FAB FOR THE INPUT DOC FILE, NAMED RNO.DOC, MUST
        ; COMPLETELY DESCRIBE THE FILE AT ASSY TIME TO SAVE RUN TIME.
        ;
        .SBTTL RMS-11 DATA BLOCKS
        .PAGE
        .EVEN
        DOCFAB: FAB$B                   ; BEGIN MINIMUM INITIALIZATION
                F$BPA   0               ; NO PRIVATE BUFFER POOL
                F$BPS   0               ; DITTO
                F$DNA   0               ; USE SYSTEM DEFAULTS
                F$DNS   0               ; DITTO
                F$FAC   FB$GET          ; ONLY GET RECORD OPERATIONS
                F$FNA   DOCNAM          ; FILE NAME ADDRESS
                F$FNS   DOCNML          ; FILE NAME LENGTH
                F$FOP   0               ; NOT CONTIGUOUS
                F$LCH   1               ; LOGICAL CHANNEL FOR FILE  1
                F$NAM   0               ; NO NAM BLOCK
                F$ORG   FB$SEQ          ; FILE ORGANIZATION
                F$RFM   FB$STM          ; DOC FILES ARE STREAM
                F$RTV   0               ; VOLUME CLUSTERSIZE
                F$SHR   0               ; NOBODY ELSE
                F$XAB   0               ; NO XABs
                FAB$E                   ; END INITIALIZATION BLOCK
        DOCNAM: .ASCII /RNO.DOC/        ; ASSUME SYSTEM DEFAULTS
        DOCNML = .-DOCNAM
        ;
        ; NEED CRT TO OUTPUT MESSAGES TO TERMINAL RUNNING JOB
        ;
        .EVEN
        CRTFAB: FAB$B                   ; BEGIN MINIMUM INITIALIZATION
                F$FAC   FB$PUT          ; OUTPUT ONLY
```

```
        F$FNA   CRTNAM          ; FILE NAME ADDRESS
        F$FNS   CRTNML          ; FILE NAME LENGTH
        F$LCH   3               ; LOGICAL CHANNEL FOR FILE  3
        F$ORG   FB$SEQ          ; FILE ORGANIZATION
        F$RFM   FB$VAR          ; CRTS ARE VARIABLE
        FAB$E                   ; END INITIALIZATION BLOCK
CRTNAM: .ASCII /TI:/            ; ASSUME SYSTEM DEFAULTS
CRTNML = .-CRTNAM
;
; INPUT RECORD IS SINGLE CHARACTER STRING, VARIABLE IN LENGTH
; THOUGH MOST WILL BE 70 CHARACTERS LONG (MARGINS 0  70).
;
.EVEN
DOCRAB: RAB$B                   ; BEGIN INITIALIZATION BLOCK
        R$FAB   DOCFAB          ; ADDRESS OF ASSOCIATED FAB
        R$MBC   1               ; ONE AT A TIME
        R$MBF   1               ; MULTI-BUFFER COUNT
        R$RAC   RB$SEQ          ; RECORD ACCESS MODE(S)
        R$UBF   DOCRCD          ; USER RECORD BUFFER ADDRESS
        R$USZ   DOCRCL          ; USER-SET RECORD SIZE
        RAB$E                   ; END INITIALIZATION BLOCK
;
CRTRAB: RAB$B                   ; BEGIN INITIALIZATION BLOCK
        R$FAB   CRTFAB          ; ADDRESS OF ASSOCIATED FAB
        R$RAC   RB$SEQ          ; RECORD ACCESS MODE
        R$ROP   RB$EOF
        RAB$E                   ; END INITIALIZATION BLOCK
.SBTTL BUFFERS AND MESSAGES
.PAGE
; BUFFERS
;
DOCRCD: .REPT   80.             ; INPUT RECORD BUFFER
        .BYTE   040             ; FILL WITH SPACES
        .ENDR
DOCRCL = .-DOCRCD
.EVEN
 WDCNT: .WORD   0               ; COUNT WORDS IN FILE
CHRCNT: .WORD   0               ; COUNT CHAR IN RCDS AS DONE
RECLEN: .WORD   0               ; STORE RCD LENGTH AFTER RMS
                        ; RETURNS IT
        .WORD   0               ; STOPPER FOR SCAN
XLTBUF: .WORD   1,10.,100.,1000.,10000.,0 ; POWERS OF 10
PAGENO: .WORD   0               ; PAGE COUNTER
LINENO: .WORD   0               ; LINE COUNTER
;
; ARGUMENTS LISTS FOR RMS I/O OPERATIONS
;
DOCLST: .WORD   1               ; LENGTH OF ARGUMENT LIST
        .WORD   DOCRAB          ; RAB FOR RECORD BEING MOVED
CRTLST: .WORD   1               ; LENGTH OF ARGUMENT LIST
        .WORD   CRTRAB          ; RAB FOR RECORD BEING MOVED
;
; VECTOR TABLES
;
CHCODE: .WORD   014             ; FORM FEED
        .WORD   040             ; SPACE
        .WORD   055             ; HYPHEN
```

```
CHCEND = .-CHCODE
;
CHPRCS: .WORD   FFEED
        .WORD   SPACE
        .WORD   HYFEN
        .WORD   NOTSPA
;
.NLIST  BIN
;
; MESSAGES
;
ERRMSG: .ASCII  /    ERROR /
ERRNBR: .ASCII  /000000/        ; FILL IN ERROR CODES
ERRMSL = .-ERRMSG
BGNMSG: .ASCIZ  /START WORDS PROGRAM /
BGNMSL = .-BGNMSG
ENDMSG: .ASCIZ  /END WORDS PROGRAM /
ENDMSL = .-ENDMSG
OPNMSG: .ASCII  /RNO.DOC OPENED/
OPNMSL = .-OPNMSG
FFMSG:  .ASCII  /PAGE   /
FFMSL = .-FFMSG
WDMSG:  .ASCIZ  /   WORDS IN THIS LINE.  TOTAL WORDS =     /
WDMSL = .-WDMSG
LINMSG: .ASCII  /LINE    IS TOO LONG./
        .ASCIZ /  CONTINUING WITH NEXT LINE./
LINMSL = .-LINMSG
BLKS3:  .ASCIZ  /   /
;
FILID:  .ASCIZ  /FILE/
DOCID:  .ASCIZ  /DOC /
STVID:  .ASCIZ  /STV /
.SBTTL MACROS
.PAGE
.LIST   BIN
;
; MACROS
;
.MACRO  MOVWD   A,B,?C
        MOV      A,R2           ; MOVE STRING A TO THE AREA
                           ; DESIGNATED BY B, COUNTING THE
        MOV      B,R3           ; CHARACTERS AS YOU GO
        CLR     R1              ; INITIALIZE COUNTER
   C:   MOVB    (R2)+,(R3)+     ; MOVE ONE CHARACTER
        INC     R1              ; COUNT IT
        CMPB     0,(R2)         ; CHECK NEXT SOURCE WORD CHAR
        BNE     C               ; GO BACK FOR MORE IF NOT ZERO
.ENDM
;
.MACRO  XLT28   A,B             ; 16-BIT NBR TO OCTAL ASCII
        MOV     R0,-(SP)        ; STORE R0 AWAY
        MOV      5,R0           ; 5 IS NBR OF CHARS IN BFR
        MOV     A,R1            ; PUT IN NBR
        MOV      B,R2           ; MOVE IN RCV BFR
        ADD      6,R2           ; MOVE TO END OF RCV BFR
        JSR     PC,XLT8         ; DO IT
        MOV     (SP)+,R0        ; BRING R0 BACK FROM THE STACK
```

