February 1979

This document describes the VAX-11 Record Management Services (RMS). It provides detailed information on the use of VAX-11 RMS facilities with the VAX-VMS operating system.

VAX-11
Record Management Services
Reference Manual

Order No. AA-D031B-TE

SUPERSESSION/UPDATE INFORMATION: This document supersedes the document of the same name, Order No. AA-D031A-TE, published August 1978.

OPERATING SYSTEM AND VERSION: VAX/VMS V01.5

SOFTWARE VERSION: VAX/VMS V01.5

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, August 1978
Revised, February 1979

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid 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:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	PDT
DATATRIEVE	TRAX	

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

TABLES

CONTENTS

PREFACE

MANUAL OBJECTIVES

The intent of this manual is to enable VAX-11 MACRO programmers to use the VAX-11 Record Management Services (RMS) facilities with the VAX/VMS operating system.

Many data operations are the same (or similar), with slight alterations depending on the application. Using VAX-11 RMS and associated control routines, you can perform these operations by simply calling a VAX-11 RMS routine, with the appropriate parameters, rather than writing your own routines.

INTENDED AUDIENCE

VAX/VMS provides record management services for all the supported languages. Except for VAX-11 MACRO, each particular language manual provides the necessary information about performing record management. However, for the VAX-11 MACRO programmers, and for those higher-level language programmers who wish to call VAX-11 RMS directly, this manual contains the user interface to record management.

STRUCTURE OF THIS DOCUMENT

This manual consists of three parts, as follows:

Part I: Introduction to VAX-11 RMS

This part discusses VAX-11 RMS in terms of who uses it and why.

Part II: VAX-11 RMS Program Interface

In this part, Chapters 3 through 7 describe the fields for VAX-11 RMS structures, such as file declaration, and the macro instructions used to initialize these fields. In addition, Chapters 8 through 15 describe the interfaces to VAX-11 RMS file and record operations and control routines.

Appendixes

The appendixes summarize the concepts of files and records, provide formulas for determining file and record size, and list completion status codes.

ASSOCIATED DOCUMENTS

The following manuals are allied to this document:

- VAX-11 MACRO User's Guide
- VAX-11 MACRO Language Reference Manual
- VAX/VMS System Services Reference Manual
- VAX-11 FORTRAN IV-PLUS User's Guide
- VAX-11 FORTRAN IV-PLUS Language Reference Manual

For FORTRAN IV-PLUS programmers, the VAX-11 FORTRAN IV-PLUS manuals provide the necessary information for performing record management. For MACRO programmers, and for higher-level programmers who want to call VAX-11 RMS directly, this manual contains the user interface to record management.

The Introduction to VAX-11 Record Management Services manual contains introductory information about file services and structures in general, and about VAX-11 RMS in particular. The VAX-11 RMS User's Guide contains detailed information on using the capabilities of VAX-11 RMS efficiently; it also contains programming examples.

For a complete list of all VAX-11 documents, including brief descriptions of each, see the VAX-11 Information Directory.

SUMMARY OF CHANGES

This manual has been revised to reflect VAX/VMS support for indexed sequential file organization.

CHAPTER 1

WHAT IS VAX-11 RMS?

The VAX-11 Record Management Services (VAX-11 RMS) are generalized routines that assist the user programs in processing and managing files and their contents. VAX-11 RMS also includes a set of macro instructions that you can use to initialize control blocks and call VAX-11 RMS service routines.

1.1 VAX-11 RMS FUNCTIONS

VAX-11 RMS provides a variety of file organizations and record access modes that let you choose the processing techniques best suited to your application. VAX-11 RMS organizes files sequentially, relatively, or in indexed form. You can access the records in these files in a number of ways:

- Sequentially
- Randomly by key
- Randomly by the record's file address (RFA)
- Dynamically, which is an intermingling of sequential and random access

You transmit file and record operation requests to VAX-11 RMS through control blocks. Through these same control blocks, such as the File Access Block or Record Access Block, VAX-11 RMS returns to you the data contents of files, attribute information about the files, and status codes.

To use VAX-11 RMS, you must:

- Allocate and initialize control blocks
- Access fields in these control blocks at run time
- Request a particular file or record operation through the use of macro instructions

1.1.1 Allocating and Initializing Control Blocks

You communicate with VAX-11 RMS through control blocks. You must allocate space in your program for the control blocks; usually, this is done at assembly time. In addition, you can establish initial values for the fields in these blocks through assembly-time initialization macros.

1.1.2 Accessing Fields in Control Blocks

At run time, you can store values in the control block data fields through the use of macro instructions. You can access data in the control block fields directly by using the defined offsets for the fields.

1.1.3 Requesting File and Record Operations

Control blocks combined with a set of VAX-11 RMS file and record operation macro instructions form the complete run-time program interface with VAX-11 RMS. Each macro instruction represents a request for a particular VAX-11 RMS file or record service. The fields of the control blocks further describe the request. Using VAX-11 RMS macro instructions, you can:

- Create new files
- Process existing files
- Extend or delete files
- Read, write, update, or delete records within files

1.2 WHO USES VAX-11 RMS

VAX-11 MACRO programmers make direct use of the VAX-11 RMS routines. Programmers writing in a higher-level language, such as VAX-11 FORTRAN IV-PLUS, can write their programs to interface with VAX-11 RMS facilities either 1) directly through the use of a call facility in the language, or 2) indirectly through the input/output (I/O) instructions of the language. The latter interface is much more commonly used. Programs that interface directly with VAX-11 RMS can use all its capabilities, whereas programs that use an I/O statement of a higher-level language are restricted to the subset of VAX-11 RMS capabilities used by that language. This manual, describing the full VAX-11 RMS interface, is therefore directed primarily to the VAX-11 MACRO user. Higher-level language users should see the VAX-11 manuals specific to their language.

1.3 DEFINITION OF TERMS

The following glossary defines terms that appear throughout this manual.

alternate key
An optional key within the data records in an indexed file; used by VAX-11 RMS to build an alternate index. See key (indexed files) and primary key.

area
VAX-11 RMS-maintained regions of an indexed file which are used for allocating buckets. An area consists of any number of buckets, and there may be from 1 to 255 areas in a file.

block
A unit of I/O transfer. A block on a Files-11 disk structure is fixed at 512 bytes and contains one or more complete or partial records. A block on tape contains one or more complete records; its size is user-determined.

block I/O
An I/O technique using a set of VAX-11 RMS procedures that allow direct access to the blocks in a file, regardless of the file organization or record format.

bucket
A structure used to store and transfer blocks of data for a relative or indexed file. A bucket consists of from 1 to 32 blocks.

buffer
An area in memory used to store data temporarily during input or output operations.

cluster
The basic unit of space allocation on a Files-11 disk. A cluster consists of one or more blocks, as defined by the initializer of the disk.

directory name
The field in a file specification that identifies the directory in which the file is listed. It begins with a left bracket ([or <) and ends with a right bracket ( ] or >). The brackets enclose either a group number and a user number separated by a comma, or an alphanumeric directory list.

dynamic access
The process of switching from one record access mode to another while processing a file.

extent
One or more adjacent clusters allocated to a file or a portion of a file.

file
A collection of data; generally used to refer to data stored on a magnetic medium, such as a disk.

file header
A block in the index file that describes a file on a Files-11 disk. Every file residing on the disk has at least one file header, which provides the location of the file's extents.

file organization
The physical arrangement of data in a file. VAX-11 RMS uses three file organizations -- sequential, relative, and indexed.

file specification
The alphanumeric character string that a user assigns to identify a file.

Files-11
The standard VAX-11 RMS physical disk structure.

fixed control area
An area, prefixed to a variable-length record, containing additional information about the record that may have no bearing on the other contents of the record. For example, the fixed control area may contain line numbering or carriage control information.

fixed-length record format
The property of a file specifying that all records must be the same length. This format allows for simplicity in determining the exact location of a record in the file and eliminates the need to prefix a record size field to each record.

home block
A block in the volume's index file that contains information pertaining to the volume as a whole, such as volume label and protection.

index
The structure which allows retrieval by key value of records in an indexed file. See key (indexed files).

index file
The file on a Files-11 volume that provides the means for identification and initial access to the volume. The index file contains the access data for all files on the volume (including itself).

indexed file organization
A file organization which allows random retrieval of records by key value and sequential retrieval of records in sorted order by key value. See key (indexed files).

key
indexed files: A character string, a packed decimal number, a 2- or 4-byte unsigned binary number, or a 2- or 4-byte signed integer within each data record in an indexed file; it is user-defined as to length and location within the records; VAX-11 RMS uses the key to build an index. See primary key, alternate key, and random access by key (indexed files only).

key
relative files: The relative record number of each data record in a data file; VAX-11 RMS uses the relative record numbers to identify and access data records in a relative file in random access mode. See relative record number.

locate mode
Record transfer technique in which records stay in place while operations are performed. The records are not copied from the I/O buffer to a user buffer; the address of the record is returned to the user.

logical block number
The number assigned to a block on a disk volume, sequentially beginning with 0 to the number of blocks that will fit on the volume. See virtual block number.

move mode
Record transfer technique in which a record is copied between an I/O buffer and a user buffer.

primary key
The mandatory key within the data records of an indexed file; used by VAX-11 RMS to build a primary index; see key (indexed files) and alternate key.

process permanent file
A file opened or created through VAX-11 RMS in supervisor or executive mode. The internal data structures of a process permanent file are allocated such that the file may be open across image activations; a restricted subset of allowable operations is available to "indirect" accessors.

random access by key
indexed files only: Retrieval of a data record in an indexed file by the primary (or optionally, alternate) key within the data record. See key (indexed files).

relative files only: Retrieval of a data record in a relative file by the relative record number of the record. See key (relative files).

random access by record's file address
The retrieval of a record by the record's unique address that VAX-11 RMS returns to the user. This record access mode is the only means of randomly accessing a sequential file containing variable-length records.

random access by relative record number
The retrieval of a record by specifying the record's number relative to the beginning of the file. For relative files, random access by relative record number is synonymous with random access by key. See random access by key (relative files only).

record
A collection of related data within a file treated as a unit of information.

record access mode
The manner in which VAX-11 RMS selects the next record to be accessed, that is, sequentially or randomly.

record cell
A fixed-length area in a relative file that is capable of containing a record. The concept of a fixed-length record cell lets VAX-11 RMS make a direct calculation of the record's actual position in the file.

record's file address
The unique address of a record in a file. This address allows records to be accessed randomly regardless of file organization.

record format
The way a record physically appears on the recording surface of the storage medium. The record format defines the method for determining record length.

record locking
A facility that prevents concurrent access to a record by more than one record stream or process until the initiating record stream or process releases the record.

record length
The size of a record, expressed as a number of bytes.

relative file organization
The arrangement of records in a file where each record occupies a cell of equal length within a bucket. Each cell is assigned a successive number, which represents its position relative to the beginning of the file.

relative record number
An identification number that specifies the position of a record cell relative to the beginning of the file; used as the key during random access by key mode to relative files.

RFA
See Record's File Address

sequential file organization
The arrangement of records in a file in a sequential fashion. Records appear in the order in which they were written.

sequential record access mode
The retrieval or storage of records starting at a designated point in the file and continuing to access additional records in the order in which they logically appear.

spooling
The technique of using a high-speed mass storage device (such as a disk) to buffer data passing between high-speed main memory and low-speed I/O devices (such as line printers). The high-speed mass storage device (the intermediate device) temporarily stores the data passing to and from the low-speed device (the spooled device). The data is queued on the intermediate device to await transmission to the printer for printing (output spooling) or to the processor for processing (input spooling).

storage allocation
The assignment of space to a file on the recording medium.

user identification code
The number assigned to a user identifying the user and, consequently, determining the files to which the user has access. It consists of a group number and a user number, separated by a comma, and enclosed in brackets.

variable-length record format
The property of a file specifying that records need not be the same length.

variable with fixed-length control record format
The property of a file specifying that records of variable-length contain an additional fixed control area capable of storing data that may have no bearing on the other contents of the record. Variable with fixed-length control record format is not applicable to indexed files.

VAX-11 Record Management Services (VAX-11 RMS)
The file and record access system for the VAX/VMS operating system. VAX-11 RMS allows programs to issue requests at the record and block level.

virtual block number

The number assigned to a block of a file. This number refers to the position of the block relative to other blocks in the same file, instead of to its position relative to other blocks on the volume. Virtual block numbers are assigned to the blocks of a file beginning with 1. The file header provides relocation information for mapping the file's virtual block numbers to the volume's logical block numbers. See logical block number.

CHAPTER 2

STATEMENT CONVENTIONS

Throughout this manual, certain conventions apply to the syntax of the VAX-11 RMS macro instructions and control routines.

In examples, parameters other than the parameter under discussion are shown. The purpose of showing these additional parameters is to illustrate and reconfirm, throughout the manual, some of the conventions that apply in coding macro instructions, such as statement continuation and parameter separation. The parameter under discussion will be shown in red print.

For example:

```
$FAB  FNA=FLNAM ALQ=132 BKS=4
```

In coding VAX-11 RMS macro instructions, you follow the same coding rules used by the VAX-11 MACRO assembler. These rules are repeated below for ease of reference.

- Comments must be separated from the rest of the code line by a semicolon (;). For example:

```
$FAB    BKS=4    ;bucket size
```

- All the parameters necessary for a macro instruction must be coded on a single macro instruction. If the parameters needed do not all fit on one line (or if you do not want them on one line), you can type the continuation character -- hyphen (-) -- as the last character on the line, and then continue typing parameters on the next line. Comments can follow the hyphen, separated by the comment-delimiter semicolon -- they are not interpreted as code. For example:

```
$FAB    FNA=FLNAM  - ;  filename address
        ALQ=132    - ;  allocation quantity
        BKS=4        ;  bucket size
```

- Parameters and sub-parameters can be separated from each other by:

  a. a single comma, with or without spaces or tabs; the preferred usage is the comma without a space or tab. That is how coding examples appear in this manual.

```
FNA=FLNAM,ALQ=132
```

  b. a blank space

```
FNA=FLNAM ALQ=132
```

c. multiple blank spaces or tabs

```
FNA=FLNAM        ALQ=132
```

- Lowercase letters and words represent information that you must supply. Such lowercase information may contain hyphens for readability. The accompanying text defines the information to be supplied. For example:

```
window-size
address
```

- Uppercase letters and words, equal signs (=), angle brackets (<>), and dollar signs ($), must be coded as shown. For example:

```
RAT=<BLK,CR>
$OPEN
```

- Information enclosed within braces indicates that you may choose any one of the enclosed values. For example:

```
FIX
VAR
VFC
UDF
```

- Each option has its own symbolic bit offset and mask value. The bit offset is formed by prefixing FAB$V_ to the option value. For example:

```
FAB$V_PUT
```

The mask value is formed by prefixing FAB$M_ to the option value. For example:

```
FAB$M_PUT
```

CHAPTER 3

THE PROGRAM INTERFACE WITH VAX-11 RMS

You gain access to the VAX-11 RMS facilities at run time by calling record management services. Your program and VAX-11 RMS exchange information by means of user control blocks defined within your program. This chapter provides an introduction to these services and user control blocks, and the macro instructions that facilitate their use.

With each request for a VAX-11 RMS service, you must place the information detailing this request in a user control block. For example, a request to open a file must be accompanied by the name of the file, information on sharing the file, and details on accessing the file. Or, as another example, a program request to read a record from a file must specify a record access mode, or perhaps a buffer size.

Once a request for a service is satisfied, VAX-11 RMS uses the same user control block to return information to your program. For example, when the file is successfully opened, VAX-11 RMS returns attribute information, such as file organization and record format. Or, when a record is retrieved from a file, VAX-11 RMS provides your program with the record's length and location in memory.

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

The following sections provide a broad overview of the interface that a program uses when requesting VAX-11 RMS services. The remaining chapters of Part II present detailed information on using the VAX-11 RMS declarative and imperative macro instructions. The declarative macro instructions allocate and initialize file access blocks (FABs), record access blocks (RABs), name blocks (NAMs), and extended attribute blocks (XABs). The imperative macro instructions invoke VAX-11 RMS operations to manipulate files and records.

3.1 USER CONTROL BLOCKS

User control blocks are formatted areas in your program, which you must allocate. Your program and VAX-11 RMS use the data fields in these blocks to exchange information.

Usually, you allocate space for user control blocks at assembly time. Optionally, you can also set values for the fields in these blocks either initially or at run time. The VAX-11 RMS declarative macro instructions perform the functions that support assembly-time allocation and initialization. For efficiency, align the control blocks on a longword boundary; if you do not, you will receive a warning message from the assembler.

Table 3-1 lists the user control blocks that are part of your program interface with VAX-11 RMS. The Macro Name column shows the VAX-11 RMS macro instruction you use to allocate space for the control block. Chapters 4 through 7 describe these macro instructions.

Table 3-1
User Control Blocks

Block Name		Function	Macro Name
File Access Block	FAB	Describes a file and contains file-related information	$FAB
Record Access Block	RAB	Describes a record and contains record-related information	$RAB
Extended Attribute Blocks	XAB	Contains file attribute information beyond that in the File Access Block	$XABxxx[1]
Name Block	NAM	Contains file specification information beyond that in the File Access Block	$NAM

[1] xxx is a 3-character XAB type specification.

3.2 VAX-11 RMS RUN-TIME OPERATIONS

To create and process VAX-11 RMS files, your program must contain calls to appropriate VAX-11 RMS routines. Generally, you make these calls by using the VAX-11 RMS imperative macro instructions for run-time processing. The expanded code of these macro instructions, when encountered at run time, causes calls to be made to the corresponding VAX-11 RMS routine. Each macro instruction, and the resultant call, represents a program request for either a file or record related service, or block I/O transfer operation.

Table 3-2 summarizes the run-time processing macro instructions. Chapters 8 through 15 describe these macro instructions.