```
.ENDM
;
.MACRO  XLT210  A,B             ; 16-BIT NBR TO DECIMAL ASCII
        MOV     R0,-(SP)        ; SAVE REGISTER ON STACK
        MOV     R1,-(SP)        ; SAVE REGISTER ON STACK
        MOV     R2,-(SP)        ; SAVE REGISTER ON STACK
        MOV     R3,-(SP)        ; SAVE REGISTER ON STACK
        MOV     R4,-(SP)        ; SAVE REGISTER ON STACK
        MOV     R5,-(SP)        ; SAVE REGISTER ON STACK
        MOV     A,R0            ; PUT IN NUMBER
        MOV      B,R5           ; LOCATION OF 1ST BYTE RCV BFR
        JSR     PC,XLT10        ; TRANSLATE IT
        MOV     (SP)+,R5        ; GET REGISTER BACK FROM STACK
        MOV     (SP)+,R4        ; GET REGISTER BACK FROM STACK
        MOV     (SP)+,R3        ; GET REGISTER BACK FROM STACK
        MOV     (SP)+,R2        ; GET REGISTER BACK FROM STACK
        MOV     (SP)+,R1        ; GET REGISTER BACK FROM STACK
        MOV     (SP)+,R0        ; GET REGISTER BACK FROM STACK
.ENDM
;
.MACRO  WRTOUT  A,B             ; OUTPUT MSG AND LENGTH
        MOV     R4,-(SP)        ; PUT RAB REFERENCE ON STACK
        MOV     R5,-(SP)        ; OUT ARG LIST REF ON STACK
        MOV      CRTRAB,R4      ; PUT CRT RAB REFERENCE IN R4
        $STORE   A,RBF,R4       ; MSG NAME IN RAB FLD
        $STORE   B,RSZ,R4       ; PUT MSG LENGTH IN RAB FLD
        MOV      CRTLST,R5      ; PUT CRT ARG LIST IN R5
        $PUT                    ; WRITE MESSAGE TO CRT
        MOV     (SP)+,R5        ; POP OFF ORIGINAL ARG LIST
        MOV     (SP)+,R4        ; POP OFF ORIGINAL RAB REF
.ENDM
.SBTTL MAIN PROGRAM
.PAGE
; INTERNAL SPACE POOL
;
.EVEN
        POOL$B                  ; BEGIN INITIALIZATION
        P$BDB   2               ; NBR =
                ; MAXBUF(1/SEQ)+MAXREL(0)+(2*MAXIDX(0))
        P$BUF   1024.           ; NBR =
                        ; DOC-STREAM-SIZE + CRT-STREAMS-SIZE
                        ; DOC-STREAM-SIZE =
                        ; BKS(1) * (512*MBC(1)) * MBF(1) = 512
                        ; CRT-STREAM-SIZE =
                        ; BKS(1) * (512*MBC(1)) * MBF(1) = 512
        P$FAB   2               ; FAB FOR EACH FILE
        P$RAB   2               ; TWO SEQUENTIAL FILES
        POOL$E                  ; END INITIALIZATION AREA
;
; RMS FUNCTIONS NEEDED
;
        ORG$    SEQ,<GET,PUT,FIN> ; SEQ FILE GETS, PUT,  FINDS
;
; !!!!!!!!!!!!!!  R U N - T I M E  !!!!!!!!!!!!!!!!!!!!
;
.EVEN
START:  $INIT                   ; INITIALIZE RMS
```

```
;
;
        $OPEN    CRTFAB          ; OPEN CRT
        MOV      CRTFAB,R4       ; SET BLOCK TO LOOK AT
        $COMPARE  0,STS,R4       ; CHECK OUT STATUS FIELD
        BGT      2$              ; GO AROUND IF NO ERROR
        JMP      FILERR          ; ERROR CODE(S) AND QUIT
;
    2$: $CONNECT  CRTRAB         ; HOOK UP CRT
        MOV      CRTRAB,R4       ; SET BLOCK TO LOOK AT
        $COMPARE  0,STS,R4       ; CHECK OUT STATUS FIELD
        BGT      4$              ; GO AROUND IF NO ERROR
        JMP      FILERR          ; ERROR CODE(S) AND QUIT
;
    4$: WRTOUT   BGNMSG,BGNMSL   ; OUTPUT IT
;
;
        $OPEN    DOCFAB          ; OPEN INPUT FILE
        MOV      DOCFAB,R4       ; SET BLOCK TO LOOK AT
        $COMPARE  0,STS,R4       ; CHECK OUT STATUS FIELD
        BGT      8$              ; GO AROUND IF NO ERROR
        JMP      FILERR          ; ERROR CODE(S) AND QUIT
;
    8$: WRTOUT   OPNMSG,OPNMSL
        $CONNECT  DOCRAB         ; HOOK UP INPUT FILE
        MOV      DOCRAB,R4       ; SET BLOCK TO LOOK AT
        $COMPARE  0,STS,R4       ; CHECK OUT STATUS FIELD
        BGT      GETLIN          ; GO AROUND IF NO ERROR
        JMP      FILERR          ; ERROR CODE(S) AND QUIT
;
GETLIN: MOV      DOCRAB,R4       ; DOCERR GETS RIGHT CODE
        MOV      DOCLST,R5       ; DOC FILE ARG LIST
        JSR      PC,INPUT        ; GET RECORD FROM DOC FILE
;
; R0 = POINTER TO INPUT LINE
; R1 = CHARACTERS/WORD COUNTER (TO HANDLE MULTIPLE SPACES)
; R2 = WORDS/LINE COUNTER
; R3 = VECTOR OFFSET INTO TABLE
; R4 = DOCRAB
; R5 = DOCLST
;
;
        CLR      R1              ; INITIALIZE CHAR/WORD COUNTER
DOWORD: CLR      R3              ; INITIALIZE VECTOR OFFSET
        INC      CHRCNT          ; CNT CHAR BEFORE LOOK AT IT
;
        CMP      CHRCNT,RECLEN   ; PAST END OF INPUT RCD YET?
        BGT      12$             ; IF SO, GO FINISH UP
;
    4$: CMPB     CHCODE(R3),(R0) ; CHECK ON NEXT CHAR IN LINE
        BEQ      8$              ; PROCESSING IF EQUAL
        TST      (R3)+           ; INCREMENT VECTOR
        CMP      R3, CHCEND      ; NOT IN TABLE?
        BLT      4$              ; CHECK NEXT ONE
;
    8$: JSR      PC,@CHPRCS(R3)  ; DO RIGHT THING
        INCB     (R0)+           ; MOVE TO NEXT CHARACTER
```

```
        BR      DOWORD
;
   12$: CMP      0,R1            ; ANY CHARACTERS MOVED?
        BEQ     16$             ; NOT A REAL LAST WORD
        INC     R2              ; COUNT LAST WORD
   16$: CMP      0,R2            ; ANY WORDS IN THIS LINE?
        BEQ     18$             ; IF NOT, DON'T DO ANYTHING
        ADD     R2,WDCNT        ; ADD THIS LINE TO TOTAL
;       XLT210  R2,WDMSG        ; PUT CURRENT LINE IN MSG
;       XLT210  WDCNT,WDMSG+38. ; PUT TOTAL CURRENT IN MSG
;       WRTOUT  WDMSG,WDMSL     ; OUTPUT IT
;       MOVWD   BLKS3,WDMSG     ; COVER UP LAST WORD COUNT
   18$: JMP     GETLIN          ; GO BACK FOR NEXT LINE
;
.SBTTL SUBROUTINES
.PAGE
; ===========
; SUBROUTINES
; ===========
;
 INPUT: $GET                    ; GET RECORD FROM INPUT FILE
        $COMPARE  0,STS,R4      ; CHECK OUT STATUS FIELD
        BGT     2$              ; GO AROUND IF NO ERROR
        JMP     DOCERR          ; ERROR CODE(S) AND QUIT
;
    2$: $COMPARE  0,RSZ,R4      ; RECORD LENGTH = 0?
        BEQ     INPUT           ; IF SO, GO GET ANOTHER ONE
        MOV      RECLEN,R3      ; ADDRESS IN REGISTER
        $FETCH  (R3),RSZ,R4     ; XFR RCD LEN TO BBFR
        BR      8$              ; BYPASS WRITE OUT
;
; SPECIAL CASE OF WRTOUT HERE BECAUSE OF RECLEN
;
    6$: MOV     R4,-(SP)        ; PUT RAB REFERENCE ON STACK
        MOV     R5,-(SP)        ; OUT ARG LIST REF ON STACK
        MOV      CRTRAB,R4      ; PUT CRT RAB REFERENCE IN R4
        $STORE   DOCRCD,RBF,R4  ; PUT MSG NAME IN RAB FLD
        $STORE  RECLEN,RSZ,R4   ; PUT MSG LENGTH IN RAB FLD
        MOV      CRTLST,R5      ; PUT CRT ARG LIST IN R5
        $PUT                    ; WRITE MESSAGE TO CRT
        MOV     (SP)+,R5        ; POP OFF ORIGINAL ARG LIST
        MOV     (SP)+,R4        ; POP OFF ORIGINAL RAB REF
;
    8$: MOV      DOCRCD,R0      ; ADDR OF 1ST CHAR TO R0
        CLR     R2              ; INITIALIZE WORD COUNTER
        CLR     CHRCNT          ; INITIALIZE CHAR COUNTER
        INC     LINENO          ; COUNT LINE/RECORD
        RTS     PC
;
;
 FFEED: INC     PAGENO          ; ADD ONE TO PAGE NUMBER
        CLR     LINENO          ; RESET LINE NUMBER
        XLT210  PAGENO,FFMSG+5. ; TURN IT INTO DECIMAL
        WRTOUT  FFMSG,FFMSL     ; ANNOUNCE FORM FEED
        MOV      5.,R3          ; FIND CTR TO TOP OF TEXT
    2$: $FIND                   ; MOVE PAST A LINE
        $COMPARE  0,STS,R4      ; CHECK OUT STATUS FIELD
```

```
        BGT     4$              ; GO AROUND IF NO ERROR
        MOV     (SP)+,R0        ; PULL RTN ADDR OFF STACK
        JMP     DOCERR          ; ERROR CODE(S) AND QUIT
    4$: SOB     R3,2$           ; SUB ONE AND BRNCH IF <> 0
        JSR     PC,INPUT        ; READ NEXT RECORD
        DECB    R0              ; FOR COMMON INCREMENT
        CLR     R1              ; GET RID OF ANY RESIDUAL THAT
                                ; CAN COUNT WORD
        RTS     PC              ; PROCESS IT
;
;
NOTSPA: INC     R1              ; COUNT IT
        RTS     PC              ; LOOK AT NEXT CHARACTER
;
;
 HYFEN: CMP     CHRCNT,RECLEN   ; LAST CHARACTER IN LINE?
        BNE     NOTSPA          ; TREAT IT LIKE ANY OTHER
        MOV     R2,-(SP)        ; CIRCUMVENT INPUT
        JSR     PC,INPUT        ; BRING IT CONTINUATION LINE
                        ; (DON'T WORRY ABOUT DOUBLE-SPACING
                        ; SINCE INPUT THROWS AWAY BLANK LINE)
        MOV     (SP)+,R2        ; BRING WORD COUNT BACK
        DECB    R0              ; ANTICIPATE AUTO INCREMENT
        CLR     R1              ; GET RID OF ANY RESIDUAL THAT
                                ; CAN COUNT WORD
        RTS     PC              ; PROCESS IT
;
;
 SPACE: CMP      0,R1            ; ANY CHARACTERS MOVED?
        BEQ     4$              ; NONE WERE
        CLR     R1              ; CLEAR CHAR/WORD COUNTER
        INC     R2              ; ONE MORE WORD
    4$: RTS     PC              ; DO NEXT ONE
;
;
  XLT8: MOV     R1,-(SP)        ; USE STACK AS SCRATCH PAD
        BIC      B1111111111111000,@SP ; CLR ALL BUT LAST 3
        ADD      '0,@SP         ; CHANGE DIGIT TO ASCII
        MOVB    (SP)+,-(R2)     ; MOVE CHARACTER INTO BUFFER
        ASR     R1              ; 3 PLACES TO THE RIGHT
        ASR     R1
        ASR     R1
        SOB     R0,XLT8         ; LOOP IF MORE DIGITS
;
; NOW CONVERT LEFTMOST BIT IN OCTAL NUMBER
;
        BIC      B1111111111111110,R1 ; GET BIT 15
        ADD      '0,R1          ; CONVERT TO ASCII
        MOVB    R1,-(R2)        ; MOVE TO BUFFER
        RTS     PC              ; GO BACK
;
;
 XLT10: MOV      XLTBUF,R4      ; POINT TO POWERS OF 10
    2$: TST     (R4)            ; END OF TABLE?
        BEQ     4$              ; YES
        CMP     R0,(R4)+        ; START YET?
        BHIS    2$              ; NO
```

```
        TST     -(R4)           ; BACK UP TABLE
    4$: TST     -(R4)           ; DITTO
        BNE     6$              ; NON-ZERO NUMBER TO PRINT
        MOVB     '0,(R5)+       ; PRINT A ZERO
        BR      10$             ; ALL DONE
    6$: MOV     R0,R3           ; MOVE NBR IN PREP FOR DIVIDE
    8$: CLR     R2              ; MORE PREP FOR DIVIDE
        DIV     (R4),R2         ; DIVIDE BY POWER OF 10
        ADD      '0,R2          ; MAKE QUOTIENT A CHARACTER
        MOVB    R2,(R5)+        ; INSERT IN RCV BUFFER
        TST     -(R4)           ; GET ANOTHER POWER OF 10
        BNE     8$              ; NO THRU YET
;
   10$: RTS     PC              ; GO HOME WHEN DONE
;
.SBTTL ERROR ROUTINES
.PAGE
; ===========
; ERROR ROUTINES
; ===========
;
;
; ERROR ROUTINE FOR FILE OPERATIONS AND DOC FILE AND CRT
;
FILERR: MOVWD   FILID,ERRMSG
FILER1: $FETCH  R2,STS,R4       ; MOVE ERROR CODE IN
        XLT28   R2,ERRNBR       ; FIGURE NBR; PUT IT IN BFR
        WRTOUT  ERRMSG,ERRMSL   ; PUT IT OUT
        $COMPARE  0,STV,R4      ; ANYTHING IN STV?
        BEQ     DUN             ; GO AROUND IF THERE'S NOT
        MOVWD   STVID,ERRMSG
        $FETCH  R2,STV,R4       ; MOVE COMPLETION VALUE IN
        MOV     R2,-(SP)        ; SO NO NEED FETCH IT AGN
        XLT28   R2,ERRNBR       ; FIGURE NBR; PUT IT IN BFR
        WRTOUT  ERRMSG,ERRMSL
        MOV     (SP)+,R2        ; BRING BACK STV VALUE
        MOV      6.,R0          ; NBR OF CHAR IN ERRNBR
    4$: CLR     (R0)+           ; ZERO THAT BYTE
        SOB     R0,4$           ; TILL ALL ARE DONE
.LIST ME
        XLT210  R2,ERRNBR       ; MAKE IT A DECIMAL NUMBER
.NLIST ME
        WRTOUT  ERRMSG,ERRMSL   ; DISPLAY IT
   DUN: EXIT$S                  ; QUIT THIS THING
;
; ERROR ROUTINE FOR RECORD OPERATION ON INPUT FILE
;
DOCERR: $COMPARE  ER$EOF,STS,R4 ; RUN OUT OF RECORDS?
        BEQ     2$              ; GO AROUND IF THAT'S IT
        $COMPARE  ER$RTB,STS,R4 ; LINE TOO LONG?
        BEQ     4$              ; GO AROUND IF THAT'S IT
        MOVWD   DOCID,ERRMSG    ; IDENTIFY ERROR
        JMP     FILER1          ; HANDLE LIKE OTHER ERRORS
    2$: XLT210  WDCNT,WDMSG+38. ; PUT TOTAL CURRENT IN MSG
        WRTOUT  WDMSG+24.,18.   ; OUTPUT IT
        WRTOUT  ENDMSG,ENDMSL   ; PUT IT ON SCREEN
        BR      32$             ; GO TO END
```

```
    4$: XLT210  LINENO,LINMSG+5.; PUT IN LINE NUMBER
        WRTOUT  LINMSG,LINMSL   ; DISPLAY MSG
        JMP     INPUT           ; GET NEXT RECORD (PC IS GOOD
                                ; 'CAUSE NOT LEFT SUBRTN)
   32$: EXIT$S                  ; QUIT IT
;
.END START
```

B.3 TESTING RELATIVE FILE CAPABILITIES

This sample program creates a Relative file, sets up a Record Access Stream, and runs five tests:

1. Test if can put a record with a relative record number greater than the Maximum Record Number set for the file.

2. Test if can get records past the Maximum Record Number set for the file.

3. Test if sequential get operations retrieve only valid records.

4. Test if sequential get operations retrieves deleted records.

5. Test if approximate match tests (KGE and KGT) work during random get operations.

```
        .MCALL  $INIT,FAB$B,NAM$B,XAB$B,RAB$B,ORG$,POOL$B
        .MCALL  $GNCAL,$FBCAL,$RBCAL,$STGDPLY,$GETSTG,EXIT$S
$GNCAL
$FBCAL
$RBCAL
;
; POOL SPACE
;
POOLAR:
        POOL$B
        P$BDB   6
        P$FAB   2
        P$RAB   2
        P$BUF   4608.
        POOL$E
;
;  FAB AREA
;
        .EVEN
FABAR:
        FAB$B                   ;ALLOCATE FAB
        F$BKS   2               ;BUCKET SIZE
        F$FAC   FB$GET!FB$PUT!FB$DEL ;DOING GETS AND PUTS
        F$FOP   FB$TMD          ;THIS FILE TO BE DELETED
        F$LCH   1               ;USE LOGICAL CHANNEL 1
        F$MRS   80.             ;MAX RECORD SIZE OF 80
        F$ORG   FB$REL          ;RELATIVE FILE
        F$RFM   FB$VAR          ;VARIABLE LENGTH RECORDS
        FAB$E
;
; RAB AREA
;
        .EVEN
RABAR:
        RAB$B                   ;ALLOCATE RAB
        R$FAB   FABAR           ;POINTS TO FAB AREA
        R$KBF   KEYNO           ;WHERE TO PUT RELATIVE KEY NUMBER
        R$KSZ   4               ;KEY IS 4 BYTES
        R$RAC   RB$SEQ          ;WRITE RECORDS SEQUENTIALLY
```

```
        R$RBF   RECBUF          ;ADDR OF RCD PUT OR GOT
        R$UBF   RECBUF          ;SAME AS RBF
        R$USZ   150.
        RAB$E
;
;
        .EVEN
FABTI:
        FAB$B
        F$FAC   FB$PUT          ;PUT TO THE FILE
        F$FNA   TFNAM           ;TERMINAL FILE NAME
        F$FNS   TFNAMS          ;TERMINAL FILE NAME SIZE
        F$FOP   FB$CTG!FB$TMD   ;CONTIGUOUS SPACE, TEMP FILE
        F$LCH   2               ;LOGICAL CHANNEL 2
        F$MRS   768.            ;MAX REC SIZE = 768. BYTES
        F$ORG   FB$SEQ          ;SEQUENTIAL FILE
        F$RAT   FB$CR           ;RECS DELIMITED BY CR AND LF
        F$RFM   FB$FIX          ;FIXED FORMAT RECS
        FAB$E
;
RABTI:
        RAB$B
        R$FAB   FABTI           ;FILE CTRL BLOCK ADDR
        R$RAC   RB$SEQ          ;SEQUENTIAL ACCESS
        R$UBF   BUFTI           ;USER BUFFER ADDR
        R$USZ   80.             ;MAX SIZE = 80. BYTES
        RAB$E
;
; LOCAL  SYMBOLS
;
        CR = 15                 ;ASCII CARRIAGE RETURN
        LF = 12                 ;ASCII LINE FEED
;
PRAMTR: .BLKW 1                 ;PARAMETER TO SEND INFO TO PRINT
;
HOLDIT: .BLKB   2               ;HOLD NUMBER FOR ASCII CONVERT
KEYNO:  .BLKB   4               ;RELATIVE KEY
KEYTST: .BLKB   2               ;KEYTST TO COMPARE RECORD RETRIEVED
 WITH RECORD PUT
DELTST: .BLKB   2               ;TEST FOR DELETED RECORDS
TSTFOR: .BLKB   1               ;TO TEST FOR FOURTH TEST
ERR:    .BLKB   1               ;TEST FOR TEST 4 ERRORS
RNFSET: .BLKB   1               ;TEST FOR RNF ERROR
;
        .EVEN
;
BUFTI:  .BLKB   80.             ;TERMINAL FILE BUFFER
;
RECBUF:
RECKEY: .BLKW   2
        .ASCII  /MISC. DATA/ <CR><LF>
RECLEN  =  . - RECBUF
;
TFNAM:  .ASCII /TI:RELTST.ERR/  ;TERMINAL FILE NAME
TFNAMS  =       . - TFNAM
;
;
```