Table 3-2
Run-Time Processing Macro Instructions

Category	Macro Name	Service
File Processing	$CREATE	Creates and opens a new file of any organization
	$OPEN	Opens an existing file and initiates file processing
	$DISPLAY	Returns the attributes of a file to user program
	$EXTEND	Extends the allocated space of a file
	$CLOSE	Terminates file processing and closes the file
	$ERASE	Deletes a file and removes its directory entry
Record Processing	$GET	Retrieves a record from a file
	$PUT	Writes a new record to a file
	$UPDATE	Rewrites an existing record in a file
	$DELETE	Deletes a record from a relative file
	$FIND	Locates and positions to a record and returns its RFA
	$CONNECT	Associates and connects a RAB to a file
	$DISCONNECT	Disconnects a RAB from a file
	$RELEASE	Unlocks a record pointed to by the contents of the RFA field of the RAB
	$FREE	Unlocks all previously locked records
	$WAIT	Determines the completion of an asynchronous record operation
	$REWIND	Positions to the first record of a file
	$TRUNCATE	Truncates a sequential file
	$FLUSH	Write modified I/O buffers and file attributes
	$NXTVOL	Causes processing of a magnetic tape file to continue to the next volume of a volume set
Block I/O	$READ	Retrieves a specified number of bytes from a file
	$WRITE	Writes a specified number of bytes to a file
	$SPACE	Spaces forward or backward in a file
File Naming	$ENTER	Enters a file name into a directory
	$PARSE	Parses a file specification
	$REMOVE	Removes a file name from a directory
	$RENAME	Assigns a new name to a file
	$SEARCH	Searches a directory for a file name

CHAPTER 4

THE FILE ACCESS BLOCK

This chapter describes the File Access Block (FAB), the fields in the FAB, and the parameters of the $FAB macro instruction.

4.1 THE PURPOSE OF THE FILE ACCESS BLOCK

The FAB is a user control block that describes a particular file. The fields of the FAB contain file-related information, such as:

- The name of the file
- The file organization
- The record format
- Disk storage space allocation information

You allocate a FAB with a $FAB macro instruction, and initialize the fields of the FAB either at assembly time (through keyword parameters) or by direct manipulation at run time. You initialize the FAB at run time through either keyword parameters with the $FAB_STORE macro instruction (see Chapter 14) or the defined symbolic offsets.

Each field in the FAB has a 3-character mnemonic name. All access to these fields is through this name (by keyword or offset). However, some of the fields are static or output-only; therefore, you need not initialize them. Table 4-1 summarizes the fields of the FAB, including the static and output-only fields.

Table 4-1
File Access Block Fields

Field & Keyword Name	Field Size (units of 1)	Description	Offset
ALQ	longword	Allocation quantity	FAB$L_ALQ
BID[1]	byte	Block identifier	FAB$B_BID
BKS	byte	Bucket size	FAB$B_BKS
BLN[1]	byte	Block length	FAB$B_BLN
BLS	word	Block size	FAB$W_BLS
CTX	longword	Context	FAB$L_CTX
DEQ	word	Default file extension quantity	FAB$W_DEQ
DEV[2]	longword	Device characteristics	FAB$L_DEV
DNA	longword	Default file specification string address	FAB$L_DNA
DNS	byte	Default file specification string size	FAB$B_DNS
FAC	byte	File access	FAB$B_FAC
FNA	longword	File specification string address	FAB$L_FNA
FNS	byte	File specification string size	FAB$B_FNS
FOP	longword	File-processing options	FAB$L_FOP
FSZ	byte	Fixed control area size	FAB$B_FSZ
IFI[2]	word	Internal file identifier	FAB$W_IFI
MRN	longword	Maximum record number	FAB$L_MRN
MRS	word	Maximum record size	FAB$W_MRS
NAM	longword	Name block address	FAB$L_NAM
ORG	byte	File organization	FAB$B_ORG
RAT	byte	Record attributes	FAB$B_RAT
RFM	byte	Record format	FAB$B_RFM
RTV	byte	Retrieval window size	FAB$B_RTV
SDC[2]	longword	Spooling device characteristics	FAB$L_SDC
SHR	byte	File sharing	FAB$B_SHR
STS[2]	longword	Completion status code	FAB$L_STS
STV[2]	longword	Status values	FAB$L_STV
XAB	longword	Extended attribute block address	FAB$L_XAB

[1] Indicates statically initialized field (by $FAB macro instruction) to identify this control block as a FAB.
[2] Indicates nonuser-initialized field.

4.2 FAB ALLOCATION

The format of the $FAB macro instruction is shown below. Every parameter is optional, depending on the function to be performed with the FAB and the combination of parameters in the macro instruction as a whole.

Format:

OPERATION	PARAMETERS
label: $FAB	ALQ=allocation-qty
	BKS=bucket-size
	BLS=block-size
	CTX=value
	DEQ=extension-qty
	DNA=address
	DNM=<filespec>
	DNS=value
	FAC=<PUT GET DEL UPD TRN BIO BRO>
	FNA=address
	FNM=<filespec>
	FNS=value
	FOP=<CBT CIF CTG DFW DLT ESC INP KFO MXV NAM NEF NFS OFP POS PPF RCK RWC RWO SCF SPL SQO SUP TEF TMD TMP UFM UFO WCK>
	FSZ=header-size
	MRN=max-rec-number
	MRS=max-rec-size
	NAM=nam-address
	ORG={REL / SEQ / IDX}
	RAT=<BLK {CR / FTN / PRN}>
	RFM={FIX / VAR / VFC / UDF}
	RTV=window-size
	SHR=<PUT GET DEL UPD NIL MSE UPI>
	XAB=xab-address

The $FAB macro instruction allocates and initializes storage for a FAB. You cannot use this macro instruction within a sequence of executable instructions.

You need one FAB for each open file in your program. After you close the file, the space used by the FAB can be reused for some other purpose (perhaps a FAB from another program).

Since VAX-11 RMS returns information in the fields of the FAB, you therefore cannot allocate a FAB in read-only sections.

label: $FAB

4.2.1 Label

You can use the label field of the $FAB macro instruction to name a FAB and thereby to refer to a particular FAB within your program. The label field is optional, but when used, must precede the symbol $FAB and be separated from $FAB by a colon (:). For example:

```
INFAB:  $FAB
```

$FAB ALQ

4.2.2 Allocation Quantity

You can use the ALQ parameter to initialize the allocation quantity field at assembly time. With this field you can specify the amount of space, in blocks, to be initially allocated to a disk file when it is created, or to be added to the file when it is explicitly extended (through a $EXTEND macro instruction).

FORMAT

```
ALQ=allocation-quantity
```

allocation-quantity
A numeric value representing a number of blocks, in the range of 0 through 4,294,967,295. A value of 0 indicates no allocation.

For example, to set an allocation quantity of 132 blocks, the coding is:

```
$FAB  ALQ=132
```

The symbolic offset for this field is:

```
FAB$L_ALQ
```

USER CONSIDERATIONS

1. When you create a new file with a $CREATE macro instruction, VAX-11 RMS interprets the value in the allocation quantity field as the number of blocks for the initial extent of the file. If the value is 0, the minimum number of blocks for the specific file organization is the allocation quantity used for the initial extent. For example, in indexed files, the number of blocks necessary to contain key and area definitions is used as the initial extent quantity when ALQ=0.

2. When an existing file is opened with a $OPEN macro instruction, VAX-11 RMS sets the allocation quantity field to indicate the highest virtual block number currently allocated to the file.

3. Before extending a file with a $EXTEND macro instruction, you must set the allocation quantity field equal to the number of blocks to be added to the file. You cannot use an extension size of 0.

4. When you use the $CREATE and $EXTEND macro instructions, the allocation quantity value is rounded up to the next cluster boundary; the number of blocks actually allocated is returned in the allocation quantity field.

NOTE

The function of the allocation quantity field with the $CREATE and $EXTEND macro instructions is different from the preceding description if allocation XABs are present during the operation. Chapter 6 describes allocation XABs and their effect on the allocation quantity field during file creation or extension.

$FAB BKS

4.2.3 Bucket Size

The BKS parameter initializes the bucket size field at assembly time. This field is used only for relative or indexed files. When you open an existing relative or indexed file, VAX-11 RMS sets the bucket size field to the defined size of the buckets in the file. However, when you create a new relative or indexed file, you must set the bucket size field before you issue the $CREATE macro instruction.

NOTE

If allocation control XABs are specified, the value specified in the XAB BKZ field will supersede the value specified in the FAB BKS field. Refer to Section 6.5.6 for a description of the XAB BKZ parameters.

FORMAT

BKS=bucket-size

bucket-size
A numeric value, in the range of 0 to 32, representing the number of blocks in each bucket of the file. If you omit this parameter or use a value of 0, you receive a default size equal to the minimum number of blocks required to contain a single record.

For example, to set the bucket size to 4, the syntax is:

```
$FAB    BKS=4,ALQ=132
```

The symbolic offset for this field is:

```
FAB$B_BKS
```

USER CONSIDERATIONS

In specifying a bucket size, you must be aware of the relationship between bucket size and record size. Since VAX-11 RMS does not allow records to cross bucket boundaries, you must ensure that the number of blocks per bucket conforms to one of the following formulas:

- Relative files with fixed-length records:

  ```
  Bsiz = ((Rlen+1)*Rnum)/512
  ```

  where

  Bsiz — is the number of blocks per bucket rounded up to the next higher integer. The result must be in the range from 1 to 32.

  Rlen — is the fixed record length.

  Rnum — is the number of records that you want in each bucket.

- Relative files with variable-length records:

  ```
  Bsiz = ((Rmax+3)*Rnum)/512
  ```

  where

  Bsiz — is the same as described above.

  Rmax — is the maximum size of any record in the file.

  Rnum — is the number of records that you want in each bucket. Variable-length records in a relative file bucket always occupy Rmax+3 bytes.

- Relative files with variable with fixed-length control records:

  ```
  Bsiz = ((Rmax+Fsiz+3)*Rnum)/512
  ```

  where

  Bsiz — is the same as described above.

  Rmax — is the maximum size of the data portion of any record in the file.

  Fsiz — is the size of the fixed control area portion of the records.

  Rnum — is the number of records that you want in each bucket. Variable with fixed-length control records in a relative file bucket always occupy Rmax+Fsiz+3 bytes.

- Indexed files with fixed-length records:

  Bsiz = ((Rlent+7)*Rnum)+15/512

  where

Bsiz	is the number of blocks per bucket rounded up to the next higher integer. The result must be in the range of from 1 to 32.
Rlen	is the fixed record length.
Rnum	is the number of records that you want in each bucket. Fixed-length records in an indexed file bucket always occupy Rlen plus seven bytes of record overhead. Fifteen bytes are required for bucket overhead.

- Indexed files with variable-length records:

  Bsiz = ((Rmax+9)*Rnum)+15/512

  where

Bsiz	is the same as described above.
Rmax	is the maximum size of any record in the file.
Rnum	is the number of records that you want in each bucket. Variable-length records in an indexed file bucket always occupy Rmax plus nine bytes of record overhead. Fifteen bytes are required for bucket overhead.

NOTE

Another consideration in choosing a bucket size for indexed file is the default cluster size of the disk. Since all extents are rounded up to the next cluster boundary, the cluster size should be an even multiple of the bucket size. This prevents the waste of blocks in an extent because there will be enough blocks to completely fill a bucket.

$FAB BLS

4.2.4 Block Size

The BLS parameter is used as input only for magnetic tape files. In all other cases, VAX-11 RMS ignores it. When you open an existing file with a $OPEN macro instruction, VAX-11 RMS returns the block size if the file is organized sequentially. However, when you create a magnetic tape file, you can set the block size field before you issue the $CREATE macro instruction.

FORMAT

```
BLS=block-size
```

block-size
The size, in bytes, of the blocks on the tape, in the range of 18 to 65535. If this parameter is 0, the default selected when the volume was mounted is used.

For example, to set the block length to 4096, the syntax is:

```
$FAB     BLS=4096,MRS=132
```

The symbolic offset for this field is:

```
FAB$W_BLS
```

NOTE

To create a magnetic tape for interchange with other DIGITAL operating systems (non-VAX/VMS), you should consult the documentation for the target system regarding possible limitations on block size. To ensure compatibility with non-DIGITAL systems, the block size should be less than or equal to 2048 bytes.

$FAB CTX

4.2.5 User Context

The CTX parameter conveys user information to a completion routine in your program. The user context field set by this parameter is intended solely for your use; VAX-11 RMS never uses it for record management activities.

FORMAT

```
CTX=value
```

value
represents any user-specified value, up to four bytes long.

For example, to pass along the symbolic value T1DONE, the syntax is:

```
$FAB     CTX=T1DONE,BKS=4
```

The offset for this field is:

```
FAB$L_CTX
```

$FAB DEQ

4.2.6 Default File Extension Quantity

The DEQ parameter sets the default file extension quantity field, which specifies the number of blocks to add when a disk file is extended automatically. This automatic extension occurs whenever your program performs an operation with a $PUT or $WRITE macro instruction and the currently allocated space is exhausted.

FORMAT

DEQ = extension-quantity

extension-quantity
The number of blocks to be added when automatic extension is required. This number must be in the range of 0 to 65,535 and is rounded up to the next cluster boundary. If you specify 0, the file will be extended using a VAX-11 RMS determined default extension value.

For example, to specify a default extension quantity of 80 blocks, the syntax is:

```
$FAB DEQ=80
```

The offset for this field is:

```
FAB$W_DEQ
```

USER CONSIDERATIONS

1. When creating a new file, you can specify the extension quantity for the file by setting the desired value in the default extension quantity field before issuing a $CREATE macro instruction. This value becomes a permanent attribute for the file.

2. When processing an existing file, you can temporarily override the default extension quantity specified when the file was created. To do this, set the desired value before issuing the $OPEN macro instruction. Once the file is closed, the default extension quantity reverts to the value set when the file was created.

$FAB DNA

4.2.7 Default File Specification String Address

You can use the DNA parameter to set program defaults in the default file specification string address field for the missing components (if any) of the file specification string pointed to by the file specification string address field.

The default file specification string is used primarily when accepting file specifications interactively; file specifications known to a user program are normally completely specified in the file specification string address and size fields (the FNA and FNS parameters). You can specify defaults for one or more of the following file specification components:

- Node
- Device
- Directory
- File name
- File type
- File version number

FORMAT

```
DNA = address
```

address
The symbolic address of an ASCII string containing one or more components of a file specification. The components in the string must be in the order in which they would occur in a complete file specification.

For example, assume an ASCII string is stored at a memory location whose symbolic address is DFNAM: To store the address of this string in the default file specification string address field, so that DFNAM will be used during execution of a $OPEN or $CREATE macro instruction, the syntax is:

```
$FAB DNA=DFNAM,DNS=4
```

This default file specification string address is only effective if the components are missing from the string whose address is stored in the file specification address field.

The offset to this field is:

```
FAB$L_DNA
```

Section 4.2.9 describes another technique -- using the DNA parameter -- for setting the default file specification string address.

$FAB DNS

4.2.8 Default File Specification String Size

The DNS parameter sets a value in the default file specification string size field. This value indicates the size, in bytes, of the string whose address is contained in the default file specification string address field.

FORMAT

```
DNS=value
```

value
A symbolic or numeric value representing the size of the default file specification string. The numeric value is in the range of 1 to 255.

For example, assume that your program contains the directive:

```
DFNAM:  .ASCII  /.DAT/
```

The following DNS parameter would set the default file specification string size field:

```
$FAB    DNS=4,DNA=DFNAM
```

The offset for this field is:

```
FAB$B_DNS
```

Section 4.2.9 describes another technique -- using the DNM parameter -- for setting the default file specification string size.

$FAB DNM

4.2.9 Default File Specification

The DNM parameter sets two fields in the FAB: the default file specification string address (DNA) and the default file specification string size (DNS). The specified default file specification string is stored in the special program section $RMSNAM.

FORMAT

```
DNM=<filespec>
```

<filespec>
The ASCII default file specification string. The angle brackets (<>) are required syntax.

For example:

```
$FAB    DNM=<.DAT>
```

$FAB FAC

4.2.10 File Access

The FAC parameter initializes the file access field at assembly time. You must indicate to VAX-11 RMS what types of operations you intend to perform on the file. After you open a file, VAX-11 RMS rejects any operation your program attempts if that operation was not specified in the file access field when you issued a $OPEN or $CREATE macro instruction for the file.

If your program will issue any of the following macro instructions, you must specify them by setting the file access field for the appropriate operation:

- $DELETE
- $FIND
- $GET

- $PUT
- $READ
- $SPACE
- $TRUNCATE
- $UPDATE
- $WRITE

FORMAT

FAC=<BIO,BRO,DEL,GET,PUT,TRN,UPD>

BIO
Used for block I/O operations involving a $READ or $WRITE macro instruction, with Get and Put access, respectively, and also with a $SPACE macro instruction. Furthermore, specifying block I/O prohibits the use of any record I/O operations (GET, PUT, DEL, UPD, TRN).

BRO
Similar to BIO, except that record I/O operations are also allowed.

DEL
Allows operations with a $DELETE macro instruction.

GET
Allows operations with a $GET or $FIND macro instruction. This is the default when you are opening this file and either the FAC parameter is not specified or the DEL, UPD, or TRN operations are specified on the FAC parameter. If you specify GET with either BIO or BRO, you can perform operations with a $READ macro instruction.

PUT
Allows operations with a $PUT macro instruction. This will be the default if you are creating this file. If you specify PUT with either BIO or BRO, you can perform operations with a $WRITE macro instruction.

TRN
Allows operations with a $TRUNCATE macro instruction. Also allows use of the truncate put (TPT) record option on a $PUT and $WRITE macro instruction (see Section 5.2.14).

UPD
Allows operations with a $UPDATE macro instruction.

You may specify more than one operation with the FAC parameter. However, if you do, the group of operations must be enclosed in angle brackets; when only one operation is specified, no angle brackets are needed. Multiple operations can be specified in any order. For example, <GET,PUT,UPD> or <UPD GET PUT>.

The following example indicates that operations with a $PUT macro instruction are going to be performed.

```
$FAB    FAC=PUT,ALQ=132,DEQ=16
```

A request for operations with $GET, $PUT, and $UPDATE macro instructions would be specified as follows:

```
$FAB    FAC=<GET,UPD,PUT>
```

Each operation has its own symbolic bit offset and mask value. The bit offset for each is formed by prefixing FAB$V_ to the operation value. For example:

```
PUT -- FAB$V_PUT
```

The mask value is formed by prefixing FAB$M_ to the operation value. For example:

```
PUT -- FAB$M_PUT
```

The offset for the file access field is:

```
FAB$B_FAC
```

$FAB FNA

4.2.11 File Specification String Address

The FNA parameter initializes the file specification string address field. This parameter works with the FNS parameter, which initializes the file specification string size field (see Section 4.2.12). The file specification string address contains the address of an ASCII string that specifies the path to a file to be processed. If this string does not contain all the components of a full file specification, VAX-11 RMS will use the defaults supplied in the default file specification string address and size fields (see Sections 4.2.7, 4.2.8, and 4.2.9). If no default string is present, or if the file specification is still incomplete, VAX-11 RMS provides further defaulting (see Section 8.2).