```
;
MSGTBL:
;
ERR1:    .ASCII /ERROR CREATING TTY FILE/ <CR><LF>
ERR1LN   = . - ERR1
;
ERR2:    .ASCII /ERROR CONNECTING TTY FILE/ <CR><LF>
ERR2LN = . - ERR2
;
ERR3:    .ASCII /ERROR CREATING RELATIVE FILE/ <CR><LF>
ERR3LN = . - ERR3
;
ERR4:    .ASCII /ERROR CONNECTING RELATIVE FILE/ <CR><LF>
ERR4LN = . - ERR4
;
MSG5:    .ASCII /TEST /
TESTNO: .BLKB   2
SUC5:    .ASCII / SUCCESSFUL!/ <CR><LF>
MSG5LN = . - MSG5
;
ERR6:    .ASCII /ERROR PUTTING RECORD TO FILE/ <CR><LF>
ERR6LN = . - ERR6
;
ERR7:    .ASCII /ERROR DELETING RECORD NUMBER  /
DELNUM: .BLKB   4
         .ASCII <CR><LF>
ERR7LN = . - ERR7
;
ERR8:    .ASCII /ERROR GETTING RECORD FROM FILE/ <CR><LF>
ERR8LN = . - ERR8
;
ERR9:    .ASCII /ERROR CLOSING FILE/ <CR><LF>
ERR9LN = . - ERR9
;
ERR10:   .ASCII /ERROR REWINDING FILE/ <CR><LF>
ER10LN = . - ERR10
;
ERR11:   .ASCII /ERROR DISCONNECTING FILE/ <CR><LF>
ER11LN = . - ERR11
;
ERR12:   .ASCII /RETRIEVED WRONG RECORD IN FILE/ <CR><LF>
ER12LN = . - ERR12
;
ERR13:   .ASCII /ERROR FINDING RECORD NUMBER /
FINDNO: .BLKB   4
CRLF13: .ASCII <CR><LF>
ER13LN = . - ERR13
;
ERR14:   .ASCII /ERROR - OVER 40. RECORDS WRITTEN ON TEST1/ <CR><LF>
ER14LN = . - ERR14
;
ERR15:   .ASCII /DID NOT GET RECORD NUMBER  /
TST5NO: .WORD   1
CRLF15: .ASCII  <CR><LF>
ER15LN = . - ERR15
;
ERR16:   .ASCII /ERROR DISCONNECTING FILE/ <CR><LF>
```

```
ER16LN = . - ERR16
;
        .EVEN
ERRTBL: .WORD   ERR1
        .WORD   ERR1LN
        .WORD   ERR2
        .WORD   ERR2LN
        .WORD   ERR3
        .WORD   ERR3LN
        .WORD   ERR4
        .WORD   ERR4LN
        .WORD   MSG5
        .WORD   MSG5LN
        .WORD   ERR6
        .WORD   ERR6LN
        .WORD   ERR7
        .WORD   ERR7LN
        .WORD   ERR8
        .WORD   ERR8LN
        .WORD   ERR9
        .WORD   ERR9LN
        .WORD   ERR10
        .WORD   ER10LN
        .WORD   ERR11
        .WORD   ER11LN
        .WORD   ERR12
        .WORD   ER12LN
        .WORD   ERR13
        .WORD   ER13LN
        .WORD   ERR14
        .WORD   ER14LN
        .WORD   ERR15
        .WORD   ER15LN
        .WORD   ERR16
        .WORD   ER16LN
        ORG$    REL,<CRE,FIN,GET,PUT,DEL>
        ORG$    SEQ,<CRE,PUT>
;
START:
        $INIT
;
        JSR     PC,CRETTY           ;CREATE, CONNECT TTY FILE

        JSR     PC,CREATE           ;CREATE, CONNECT RELATIVE FILE
;
        JSR     PC,TEST1            ;TEST 1 BEGINS
;
        JSR     PC,TEST2            ;TEST 2 BEGINS

        JSR     PC,CLOSE            ;DISCONNECT AND CLOSE FILE
;
        JSR     PC,TEST3            ;TEST 3 BEGINS
;
        JSR     PC,TEST4            ;TEST 4 BEGINS
;
        JSR     PC,TEST5            ;TEST 5 BEGINS
;
```

```
        JSR     PC,CLOSE          ;DISCONNECT AND CLOSE FILE
;
        EXIT$S
;
;
;CREATE AND CONNECT THE TTY FILE
;
CRETTY:
        MOV      0,PRAMTR         ;SET UP TTY CREATE ERRMSG
        $CREATE  FABTI, ENDIT     ;CREATE TTY FILE
        MOV      4,PRAMTR         ;SET UP TTY CONNECT ERR
        $CONNECT  RABTI, ENDIT    ;CONNECT TTY FILE
        RTS     PC
;
;CREATE THE RELATIVE FILE
;
CREATE:
        MOV      RABAR,R4         ;INITIALIZE RAB POINTER
        $STORE   RECLEN,RSZ,R4    ;SET UP RECORD SIZE
        CLR     R1                ;R0  R1 SET UP 2 WORD MRN
        MOV      40.,R0           ;SET UP R0 FOR MRN
        MOV      FABAR,R5         ;SET UP FAB POINTER
        $STORE  R0,MRN,R5         ;INITIALIZE 40. AS MRN
        MOV      10,PRAMTR        ;SET UP CREATE REL FILE ERR
        $CREATE R5, ENDIT         ;CREATE THE FILE
        RTS     PC
CONECT:
        MOV      14,PRAMTR        ;SET UP CONNECT REL FILE ERR
        MOV      RABAR,R5         ;SET UP RAB POINTER
        $CONNECT R5, ENDIT        ;CONNECT REL FILE
        RTS     PC
;
;REWIND THE FILE
;
REWIND:
        MOV      44,PRAMTR        ;SET UP FOR REWIND ERROR
        $REWIND  RABAR, ENDIT     ;GO TO B-O-F FOR NEXT TEST
        RTS     PC
;
;DISCONNECT AND CLOSE THE FILE
;
CLOSE:
        MOV      74,PRAMTR        ;SET UP FOR DISCONNECT ERROR
        $DISCONNECT  RABAR, ENDIT ;DISCONNECT FILE
        MOV      40,PRAMTR        ;SET UP FOR CLOSE ERROR
        $CLOSE   FABAR, ENDIT     ;CLOSE FILE
        RTS     PC
;
;PRINT THE MESSAGES
;
PRINT:
        MOV      0,R2             ;CLEAR R2
        MOV     PRAMTR,R2         ;SET UP INDEX WITHIN TABLE
        ADD      ERRTBL,R2        ;POINTER TO ERROR MESSAGE
        MOV      RABTI,R3         ;POINTER TO ERROR TABLE
        $STORE  (R2),RBF,R3       ;STORE ADDRESS OF MESSAGE
        $STORE  2(R2),RSZ,R3      ;STORE SIZE OF MESSAGE
```

```
        $PUT    R3,BOMB         ;TTY MSG - BOMB ON ERROR
        RTS     PC
;
ENDIT:
        JSR     PC,PRINT         ;PRINT ERROR MESSAGE

BOMB:
        BPT                      ;BOMB
;
;-----------------------------------------------
;TEST 1
;       THIS IS A TEST OF MRN.  MRN SET TO 40.
;       WILL ATTEMPT TO WRITE 50. RECORDS.
;       SHOULD ENCOUNTER ER$MRN ERROR.
;-----------------------------------------------
;
TEST1:

        CLR     R1              ;INITIALIZE R1
        JSR     PC,CONECT        ;CONNECT THE FILE

PUTIT:
        INC     R1              ;NEXT RECORD
        $PUT    R4              ;WRITE THE RECORD
        $COMPARE  SU$SUC,STS,R4 ;PUT ERROR?
        BNE     T1ERR           ;IF YES, GO TO T1ERR
        CMP      50.,R1         ;WRITTEN 50 RECORDS YET?
        BHI     PUTIT           ;NO, GO WRITE SOME MORE
        MOV      64,PRAMTR      ;SET UP FOR TEST 1 ERROR
        JSR     PC,ENDIT         ;PRINT ERROR MESSAGE, BOMB
T1ERR:
        MOV      24,PRAMTR      ;SET UP FOR PUT ERROR
        $COMPARE  ER$MRN,STS,R4 ;MRN ERROR?
        BNE     ENDIT           ;NOT MRN ERROR--BOMB
        MOV      20,PRAMTR      ;SET UP FOR SUCCESS MSG
        MOV      61,TESTNO      ;SET UP FOR PRINTING TEST 1
        JSR     PC,PRINT         ;PRINT SUCCESS MESSAGE
        JSR     PC,REWIND        ;RETURN TO B-O-F
        RTS     PC               ;GO ON TO TEST 2
;
;-------------------------------------------------------------
;       IF TEST 1 IS SUCCESSFUL, RECORDS HAVE BEEN WRITTEN
;       UP TO MRN.  THAT IS, 40 RECORDS HAVE BEEN WRITTEN.
;       THE FILE HAS BEEN REWOUND, READY FOR TEST 2.
;-------------------------------------------------------------
;
;-----------------------------------------------
;TEST 2
;       THIS TEST ATTEMPTS TO RETRIEVE RECORDS
;       SEQUENTIALLY BEYOND MRN.
;       SHOULD ENCOUNTER AN ER$EOF ERROR.
;       IT ATTEMPTS TO RETRIEVE 50.. RECORDS.
;       AFTER THE 40.TH RECORD, AN ER$EOF ERROR
;       SHOULD RESULT.
;-----------------------------------------------
;
TEST2:
```

```
        MOV      RABAR,R4        ;SET UP RAB POINTER
        CLR      R1              ;INITIALIZE RECORD COUNTER
        MOV      34,PRAMTR       ;SET UP FOR GET ERROR
        MOV      62,TESTNO       ;SET UP FOR PRINTING TEST 2
T2INC:
        INC      R1              ;INCREMENT RECORD COUNTER
        $GET     RABAR           ;GET ANOTHER RECORD
        $COMPARE  SU$SUC,STS,R4  ;SUCCESSFUL GET?
        BNE      GETERR          ;IF NOT, GO TO GETERR
        CMP      50.,R1          ;GOTTEN 50 RECORDS YET?
        BNE      T2INC           ;IF NOT, CONTINUE
        RTS      PC               ;YES, RETURN
GETERR:
        $COMPARE  ER$EOF,STS,R4  ;EOF ERROR?
        BNE      ENDIT           ;NOT AN EOF ERROR - BOMB
        MOV      20,PRAMTR       ;SET UP FOR TEST SUCCESS MESSAGE
        JSR      PC,PRINT         ;PRINT SUCCESS MESSAGE
        JSR      PC,REWIND        ;GO TO BEGINNING OF FILE
        RTS      PC               ;GO ON TO NEXT TEST
;
;-------------------------------------------------------
;TEST 3
;       CREATE A NEW FILE.
;       PUT EVERY OTHER RECORD RANDOMLY UP TO AN MRN OF 40.
;       REWIND, AND GET SEQUENTIALLY.
;       SHOULD GET ONLY THOSE RECORDS JUST PUT.
;-------------------------------------------------------
;
TEST3:
        JSR      PC,CREATE        ;CREATE NEW RELATIVE FILE
        $STORE   RB$KEY,RAC,R4   ;RELATIVE ACCESS MODE
        JSR      PC,CONECT        ;CONECT THE FILE
        MOV      1,KEYNO         ;INITIALIZE THE RELATIVE KEY
        CLR      KEYNO+2         ;CLEAR 2ND WORD OF KEY
        CLR      RECKEY+2        ;CLEAR 2ND WD OF KEY IN FILE
;
;SET UP FILE - WRITE EVERY OTHER RECORD
;
SETUP:
        MOV      24,PRAMTR       ;SET UP FOR PUT ERROR
        MOV      KEYNO,RECKEY    ;PUT KEY NUMBER IN RECORD
        $PUT     RABAR, ENDIT    ;PUT RCD; ERROR = MSG, END.
        ADD      2,KEYNO         ;SKIP A RECORD
        CMP      40.,KEYNO       ;FILE FULL?
        BHI      SETUP           ;IF NOT, WRITE SOME MORE
        MOV      20,PRAMTR       ;SET UP FOR TEST 3 SUCCESS
        MOV      63,TESTNO       ;SET UP FOR PRINTING TEST 3
;
;******** COMMON ROUTINE FOR TESTS 3 AND 4 **********
;
;
;REWIND AND REINITIALIZE
;
SEQGET:
        CALL     REWIND          ;RETURN TO BEGINNING OF FILE
        MOV      -1,KEYTST       ;SET UP TO TEST FOR CORRECT RCD
        $STORE   RB$SEQ,RAC,R4   ;GET SEQUENTIALLY
```

```
;
;GET THE RECORDS SEQUENTIALLY
;
GETIT:
        $GET    R4                ;GET RECORD
        $COMPARE  SU$SUC,STS,R4 ;SUCCESSFUL GET?
        BNE     GETERR            ;IF NOT, TEST FOR E-O-F
                                  ;GETERR IN TEST 2
RECTST:
        ADD      2,KEYTST         ;EVERY OTHER RECORD
        CMP     KEYTST,RECKEY     ;RIGHT RECORD?
        BEQ     GETIT             ;IF SO, LET'S GET ANOTHER
BADREC:
        CMPB     1,TSTFOR         ;IS IT TEST  4?
        BNE     BADCON            ;IF NOT, WRONG RCD
        ADD      6,DELTST         ;HOLD KEY OF NXT DELETED RCD
        CMP     DELTST,KEYTST     ;IS IT A DELETED RECORD?
        BEQ     RECTST            ;YES, ADD 2 TEST AGAIN
BADCON:
        MOV      54,PRAMTR        ;WRONG RECORD = MSG
        JMP     ENDIT             ;PRINT OUT MESSAGE, BOMB
;
;-------------------------------------------------------------
;TEST 4
;       TEST 4 USES THE FILE CREATED IN TEST 3.
;       SOME RECORDS ARE DELETED.
;       GET SEQUENTIALLY AND SEE IF DELETED RECORDS
;       ARE FOUND.
;-------------------------------------------------------------
TEST4:
        MOV      RABAR,R4         ;SET UP RAB POINTER
        $STORE   RB$KEY,RAC,R4    ;DELETE USING RANDOM
        MOV      -5,KEYNO         ;INIT REL KEY FOR DELETES
        MOV      1,TSTFOR         ;SET UP FOR FOURTH TEST
        MOV      -5,DELTST        ;INIT FOR DELETED RCD KEY
NXTDEL:
        ADD      6,KEYNO          ;DELETE EVERY 4TH RECORD
        JSR     PC,FINDIT          ;GO DELETE IT
        CMP      37.,KEYNO        ;LAST RECORD
                                  ;TO BE DELETED BEFORE EOF?
        BHI     NXTDEL            ;IF NOT, DELETE SOME MORE
        CMPB     0,ERR            ;ANY ERRORS ENCOUNTERED?
        BEQ     GET4              ;NO ERRORS, GET SEQUENTIALLY
        JMP     BOMB              ;ERRORS FOUND - BOMB
GET4:
        MOV      20,PRAMTR        ;SET UP FOR TEST 4 SUCC MSG
        MOV      64,TESTNO        ;SET UP FOR PRINTING TEST 4
        JSR     PC,SEQGET          ;READY TO GET SEQUENTIALLY
        RTS     PC                 ;GO ON TO NEXT FILE
FINDIT:
        $FIND    RABAR
        $COMPARE  SU$SUC,STS,R4 ;FIND SUCCESSFUL?
        BEQ     DELETE            ;IF IT WAS, DELETE RECORD
        MOV      60,PRAMTR        ;SET UP FOR FIND ERROR
        MOV     KEYNO,R0          ;SET UP FOR KEY NO.
                        ;CONVERSION TO ASCII
        JSR     PC,CONVER          ;CONVERT RCD NBR TO ASCII
```

```
        MOV     HOLDIT,FINDNO   ;PUT RCD NBR IN ERRMSG
        JSR     PC,PRINT         ;PRINT ERROR MESSAGE
        RTS     PC               ;GO ON TO NEXT RECORD
DELETE:

        $DELETE  RABAR           ;DELETE RECORD
        $COMPARE  SU$SUC,STS,R4 ;SUCCESSFUL DELETE?
        BEQ     RETURN           ;IF SUCCESSFUL, CONTINUE
        MOV      30,PRAMTR       ;SET UP FOR DELETE ERRMSG
        MOV     KEYNO,R0         ;SET UP FOR ASCII CONVERT
        CALL    CONVER           ;CNV KEY NBR TO ASCII FOR ERRMSG
        MOV     HOLDIT,DELNUM    ;PUT RCD NBR IN ERRMSG
        JSR     PC,PRINT         ;PRINT ERROR MESSAGE
;
RETURN:
        RTS     PC               ;GO ON TO NEXT RECORD
;
;CONVERSION TO ASCII TO PRINT OUT RCD NBR IN ERRMSG
;
CONVER:
        MOVB     1,ERR           ;NOTE THAT ERROR OCCURRED
        MOV      RABAR,R4        ;SET UP RAB PNTR AFTER ERR
        $STORE   1,STS,R4        ;RESET TO GOOD STATUS
        MOV      1,R3            ;SET UP R3 FOR INDEXING
        MOV      2,R2            ;SET UP R2 TO LOOP 2 TIMES:
                        ;ONCE FOR EACH BYTE