FORMAT

```
FNA=address
```

address
The symbolic address of an ASCII string containing the file specification.

For example, assume that the following directive is in your program:

```
FLNAM:  .ASCII  /MASTER.OLD/
```

The syntax for the FNA parameter is:

```
$FAB    FNA=FLNAM,FNS=10
```

The offset for this field is:

```
FAB$L_FNA
```

See 4.2.13 for an alternate method of setting the file specification string address field.

$FAB FNS

4.2.12 File Specification String Size

The FNS parameter initializes the file specification string size field. This field describes the length, in bytes, of the ASCII string pointed to by the file specification string address field (FNA).

FORMAT

```
FNS=value
```

value
A numeric or symbolic value representing the size, in bytes, of the file specification string, in the range of 0 to 255.

For example, assume that the following directive is in your program:

```
FLNAM:  .ASCII  /INPUTFILE:/
```

The syntax for the FNS parameter is:

```
$FAB    FNS=10,FNA=FLNAM
```

The offset for this field is:

```
FAB$B_FNS
```

Section 4.2.13 describes another technique -- using the FNM parameter -- for setting the file specification string size field.

$FAB FNM

4.2.13 File Specification

The FNM parameter sets two fields in the FAB: the file specification string address and the file specification string size. It causes the specified string to be stored in the special program section named $RMSNAM.

FORMAT

```
FNM = <filespec>
```

<filespec>
The ASCII file specification string; the angle brackets (<>) are required syntax.

For example:

```
$FAB    FNM=<DISK:[DATA]FILE.DAT>,ALQ=132
```

4.2.14 File Process Options

The FOP parameter sets indicators in the file-processing options field that represent requests for optional file-handling operations.

FORMAT

```
FOP=<CBT,CIF,CTG,DFW,DLT,ESC,INP,KFO,MXV,NAM,NEF,NFS,OFP,POS,
     PPF,RCK,RWC,RWO,SCF,SPL,SQO,SUP,TEF,TMD,TMP,UFM,UFO,WCK>
```

With the exception of the CBT, CTG, RCK, and WCK bits, the contents of this field are not modified by VAX-11 RMS operations.

Each option is interpreted as follows:

CBT

Contiguous best try; indicates that the file is to be allocated contiguously on a "best effort" basis. It is input to the create service, and is output from the open service to indicate the file status. Note that the file will take on the contiguous best try attribute only if a space allocation is actually performed. The CBT option takes precedence over the CTG option (below).

CIF

Create if; causes the file to be created if it does not exist; otherwise the create service acts as an open service. That is, it performs all processing as described for an open operation. The CIF option takes precedence over the SUP option.

CTG

Contiguous; indicates that the space for the file is to be allocated contiguously. If this cannot be done, the operation fails. It is input to the create service, and is output by the open service to indicate the status of the file. The CBT option (above) takes precedence over the CTG option.

DFW

Deferred write; indicates that writing back to the file of modified I/O buffers is to be deferred until the buffer must be used for other purposes. This applies to relative files and indexed files.

DLT

Delete; indicates that the file is to be deleted when it is closed; this option may be specified on a close, create, or open service. You can specify the DLT option with the SCF or SPL option. When using this option, you should normally use a NAM block so that the file's directory entry is also removed.

ESC

Escape; indicates nonstandard VAX-11 RMS processing; for DIGITAL-supplied component usage.

INP

Input; indicates that this process permanent file is the system command file named SYS$INPUT; for DIGITAL-supplied component usage.

KFO

Known file open; indicates a search of the known file list for DIGITAL-supplied component usage.

MXV

Maximize version; indicates that the version number of the file should be the maximum of the explicit version number given in the file specification and should be one greater than the highest version number for an existing file in the same directory with the same file name and file type.

NAM

NAM block inputs; indicates that the NAM block specified in the name block address field is to be used to provide:

- The device identification, file identification, and/or the directory identification when the file is being opened, closed, or deleted
- The device identification and the directory identification when the file is being created

NEF

Not end of file; inhibits the positioning to the end of file when a tape file is opened and the file access field of this FAB indicates a PUT operation.

NFS

Nonfile structured; indicates on open or create that the volume is to be processed in a nonfile-structured manner. This allows the use of volumes created on non-DIGITAL systems.

OFP

Output file parse; specifies that the related file resultant file specification string, if used, is to provide file name and file type defaults only (see Section 8.2).

POS

Current position; indicates that the magnetic tape volume set should be positioned immediately after the most recently closed file when the next file is created. However, if the RWO option of this field is also set, it overrides the POS option and positions to the beginning of the volume set.

PPF

Process-permanent file; specifies that the file's internal VAX-11 RMS structures are to be allocated in the process I/O segment. The file can then be left open across images. This option applies only to DIGITAL-supplied component usage.

RCK

Read-check; specifies that transfers from disk volumes are to be checked by a follow-up read-compare operation. This is an input to the create service, and is filled in by the open service to indicate the default for the file. If RCK is set when you open the file, checking is performed for the duration of the open, regardless of the default.

RWC

Rewind on close; specifies that the magnetic tape volume is to be rewound when the file is closed. This option can be specified for the close, create, or open services.

RWO

Rewind on open; specifies that the magnetic tape volume is to be rewound before the file is opened or created. The RWO option takes precedence over the POS option (above).

SCF

Submit command file; indicates that the file is to be submitted as a batch-command file to the process-default batch queue when the file is closed. This option can be specified for the close, create, or open services.

SPL

Spool; indicates that the file is to be spooled to the process default print queue when the file is closed. When using this option, you should normally use a NAM block and specify the NAM option (of this file-processing options field) so that the resultant file specification string is available. This option can be specified for the close, create, or open services for sequential files only.

SQO

Sequential only; indicates that this file can be processed sequentially only, thus allowing certain processing optimizations. Any attempt to perform random access will result in an error. This option is input to the create and open services and applies to the find, get, and put services for sequential files, and to all network operations.

SUP

Supersede; allows an existing file to be superseded on a create service by a new file of the same name, type, and version. The CIF option (above) takes precedence over the SUP option.

TEF

Truncate at end of file; indicates that unused space allocated to a file is to be deallocated on a close service.

TMD

Temporary marked for delete; indicates that a temporary file is to be created, and then deleted when the file is closed. The TMD option takes precedence over the TMP option (below).

TMP

Temporary; indicates that a temporary file is to be created and retained, but that no directory entry will be made for this file. The TMD option (above) takes precedence over the TMP option.

UFM

User file mode; indicates that the channel for the file is to be assigned in user mode. This applies only if the ESC and either the NFS or UFO options are also set. This option is provided for DIGITAL-supplied component usage.

UFO

User file open; indicates that VAX-11 RMS will open or create the file only. No further VAX-11 RMS operations can be done with this file. To perform any further processing on the file, you must use the QIO system service with the channel number that is returned in the status value field (STV). For the create service, the end of file mark will be set to the end of the block specified in the allocation options field on input (see Section 4.2.2). For either the open or create services, the IFI field is set to 0 on return to indicate that VAX-11 RMS cannot perform any more operations on the file.

WCK

Write-check; indicates that transfers to disk are to be checked by a follow-up read-compare. Similar to the RCK option.

You can specify more than one option with the FOP parameter. However, if you do, you must enclose the group of options in angle brackets. When you specify only one option, no angle brackets are needed. The options can be specified in any order.

For example, to rewind a tape file as part of the close operation, the syntax is:

```
$FAB    BLS=4096,FOP=RWC
```

Each option has its own symbolic bit offset and mask value. The bit offset for each is formed by prefixing FAB$V_ to the option value. For example:

```
CIF -- FAB$V_CIF
```

The mask value is formed by prefixing FAB$M_ to the option value. For example:

```
CIF -- FAB$M_CIF
```

The offset for the file-processing options field is:

```
FAB$L_FOP
```

$FAB FSZ

4.2.15 Fixed Control Area Size

The FSZ parameter initializes the fixed control area size field, which is used when dealing with variable with fixed control records. When you create a file with this type of record, you must set the value for the fixed control area before you issue the $CREATE macro instruction. When you open an existing file that contains variable with fixed control records, VAX-11 RMS sets this field equal to the value specified when the file was created. The FSZ parameter is not applicable to indexed files.

FORMAT

```
FSZ=header-size
```

header-size
The numeric value, in bytes, of the size of the fixed control area, in the range of 1 to 255. The default size is 2 bytes. If you specify 0, then the default size is used.

For example, if each variable with fixed control record is to have an 8-byte fixed control area, the syntax is:

```
$FAB    FOP=WCK,FSZ=8
```

The offset for this field is:

```
FAB$B_FSZ
```

$FAB MRN

4.2.16 Maximum Record Number

The MRN parameter sets the maximum record number field, which indicates the highest record number that can be written into this file. You can use this parameter only for relative files. If you attempt to put or get a record with a higher relative record number than the specified limit, an error will occur and VAX-11 RMS will return a message indicating an illegal maximum record number. If, however, you specify 0, checking is suppressed.

FORMAT

```
MRN=max-rec-number
```

max-rec-number
Numeric value of the highest numbered record allowed in the file, in the range of 0 to 2,147,483,647. The default for this parameter is 0.

For example, to set the highest relative record number at 10000, the syntax is:

```
$FAB    MRN=10000,FOP=WCK
```

The offset for this field is:

```
FAB$L_MRN
```

NOTE

VAX-11 RMS does not maintain the relative record number of the highest existing record in the file. If you require this information, you must maintain the relative record number yourself.

$FAB MRS

4.2.17 Maximum Record Size

The MRS parameter sets the maximum record size field, which indicates, in bytes, the size of the records in the file.

For fixed-length records, the value represents the actual size of each record in the file. You must specify a size when you create a file.

For variable-length records, the value represents the size of the largest record that can be written into the file. If the file is not a relative file, a value of 0 is used to suppress record size checking, thus indicating that there is no user limit on record size. However, the record size must conform to physical limitations, such as that, in indexed and relative files, records may not cross bucket boundaries.

For variable with fixed control records, the value includes only the data portion; it does not include the size of the fixed control area.

For all relative files, the size is used to determine the size of the record cell, and is used in conjunction with the bucket size field (see Section 4.2.3).

You specify a value when you issue a $CREATE (or $MODIFY) macro instruction. VAX-11 RMS returns the maximum record size when you issue a $OPEN macro instruction.

FORMAT

```
MRS=max-rec-size
```

max-rec-size
The record size, in bytes, in the range of 0 to 32,767 (16,383 for relative files) less any fixed control area (variable to 255 bytes) and control overhead information (variable to 4 bytes).

For example, to set a maximum record size of 512 bytes, the syntax is:

```
$FAB    MRS=512,MRN=10000
```

The offset for this field is:

```
FAB$W_MRS
```

NOTE

The length of the largest record actually existing in a sequential file with variable or VFC record format is also maintained by VAX-11 RMS and is available through the file header characteristics XAB (LRL field of $XABFHC) (see Section 6.9).

$FAB NAM

4.2.18 Name Block Address

The NAM parameter lets you set a symbolic address in the name block address field of the FAB. This address points to the NAM block you want to use when performing an operation, such as an open or create, on a file. The NAM block, which is described in Chapter 7, is required only in conjunction with the file specification processing macro instructions (see Chapter 13).

FORMAT

```
NAM=nam-address
```

nam-address
The symbolic address of the NAM block.

For example, if a $NAM macro instruction for a NAM block has a label of NMBLK, the syntax is:

```
$FAB    MRS=512,MRN=1000,NAM=NMBLK
```

The offset for this field is:

FAB$L_NAM

$FAB ORG

4.2.19 File Organization

The ORG parameter sets the file organization field, indicating the arrangement of the data in the file. You must set this field before you issue a $CREATE macro instruction. VAX-11 RMS returns the contents of this field when you issue a $OPEN macro instruction.

FORMAT

$$\text{ORG=} \begin{Bmatrix} \text{REL} \\ \text{INDX} \\ \text{SEQ} \end{Bmatrix}$$

REL
Relative file organization.

IDX
Indexed file organization.

SEQ
Sequential file organization. This is the default.

For example, to set the file organization field to relative, the syntax is:

```
$FAB    MRN=1000,ORG=REL,MRS=512
```

Each organization has its own symbolic value.

- REL -- FAB$C_REL
- IDX -- FAB$C_IDX
- SEQ -- FAB$C_SEQ

The offset for this field is:

FAB$B_ORG

$FAB RAT

4.2.20 Record Attributes

The RAT parameter initializes the record attributes field with special control information pertaining to the records in the file. If you need this information, set this field before you issue a $CREATE macro instruction. VAX-11 RMS sets the field when you issue a $OPEN macro instruction.

FORMAT

RAT=<BLK {CR / FTN / PRN} >

BLK

Indicates that records do not cross block boundaries. This information applies to sequential files only.

CR

Indicates that each record is to be preceded by a line feed and followed by a carriage return when the record is written to a carriage control device such as a line printer or terminal.

FTN

Indicates that the first byte of each record contains a FORTRAN (ASA) carriage control character, defined as follows:

Byte 0 Value (hexadecimal)	ASCII Character	Meaning
20	(space)	Single-space carriage control. (Sequence: LINE FEED, print buffer contents, RETURN.)
30	0	Double-space carriage control. (Sequence: LINE FEED, LINE FEED, print buffer contents, RETURN.)
31	1	Page eject carriage control. (Sequence: FORM FEED, print buffer contents, RETURN.)
2B	+	Overprint carriage control. (Sequence: print buffer contents, RETURN.) Allows double printing for emphasis.
24	$	Prompt carriage control. (Sequence: LINE feed, print buffer contents.)
All other values		Same as ASCII space character: single-space carriage control.

PRN

Indicates the print file format for variable with fixed control records, where the fixed control area contains the print file information, including carriage control. The first byte of the fixed control area constitutes a "prefix" area, and the second byte constitutes a "postfix" area, specifying carriage control to be performed before and after printing the record respectively. The encoding scheme of both bytes is as follows (even though they are interpreted separately):

Bit 7	Bits 0-6	Meaning
0	0	No carriage control is specified, that is, NULL.
0	1-7F	Bits 0 through 6 are a count of line feeds.

Bit 7	Bit 6	Bit 5	Bits 0-4	Meaning
1	0	0	0-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 (7-bit character set C0).
1	1	0	0-1F	Output the single ASCII control character specified by the configuration of bits 0 through 4 which are translated as ASCII characters 128 through 159 (8-bit character set C1).
1	1	1	0-1F	Reserved

Only the BLK attribute can be paired with another attribute. You cannot use CR, FTN, and PRN together in any combination. When BLK is used with another attribute, you can specify them in any order; the angle brackets are part of the required syntax when BLK is used with another attribute.

The following example indicates that records do not cross block boundaries.

```
FAB$    ORG=SEQ,RAT=BLK
```

Each option has its own symbolic bit offset and mask value. The bit offset for each is formed by prefixing FAB$V_ to the attribute value. For example:

```
BLK -- FAB$V_BLK
```

The mask value is formed by prefixing FAB$M_ to the attribute value. For example:

```
BLK -- FAB$M_BLK
```

The offset for this field is:

```
FAB$B_RAT
```

$FAB RFM

4.2.21 Record Format

The RFM parameter initializes the record format field to indicate the type of records in the file. When you create the file, you must set this field before you issue the $CREATE macro instruction. VAX-11 RMS returns the record format when you issue a $OPEN macro instruction.

FORMAT

```
        FIX
RFM=   {VAR }
        VFC
        UDF
```

FIX
Indicates fixed-length record format.

VFC
Indicates variable-length with fixed control record format. This format is not valid for indexed files.

VAR
Indicates variable-length record format. This is also the default value if the FAB is initialized through a $FAB macro instruction (assembly time default).

UDF
Indicates undefined record format. The undefined record format is valid for sequential file organization only, and can be processed only through the use of block I/O. This is the default value if the FAB is not initialized with a $FAB macro instruction.

For example, to indicate that records are fixed-length, the syntax is:

```
$FAB    RFM=FIX,FAC=GET
```

Each record format has its own symbolic value.

- FIX -- FAB$C_FIX
- VAR -- FAB$C_VAR
- VFC -- FAB$C_VFC
- UDF -- FAB$C_UDF

The offset for this field is:

FAB$B_RFM

$FAB RTV

4.2.22 Retrieval Window Size

The RTV parameter initializes the retrieval window size field. This field identifies the number of retrieval pointers you want VAX-11 RMS to maintain in memory for the file.

FORMAT

RTV=window-size

window-size
The number of retrieval pointers, in the range of 0 to 127, or 255. A value of 0 indicates that VAX-11 RMS is to use the system default number of retrieval pointers. A value of 255 means to map the entire file, if possible. Values between 128 and 254 inclusive are reserved for future use.

For example, to reserve ten retrieval pointers, the syntax is:

```
$FAB    FAC=GET,RTV=10,RFM=FIX
```

The offset for this field is:

FAB$B_RTV

$FAB SHR

4.2.23 File Sharing

The SHR parameter sets a value in the file-sharing field, indicating the operations other users can perform when they are sharing access to the file with you. File sharing pertains only to relative and indexed file operations, unless the UPI bit is set.

FORMAT

SHR=<PUT,GET,DEL,UPD,NIL,UPI,MSE>

PUT
Allows other users to write records to the file.

GET
Allows other users to read the file.

DEL
Allows other users to delete records from the file.

UPD
Allows other users to update records that currently exist in the file.

NIL
Prohibits any type of file sharing by other users. (If specified along with other operations, NIL takes precedence.)

UPI
Allows one or more writers for a sequential file or a shared file which is open for block I/O. The user assumes the responsibility for any required interlocking. This operation is set in combination with PUT, GET, UPD, and/or DEL.

MSE
Allows multistream access. You must specify MSE whenever you are going to issue $CONNECT macro instructions for multiple RABs for this FAB.

You can specify one or more file-sharing operations in any order.

For example, to allow read, write, and delete operations by other users, the syntax is:

```
$FAB    RTV=10,RFM=FIX,SHR=<DEL,PUT,GET>
```

Each file-sharing operation has its own symbolic bit offset and mask value. The bit offset is formed by prefixing FAB$V_ to the operation. For example:

PUT -- FAB$V_PUT

The mask value is formed by prefixing FAB$M_ to the operation. For example:

```
PUT -- FAB$M_PUT
```

The offset for the file-sharing field is:

```
FAB$B_SHR
```

NOTE

If you do not specify the SHR, VAX-11 RMS enters a value of 0 in the file-sharing field. Defaults apply as follows:

- If the file access field (FAC parameter) is set or defaulted to GET, the file-sharing field is interpreted as if set to GET.
- If the file access field is set or defaulted to either PUT, DEL, UPD, or TRN, the file-sharing field is defaulted to NIL.