LOOP:
        MOV     R0,-(SP)         ;SAVE REMAINDER FOR CNV
        BIC      -10,R0          ;CLEAR ALL BUT LOW ORDER 3 BITS
        ADD      60,R0           ;CONVERT TO ASCII
        MOVB    R0,HOLDIT(R3)    ;STORE IT IN RECORD
        MOV     (SP)+,R0         ;GET COUNTER
        ASR     R0               ;SHIFT RIGHT 3 BITS
        ASR     R0               ;TO WORK ON EACH DIGIT
        ASR     R0
        DEC     R3               ;MOVE TO NXT BYTE OF RECKEY
        SOB     R2,LOOP          ;DIGIT IN EACH BYTE??
        RTS     PC
;-------------------------------------------------------
;TEST 5
;       THIS TEST WILL TEST THE ROP FIELDS OF
;       KGT AND KGE.
;       AFTER SUCCESSFUL COMPLETION OF TEST 4,
;       THE FILE WILL NOW HAVE OCTAL RECORDS:
;       3,5,11,13,17,21,25,27,33,35,41,43,47.
;       USING THIS INFORMATION, WE WILL TEST
;       'KEY GREATER THAN' AND 'KEY GREATER THAN
;       OR EQUAL TO'.
;-------------------------------------------------------
;
TEST5:
        MOV      RABAR,R4        ;SET UP RAB POINTER
        $STORE   RB$KEY,RAC,R4   ;USING RANDOM
        $STORE   RB$KGT,ROP,R4   ;SET UP KEY GREATER THAN
        JSR     PC,KGT           ;DO KEY GREATER THAN TEST
        $STORE   RB$KGE,ROP,R4   ;SET UP KEY GTR'N OR = TO
```

```
        JSR     PC,REWIND       ;GO TO BEGINNING OF FILE
        JSR     PC,KGE          ;DO KEY GTR'N OR = TEST
        CMPB    0,ERR           ;ANY ERRORS DETECTED?
        BEQ     GOOD5           ;NO ERRORS = SUCCESS MSG
        RTS     PC
GOOD5:
        MOV     65,TESTNO       ;MOV 5 INTO SUCCESS MESSAGE
        MOV     20,PRAMTR       ;SET UP FOR TEST 5 SUCC MSG
        CALL    PRINT           ;PRINT MESSAGE
        RTS     PC              ;DONE WITH TEST 5
KGT:
        MOV     5,KEYNO         ;WE WANT THE NEXT REC GT 5
        MOV     11,R0           ;IT SHOULD BE RECORD 11
        JSR     PC,GET5         ;GET THE RECORD
        MOV     20,KEYNO        ;TRY NEXT REC GTR'N 20
        MOV     21,R0           ;WE SHOULD GET RECORD 21
        CALL    GET5            ;GET THE RECORD
        MOV     27,KEYNO        ;WANT THE NEXT REC GTR'N 27
        MOV     33,R0           ;WE SHOULD GET RECORD 33
        JSR     PC,GET5         ;GET THE RECORD
        MOV     47,KEYNO        ;GET REC GT 47
        MOVB    1,RNFSET        ;SET UP TO TEST FOR RNF
        JSR     PC,GET5         ;SHOULD GET RNF ERROR
        RTS     PC              ;GO ON TO KGE TEST
KGE:
        MOV     5,KEYNO         ;WE WANT THE REC. GE 5
        MOV     5,R0            ;SHOULD GET RECORD 5
        JSR     PC,GET5         ;GET THE RECORD
        MOV     20,KEYNO        ;WE WANT THE REC. GE 20
        MOV     21,R0           ;WE SHOULD GET RECORD 21
        CALL    GET5            ;GET THE RECORD
        MOV     27,KEYNO        ;WE WANT THE REC. GE 27
        MOV     27,R0           ;SHOULD GET RECORD 27
        CALL    GET5            ;GET THE RECORD
        MOV     47,KEYNO        ;WE WANT THE REC. GE 47
        MOV     47,R0           ;SHOULD GET RECORD 47
        JSR     PC,GET5         ;GET THE RECORD
        RTS     PC              ;DONE WITH TEST 5
GET5:
        $GET    R4              ;GET THE RECORD
        $COMPARE  SU$SUC,STS,R4 ;SUCCESSFUL GET?
        BEQ     COMPAR          ;YES = TEST FOR CORRECT RCD
        CMPB    1,RNFSET        ;TESTING FOR RNF ERROR?
        BNE     GT5ERR          ;NO = CONT ERR PROCESSING
        MOVB    0,RNFSET        ;RESET FOR NO RNF ERROR
        $COMPARE  ER$RNF,STS,R4 ;DID WE GET AN RNF ERROR?
        BNE     GT5ERR          ;NO, THERE'S REALLY AN ERROR
        RTS     PC              ;RNF ERR = GOOD, CONTINUE
GT5ERR:
        MOV     34,PRAMTR       ;SET INDEX FOR GET ERR
        $STORE  1,STS,R4        ;RESET TO GOOD STATUS
        MOVB    1,ERR           ;SET ERR BYTE
        JSR     PC,PRINT        ;PRINT GET ERROR MESSAGE
        RTS     PC
COMPAR:
        CMP     R0,RECKEY       ;CORRECT RECORD?
        BNE     ERROR5          ;IF NOT GO TO ERROR ROUTINE
```

```
        RTS     PC                  ;WE DID, CONTINUE
ERROR5:
        MOV      70,PRAMTR          ;SET INDEX IN TABLE FOR ERROR MSG
        JSR     PC,CONVER           ;CNV RCD NBR TO ASCII FOR PRINT
        MOV     HOLDIT,TST5NO       ;PUT RCD NBR IN MSG
        CALL    PRINT               ;PRINT THE MESSAGE
        RTS     PC
;
        .END    START
```

APPENDIX C

DATE CONVERSION ROUTINE

```
        .TITLE  $CDTTA - CONVERT DATE AND TIME TO ASCII

        .IDENT  "X0001"
;
;  COPYRIGHT (C) 1977 BY DIGITAL EQUIPMENT CORPORATION,
;  COPYRIGHT (C) 1976 BY DIGITAL EQUIPMENT CORPORATION,
;  MAYNARD, MASSACHUSETTS
;
;  THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
;  SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE IN-
;  CLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS SOFTWARE, OR
;  ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
;  MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
;  SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS.  TITLE
;  TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
;  IN DIGITAL.
;
;  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITH-
;  OUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY
;  DIGITAL EQUIPMENT CORPORATION.
;
;  DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR
;  THE USE OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT WHICH IS
;  NOT SUPPLIED BY DIGITAL.
;
;  ANDREW C.  GOLDSTEIN 20 JUN 76 14:12

        SMACIT

        .MACRO  $SAVRG
        JSR     R5,$SAVRG
        .ENDM   $SAVRG



        .MACRO  .QUAD           WORD
        $$$B1=0
        $$$B2=0
        $$$B3=0
        $$$B4=0
        $$$B5=0
        $$$B6=0
        $$$B7=0
        $$$B8=0
```

```
        .IRPC   D,WORD
        $$$B1=$$$B1*10.+D'.
        $$$C=$$$B1/256.
        $$$B1=$$$B1O377
        $$$B2=$$$B2*10.+$$$C
        $$$C=$$$B2/256.
        $$$B2=$$$B2O377
        $$$B3=$$$B3*10.+$$$C
        $$$C=$$$B3/256.
        $$$B3=$$$B3O377
        $$$B4=$$$B4*10.+$$$C
        $$$C=$$$B4/256.
        $$$B4=$$$B4O377
        $$$B5=$$$B5*10.+$$$C
        $$$C=$$$B5/256.
        $$$B5=$$$B5O377
        $$$B6=$$$B6*10.+$$$C
        $$$C=$$$B7/256.
        $$$B6=$$$B6O377
        $$$B7=$$$B7*10.+$$$C
        $$$C=$$$B7/256.
        $$$B7=$$$B7O377
        $$$B8=$$$B8*10.+$$$C
        $$$C=$$$B8/256.
        $$$B8=$$$B8O377
        .ENDM
        $$$W1=$$$B2*256.+$$$B1
        $$$W2=$$$B4*256.+$$$B3
        $$$W3=$$$B6*256.+$$$B5
        $$$W4=$$$B8*256.+$$$B7
        .LIST   MEB
        .WORD   $$$W1,$$$W2
        .WORD   $$$W3,$$$W4
        .NLIST  MEB
        .ENDM   .QUAD
;
;  SOME CONSTANTS
;

        .RADIX  10

QNS:    .QUAD   10000000        ;  TENTHS OF MICROSECONDS IN A SECOND
        .WORD   $$$W1,$$$W2
        .WORD   $$$W3,$$$W4
QND:    .QUAD   864000000000    ;  TENTHS OF MICROSECONDS IN A DAY
        .WORD   $$$W1,$$$W2
        .WORD   $$$W3,$$$W4

FDAYS = 365*4+1                 ;  NUMBER OF DAYS IN 4 YEARS
                                ;  AND NUMBER OF QUARTER DAYS IN A YEAR
CDAYS = FDAYS*25-1              ;  NUMBER OF DAYS IN A CENTURY
QDAYL = 15025                   ;  NUMBER OF DAYS IN QUADRICENTURY, LOW
QDAYH = 2                       ;  NUMBER OF DAYS IN QUADRICENTURY, HI
DS15L = 65176                   ;  NUMBER OF DAYS BETWEEN 1-JAN-1501
DS15H = 1                       ;  AND 17-NOV-1858, LOW AND HIGH
```

```
        .RADIX
;+
;
;   *** - $CDTTA CONVERT DATE/TIME TO ASCII
;
;   THIS ROUTINE CONVERTS THE STANDARD DATE/TIME REPRESENTATION INTO
;   AN ASCII STRING.  THE FORMATTING OF THE STRING IS CONTROLLED BY
;   OPTION BITS ON INPUT.
;
;   INPUTS:
;
;        R0 = ADDRESS TO STORE ASCII STRING
;        R1 = ADDRESS OF 64 BIT DATE/TIME
;        R2 = OPTION FLAGS:
;          BIT 0 = 1 IF DATE NOT WANTED
;          BIT 1 = 1 IF TIME NOT WANTED
;          BIT 2 = 1 IF TIME WANTED IN AM/PM
;          BIT 3 = 1 IF SECONDS NOT WANTED
;
;   OUTPUTS:
;
;        R0 UPDATED TO END OF GENERATED STRING
;        OTHER REGISTERS PRESERVED
;
;   THE DATE ALGORITHM BELOW ACCOUNTS FOR THE COMMON LEAP YEAR CYCLES -
;   EVERY FOURTH YEAR IS, EXCEPT EVERY 100TH ISN'T, EXCEPT EVERY 400TH
;   IS.  YEAR 2000 IS A LEAP YEAR.  THE CODE IS BASED ON ALGORITHMS
;   WRITTEN BY P.CONKLIN, J.BARNABY, D.ROSENBERG, AND M.SPEIR.
;
;-

$CDTTA::
        $SAVRG                  ;  SAVE ALL REGISTERS
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R0,R5
        SUB     #16.,SP         ;  ALLOCATE TWO QUAD SLOTS
        MOV     SP,R3
        ADD     #8.,R3
        MOV     #QND,R0         ;  SPLIT DATE/TIME INTO DAY AND TIME
        MOV     SP,R2
        JSR     PC,$DIVQ
;
;   TOP OF STACK CONTAINS DAYS SINCE 17-NOV-1858.  CONVERT THIS TO A
;   REAL DATE.
;
        BIT     #1,16.(SP)      ;  DO OF DATE REQUESTED
        BNE     L0
        MOV     (SP),R3         ;  DAY SINCE 17-NOV-1858 IS
        MOV     2(SP),R2        ;  LESS THAN 32 BITS
                                ;  ADD THE OFFSET SO WE HAVE THE NUMBER
                                ;  OF DAYS SINCE 1-JAN-1501, ALIGNING
                                ;  CYCLES SO THAT THE LATER
                                ;  TRUNCATIONS AND ROUND-OFFS WORK.
        ADD     #DS15L,R3
        ADC     R2
        ADD     #DS15H,R2
```

```
                                ;  DIVIDE BY THE NUMBER OF DAYS IN
                                ;  A QUADRICENTURY
        MOV     #QDAYH,R0
        MOV     #QDAYL,R1
        JSR     PC,$DIVD
                                ;  R0,R1 = DAYS INTO THIS QUADRICENTURY
                                ;  R3 = QUADRICENTURIES SINCE 1501
        ASL     R3              ;  CONVERT TO CENTURIES
        ASL     R3
        MOV     R3,-(SP)        ;  AND SAVE
        ASL     R1              ;  CONVERT NUMBER OF DAYS INTO
        ROL     R0              ;  NUMBER OF QUARTER DAYS
        ASL     R1              ;  BY MULTIPLYING BY 4
        ROL     R0
        MOV     R1,R3
        MOV     R0,R2
                                ;  DIVIDE BY THE AVERAGE NUMBER OF
                                ;  QUARTER DAYS IN A CENTURY
        MOV     #QDAYH,R0
        MOV     #QDAYL,R1
        JSR     PC,$DIVD
                                ;  R0,R1 = QUARTER DAYS INTO CENTURY
                                ;  R3 = CENTURIES IN THIS QUAD CENTURY
        ADD     R3,(SP)         ;  ADD TO ACCUMULATED TIME
;
;  BY BASING THE TIME AT 1501, THE CENTURY HAS AN EXTRA LEAP DAY
;  INTO THE FIRST CENTURY OF THE QUADRICENTURY.  NOW DISCARD ANY
;  FRACTION OF A DAY LEFT FROM THE PREVIOUS DIVISION AND ADD IN 3/4
;  OF A DAY TO FORCE THE LEAP YEAR INTO THE LAST YEAR OF EACH 4 YEAR
;  CYCLE.
;
        BIS     #3,R1
        MOV     R1,R3           ;  NOW DIVIDE BY THE AVERAGE NUMBER OF
        MOV     R0,R2           ;  QUARTER DAYS IN A YEAR
        MOV     #0,R0
        MOV     #FDAYS,R1
        JSR     PC,$DIVD
        MOV     R1,R2           ;  NUMBER OF DAY IN YEAR
        ASR     R2
        ASR     R2
        INC     R2
        MOV     #100.,R0        ;  COMPUTE # YEARS IN CENTURIES
        MOV     (SP)+,R1
        JSR     PC,$MULUS
        ADD     R3,R1           ;  COMPUTE TRUE CALENDAR YEAR
        ADD     #1501.,R1
        MOV     R1,-(SP)        ;  SAVE FOR FINAL OUTPUT
;
;  FIND OUT WHETHER THIS YEAR IS A LEAP YEAR.  IF NOT, BIAS THE DAY IF
;  IT IS PAST FEBRUARY:  THE MONTH TABLE IS WRITTEN FOR A LEAP YEAR.
;
        MOV     #1,R4           ;  INIT LEAP YEAR FLAG TO NO
        MOV     (SP),R0         ;  EVERY 400TH IS A LEAP YEAR
        MOV     #400.,R1
        JSR     PC,$DIV         ;  FROM SYSLIB
        TST     R1
        BNE     L1
```

```
        CLR     R4
        BR      L2
L1:
        MOV     (SP),R0         ;  EVERY 100TH IS NOT
        MOV     #100.,R1
        JSR     PC,$DIV         ;  FROM SYSLIB
        TST     R1
        BEQ     L3
        MOV     (SP),R0         ;  AND EVERY 4TH IS
        MOV     #4,R1
        JSR     PC,$DIV         ;  FROM SYSLIB
        TST     R1
        BNE     L4
        CLR     R4
L4:
L3:
L2:
                                ;  IF THIS IS NOT A LEAP YEAR,
                                ;  BIAS THE DAY UP ONE IF IT IS
                                ;  PAST FEBRUARY.
        CMP     R2,#31.+28.
        BLOS    L5
        ADD     R4,R2
L5:
;
;  SCAN THE MONTH TABLE AND FIND OUT WHAT MONTH THIS IS.
;
        CLR     R1
B0:
        MOVB    $DAYTB(R1),R0 ;  GET NUMBER OF DAYS IN MONTH
        CMP     R2,R0
        BLOS    E0
        SUB     R0,R2
        INC     R1
        CMP     R1,#11.
        BLT     B0
E0:
;
;  R2 = DAY, R1 = MONTH
;  START TO CRANK OUT THE ASCII DATE
;
        MOV     R1,-(SP)
        MOV     R5,R0           ;  DAY NUMBER
        MOV     R2,R1
        MOV     #0,R2
        JSR     PC,$CBDMG
        MOVB    #'-,(R0)+       ;  DASH
                                ;  INDEX INTO THE MONTH NAME TABLE
        MOV     (SP),R1
        ASL     R1
        ADD (SP)+,R1
        ADD     #$MONTB,R1
        MOVB    (R1)+           ;  FILL IN MONTH NAME
        MOVB    (R1)+,(R0)+
        MOVB    (R1)+,(R0)+
        MOVB    #'-,(R0)+       ;  DASH
        MOV     (SP)+,R1        ;  ADD YEAR
```