$FAB XAB

4.2.24 Extended Attribute Block Pointer

For some operations, you must associate Extended Attribute Blocks (XABs) with a FAB to convey additional attributes about a file (see Chapter 6 for a description of an XAB). The XAB parameter sets the extended attribute block pointer field with the address of the first associated block (of a potential chained list of such blocks) for the file.

FORMAT

```
XAB=xab-address
```

xab-address
The symbolic address of the first XAB. A value of 0 (the default) indicates no XABs for the file.

For example, if the $XAB macro instruction has a label of HDRXAB, the syntax is:

```
$FAB    XAB=HDRXAB
```

The offset for this field is:

```
FAB$L_XAB
```

NOTES

1. If you specify an XAB for either a $OPEN or $DISPLAY macro instruction, VAX-11 RMS returns the attributes for the file to the XAB.

2. If you specify an XAB for a $CLOSE, $CREATE, or $EXTEND macro instruction, VAX-11 RMS uses the XAB as input to those functions.

4.3 NONINITIALIZABLE FAB FIELDS

The following list describes the FAB fields that you cannot initialize at assembly time. Either they are static, or VAX-11 RMS sets them for you.

BID

Block identifier field; identifies the block as a FAB to VAX-11 RMS. This field is set by the $FAB macro instruction to the symbolic value FAB$C_BID, and must not be altered.

BLN

Block length field; defines the length of the FAB to VAX-11 RMS. This field is set by the $FAB macro instruction to the symbolic value FAB$C_BLN, and must not be altered.

DEV

The device characteristics field is set by VAX-11 RMS when you issue a $OPEN or $CREATE macro instruction. This field allows VAX-11 RMS to communicate to your program the generic characteristics of the device containing the file. Although you cannot initialize this field at assembly time, you can interrogate the contents of the fields through the symbolic offsets. Table 4-2 lists the bits in the device characteristics field. Each bit described in this table has its own symbolic bit offset and mask value. These definitions can be made available to your program by issuing the $DEVDEF macro instruction. The bit offset is formed by prefixing the characteristic name with DEV$V_. For example:

```
REC -- DEV$V_REC
```

The mask value is formed by prefixing the characteristic name with DEV$M_. For example:

```
REC -- DEV$M_REC
```

The offset to this field is FAB$L_DEV.

IFI

Internal file identifier field; associates the FAB with a corresponding internal control structure. It is set by VAX-11 RMS following a successful $OPEN or $CREATE macro instruction. (When the UFO option is specified in the FOP parameter, this field is set to 0 following $OPEN or $CREATE.) A $CLOSE macro instruction clears this field. This field should not be altered.

The offset to this field is FAB$W_IFI.

SDC

Spooling device characteristics field; equivalent to the device characteristics field (DEV), except that spooling device characteristics refer to the intermediate device used for spooling. The spooling characteristic fields are the same as the device characteristics fields.

The offset to this field is FAB$L_SDC.

STS

Completion status code field; VAX-11 RMS sets this field with success or failure codes before control is returned to your program. Appendix A lists the symbolic completion codes that your program can use to test the contents of this field.

The offset to this field is FAB$L_STS.

STV

Status value field; communicates additional completion information to your program, based on the type of operation performed and the contents of the completion status code field. See Appendix A for the instances when VAX-11 RMS uses the status value field.

The offset to this field is FAB$L_STV.

Table 4-2
Device Characteristics

Bit Name	Description
ALL	Device is allocated
AVL	Device is available for use
CCL	Carriage control device
DIR	Directory structured device
DMT	Device is marked for dismount
ELG	Device is error log enabled
FOD	File-oriented device (disk and magnetic tape)
IDV	Device can provide input
MBX	Device is mail box
MNT	Device is currently mounted
ODV	Device can accept output
REC	Record-oriented device (terminal, line printer, etc.). If field is 0, device is assumed to be block-oriented (disk, magnetic tape). All record-oriented devices are considered sequential in nature.
RND	Device is random access in nature
RTM	Device is realtime in nature; not suitable for VAX-11 RMS usage
SDI	Single directory device (master file directory only)
SHR	Shareable device
SPL	Device is being spooled
SQD	Sequential block-oriented device (magnetic tape)
SWL	Device is currently software write-locked
TRM	Terminal device

CHAPTER 5

THE RECORD ACCESS BLOCK

This chapter describes the Record Access Block (RAB), the fields in the RAB, and the parameters of the $RAB macro instruction.

5.1 THE PURPOSE OF THE RECORD ACCESS BLOCK

The RAB is the second type of user control block that you allocate either at assembly time or run time to communicate with VAX-11 RMS. During program execution, you associate a RAB with a File Access Block (FAB) to establish a record stream using a $CONNECT macro instruction. Once you have established a record stream, you use the fields of the RAB to define to VAX-11 RMS the next record you want to access in the file.

Each RAB is linked to a FAB, and represents a record request stream on the file associated with the FAB. Once you establish this link, you can use the fields of the RAB to define for VAX-11 RMS the next logical record you want to access and various characteristics about that record.

You allocate a RAB with a $RAB macro instruction, and initialize the fields either at assembly time (through keyword parameters) or by direct manipulation at run time. You initialize the RAB at run time through either keyword parameters with the $RAB_STORE macro instruction (see Chapter 14) or the defined symbolic offsets.

Each field in the RAB has a 3-character mnemonic name. All access to these fields is through this name (by keyword or offset). However, some of the fields, as in the FAB, are static or output only; therefore, you need not initialize them. Table 5-1 summarizes the fields of the RAB, including the static and output-only fields.

Table 5-1
Record Access Block Fields

Field & Keyword Name	Field Size	Description	Offset
BID[2]	byte	Block identifier	RAB$B_BID
BKT	longword	Bucket code	RAB$L_BKT
BLN[2]	byte	Block length	RAB$B_BLN
CTX	longword	Context	RAB$L_CTX
FAB	longword	File access block address	RAB$L_FAB
ISI[1]	word	Internal stream identifier	RAB$W_ISI
KBF	longword	Key buffer address	RAB$L_KBF
KRF	byte	Key of reference	RAB$B_KRF
KSZ	byte	Key size	RAB$B_KSZ
MBC	byte	Multiblock count	RAB$B_MBC
MBF	byte	Multibuffer count	RAB$B_MBF
PBF	longword	Prompt buffer address	RAB$L_PBF
PSZ	byte	Prompt buffer size	RAB$B_PSZ
RAC	byte	Record access mode	RAB$B_RAC
RBF	longword	Record address	RAB$L_RBF
RFA[1]	3 words	Record's file address	RAB$W_RFA
RHB	longword	Record header buffer	RAB$L_RHB
ROP	longword	Record-processing options	RAB$L_ROP
RSZ	word	Record size	RAB$W_RSZ
STS[1]	longword	Completion status code	RAB$L_STS
STV[1]	longword	Status value	RAB$L_STV
TMO	byte	Timeout period	RAB$B_TMO
UBF	longword	User record area address	RAB$L_UBF
USZ	word	User record area size	RAB$W_USZ

[1] Indicates nonuser-initialized field.
[2] Indicates statically initialized field (by the $RAB macro instruction) to identify this control block as a RAB.

5.2 RAB ALLOCATION

The format of the $RAB macro instruction is shown below. Every parameter is optional, depending on the function to be performed and the combination of parameters in the macro instruction as a whole.

Format:

OPERATION	PARAMETERS
label: $RAB	BKT=number
	CTX=value
	FAB=fab-address
	KBF=buffer-address
	KRF=key-number
	KSZ=size
	MBC=blocks
	MBF=buffers
	PBF=prompt-address
	PSZ=prompt-size
	RAC={SEQ / KEY / RFA}
	RBF=buffer-address
	RHB=header-address
	ROP=<ASY BIO CCO CVT EOF KGE KGT LIM LOA LOC NLK NXR PMT PTA RAH RLK RNE RNF TMO TPT UIF ULK WBH>
	RSZ=record-size
	TMO=seconds
	UBF=buffer-address
	UBZ=buffer-size

The $RAB macro instruction allocates and initializes storage for a RAB. You cannot use this macro instruction within a sequence of executable instructions. Keyword parameters initialize the RAB fields at assembly time. In some cases, specific default values are assigned automatically when you omit a parameter. These specific defaults are noted in the text that explains each parameter. If there is no specific default, VAX-11 RMS uses a default value of 0.

Because VAX-11 RMS returns information in the fields of the RAB, you cannot allocate a RAB in a read-only program section.

label: $RAB

5.2.1 Label

The label for the $RAB macro instruction lets you name a RAB, and thereby provides symbolic access to a particular RAB within your program. The label is optional but, when used, must precede the symbol $RAB and be separated from $RAB by a colon (:). For example:

```
INPUT:  $RAB
```

$RAB BKT

5.2.2 Bucket Code

The BKT parameter initializes the bucket code field of the RAB at assembly time. This field is used as follows:

1. With records in a relative file
2. When performing block I/O

For relative files, the relative record number of the record acted upon (or which produced an error) is returned to the bucket code field only after the completion of a sequential operation. That is, VAX-11 RMS returns the relative record number when you set the record access mode for sequential access (RAC=SEQ) on the execution of a $GET, $PUT, or $FIND macro instruction.

When performing block I/O on disk devices, you must store (in the bucket code field) the virtual block number (VBN) of the first block you want to read or write. For all other devices, this field is not used. If you specify a VBN of 0, VAX-11 RMS will begin the block transfer at the block pointed to by the Next Block Pointer (NBP). The NBP is an internal pointer maintained by VAX-11 RMS, and is described in Chapter 12.

FORMAT

```
BKT=number
```

number
A relative record number or a numeric value representing the virtual block number to be accessed.

For example, to indicate access to the tenth block of the file when the program performs its first block I/O operation, the syntax is:

```
$RAB BKT=10,CTX=RECOK
```

The offset for this field is:

```
RAB$L_BKT
```

$RAB CTX

5.2.3 Context

The CTX parameter initializes the context field, which is a field devoted exclusively to your use. VAX-11 RMS makes no use of the contents of this field; therefore, you can set any value you want in this field. For example, you could use this field to communicate a completion routine to your program.

FORMAT

```
CTX=value
```

value
Any user-selected value, up to one longword in length.

For example, to initialize the context field to the value of the symbol RECOK, the syntax is:

```
$RAB CTX=RECOK,BKT=10
```

The offset for this field is:

```
RAB$L_CTX
```

$RAB FAB

5.2.4 File Access Block Address

The FAB parameter initializes the file access block address field of the RAB. When you issue a $CONNECT macro instruction, you must set this field to indicate the address of the FAB associated with the open file.

FORMAT

```
FAB=fab-address
```

fab-address
The symbolic address of the FAB for the file in question.

For example, if you define the label of the FAB for the file as MASTER, the syntax is:

```
$RAB    FAB=MASTER,CTX=RECOK
```

The offset for this field is:

RAB$L_FAB

$RAB KBF

5.2.5 Key Buffer Address

The KBF parameter initializes the key buffer address field. You use this field when the record access mode (RAC) field specifies random access by key value (see Section 5.2.12), and you set it to the address of the buffer that contains the key of the desired record. For a relative file (or for a sequential disk file with fixed-length records), the key is the relative record number. For an indexed file, the key is the key value within the record for the key of reference (KRF) (see Section 5.2.6).

FORMAT

KBF=buffer-address

buffer-address

The symbolic address of the buffer containing the key.

For example, if the label of the buffer that provides the relative record number is RELKEY, you initialize the KBF parameter as follows:

```
$RAB    KBF=RELKEY,CTX=RECOK
```

The offset for this field is:

RAB$L_KBF

NOTE

Before issuing a $GET or $FIND macro instruction in random mode to an indexed file, you place in KBF the address of a location containing a key value. The size of this key value must be specified in the KSZ field. During execution of the $GET or $FIND operation, VAX-11 RMS uses the key value described by the KBF and KSZ fields to search an index (which you specify through the contents of the KRF field of the RAB) and locate the desired record in the file. The type of match (that is, exact, generic, approximate, or approximate and generic) that VAX-11 RMS attempts between the key value you specify and key values in records of the file is determined by the KSZ field and the ROP field.

NOTE

The key buffer address field uses the same location in the RAB as the prompt buffer address field. There is no conflict between these two fields, however, because the prompt buffer address field is used only for terminals, while the key buffer address field is used only for randomly accessed disk files.

$RAB KRF

5.2.6 Key of Reference

The KRF parameter initializes the key of reference field, which specifies the key or index (primary, first alternate, and so on) to which the operation applies. The KRF field is applicable to indexed files only.

When your program issues a $GET or $FIND macro instruction in random access mode, the key of reference specifies the index to search for a match on the key value which is described by the key buffer address (KBF) and key size (KSZ) field. When your program issues a $CONNECT or $REWIND macro instruction, the key of reference identifies the index in the file of the next record in the stream. The next record is important in sequential retrieval of records; the Next Record is described in Section 10.2.2.

The format of this parameter is:

```
KRF=key-number
```

where

key-number

is the numeric value representing a key in the records of a file. The value 0 indicates the primary key. The values 1 through 254 indicate alternate keys. The assembly-time default value is 0 (primary key).

As an example, if the first alternate key is the index to search for a match (approximate, generic, or generic-approximate) on the key value described by the KBF and KSZ fields, the KRF parameter would be initialized as follows:

```
$RAB KRF=1,KBF=KEYBUF,KSX=KEYSIZE
```

The offset for this field is:

```
RAB$B_KRF
```

$RAB KSE

5.2.7 Key Size

The KSZ parameter initializes the key size field, which contains the size, in bytes, of the key pointed to by the key buffer address field.

5.2.7.1 **Relative Files** - The size of the relative record number of a record in a relative file is a longword, positive, integer value; therefore, the key size is 4.

5.2.7.2 **Indexed Files.** - The size of key values in bytes of an indexed file can be from 1 to 255 bytes.

When you access an indexed file in random mode, the contents of the KSZ and the contents of the ROP field determine the type of match to make on the key value specified in the key buffer address. For string key data type, the contents of the KSZ field can be less than the defined key size. For the other (numeric) key types, the contents of KSZ must be the defined length or 0, which defaults to the defined length. The following chart shows the relationships of the KSZ/ROP field contents and the type of match. Since KSZ for numeric key types must be the defined length, only exact and approximate matches are made on these types.

KGE or KGT Specified in ROP	Specified KSZ/Defined Key Size Relationship	Type of Match
NO	EQUAL	EXACT
NO	LESS THAN	GENERIC
YES	EQUAL	APPROXIMATE
YES	LESS THAN	GENERIC-APPROXIMATE

5.2.7.3 **Relative and Indexed Files**

FORMAT

```
KSZ=size
```

size
The numeric value of the size of the record key. For relative record numbers, the default value of 0 causes a key size of 4 to be used. For string keys a value from 1 to the size of the key field and for the numeric key data types a value of 0 cause the defined size to be assumed; a nonzero value is checked against the defined size and an error (RMS$_RSZ) is returned if they are not equal.

For example, for relative files the KSZ parameter must be coded as:

```
$RAB    KSZ=4,KBF=RELKEY
```

The offset for this field is:

```
RAB$B_KSZ
```

NOTE

The key size field uses the same location in the RAB as the prompt buffer size field. There is no conflict between these fields, however, because one field (PBF) is used only for terminal I/O, while the other field (KSZ) is used only for randomly accessed disk files.

$RAB MBC

5.2.8 Multiblock Count

The MBC parameter initializes the multiblock count field, and applies only when the RAB accesses a sequential disk file.

VAX-11 RMS examines the multiblock count field during the execution of a $CONNECT macro instruction. The value in this field is used as the number of blocks to be transferred as a single entity during an I/O operation for the record stream represented by this RAB. A buffer is allocated that can contain the specified number of blocks. In addition, more than one buffer (of this size) can be allocated for the record stream, as determined by the value of the multibuffer count field (see Section 5.2.9).

The use of the multiblock count field optimizes data throughput especially for sequential operations and in no way affects the structure of the file. It reduces the number of disk accesses you would normally require for your record operations and can thereby greatly increase execution speed. On the other hand, the extra buffering increases memory requirements.

FORMAT

```
MBC=blocks
```

blocks
The number of blocks, in the range of 1 to 127, to be allocated to each I/O buffer. If you omit this parameter, the multiblock count field is initialized to 0 at assembly time, which specifies that the process default for the multiblock count is to be used. If the process default is also 0, VAX-11 RMS uses the system default. If the system default is also 0, then the default size for each I/O buffer is one block.

For example, to allocate 16 blocks to each I/O buffer, the syntax is:

```
$RAB   MBC=16,CTX=RECOK
```

The offset for this field is:

```
RAB$B_MBC
```

NOTE

The MBC parameter is not used with block I/O.

$RAB MBF

5.2.9 Multibuffer Count

The MBF parameter sets the multibuffer count field to indicate the number of I/O buffers you want VAX-11 RMS to allocate when you issue a $CONNECT macro instruction for this RAB.

VAX-11 RMS requires that at least one buffer be allocated for each file, unless the file is to be processed with block I/O operations only. Multiple buffers can be used efficiently to overlap I/O time with program compute time, particularly in read-ahead or write-behind processing (see Section 5.2.15).

FORMAT

```
MBF=buffers
```

buffers
A numeric value, in the range of -128 to +127, which represents the number of buffers to be allocated. If the number is positive, the buffers are locked in the working set of the process. If negative, the absolute value gives the number of buffers to allocate, but they are not locked in the working set of the process.

If the MBF parameter is omitted, the field is initialized to 0 at assembly time. A 0 value indicates the use of the process default for the particular file organization and device type.

If the process default is also 0, the system default for the particular file organization and device type applies.

If the system default is likewise 0, one buffer is allocated, unless read-ahead or write-behind is specified, in which case two buffers are allocated; in either case, the buffer or buffers are not locked in the working set of the process.

If either the process or system default is nonzero, the allocated buffers are either locked or not locked in the process' working set based on whether the default value is positive or negative, as described above.

For example, to allocate four buffers, the syntax is:

```
$RAB MBF=4,CTX=RECOK
```

The offset for this field is:

```
RAB$B_MBF
```

NOTE

The MBF parameter is not used with block I/O. No buffers are allocated either if block I/O access is specified in the file access (FAC) field of the FAB on open or create, or if mixed block I/O and record I/O is specified in the file access field, but the block I/O record option is set in the record processing options (ROP) field for the connect service.

$RAB PBF

5.2.10 Prompt Buffer Address

The PBF parameter initializes the prompt buffer address field. This field points to a character string to be used as a prompt for terminal input. If you select the PMT option of the ROP parameter (see Section 5.2.14) when you issue a $GET macro instruction, this character string is output to the terminal before the read operation is performed.

To perform any carriage control on the terminal, you must insert the appropriate carriage control characters into this character string.

FORMAT

```
PBF=prompt-address
```

prompt-address
The symbolic address of the buffer containing the prompt character string.

For example, if the buffer containing the prompt character string has a symbolic label of PROMPT, the PBF parameter is:

```
$RAB   PBF=PROMPT,ROP=PMT,PSZ=2
```

The offset for this field is:

```
RAB$L_PBF
```

NOTE

The prompt buffer address field uses the same location in the RAB as the key buffer address field. There is no conflict between these two fields, however, because the prompt buffer address field is used only for terminals, while the key buffer address field is used only for randomly accessed disk files.

$RAB PSZ

5.2.11 Prompt Buffer Size

The PSZ parameter initializes the prompt buffer size field. This field contains the size, in bytes, of the character string for terminal I/O prompting.

FORMAT

```
PSZ=prompt-size
```

prompt-size
The size, in bytes, of the prompt character string, in the range of 0 to 255.

If, for example, the character string is only two bytes long, the syntax is:

```
$RAB    PBF=PROMPT,PSZ=2,ROP=PMT
```

The offset for this field is:

```
RAB$B_PSZ
```

NOTE

The prompt buffer size field uses the same location in the RAB as the key size field. There is no conflict between these fields, however, because the prompt buffer size (PSZ) field is used only for terminal I/O, while the key size (KSZ) is used only for ramdomly accessed disk files.

$RAB RAC

5.2.12 Record Access Mode

The RAC parameter initializes the record access mode field to indicate the method of retrieving or storing records in the file.

FORMAT

```
RAC=  {SEQ}
      {KEY}
      {RFA}
```

SEQ
Indicates sequential record access mode (the default); can be specified with any type of file organization.

KEY
Indicates random access by key; used with relative files (and with sequential files on disk with fixed-length records) to indicate access by relative record number; used with indexed files to indicate access by key value.

RFA
Indicates random access by record's file address; used for disk files only.

For example, to set the record access mode field to indicate the sequential record access mode, the syntax is:

```
$RAB    RAC=SEQ,CTX=RECOK
```

The offset for this field is:

```
RAB$B_RAC
```

Each record access mode has a symbolic constant that can be used to set the record access mode field at run time, as follows:

- SEQ - RAB$C_SEQ
- KEY - RAB$C_KEY
- RFA - RAB$C_RFA

NOTES

1. You can specify the record access mode on a per-operation basis.
2. For block I/O, you do not use the record access mode field.

$RAB RBF

5.2.13 Record Address

The RBF parameter initializes the record address field. When you issue a $PUT or $WRITE macro instruction, this field must specify the address of the record to be written to the file.

When you issue a $GET or $READ macro instruction, VAX-11 RMS sets this field to the address of the record just read from the file; you need not initialize this field.

FORMAT

RBF=buffer-address

buffer-address
The symbolic address of the buffer in your program that contains the record to be written.

For example, to initialize the record address field with the address of a buffer having the label of RECBUF, the syntax is:

```
$RAB   RBF=RECBUF,CTX=RECOK
```

The offset for this parameter is:

RAB$L_RBF

$RAB RHB

5.2.14 Record Header Buffer

The RHB parameter initializes the fixed-length record header field. This buffer is used only when processing records of variable with fixed-length control. For a $GET macro instruction, VAX-11 RMS strips the fixed control area portion of the record and places it in the buffer whose address is specified in this field. For the $PUT or $UPDATE macro instructions, VAX-11 RMS writes the contents of the specified buffer to the file as the fixed control area portion of the record.

The size of this fixed control area is defined in the FAB, through the FSZ parameter. You must ensure that the size of the buffer described in the record header buffer field is equal to the value specified by the FSZ parameter.

FORMAT

RHB=header-address

header-address
The symbolic address of the record header buffer. If omitted, an address of 0 is assumed, which indicates the absence of a buffer; the fixed control area is discarded for a $GET macro instruction, zeroed for a $PUT macro instruction, and left unchanged for a $UPDATE macro instruction.

For example, if the buffer is defined with a label of FCABUF, the syntax is:

```
$RAB    RHB=FCABUF,CTX=RECOK
```

The offset for this field is:

RAB$L_RHB

$RAB ROP

5.2.15 Record-Processing Options

The ROP parameter sets indicators in the record-processing options field that let you request optional functions during execution of a record operation. VAX-11 RMS operations never modify the contents of this field.

FORMAT

```
ROP= <ASY,BIO,CCO,CVT,EOF,LOC,KGE,KGT,LOA,LIM,NLK,NXR,PMT,PTA,
      RAH,RLK,RNE,RNF,TMO,TPT,UIF,ULK,WBH>
```

ASY
Asynchronous; indicates that any required I/O operation is to be performed asynchronously. When you specify ASY, VAX-11 RMS will return control to your program as soon as an I/O operation is initiated, even though that operation may not yet be completed.

BIO
Block I/O; specifies that only block I/O operations will be performed. Used as a flag to the $CONNECT macro instruction to indicate that no VAX-11 RMS I/O buffers are to be allocated, even though the file access field of the FAB (Section 4.2.10) for the $OPEN or $CREATE macro instruction specifies mixed block and record operations (BRO).

CCO
Cancel control O; guarantees that terminal output will not be discarded if the operator enters CTRL/O.

CVT
Convert; changes characters to uppercase on a read from a terminal.

EOF

End-of-file; indicates that VAX-11 RMS is to position to the end of the file when a $CONNECT macro instruction executes. This applies only to sequential disk files.

LOC

Locate mode; indicates that record operations involving the $GET macro instruction will use locate mode (see Section 10.1.2).

KGE

Key is greater than or equal to; requests VAX-11 RMS to access the first record in an indexed file, which contains a value for the specified key of reference (KRF) (see Section 5.2.6) that is greater than or equal to the value described by the key buffer address (KBF) and key size (KSZ) fields (see Sections 5.2.5 and 5.2.7.2, respectively). This applies to indexed and relative files only. If neither KGE nor KGT is specified, a key equal match is made.

KGT

Key is greater than; requests VAX-11 RMS to access the first record in an indexed file, which contains a value for the specified key of reference (KRF) (see Section 5.2.) that is greater than the value described by the key buffer address (KBF) and key size (KSZ) fields (see Sections 5.2.5 and 5.2.7.2, respectively). This applies to indexed and relative files only. If neither KGE nor KGT is specified, a key equal match is made.

LOA

Load; specifies that VAX-11 RMS is to load buckets according to the fill size established at file creation time. The bucket fill size is established at file creation time by the data bucket fill size (DFL) and index bucket fill size (IFL) fields of the key extended attribute blocks (XABs). The XABs are described in Chapter 6. Assembly-time default (LOA not specified) causes VAX-11 RMS to ignore the established bucket fill size (that is, buckets will be completely filled). This applies to indexed files only.

LIM

Limit; key value described by the key buffer address (KBF) and key size (KSZ) fields (see Sections 5.2.5 and 5.2.7.2, respectively) is to be compared to the value in the record accessed in sequential mode. If the record's key value is greater than the limit key value, an RMS$_OK-LIM status code is returned.

NLK

No lock; specifies that the record accessed through a $GET or $FIND macro instruction is not to be locked. This does not apply to sequential files. The NLK option takes precedence over the ULK option (below).

NXR

Nonexistent record processing; specifies that if the record randomly accessed through a $GET or $FIND macro instruction does not exist (was never inserted into the file or was deleted), the service is to be performed anyway, locking the record cell if required. For the $GET macro instruction, the previous contents of a deleted record are returned; 0s are returned if the record never existed. The processing of a deleted record returns a completion status code of RMS$_OK_DEL, and the processing of a record that never existed returns RMS$_OK_RNF. This option applies only to relative files.

PMT

Prompt; indicates that the contents of the prompt buffer are to be used as a prompt on a read from a terminal (see Section 5.2.10).

PTA

Purge type-ahead; eliminates any information that may be in the type-ahead buffer on a read from a terminal.

RAH

Read-ahead; used with multibuffers (see Section 5.2.8) to indicate read-ahead operations. When a buffer is filled, the next record will be read into the next buffer. This permits an overlapping of input and computing. Read-ahead is ignored for unit record device I/O.

RLK

Read of locked record allowed; specifies that a user who locks a record is allowing the locked record to be read by other accessors. This option does not apply to sequential files.

RNE

Read no echo; indicates that input data is not echoed (displayed) on the terminal as it is entered on the keyboard.

RNF

Read no filter; indicates that CTRL/U, CTRL/R, and DELETE are not to be considered control commands on terminal input, but are to be passed to the user program.

TMO

Time-out; indicates that the content of the time-out period field of the RAB is to be used to determine the number of seconds that a VAX-11 RMS operation has to complete its operation. If the time-out period expires, VAX-11 RMS returns an error status (see Section 5.2.17).

TPT

Truncate put; specifies that a put service with a record access mode of sequential can occur at any point in the file, truncating the file at that point. On a write service, this causes the end of file mark to immediately follow the last byte written. This applies only to sequential files.

UIF

Update if; indicates that if a $PUT macro instruction is issued for a record that already exists in the file, the operation is converted to an update. This option applies to random access by relative record number to sequential, relative, and indexed files.

ULK

Manual unlocking; specifies that VAX-11 RMS cannot automatically unlock records. Instead, once locked (through a $GET, $FIND, or $PUT macro instruction), a record must be specifically unlocked by a $FREE or $RELEASE macro instruction. This option does not apply to sequential files. The NLK option (above) takes precedence over the ULK option.

WBH

Write-behind; used with multibuffers (see Section 5.2.8). When a buffer is filled, the next record written will be placed in the next buffer while the previous buffer is output. This allows for an overlapping of computing and output. Write-behind is ignored for unit record devices.

You can use one or more options with the ROP parameter. For example, to indicate that a terminal read should convert from lower- to uppercase, and use locate mode, the prompt buffer, and the specified time-out period, the ROP parameter would be:

```
$RAB   ROP=<CVT,LOC,PMT,TMO>,PBF=PROMPT,PSZ=PROMPT_SIZE,TMO=30
```

Each option has its own symbolic bit offset and mask value. The bit offset for each is formed by prefixing RAB$V_ to the option value. For example:

```
ASY --  RAB$V_ASY
```

The mask value is formed by prefixing RAB$M_ to the option value. For example:

```
WBH --  RAB$M_WBH
```

The offset for the ROP field is:

```
RAB$L_ROP
```

$RAB RSZ

5.2.16 **Record Size**

The RSZ parameter sets the record size field. This field controls the size of a record or the number of bytes that, respectively, a $PUT or $WRITE (block I/O) macro instruction can write.

On input from a file, VAX-11 RMS sets this field to indicate the length, in bytes, of the record that a $GET macro instruction transfers or that a $READ macro instruction reads.

FORMAT

```
RSZ=record-size
```

record-size

The size, in bytes, of the record. For operations with a $PUT macro instruction, the size must be in the range of 0 to 32767. For operations with a $WRITE macro instruction, the range is 1 to 65535.

For example, to indicate a record size of 150 bytes, the syntax is:

```
$RAB   RBF=RECBUF,RSZ=150
```

The offset for this field is:

```
RAB$W_RSZ
```

NOTES

1. After a get operation, VAX-11 RMS places the size of the record retrieved into the record size field. On a read operation, VAX-11 RMS sets the record size field to the number of bytes actually transferred.

2. For variable with fixed control records, VAX-11 RMS does not include the size of the fixed control area in the record size field.

$RAB TMO

5.2.17 Time-Out Period

The TMO parameter initializes the time-out period field, which indicates the maximum number of seconds that VAX-11 RMS can use to complete an operation. If the time-out period expires before the operation completes, VAX-11 RMS returns an error status code.

To use this field, you must also specify the TMO option when you set the record-processing option field (ROP parameter).

FORMAT

```
TMO=seconds
```

seconds
The maximum number of seconds, in the range of 0 to 255, that a VAX-11 RMS operation can use. If you specify 0, VAX-11 RMS must complete the operation in less than one second.

For example, to indicate that the operation for this RAB must complete in 20 seconds or less, the syntax is:

```
$RAB    TMO=20,ROP=TMO
```

Note that the TMO option is specified on the ROP parameter.

The offset for this parameter is:

```
RAB$B_TMO
```

NOTE

The time-out period currently applies only when you use a $GET or $READ macro instruction to input from a terminal or a mailbox.

$RAB UBF

5.2.18 User Record Area Address

The UBF parameter initializes the user record area address field, which indicates the location of a record or block buffer.

When you issue a $GET macro instruction, this field must contain the buffer address regardless of the record transfer mode (locate or move). This also applies when you issue a $READ macro instruction for block I/O. However, operations with a $PUT macro instruction never need a user buffer.

FORMAT

```
UBF=buffer-address
```

buffer-address
The symbolic address of a work area (buffer) within your program. (The size of this buffer must be defined in the user record area size field; the USZ parameter.)

For example, if the buffer area has a label of USRBUF, the syntax is:

```
$RAB    UBF=USRBUF,USZ=2048
```

The offset for this field is:

```
RAB$L_UBF
```

$RAB USZ

5.2.19 User Record Area Size

The USZ parameter initializes the user record area size field, which indicates the length, in bytes, of the user record or block buffer. This buffer area should be large enough to contain the largest record in the file. If the buffer is not large enough on an operation with a $GET macro instruction, VAX-11 RMS will move as much of the record as possible into the buffer, and return a warning status code.

The value in this field specifies the transfer length, in bytes, for block I/O operations with a $READ macro instruction.

FORMAT

```
USZ=buffer-size
```

buffer-size

A numeric value representing the size, in bytes, of the buffer. This value must be in the range of 1 to 65535.

For example, for a user buffer area with a label of USRBUF and a size of 2048 bytes, the syntax is:

```
$RAB    UBF=USRBUF,USZ=2048
```

The offset for this field is:

```
RAB$W_USZ
```

5.3 NONINITIALIZABLE RAB FIELDS

The following list describes the RAB fields that you cannot initialize at assembly time. Either they are static, or VAX-11 RMS sets them for you.

BID

Block identifier field; identifies the block as a RAB. The $RAB macro instruction sets this field to the symbolic value RAB$C_BID; this field must not be altered.

BLN

Block length field; defines the length, in bytes, of the RAB to VAX-11 RMS. The $RAB macro instruction sets this field to the symbolic value RAB$C_BLN; this field must not be altered.

ISI

Internal stream identifier field; associates the RAB with a corresponding FAB. VAX-11 RMS sets this field after the execution of a $CONNECT macro instruction. A $DISCONNECT macro instruction clears this field. This field should not be altered.

The offset of this field is RAB$W_ISI.

STS

Completion status code field; VAX-11 RMS sets this field with the success or failure status codes for a record operation before returning control to your program. (In the case of an asynchronous operation that has been initiated but not yet completed, this field is 0.) Appendix A lists the symbolic completion status codes that your program can use to test the contents of this field.

The offset to this field is RAB$L_STS.

STV

Status value field; communicates additional completion information to your program, based on the type of operation and the contents of the completion status code field. See Appendix A for the instances when VAX-11 RMS uses the status value field.

The offset for this field is RAB$L_STV.

5.3.1 The Record's File Address

After the successful execution of a $GET, $PUT, or $FIND macro instruction, VAX-11 RMS sets the record's file address (RFA) field to the address of the record acted on by the operation. This address is meaningful only for disk files; it provides an unambiguous means of randomly locating this same record at some later time.

You can store the contents of the record's file address field for future use. When you want to retrieve the record again, merely restore the saved contents of the field, set the record access mode to random by RFA, and issue a $GET or $FIND macro instruction.

The offset for this field is:

RAB$W_RFA

NOTES

1. This field is six bytes long.

2. RFA values remain valid for a record in a sequential file as long as the record is within the space defined by the logical file, that is, until the file is truncated to a point before the record.

3. RFA values remain valid for a record in a relative or indexed file for the life of the file, that is, until the file is deleted.

CHAPTER 6

THE EXTENDED ATTRIBUTE BLOCKS

This chapter describes the various Extended Attribute Blocks (XABs), their fields, and the macro instructions and parameters you use to initialize the fields at assembly time.

6.1 THE PURPOSE OF EXTENDED ATTRIBUTE BLOCKS

The XABs are optional additional control blocks, which you can use to communicate to VAX-11 RMS any file attributes beyond those expressed in the FAB. You use these control blocks only when you want to specify exactly, or retrieve information on, the attributes handled by a particular XAB.

You can use XABs to set file attributes by specifying them as inputs to the $CREATE, $CLOSE, or $EXTEND macro instructions. Retrieve the attributes by specifying the XAB as input to the $OPEN or $DISPLAY macro instructions. If the CIF bit is set in the file processing options field of the FAB on a create service, VAX-11 RMS uses the XAB fields as input or output depending on whether the file is opened or created, respectively.

When you need more than one XAB, you can chain them together. Each XAB has a next XAB address field, which can be set at assembly time through the NXT parameter, or at run time. You can set this field at run time by storing the appropriate address into the next XAB address field. The extended attribute block pointer field of the FAB (see Section 4.2.24) points to the first block in the chain. Section 6.2 below describes chaining in detail.

Currently, VAX-11 RMS supports seven types of XABs, each with its own macro instructions for allocation and initialization. These blocks and their macro instructions are as follows:

- Date and time -- $XABDAT
- File protection -- $XABPRO
- Allocation control -- $XABALL
- Key definition -- $XABKEY
- Summary -- $XABSUM
- File header characteristics -- $XABFHC
- Revision date and time -- $XABRDT

The last three characters of each macro instruction (DAT, PRO, ALL, KEY, SUM, FHC, and RDT) define the specific type of the XAB to VAX-11 RMS, and cause the value for this specific type to be stored in the type code field of each block. The symbolic offset for this field is:

XAB$B_COD

In addition, a length value is stored in the block length field of each block. The symbolic offset for this field is:

XAB$B_BLN

Because each block has its own initialization macro instruction, each is discussed separately in Sections 6.3 through 6.9 below.

Table 6-1 indicates which XAB types are processed by which service.