```
        MOV     #0,R2
        JSR     PC,$CBDMG
        MOVB    #40,(R0)+       ;  AND A SPACE
        MOV     R0,R5           ;  SAVE THE POINTER WHILE WE MESS
                                ;  WITH THE TIME OF DAY
L0:
;
;  NOW CONVERT THE TIME OF DAY TO ASCII
;
        BIT     #2,16.(SP)      ;  DO IF TIME REQUESTED
        BNE     L6
        MOV     SP,R1           ;  POINT TO TIME ON STACK
        ADD     #8.,R1
        MOV     SP,R3
        SUB     #8.,SP          ;  ALLOCATE ONE MORE QUAD
        MOV     #QNS,R0         ;  REDUCE TO TIME IN SECONDS
        MOV     SP,R2
        JSR     PC,$DIVQ
        MOV     (SP)+,R3        ;  WHICH IS EXPRESSIBLE IN 32 BITS
        MOV     (SP)+,R2
        CMP     (SP)+,(SP)+     ;  CLEAN GARBAGE FROM STACK
        MOV     #0,R0           ;  DIVIDE OUT SECONDS
        MOV     #60.,R1
        JSR     PC,$DIVD
        MOV     R1,-(SP)        ;  AND SAVE THEM
        MOV     #60.,R1         ;  DIVIDE OUT MINUTES
        JSR     PC,$DIVD
        MOV     R1,-(SP)        ;  AND SAVE
        MOV     #24.,R1         ;  AND DIVIDE OUT HOURS
        JSR     PC,$DIVD
                                ;  OUTPUT ASCII TIME OF DAY
        MOV     R1,R0
        JSR     PC,C2D
        MOVB    #':,(R5)+
        MOV     (SP)+,R0
        JSR     PC,C2D
        MOV     (SP)+,R0        ;  GET SECONDS
        BIT     #10,16.(SP)     ;  IF SECONDS WANTED
        BNE     L7
        MOVB    #':,(R5)+
        JSR     PC,C2D
L7:
        MOV     R5,R0           ;  RETURN FINAL STRING POINTER
L6:
        ADD     #16.,SP         ;  FINAL STACK CLEANUP
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        RTS     PC
;+
;
;  *** - C2D    CONVERT BINARY TO 2 DECIMAL DIGITS
;
;  THIS ROUTINE IS USED TO OUTPUT THE COMPONENTS OF THE TIME
;
;  INPUTS:
;
;       R0 = BINARY VALUE
```

```
;
;  OUTPUTS:
;
;       TWO DIGITS STORED AT R5
;       R5 BUMPED BY 2
;       R0,R1 CLOBBERED
;
;-
C2D:
        MOV     #10.,R1         ;  SPLIT THE DIGITS
        JSR     PC,$DIV         ;  FROM SYSLIB
        ADD     #'0,R0          ;  CONVERT TO ASCII
        ADD     #'0,R1
        MOVB    R0,(R5)+        ;  AND STORE
        MOVB    R1,(R5)+
        RTS     PC
;
;+
;
;  *** - $DIVD  DOUBLE PRECISION DIVIDE
;
;  THIS ROUTINE PERFORMS AN UNSIGNED DIVIDE WITH 32 BIT DIVISOR,
;  DIVIDEND, QUOTIENT, AND REMAINDER.  DIVIDE BY ZERO IS NOT CHECKED.
;
;  ANDREW C.  GOLDSTEIN 4 MAR 76 11:41
;
;  BASED ON THE ALGORITHM IN RT-11'S LIBRARY ROUTINE JDIV
;
;  INPUTS:
;
;       R0 = HIGH ORDER DIVISOR
;       R1 = LOW ORDER DIVISOR
;       R2 = HIGH ORDER DIVIDEND
;       R3 = LOW ORDER DIVIDEND
;
;  OUTPUTS:
;
;       R0 = HIGH ORDER REMAINDER
;       R1 = LOW ORDER REMAINDER
;       R2 = HIGH ORDER QUOTIENT
;       R3 = LOW ORDER QUOTIENT
;
;       R4,R5 PRESERVED
;
;-
$DIVD::
        MOV     R4,-(SP)        ;  SAVE R4  R5
        MOV     R5,-(SP)
        MOV     #33.,-(SP)      ;  SET UP ITERATION COUNT
        CLR     R4              ;  QUOTIENT ENDS UP IN R2:R3
        CLR     R5              ;  REMAINDER ENDS UP IN R4:R5
10$:    ROL     R5              ;  EXPOSE NEW BIT OF NUMERATOR
        ROL     R4
        CMP     R0,R4           ;  DOES DENOM FIT?
        BHI     30$             ;  BRANCH IF NOT, C=0
        BNE     20$             ;  BRANCH IF YES
        CMP     R1,R5           ;  HIGH PARTS SAME, CHECK LOW
```

```
        BHI     30$             ;  BRANCH IF NOT, C=0
20$:    SUB     R1,R5           ;  SUBTRACT DENOM FROM REMAINDER
        SBC     R4
        SUB     R0,R4
        SEC                     ;  INDICATE NEW QUOTIENT BIT
30$:    ROL     R3              ;  SHIFT IN NEW BIT OF QUOTIENT
        ROL     R2
        DECB    (SP)            ;  CHECK LOOP COUNT
        BGT     10$             ;  BRANCH TO LOOP
        TST     (SP)+           ;  CLEAN THE STACK
        MOV     R4,R0           ;  MOVE REMAINDER TO R0  R1
        MOV     R5,R1
        MOV     (SP)+,R5        ;  RESTORE R4  R5
        MOV     (SP)+,R4
        RTS     PC
;
;  *** - $DIVQ  QUAD DIVIDE ROUTINE
;
;  THIS ROUTINE DOES A 64 BIT UNSIGNED DIVIDE OPERATION.  DIVIDE
;  BY ZERO IS NOT CHECKED.  ALL NUMBERS ARE STORED WITH INCREASING
;  SIGNIFICANCE IN INCREASING ADDRESSED BYTES.
;
;  ANDREW C.  GOLDSTEIN 17-NOV-76 10:47
;+
;  INPUTS:
;
;       R0 = ADDRESS OF DIVISOR
;       R1 = ADDRESS OF DIVIDEND
;       R2 = ADDRESS OF QUOTIENT
;       R3 = ADDRESS OF REMAINDER
;
;  OUTPUTS:
;
;       QUOTIENT AND REMAINDER RETURNED IN SPECIFIED ADDRESSES
;
;-
$DIVQ::
        CALL    $SAVAL
        MOV     #65.,-(SP)      ;  SET UP LOOP COUNT
        MOV     R2,R5           ;  POINT TO QUOTIENT AREA
        MOV     (R1)+,(R5)+     ;  COPY DIVIDEND INTO QUOTIENT AREA
        MOV     (R1)+,(R5)+
        MOV     (R1)+,(R5)+
        MOV     (R1)+,(R5)+
        MOV     R3,R4           ;  COPY REMAINDER POINTER
        CLR     (R4)+           ;  CLEAR OUT REMAINDER AREA
        CLR     (R4)+
        CLR     (R4)+
        CLR     (R4)+           ;  R4 POINTS TO END OF REMAINDER
10$:    MOV     R3,R4
        ROL     (R4)+           ;  SHIFT IN NEXT BIT OF DIVIDEND
        ROL     (R4)+
        ROL     (R4)+
        ROL     (R4)+
        MOV     R0,R1
        ADD     #8.,R1          ;  POINT TO HIGH ORDER OF DIVISOR
        MOV     R4,R5           ;  AND OF REMAINDER AREA
```

```
        CMP     -(R1),-(R5)     ;  SEE IF DIVISOR IS GREATER THAN
        BNE     20$             ;  CURRENT REMAINDER VALUE
        CMP     -(R1),-(R5)
        BNE     20$
        CMP     -(R1),-(R5)
        BNE     20$
        CMP     -(R1),-(R5)
20$:    BHI     30$             ;  YES:  BRANCH, NO SUBTRACT, C IS CLEAR
        MOV     R0,R1           ;  POINT TO START OF DIVISOR
        MOV     R3,R4           ;  AND START OF REMAINDER
        SUB     (R1)+,(R4)+     ;  SUBTRACT DIVISOR FROM REMAINDER
        SBC     (R4)+
        SBC     (R4)+
        SBC     (R4)
        CMP     -(R4),-(R4)
        SUB     (R1)+,(R4)+
        SBC     (R4)
        SBC     2(R4)
        SUB     (R1)+,(R4)+
        SBC     (R4)
        SUB     (R1)+,(R4)+
        SEC                     ;  SET BIT TO SHIFT INTO QUOTIENT
30$:    MOV     R2,R5           ;  POINT TO QUOTIENT
        ROL     (R5)+           ;  SHIFT NEW BIT IN FROM RIGHT
        ROL     (R5)+
        ROL     (R5)+
        ROL     (R5)+           ;  C = NEXT BIT OF DIVIDEND
        DEC     (SP)            ;  COUNT ITERATIONS
        BNE     10$             ;  AND LOOP
        TST     (SP)+           ;  CLEAN THE STACK
        RTS     PC

$MULUS::        MOV             R0,-(SP)
        MOV     #21,-(SP)
        CLR     R0
10$:    ROR     R0
        ROR     R1
        BCC     20$
        ADD     2(SP),R0
20$:    DEC     @SP
        BGT     10$
        CMP     (SP)+,(SP)+
        RETURN
        .END
```

INDEX

RMS-11 MACRO-11
Reference Manual
Order No. AA-H683A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)_____

Name_____ Date_____

Organization_____

Street_____

City_____ State_____ Zip Code_____
or
Country

— — — — —Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed in th
United State

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

ATTN: Commercial Engineering Publications MK1-2/2H3
DIGITAL EQUIPMENT CORPORATION
CONTINENTAL BOULEVARD
MERRIMACK N.H. 03054

— — — — Do Not Tear - Fold Here and Tape —