Table 6-1
XAB Types Processed by Service

Service / Type	Close	Create	Display	Extend	Open
Allocation Control		Input Output	Output	Input Output	Output
Key Definition		Input Output	Output		Output
Summary			Output		Output
Date and Time		Input Output[1]	Output		Output
File Header Characteristics		Input Output[1]	Output		Output
File Protection	Input[2]	Input Output[1]	Output		Output
Revision Date and Time	Input[2]	Input Output[1]	Output		Output

[1] Fields of the XAB are output only if the create if (CIF) bit is set and the file is opened, not created.
[2] Processed only if file is write-accessed.

At assembly time, you can initialize the fields of the particular XAB through keyword parameters. At run time, you can use the keyword parameters with the appropriate $XABxxx_STORE macro (see Chapter 14) or the defined symbolic offsets.

6.2 CHAINING EXTENDED ATTRIBUTE BLOCKS

Every XAB has a next XAB address field, regardless of the type of information that the block contains, such as date/time or file protection information. When you need one or more XAB for a particular operation, place the symbolic address of the first XAB of the chain into the extended attribute block pointer field of the FAB. Then, place the address of the second XAB in the chain, if any, in the next XAB address field of the first XAB (NXT parameter). You continue this process until you have chained all the XABs you need. You must set the next XAB address field of the last XAB to 0 to indicate the end of the chain. You can either set this field explicitly, or allow the system to default to this 0 value.

Within the XAB chain, the different types of XABs need not be in any specific order. For example, at assembly time, you could allocate a date and time XAB, a file protection XAB, and an allocation control XAB. You can chain these different types of XABs in any order, by appropriately setting the contents of the next XAB address field in each block. For indexed files, however, VAX-11 RMS permits multiple instances of the same type of XAB in an allocation control or key definition XAB chain. For $CREATE macro instructions, these must appear in a specific order and allocation control XABs must be linked together in ascending order. based on the contents of the area identification number (AID) field (see Section 6.5.2). Also, for the $CREATE macro instruction, key definition XABs must be linked together in ascending order based on the contents of the key of reference (REF) field (see Section 6.6.12). In these cases, there cannot be any intervening XABs of another type in the sub-chain of XABs of one type. Further, the operation for which the allocation control or key definition XABs is present determines whether the ascending order must be dense. For create operations, allocation control and key definition XABs, if present, must appear in densely ascending order by area identification (AID) number or key of reference (REF) value, respectively. For extend operations, allocation control and key definition XABs, if present, must be in ascending order but need not be dense. For open and display operation, RMS-32 verifies that the number of XABs specified does not exceed the number specified for the file. If the number of XABs specified does exceed the number defined for the file, a RMS$_AID error is returned for allocation XABs and a RMS$_REF error is returned for key definition XABs.

The NXT parameter appears in the format of each of the XAB macro instructions. This parameter is explained below, rather than repeated throughout Sections 6.3 through 6.10.

FORMAT

```
NXT=address
```

address
The symbolic address of the next XAB in the chain. A value of 0 (the default) indicates the last (or only) XAB in the chain.

The offset for this field is:

```
XAB$L_NXT
```

$XABDAT

6.3 DATE AND TIME XAB

The $XABDAT macro instruction allocates and initializes an XAB for date and time. This block allows for extended control of the date and time of the file's creation, revision (update), and expiration. Table 6-2 summarizes the fields comprising the date and time XAB.

Table 6-2
Date and Time Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
BLN[2]	byte	Block length	XAB$B_BLN
CDT[1]	quadword	Creation date and time	XAB$Q_CDT
COD[2]	byte	Type code	XAB$B_COD
EDT	quadword	Expiration date and time	XAB$Q_EDT
NXT	longword	Next XAB address	XAB$L_NXT
RDT[1]	quadword	Revision date and time	XAB$Q_RDT
RVN[1]	word	Revision number	XAB$W_RVN

[1] Indicates no assembly time initialization.
[2] Indicates that this field is set automatically by the type of macro instruction.

FORMAT

OPERATION	PARAMETERS
label: $XABDAT	EDT=date-time NXT=address

Both parameters and the label are optional, depending on how you use the particular macro instruction.

label: $XABDAT

6.3.1 Label

The optional label for the $XABDAT macro instruction lets you assign a name to a date and time XAB, thereby allowing symbolic access to the XAB.

For example, suppose a $XABDAT macro instruction represents the first or only XAB in the chain, and has the following label:

```
HDRXAB:   $XABDAT
```

Then, your $FAB macro instruction would have an XAB parameter as follows:

```
$FAB   XAB=HDRXAB
```

Note that the label must be separated from the $XABDAT macro name by a colon (:).

$XABDAT EDT

6.3.2 Expiration Date and Time

The EDT parameter sets the expiration date and time field. This field indicates the date and time after which a magnetic tape file can be deleted.

FORMAT

```
EDT=date-time
```

date-time
A 64-bit binary value in either absolute (positive) or delta (negative) format. (See the VAX/VMS System Services Reference Manual.)

The offset for this field is:

```
XAB$Q_EDT
```

$XABDAT NXT

6.3.3 Next XAB Address

The NXT parameter sets the next XAB address field. See Section 6.2 for a complete description of this parameter.

6.3.4 Creation/Revision Date and Time, and Revision Number

VAX-11 RMS sets certain values for date and time, and returns them in date and time XAB fields for your inspection. You can override these system-supplied values through the use of a date and time XAB as input to a $CREATE macro instruction. However, the $XABDAT macro instruction does not contain parameters for the assembly-time initialization of these fields. As outlined in Table 6-2, these fields are:

- Creation date and time (CDT) -- this is a 64-bit binary value expressing the date and time at which the file was created. The symbolic offset for this field is:

  XAB$Q_CDT

- Revision date and time (RDT) -- this is a 64-bit binary value expressing the date and time at which the file was last updated. The symbolic offset for this field is:

  XAB$Q_RDT

- Revision Number (RVN) -- this field provides the number of times this file was opened for write operations. The symbolic offset for this field is:

  XAB$W_RVN

6.3.5 Date/Time Type Code and Block Length

The $XABDAT macro instruction sets the values for both the type code and block length fields. VAX-11 RMS interprets the last three characters (DAT) of the macro instruction to determine the value. For the date and time XAB, the following symbolic values are stored:

- Type code field -- XAB$C_DAT
- Block length field -- XAB$C_DATLEN

$XABPRO

6.4 FILE PROTECTION XAB

The $XABPRO macro instruction allocates and initializes an XAB that you can use to explicitly specify file ownership and file protection. Table 6-3 summarizes the fields comprising the file protection XAB.

Table 6-3
File Protection Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
BLN[1]	byte	Block length	XAB$B_BLN
COD[1]	byte	Type code	XAB$B_COD
GRP	word	Group number of file owner	XAB$W_GRP
MBM	word	Member number of file owner	XAB$W_MBM
NXT	longword	Next XAB address	XAB$L_NXT
PRO	word	File protection; contains four separate fields denoting the protection for system, owner, group, and world	XAB$W_PRO

[1] Indicates that this field is set automatically by the type of macro instruction.

FORMAT

OPERATION	PARAMETERS
label: $XABPRO	PRO=<system,owner,group,world>
	UIC=<group,member>
	NXT=address

label: $XABPRO

6.4.1 Label

VAX-11 RMS uses the label for the $XABPRO macro instruction in the same way that it uses the label for the $XABDAT macro instruction; see Section 6.3.1.

$XABPRO PRO

6.4.2 File Protection

The PRO parameter initializes the four subfields of the file protection field and it specifies the file access privileges of the four classes of users. The subfields for the four classes are:

1. System -- this subfield specifies access rights for users executing under a system UIC, that is, having an octal group number less than 10.

2. Owner -- this subfield specifies access rights for the owner of the file. A user is considered the owner of the file only if both the group and member number fields (see Section 6.4.3) of the accessing process match the group and member number fields of the file owner's UIC stored with the file.

3. Group -- this subfield specifies the access rights for users whose group number matches the group number field of the file owner.

4. World -- this subfield specifies the access rights for any user. It is normally allowed for users not within the system, owner, or group classifications (items 1, 2, and 3, above).

A user is granted the maximum number of types of access rights for each of the classes to which he belongs.

The entire file protection field is one word, and each classification subfield occupies four bits of this word. The field is organized as shown in Figure 6-1.

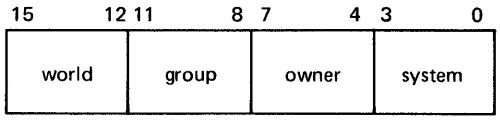


Figure 6-1 File Protection Field

FORMAT

PRO=<system,owner,group,world>

<system,owner,group,world>
The access code for the four classifications of users. An access code consists of four bits, each of which represents the type of access granted to a user in the class. These access rights and the characters that signify them are:

R - read access
W - write access
E - execute access
D - delete access

You can specify any number of access characters, in any order, for each classification. For example, you could specify RWD, RWED, DREW, or any combination, up to four characters per classification. The access rights for one classification are not separated by a comma. However, the classifications must be separated from each other by a comma or other valid separator to delimit the end of one classification and the start of the next. For example, the access rights for one classification may have a syntax of:

```
RWD
```

However, the syntax for three separate classifications might be:

```
RWD,DWRE,R
```

Note that when you use less than all four access rights characters, you need not supply a delimiter or code to indicate the omission.

The four different classifications of user, however, must be coded in the following order:

```
<system,owner,group,world>
```

The angle brackets are required syntax, and each classification must be separated from the others by a comma. In addition, when you omit a classification, the comma must be retained to indicate the omission, unless no other classifications follows. For example, to specify all access rights for system, owner, and world, you would write:

```
$XABPRO   PRO=<RWED,RWED,,RWED>
```

However, to specify all access rights to only system and owner, you would write:

```
$XABPRO   PRO=<RWED,RWED>
```

The absence of a code specifies that the access associated with the code is denied to the user.

The offset for the PRO field is:

```
XAB$W_PRO
```

Each 4-bit subfield also has its own symbolic offset, as follows:

- System -- XAB$V_SYS
- Owner -- XAB$V_OWN
- Group -- XAB$V_GRP
- World -- XAB$V_WLD

Additionally, each separate access specification has the following mask values:

- No read access -- XAB$M_NOREAD
- No write access -- XAB$M_NOWRITE
- No execute access -- XAB$M_NOEXE
- No delete access -- XAB$M_NODEL

USER CONSIDERATION

The bit values in the protection word are set to 1 to deny access. Thus, specifying a particular access right code clears the bit to 0.

NOTE

If you do not provide a file protection XAB for a $CREATE macro instruction, or if the PRO parameter is not specified or is specified as no access to all classes (all 1 bits), the default file protection for the process will be used for the newly created file.

$XABPRO UIC

6.4.3 Group and Member Number

The UIC parameter initializes both the group and member number fields, thus supplying both portions of the user identification code (UIC) of the file's owner.

FORMAT

UIC=<group,member>

<group,member>
The group number and member number, respectively, of the owner of the file. Both numbers are octal numbers in the range of 0 through 177777. The group number and member number must be enclosed within angle brackets, placed in the order shown in the format, and separated by a comma.

For example, if your group number is 126 and your member number is 1, the syntax is:

$XABPRO UIC=<126,1>

The symbolic offsets for these fields are:

- Group number -- XAB$W_GRP
- Member number -- XAB$W_MBM

The total user identification field, comprised of both the group and member number fields, has a symbolic offset of:

XAB$L_UIC

NOTE

If no file protection XAB is provided, or the user identification field is null for a $CREATE macro instruction, the UIC of the process will be used as the owner's UIC for the newly created file.

$XABPRO NXT

6.4.4 Next XAB Address

The NXT parameter sets the next XAB address field. See Section 6.2 for a complete description of this parameter.

6.4.5 File Protection Type Code and Block Length

The $XABPRO macro instruction sets the values for both the type code and block length fields. VAX-11 RMS interprets the last three characters (PRO) of the macro instruction name to determine the value. For the file protection XAB, the following values are stored:

- Type code field -- XAB$C_PRO
- Block length field -- XAB$C_PROLEN

$XABALL

6.5 ALLOCATION CONTROL XAB

The $XABALL macro instruction allocates and initializes an XAB that allows extended control of file disk space allocation, both for initial allocation and later extension. When you use an allocation control XAB as input to a create or extend service, certain fields override corresponding fields of the FAB, in particular, the allocation quantity (ALQ), bucket size (BKZ, which is the BKS in the FAB), default extension quantity (DEQ), and the CBT and CTG bits of the allocation options (AOP, which are the CBT and CTG bits of the FOP field in the FAB) fields. On an open or display service, VAX-11 RMS fills in these fields with the values that pertain to the file. Table 6-4 summarizes the fields comprising the allocation control XAB.

Table 6-4
Allocation Control Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
AID	byte	Area identification number	XAB$B_AID
ALN	byte	Alignment boundary type	XAB$B_ALN
ALQ	longword	Allocation quantity	XAB$L_ALQ
AOP	byte	Allocation options	XAB$B_AOP
BKZ	byte	Bucket size	XAB$B_BKZ
BLN[1]	byte	Block length	XAB$B_BLN
COD[1]	byte	Type code	XAB$B_COD
DEQ	word	Default extension quantity	XAB$W_DEQ
LOC	longword	Location	XAB$L_LOC
NXT	longword	Next XAB address	XAB$L_NXT
VOL	word	Relative volume number	XAB$W_VOL

[1] Indicates that this field is set automatically by the type of macro instruction.

FORMAT

OPERATION	PARAMETERS
label: $XABALL	AID=area-number
	ALN={CYL / LBN / VBN}
	ALQ=allocation-qty
	AOP=allocation-option
	BKZ=bucket-size
	DEQ=extension-qty
	LOC=number
	NXT=address
	VOL=volume-number

label: $XABALL

6.5.1 Label

VAX-11 RMS uses the label for the $XABALL macro instruction in the same way that it uses the label for the $XABDAT macro instruction; see Section 6.3.1.

$XABALL AID

6.5.2 Area Identification Number

The AID parameter initializes the area identification number field, which identifies the area of the file described by the current XAB. You are always responsible for the contents of this field; it is never set by VAX-11 RMS. Rather, VAX-11 RMS uses the contents of this field for the following:

- Checks the sequencing of allocation control XABs in an XAB chain. The allocation XABs in an XAB chain must appear in ascending order, based on the contents of the AID field in $CREATE and $EXTEND macro instructions; the order is irrelevant in the $OPEN an $DISPLAY macro instructions.

- Identifies the target area for a specific operation (for example, create, extend, and so on).

FORMAT

```
AID=area-number
```

area-number
Is a numeric value indicating which area, in a range of 0 through 254, of the file is described by the current XAB. If the file is a sequential or relative file, only a single allocation XAB can be used for any operation and its AID field must contain 0. The assembly-time default for this field is 0.

For example, to establish an allocation XAB for area 3 of an indexed file, you would write:

```
$XABALL   AID=3
```

The offset for this field is:

```
XAB$B_AID
```

$XABALL ALN

6.5.3 Alignment Boundary Type

The ALN parameter initializes the boundary type field, which specifies the type of alignment for the area to be allocated. This gives you control over the placement of your file. If you need this placement control on either a create or extend operation, you use the alignment boundary type field to specify whether the location field (LOC parameter) contains a starting cylinder number, logical block number, or virtual block number.

FORMAT

ALN= {CYL / LBN / VBN}

CYL
Indicates that the alignment starts at the cylinder number specified in the location field.

LBN
Indicates that the alignment starts at the logical block number specified in the location field.

VBN
Indicates that the alignment starts as near as possible to the virtual block number specified in the location field.

For example, if you want the file you are going to create or extend to be aligned at the tenth cylinder on the volume, you would write:

```
$XABALL   ALN=CYL,LOC=10
```

Each alignment type has its own symbolic value.

- CYL - XAB$K_CYL
- LBN - XAB$K_LBN
- VBN - XAB$K_VBN

The offset for this field is:

XAB$B_ALN

NOTE

If you do not set a value in this field, VAX-11 RMS assumes that you do not want to exercise control over the placement of your file.

$XABALL ALQ

6.5.4 Allocation Quantity

The ALQ parameter sets the allocation quantity field. This field indicates the number of blocks (or area via AID parameters for indexed files) for the initial file allocation ($CREATE macro instruction), or the number of blocks to add (or area via AID parameters for indexed files) when the file is extended ($EXTEND macro instruction).

In either case (create or extend operation), the value in this field overrides the contents of the allocation quantity field of the FAB (see Section 4.2.2).

The open, create, and display services fill in this field with the actual allocation size of the file or area for indexed files. The extend service fills in the field with the actual size of the extended space.

FORMAT

```
ALQ=allocation-quantity
```

allocation-quantity
A numeric value in the range of 0 to 4,294,967,295. A value of 0 (the default) indicates that no allocation is to be performed.

For example, to indicate that the allocation amount is 30 blocks, the syntax is:

```
$XABALL  ALQ=30
```

The offset for this field is:

```
XAB$L_ALQ
```

$XABALL AOP

6.5.5 Allocation Option

The AOP parameter sets the allocation option field, which lets you specify a particular type of allocation.

FORMAT

```
AOP=<CBT,CTG,HRD,ONC>
```

The ROP parameter can indicate any number of options; however, HRD is applicable only when CYL or LBN is specified for the ALN parameter of the allocation XAB. When only one option is chosen, angle brackets (< and >) are not required; otherwise, they are required syntax. The allocation options may be specified in any order.

CBT

Contiguous best try; indicates that VAX-11 RMS is to perform the initial allocation (or a later extension) using contiguous blocks, on a "best effort" basis. This overrides the CBT bit in the file processing options (FOP) field of the FAB.

CTG

Contiguous; indicates that the initial allocation (or later extension) must use contiguous blocks only; the allocation fails if the requested number of contiguous blocks is not available. If this is the initial allocation, the file is marked contiguous. Overrides the CTG bit in the file-processing options field of the FAB.

HRD

Hard; indicates that if the requested alignment cannot be performed, an error will be returned. The default is that allocation is to be performed as near as possible to the requested alignment.

NOTE

The HRD option is applicable only to CYL and LBN alignment boundary types, specified by the ALN parameter of the allocation XAB.

ONC

On cylinder boundary; indicates that VAX-11 RMS is to start the allocation on any available cylinder boundary.

For example, suppose you want 30 blocks allocated contiguously starting at logical block number 1024 with an error returned if not possible. You would write:

```
$XABALL   ALQ=30,          -; allocation amt
          ALN=LBN          -; start at logical blk.  no.
          LOC=1024         -;  1024
          AOP=<CTG,HRD>    ; contig.  or rtn.  error
```

Each allocation request option has its own symbolic bit offset and mask value, formed by prefixing the option value with XAB$V_ and XAB$M_, respectively. For example:

XAB$V_CBT and XAB$M_CBT

The offset for this field is:

XAB$B_AOP

$XABALL BKZ

6.5.6 Bucket Size

The BKZ parameter initializes the bucket size field, which is used only with the relative and indexed file organizations. When you create a relative or indexed file, you specify the bucket size field before issuing the $CREATE macro instruction. For a relative file, the BKZ parameter specifies the bucket size for the file, since a

relative file may have only one area. However, for an indexed file, the BKZ parameter specifies the bucket size for the area described by this allocation XAB; this allows you to vary the size of buckets among the multiple areas of your indexed file. When you open an existing file, VAX-11 RMS sets this field to the defined size of the buckets in the file for a relative file or the defined size of the buckets in this area (defined by the AID parameter) for an indexed file.

The value in this field overrides the contents of the bucket size field (BKS) of the FAB on a create service (see Section 4.2.3).

FORMAT

```
BKZ=bucket-size
```

bucket-size
A numeric value, in the range of 0 to 32, representing the number of blocks in a bucket. If this parameter is omitted or if a value of 0 is used, then a default size will be used equal to the minimum number of blocks required to contain a single record.

For example, to specify a bucket size of two blocks, you would write:

```
$XABALL  BKZ=2
```

The offset for this field is:

```
XAB$B_BKZ
```

$XABALL DEQ

6.5.7 Default Extension Quantity

The DEQ parameter initializes the default extension quantity field, which specifies the number of blocks to add to the file whenever it is extended automatically.

The value in this field overrides the contents of the default extension quantity field (DEQ) of the FAB (see Section 4.2.6).

FORMAT

```
DEQ=extension-quantity
```

extension-quantity
The number of blocks to be added when automatic extension is required. This number must be in the range of 0 to 65,535. If you specify 0, the file will be extended using a VAX-11 RMS-determined default extension value.

For example, to specify a default extension quantity of 50 blocks, you would write:

```
$XABALL  DEQ=50
```

The offset for this field is:

```
XAB$W_DEQ
```

$XABALL LOC

6.5.8 Location

The LOC parameter initializes the location field, indicating the starting point for file allocation. The exact interpretation of this field depends on the contents of the alignment boundary type field (ALN) (see Section 6.5.3). VAX-11 RMS uses the contents of the location field on a $CREATE or $EXTEND macro instruction, but only if the alignment boundary type field (ALN) is also initialized.

FORMAT

```
LOC=number
```

number
The starting point for the allocation is determined from the contents of the alignment boundary type field as follows:

- If CYL is specified for the ALN parameter, the LOC number specified is the starting cylinder number where the allocation is to start, in the range of 0 to the maximum cylinder number on the volume.
- If LBN is specified for the ALN parameter, the LOC number specified is the logical block number where the allocation is to start, in the range of 0 to the maximum number of blocks on the volume.
- If VBN is specified for the ALN parameter, the LOC number specified is the virtual block number where the allocation is to start, in a range from 1 to the maximum number of blocks in the file. This is used only in conjunction with a $EXTEND macro instruction. If the number 0 is specified, or if the number is omitted during an extend operation, VAX-11 RMS extends as near as possible to the end of the file.

For example, to indicate that you want to allocate 30 blocks contiguously starting at or near logical block 1024, you would write:

```
$XABALL ALQ=30          -; allocate 30 blocks
        ALN=LBN         -; start at logical block
        LOC=1024        -; number 1024
        AOP=CTG          ; contiguously
```

The offset for this field is:

```
XAB$L_LOC
```

$XABALL VOL

6.5.9 Relative Volume Number

The VOL parameter initializes the relative volume number field. It indicates the specific member of a volume set upon which the file is to be allocated.

FORMAT

```
VOL=volume-number
```

volume-number
An integer in the range 0 to 65535. Assembly-time default is 0, specifying the "current" member of the volume set.

For example, to indicate that the file is to reside on relative volume number 3 of the volume set, you would write:

```
$XABALL  VOL=3,ALQ=30,ALN=CYL,LOC=1
```

The offset for this field is:

```
XABALL$W_VOL
```

$XAVALL NXT

6.5.10 Next XAB Address

The NXT parameter sets the next XAB address field. See Section 6.2 for a complete description of this parameter.

6.5.11 Allocation Control Type Code and Block Length

The $XABALL macro instruction sets the values for both the type code and block length fields. VAX-11 RMS interprets the last three characters (ALL) of the macro instruction name to determine the value. For the allocation control XAB, the following values are stored:

- Type code field -- XAB$C_ALL
- Block length field -- XAB$C_ALLLEN

$XABKEY

6.6 KEY DEFINITION XAB

The $XABKEY macro instruction allocates and initializes an XAB that defines the key fields of an indexed file at file creation; it also allows retrieval of the key definition at file open and display. Each key definition XAB describes one key of an indexed file.

When you create an indexed file, you must set the contents of the fields of this XAB before you issue the $CREATE macro instruction. Further, you must provide one key definition for each key that you want the file to have. Since every indexed file must have at least one key, the primary key, you will always require at least one key definition XAB.

When you open an existing indexed file or issue a $DISPLAY operation for such a file, you use key definition XABs only if you want VAX-11 RMS to provide your program with one or more of the key definitions specified when the file was created.

Table 6-5 summarizes the fields that comprise the key definition XAB.

Table 6-5
Key Definition Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
DAN	byte	Data bucket area number	XAB$B_DAN
DBS[1]	byte	Data bucket size	XAB$B_DBS
DFL	word	Data bucket fill size	XAB$W_DFL
DTP	byte	Data type of the key	XAB$B_DTP
DVB[1]	longword	First data bucket start virtual block number	XAB$L_DVB
FLG	byte	Key options flag	XAB$B_FLG
IAN	byte	Index buckets area number	XAB$B_IAN
IBS[1]	byte	Index bucket size	XAB$B_IBS
IFL	word	Index bucket file size	XAB$W_IFL
KNM	longword	Key name buffer address	XAB$L_KNM
LAN	byte	Lowest level of index area number	XAB$B_LAN
LVL[1]	byte	Level of root buckets	XAB$B_LVL
MRL[1]	word	Minimum record length	XAB$W_MRL
NSG[1]	byte	Number of key segments	XAB$B_NSG
NUL	byte	Null key value	XAB$B_NUL
POS	word	Key position	XAB$W_POSO through XAB$W_POS7

Table 6-5 (Cont.)
Key Definition Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
REF	byte	Key of reference	XAB$B_REF
RVB[1]	longword	Root bucket start virtual block number	XAB$L_RVB
SIZ	byte	Key size	XAB$B_SIZO through XAB$B_SIZ7
TKS[1]	byte	Total key field size	XAB$B_TKS

[1] Indicates nonuser-initialized field

FORMAT

OPERATION	PARAMETERS
label:$XABKEY	DAN=area-number
	DFL=bytes
	DTP=data-type-code
	FLG=<option,option,...>
	IAN=area-number
	IFL-bytes
	KNM=address
	LAN=area-number
	NUL=value
	POS=<position,...>
	REF=value
	SIZ=<size,...>

label: $XABKEY

6.6.1 Label

VAX-11 RMS uses the label for the $XABKEY macro instruction in the same way that it uses the label for the $XABDAT macro instruction; see Section 6.3.1.

$XABKEY DAN

6.6.2 Data Buckets Area Number

The DAN parameter initializes the data buckets area number field of the key definition XAB. You use this parameter to specify the area of the file that the data buckets are to reside in only when both of the following are true:

- You are creating a new indexed file
- You are using allocation XABs (described in Section 6.5) to control placement

When the key definition XAB describes the primary key, the data level of the index consists of buckets that contain the actual data records of the file. However, when the key definition describes an alternate key, the data level of the index consists of buckets in which VAX-11 RMS maintains pointers to the actual data records.

FORMAT

```
DAN=area-number
```

area-number
A numeric value in the range 0 through 254, representing an identification number contained in the AID field of an allocation XAB present in the same chain (see Section 6.5.2). The assembly-time default is 0, that is, area 0.

For example, to indicate that these data buckets are to reside in area 3 of an indexed file, you would write:

```
$XABKEY DAN=3
```

The offset for this field is:

```
XAB$B_DAN
```

$XABKEY DFL

6.6.3 Data Buckets Fill Size

The DFL parameter initializes the data buckets fill size field of the key definition XAB. When you create an indexed file, you use this parameter to specify the number of bytes (of data) you want in each data level bucket. If you specify less than the total possible bucket size, you thereby indicate that the data buckets are to contain some amount of free space. At run time, VAX-11 RMS follows the fill size specified at $CREATE time only if the RAB$V_LOA bit is set in the record processing options (ROP) field of the RAB. The ROP field is described in Section 5.

When the key definition XAB describes the primary key, the DFL field describes the space in the buckets containing actual user data records. When the key definition XAB describes an alternate key, the DFL field describes the space in the buckets containing pointers to the user data records.

It is advantageous to use the DFL field in the following situation:

If you expect to perform numerous $PUT and $UPDATE operations on the file after it has been initially populated, you can minimize the resultant movement of records (known as bucket splitting) by specifying less than the maximum bucket fill size at $CREATE time. To utilize the free space thereby reserved in the buckets, programs that perform $PUT or $UPDATE operations on the file should not place the value RAB$LOA in the ROP field of the RAB.

FORMAT

```
DFL=bytes
```

bytes
A numeric value representing the maximum number of bytes (of data) in a data bucket. The assembly-time default value is 0, which is interpreted by VAX-11 RMS as meaning the maximum available space (i.e., no unused space).

For example, to specify that each bucket at the data level is to be filled to a maximum of 400 bytes, you would write:

```
$XABKEY DFL=400
```

The offset for this field is:

```
XAB$W_DFL
```

$XABKEY DTP

6.6.4 Key Data Type

The DTP parameter initializes the data type of the key field of the XAB. When you create an indexed file, you use this parameter to specify the type of data in the record key field.

Key field data types and the data type codes are summarized and the associated global symbols are listed in Table 6-6.

Table 6-6
Key Field Data Types, Data Type Codes and Global Symbols

Key Field Data Type	Data Type Code	Global Symbol
String	STG	XAB$K_STG
Signed 2-byte integer	IN2	XAB$K_IN2
Signed 4-byte integer	IN4	XAB$K_IN4
Unsigned 2-byte binary	BN2	XAB$K_BN2
Unsigned 4-byte binary	BN4	XAB$K_BN4
Packed decimal	PAC	XAB$K_PAC

String data type (STG) is defined as a left-justified string of unsigned 8-bit bytes.

The string key field can comprise from one to eight disjoined key field segments (see Sections 6.6.1 and 6.6.13).

Integer, binary, and packed decimal key fields must be a contiguous set of bytes.

The null value (that is, NUL option in FLG parameter is set) for integer, binary, and packed decimal is zero and the NUL parameter (field) is ignored (see Sections 6.6.5 and 6.6.10).

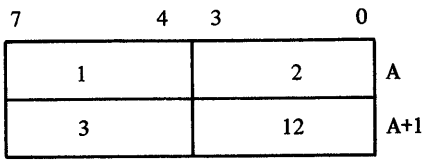
A packed decimal is a contiguous sequence of bytes and is specified by two attributes: the address, A, of the first byte of the string and a length, L, that is the number of digits in the packed decimal. The bytes of a packed decimal are divided into two 4-bit fields (nibbles) that must contain decimal digits, except for the low nibble (bits 3:0) of the last (highest addressed) byte, which must contain a sign. The representation for the digits and signs is shown in Table 6-7.

Table 6-7
Packed Decimal Digits and Signs Representation

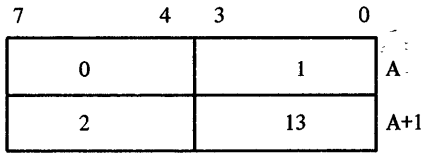
Digit or Sign	Decimal	Hex
0	0	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
+	10, 12, 14 or 15	A, C, E or F
–	11 or 13	B or D

The preferred sign representation is 12 for plus (+) and 13 for minus (-). The length L is the number of digits in the packed decimal (not counting the sign) and must be in the range 0 through 31. When the number of digits is even, it is required that an extra 0 digit appear in the high nibble (bits 7:4) of the first byte. Again the length in bytes of the packed decimal is L/2 + 1. The value of a 0 length-packed decimal is identically 0; it contains only the sign byte, which also includes the extra 0 digit.

The address, A, of the packed decimal specifies the byte containing the most significant digit in its high nibble. Digits of decreasing significance are assigned to increasing byte addresses and from high nibble to low nibble within a byte. Thus +123 has length 3 and is represented as follows:



and -12 has length 2 and is represented as follows:



Integer and binary key field data have the following formats:

```
IN2:   LSB at A
       MSB and sign at A+1
IN4:   LSB at A
       MSB and sign at A+3
BN2:   LSB at A
       MSB at A+1
BN4:   LSB at A
       MSB at A+3
```

FORMAT

```
DTP=data-type-code
```

data-type-code
One of the following, as appropriate:

```
STG, string (left-justified, unsigned 8-bit bytes)
IN2, signed 2-byte integer key data
IN4, signed 4-byte integer key data
BN2, unsigned 2-byte binary key data
BN4, unsigned 4-byte binary key data
PAC, packed decimal key data
```

For example, to specify that the key data type is a signed 4-byte integer, you would write:

```
$XABKEY DTP=IN4
```

The offset for this field is:

```
XAB$B_DTP
```

$XABKEY FLG

6.6.5 Key Options Flag

The FLG parameter initializes the key options flag field of the key definition XAB. When you create an indexed file, you specify the following optional characteristics of the key represented by this XAB:

- Key values can change
- Duplicate key values are permitted
- Null key value

The allowed combinations of the changeable key values and duplicate key values options depend on the type of key (that is, primary or alternate) represented by this XAB; table 6-8 summarizes these combinations.

Table 6-8
Key Options Flag Combinations

Key Type	Combinations			
	CHG + DUP	CHG + NO DUP	NO CHG + DUP	NO CHG + NO DUP
Primary	Error	Error	Allowed	Default
Alternate	Default	Allowed	Allowed	Allowed

You should note that the entire FLG field is affected when an option is specified, setting those bits specified and clearing those not specified. When a bit is set, it means the corresponding option is specified for the key. Conversely, when a bit is cleared (to 0), it means that the corresponding option is not specified for the key. Therefore, when defining the characteristics of a key, you should specify exactly what you want the field to contain, expressing bit settings as symbolic values. Any symbolic values you omit will result in a 0 in the corresponding bit position.

FORMAT

```
FLG=<option,option,...>
```

option
one of the following:

CHG
The key value within the record in the file can be changed by a program during a $UPDATE operation. This option can be specified only for alternate keys.

DUP
The key value within the record in the file may have the same key value as another record (or other records) within the file. (Refer to the chart above for allowable combinations of CHG and DUP options relative to key type.)

NUL
The NUL field of the XAB contains a null key value if the key data type is string. If the key data type is other than string (i.e., is integer, binary, or packed decimal), then the null key value is 0. This option can be specified only for alternate keys. Refer to Section 6.6.10 for a description of the XAB NUL field.

The assembly-time defaults for the FLG field depend on the type of key defined by the XAB.

The defaults for a primary key are as follows:

- Duplicate key values are not allowed
- Key values cannot change

The defaults for an alternate key are as follows:

- Duplicate key values are allowed
- Key values can change
- No null key values

When you specify more than one option with the FLG parameter, you must enclose the options in angle brackets. The options can be specified in any order. When you specify only one option, no angle brackets are required.

For example, to specify that duplicate key values are allowed, that a null key value is allowed, and that key values cannot change (through absence of CHG), you would write the following:

```
$XABKEY FLG=<DUP,NUL>
```

Each key option flag operation has its own symbolic bit offset and mask value. The bit offset is formed by prefixing XAB$V_ to the operation. For example:

```
XAB$V_DUP
```

The mask value is formed by prefixing XAB$M_ to the operation. For example:

```
XAB$M_DUP
```

The offset for the key option flag field is:

```
XAB$B_FLG
```

$XABKEY IAN

6.6.6 Index Buckets Area Number

The IAN parameter initializes the index buckets area number field of the key definition XAB. When you create an indexed file, you use this parameter to specify the area of the file that the index buckets are to reside in only when both of the following are true:

- You are creating a new indexed file.
- You are using allocation XABs (described in Section 6.5) to control placement and structure of the file.

When the key definition XAB describes the primary key, the index level of the index consists of all levels of the tree- (pyramid-)structured primary index down to and including the level containing pointers to the user data records themselves. However, when the key definition describes an alternate key, the index level of the index comprises all levels of the pyramid-structured alternate index down to, but not including, the level containing buckets in which VAX-11 RMS maintain pointer arrays describing the user data records. Refer to the LAN parameter for a description of how to place the lowest level of the index in a location separate from the higher levels.

FORMAT

```
IAN=area-number
```

area-number
A numeric value in the range 0 through 254, representing an identification number contained in the AID field of an allocation XAB present in the same chain (see Section 6.5.2). The assembly-time default is 0, that is, area 0.

For example, to indicate that these index buckets are to reside in area 3 of an indexed file, you would write:

```
$XABKEY IAN=3
```

The offset for this field is:

```
XAB$B_IAN
```

$XABKEY IFL

6.6.7 Index Buckets Fill Size

The IFL parameter initializes the index buckets fill size field of the key definition XAB. When you create an indexed file, you use this parameter to specify the number of bytes you want in each index bucket. If you specify less than the total possible bucket size, you indicate that the index buckets are to contain some amount of free space. At run time, VAX-11 RMS adheres to the fill size specified at $CREATE time only if the RAB$V_LOA bit is set in the record-processing options (ROP) field of the RAB. The ROP field is described in Section 5.

When the key definition XAB describes the primary key, the IFL field describes the space in the buckets in all levels of the primary index down to and including the level containing pointers to the user data records. When the key definition XAB describes an alternate key, the IFL field describes the space in the buckets in all levels of the alternate index down to, but not including, the level containing buckets in which VAX-11 RMS maintains pointer arrays describing the user data records.

It is advantageous to use the IFL field in the following situation:

If you expect to perform numerous $PUT and $UPDATE operations on the file after it has been initially populated, you can minimize the resultant movement of records (known as bucket splitting) by specifying less than the maximum bucket fill size at $CREATE time. To utilize the free space thereby reserved in the buckets, programs that perform $PUT or $UPDATE operations on the file should not place the value RAB$LOA in the ROP field of the RAB.

FORMAT

```
IFL=bytes
```

bytes
A numeric value representing the maximum number of bytes in an index bucket. The assembly-time default value is 0, which is interpreted by VAX-11 RMS as meaning the maximum available space (that is, no unused space).

For example, to specify that each index bucket is to be filled to a maximum of 256 bytes, you would write:

```
$XABKEY IFL=256
```

The offset for this field is:

```
XAB$W_IFL
```

$XABKEY KNM

6.6.8 Key Name Address

The KNM parameter initializes the key name buffer address field of the key definition XAB. When you define a key during creation of an indexed file, you can associate any 32-character string you choose with the key field represented by the XAB. VAX-11 RMS never examines this character string, but it retains it in the file as part of the key definition information.

FORMAT

```
KNM=address
```

address
The symbolic address of a buffer, which must always be at least 32 bytes in length. A value of 0 in this field indicates that no key name is defined during a $CREATE operation or is to be displayed during a $OPEN or $DISPLAY operation.

For example, if the key buffer area has a label of KEYBUF, you would write:

```
$XABKEY KNM=KEYBUF
```

The offset for this field is:

```
XAB$L_KNM
```

$XABKEY LAN

6.6.9 Lowest Level of Index Area Number

The LAN parameter initializes the lowest level of index area number field of the key definition XAB. It permits you to separate the lowest level (level 1) of the index from all higher levels (levels 2 +) of the index in an indexed file; that is, you can use the LAN parameter to specify an area of the index wherein the lowest level of the index will reside, which is separate from the area (or areas) specified by the IAN parameter (wherein higher levels of the index will reside). The IAN parameter is described in Section 6.6.6.

You can utilize the LAN parameter only when both of the following are true:

- You are creating a new indexed file.
- You are using allocation XABs (described in Section 6.5) to control placement and structure of the file.

NOTE

The bucket size of the area specified by the LAN parameter must be the same as the bucket size specified by the IAN parameter.

FORMAT

LAN=area-number

area-number
A numeric value in the range 0 through 254, representing an identification number contained in the AID field of an allocation XAB present in the same chain (see Section 6.5.2). The assembly-time default is 0; that is, the lowest level of the index will occupy the same area of the file as the remainder of the index.

For example, to indicate that the lowest level of the index is to reside in area 3 of an indexed file, you would write:

```
$XABALL  AID=3      ;area identification
$XABKEY  IAN=5     -;index area number
         LAN=3      ;lowest level of index area number
```

The offset for this field is:

XAB$B_LAN

$XABKEY NUL

6.6.10 Null Key Value

The NUL parameter initializes the null field of the key definition XAB. Normally, VAX-11 RMS updates all indexes to reflect the values in the corresponding key fields of the records written to an indexed file. The NUL parameter, however, allows you to instruct VAX-11 RMS not to make an entry in an alternate index if a record being entered in an indexed file contains the specific (null) alternate key value. The following prerequisites must be satisfied for you to use the NUL parameter:

- The file must be an indexed file.
- The XAB must define an alternate key.
- The NUL option of the FLG parameter must have been set at file creation (refer to Section 6.6.5 for a description of the FLG parameters).

FORMAT

NUL=value

value
Any user-selected character value

For example, to indicate that a record with an alternate key value of 127 (ASCII delete) is not to have an entry made for it in the associated alternate index (in this case the second alternate index), you would write:

```
$XABKEY  FLG=NUL   -;set null flag
         NUL=127   -;null key value
         REF=2      ;second alternate key
```

The offset for this field is:

XAB$B_NUL

$XABKEY POS

6.6.11 Key Position

The POS parameter initializes the key position field of the key definition XAB. The key position field defines the location of the key within each record of an indexed file, and is eight words in length. Two types of keys can be defined: simple keys and segmented keys.

A simple key is a single string of contiguous bytes in the records. The first word of the position field specifies the starting position of the string and the remaining words contain 0s. You can use simple keys with any data type (string, integer,) binary, or packed decimal (see Section 6.6.4).

Segmented keys can be used only with key fields that contain string data. A segmented key consists of two to eight strings of bytes in the record. Each individual string (segment) is a set of contiguous bytes, but the strings do not need to be contiguous; additionally, the strings can be in any order and may overlap. Each successive word of the position field specifies a starting position of one of the segments. When processing records that contain segmented keys, VAX-11 RMS regards the key field as a single, logically contiguous string beginning with the first segment and ending with the last.

You should note that the key position and the key size field (see Section 6.6.13) must define an equal quantity of key position values and key size values.

FORMAT

POS=position

or

POS=<position0,position1,...,position7>

position
Is a numeric value representing the starting (byte) position of the key within each record. The first byte of a record is represented by the value 0, the second by the value 1, etc. A simple key has only one starting position, while a segmented key may have up to eight starting positions.

For example, to indicate that a record contains a simple key which starts in the first byte of each record, you would write:

```
$XABKEY  POS=0,  -;  key starts in first byte
         SIZ=8    ;  key length 8 bytes
```

To indicate that a record contains a segmented key consisting of 4 segments with the 1st segment starting in the 20th byte, the 2nd segment starting in the 14th byte, the 3rd segment starting in the 1st byte, and the 4th segment starting in the 29th byte, you would write:

```
$XABKEY  POS=<19,13,0,28>,  -;  segmented key
         SIZ=<8,2,5,32>      ;  length in bytes
```

You must include the angle brackets for multiple argument key positions.

The offset for this field is:

```
XAB$W_POSO,...,XAB$W_POS7
```

$XABKEY REF

6.6.12 Key of Reference

The REF parameter initializes the key of reference field in the key definition XAB. The key of reference field identifies which key (that is, primary, first alternate, second alternate, and so on) in an indexed file is defined by the XAB.

NOTE

VAX-11 RMS can process an indexed file with 255 defined keys; however, you should be aware that each key field defined has associated with it a cost in processing and I/O time. The time to build and maintain the index for the key field and the disk storage required to contain the index for each key field should be considered when you make the design decision as to whether the field should be an alternate key field. A file with six to eight defined keys (the primary and five to seven alternate keys) should be considered as a maximum; a file with two or three defined keys should be considered normal.

FORMAT

```
REF=value
```

value

Is a numeric value in the range 0 through 254 indicating which key is represented by the XAB. A value of 0 indicates the primary key, 1 indicates the first alternate key, 2 indicates the second atlternate key, and so on. For the $CREATE and $EXTEND macro instructions, the key references must be listed consecutively, in ascending order. The order is irrelevant for the $OPEN and $DISPLAY macro instructions.

For example, to indicate the primary key, you would write:

```
$XABKEY REF=0
```

The offset for this field is:

```
XAB$B_REF
```

$XABKEY SIZ

6.6.13 Key Size

The SIZ parameter initializes the key size field of the key definition XAB. The key size field defines the length (in bytes) of the key (whose starting position is defined in the key position field of the same XAB) within each record of an indexed file. Two types of keys can be defined: simple keys and segmented keys (see Section 6.6.11). The key size field defining a simple key will contain only one key size value. The key size field defining a segmented key must contain a key size value for each segment of the key. You should note that the key size field and the key position field (see Section 6.6.11) must contain an equal quantity of key size values and key position values. VAX-11 RMS associates the first key position value specified with the first key size value specified which together define the location and length of the first segment of a segmented key, and so forth.

FORMAT

```
SIZ=size0
```

or

```
SIZ=<size0,size1,...,size7>
```

size

Is a numeric value representing the length, in bytes, of the key within the record. Up to eight values can be assigned.

When the data type of the key (see Section 6.6.4) is string, the total size (sum of <size,size,...>) of the key must be less than 256 bytes.

When the data type of the key (see Section 6.6.4) is 2-byte integer or 2-byte binary, size0 must equal 2 and size1 through size7 must be 0s. If size0 is 0, it is defaulted to 2.

When the data type of the key (see Section 6.6.4) is 4-byte integer or 4-byte binary, size0 must equal 4 and size1 through size7 must be 0s. If size0 is 0, it is defaulted to 4.

When the data type of the key (see Section 6.6.4) is packed decimal, the size specified by size 0 must be from 1 through 16 and size 1 through 7 must be 0s.

For example, to indicate that a record contains a simple key eight bytes in length, you would write:

```
$XABKEY  POS=0,  -;  key starts in first byte
         SIZ=8    ;  key length 8 bytes
```

To indicate that a record contains a segmented key consisting of 4 segments with the 1st segment 8 bytes in length, the 2nd segment 2 bytes in length, the 3rd segment 5 bytes in length, and the 4th segment 32 bytes in length, you would write:

```
$XABKEY  POS=<19,13,0,28>,  -; key segment start locations
         SIZ=<18,2,5,32>     ; key length in bytes
```

The offset for this field is:

XAB$B_SIZ0,...,XAB$B_SIZ7

6.7 NONINITIALIZABLE KEY FIELDS

The following list describes the KEY fields that you cannot initialize at assembly time; VAX-11 RMS sets them for you.

DBS

Data bucket size field. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the size of the data level (level 0) buckets, in virtual blocks, for the key described by the XAB.

The offset to this field is XAB$B_DBS.

DVB

First data bucket start virtual block number. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the start virtual block number for the first data level bucket for the key described by the XAB.

The offset to this field is XAB$L_DVB.

IBS

Index bucket size. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the size of the index level (level 1 to n) buckets, in virtual blocks, for the key described by the XAB.

The offset to this field is XAB$B_IBS.

LVL

Level of root bucket. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the level of the root bucket for the key described by the XAB.

The offset to this field is XAB$B_LVL.

MRL

Minimum record length. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the minimum record length in bytes, which will totally contain the key field for the key described by the XAB.

If the key described by the XAB is the primary key (REF=0), then a record must be equal to or greater than the minimum record length returned in MRL to be inserted/updated in the file.

If the key described by the XAB is an alternate key (REF=1 to n), then a record must be equal to or greater than the minimum record length returned in MRL to be recorded in the associated index for that alternate key. The offset to this field is XAB$W_MRL.

NSG

Number of key segments. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the number of key segments which make up the key field for the key described by the XAB (see Section 6.6.11). This field must not be altered.

The offset to this field is XAB$B_NSG.

RVB

Root index bucket start virtual block number. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the start virtual block number for the root bucket of the index for the key described by the XAB.

The offset to this field is XAB$L_RVB.

TKS

Total key size. When a key definition XAB is present during an open or display operation, VAX-11 RMS sets this field to the total key size, in bytes (the sum of SIZ0 through SIZ7), for the key described by the XAB (see Section 6.6.13).

The offset to this field is XAB$B_TKS.

$XABSUM

6.8 SUMMARY XAB

The $XABSUM macro instruction allows you to determine the number of keys and/or the number of allocation areas defined and the prologue version number for an existing file.

The summary XAB is ignored with a $CREATE macro call. However, one summary XAB can be associated with a FAB at the time a $OPEN or $DISPLAY macro call is issued. The presence of this XAB during these calls allows VAX-11 RMS to return to your program the total number of keys and allocation areas defined and the prologue version number when the file was created.

There are no assembly-time initialization key words for this macro.

Table 6-9 summarizes the fields that comprise the summary XAB.

NOTE

The summary XAB is used only with indexed files.

Table 6-9
Summary Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
NOA	byte	Number of allocation areas defined for the file	XAB$B_NOA
NOK	byte	Numbers of keys defined for the file	XAB$B_NOK
PVN	word	Prologue version number	XAB$W_PVN

$XABFHC

6.9 FILE HEADER CHARACTERISTICS XAB

The $XABFHC macro instruction allocates and initializes a file header characteristics XAB. You can use this block to display information about the file as stored in the file header.

The only field in this XAB that you initialize at assembly time is the next XAB address field (if this XAB points to a succeeding XAB in the chain). VAX-11 RMS copies the file characteristics (including the starting logical block number if the file is contiguous) into this XAB whenever an operation is performed with a $OPEN or $DISPLAY macro instruction. The field is then available for you to examine during processing. Note that for shared sequential files, the values in the end-of-file block, first free byte in the end-of-file block, and longest record length fields correspond to the values at the time of the last close or flush service.

On a create service, only the longest record length field of this XAB is used as an input attribute, and then only if the record format is not fixed-length.

Table 6-10 summarizes the fields comprising the file header characteristics XAB. Note that many of these fields are also available in the FAB.

FORMAT

OPERATION	PARAMETERS
label: $XABFHC	NXT=address

Table 6-10
File Header Characteristics
Extended attribute Block Fields

Field Name	Field Size	Description	Offset
ATR	byte	Record attributes; equivalent to the RAT field of the FAB	XAB$B_ATR
BKZ	byte	Bucket size; equivalent to the BKS field of the FAB	XAB$B_BKZ
BLN[2]	byte	Block length	XAB$B_BLN
COD[2]	byte	Type code	XAB$B_COD
DXQ	word	Default file extension quantity; equivalent to the DEQ field of the FAB	XAB$W_DXQ
EBK	longword	End-of-file block	XAB$L_EBK
FFB	word	First free byte in the end-of-file block	XAB$W_FFB
HBK	longword	Highest virtual block in the file; the execution of a $OPEN macro instruction sets the allocation quantity field of the FAB to this value	XAB$L_HBK
HSZ	byte	Fixed length control header size; equivalent to the FSZ field of the FAB	XAB$B_HSZ
LRL	word	Longest record length	XAB$W_LRL
MRZ	word	Maximum record size; equivalent to the MRS field of the FAB	XAB$W_MRZ
NXT[1]	longword	Next XAB address	XAB$L_NXT
RFO	byte	File organization and record format; combines the RFM and ORG fields of the FAB	XAB$B_RFO
SBN	longword	Starting logical block number for the file if it is contiguous, otherwise this field is 0	XAB$L_SBN

[1] This field can be initialized at assembly time.
[2] Indicates that this field is set automatically by the type of macro instruction.

6.9.1 Label

VAX-11 RMS uses the label for the $XABFHC macro instruction in the same way that it uses the label for the $XABDAT macro instruction; see Section 6.3.1.

$XABFHC NXT

6.9.2 Next XAB Address

The NXT parameter sets the next XAB address field. See Section 6.2 for a complete description of this parameter.

6.9.3 File Header Type Code and Block Length

The $XABFHC macro instruction sets the values for the type code and block length fields. VAX-11 RMS interprets the last three characters (FHC) of the macro instruction name to determine the value, as follows:

- Type code field -- XAB$C_FHC
- Block length field -- XAB$C_FHCLEN

$XABRDT

6.10 REVISION DATE AND TIME XAB

The $XABRDT macro instruction allocates and initializes an XAB for revision date and time. This XAB operates much like the date and time XAB (see Section 6.3) when input to the $OPEN, $DISPLAY, or $CREATE macro instructions. However, when you gain access to a file for writing, issuing a $CLOSE macro instruction for that file causes the revision date and time to be set from the current date and time and the revision number to be incremented. Thus, any revision date and time you specify through the XAB on a $CREATE macro instruction is lost.

For this reason, you can input the revision date and time XAB to the $CLOSE macro instruction, and cause the file's revision date and time and revision number to take on the specified values.

Table 6-11 summarizes the fields comprising the revision date and time XAB.

Table 6-11
Revision Date and Time Extended Attribute Block Fields

Field Name	Field Size	Description	Offset
BLN[2]	byte	Block length	XAB$B_BLN
COD[2]	byte	Type code	XAB$B_COD
NXT	longword	Next XAB address	XAB$L_NXT
RDT[1]	quadword	Revision date and time	XAB$Q_RDT
RVN[1]	word	Revision number	XAB$W_RVN

[1] Indicates no assembly time initialization.
[2] Indicates that this field is set automatically by the type of macro instruction.

FORMAT

OPERATION	PARAMETERS
label: $XABRDT	NXT=address

label: $XABRDT

6.10.1 Label

VAX-11 RMS uses the label for the $XABRDT macro instruction in the same way it uses the label for the $XABDAT macro instruction; see Section 6.3.1.

$XABRDT NXT

6.10.2 Next XAB Address

The NXT parameter sets the next XAB address field. See Section 6.2 for a complete description of this parameter.

6.10.3 Revision Date and Time

VAX-11 RMS sets certain values for the revision date and time, and returns them in the revision date and time XAB fields for your inspection. You can override these system-supplied values through the

use of a revision date and time XAB as input to a $CLOSE or $CREATE macro instruction. However, the $XABRDT macro instruction does not contain parameters for the assembly time initialization of these fields. As outlined in Table 6-8, these fields are:

- Revision date and time (RDT) -- this is a 64-bit binary field, indicating the date and time at which the file was last updated. The symbolic offset for this field is:

  XAB$Q_RDT

- Revision Number (RVN) -- this field provides the number of times this file was opened for write operations. The symbolic offset for this field is:

  XAB$W_RVN

6.10.4 Revision Date and Time Type Code and Block Length

The $XABRDT macro instruction sets the values for both the type code and block length fields. VAX-11 RMS interprets the last three characters (RDT) of the macro instruction to determine the value. For the revision date and time XAB, the following symbolic values are stored:

- Type code field -- XAB$C_RDT
- Block length field -- XAB$C_RDTLEN

CHAPTER 7

THE NAME BLOCK

This chapter describes the Name (NAM) Block, its fields, and the macro instruction and parameters that initialize the fields at assembly time.

7.1 THE PURPOSE OF THE NAME BLOCK

The NAM block contains supplementary information for use with the file specification, and is useful as a means to optimize file opening. The fields of the NAM block include the following information:

- Device identification
- Directory identification
- File identification
- Expanded and resultant file name strings
- Address of a related file's NAM block
- Wildcard context

To use a NAM block, you must specify its symbolic address as the value in the name block address field (NAM parameter) of the associated FAB.

The $NAM macro instruction allocates a NAM block. At assembly time, you can initialize the fields in the NAM block through keyword parameters. For run-time access to these fields, you can use the keyword parameters with the $NAM_STORE macro instruction (see Chapter 14), or the symbolic offsets.

Table 7-1 summarizes the fields comprising the NAM block. Some of these fields, however, are set by VAX-11 RMS or are static; therefore, you cannot initialize them at assembly time by keyword parameters.

Table 7-1
Name Block Fields

Field Name	Field Size	Description	Offset
BID[1]	byte	Block identifier	NAM$B_BID
BLN[1]	byte	Block length	NAM$B_BLN
DID[1]	3 words	Directory identification	NAM$W_DID
DVI[1]	16 bytes	Device identification	NAM$T_DVI
ESA	longword	Expanded string area address	NAM$L_ESA
ESL[1]	byte	Expanded string length	NAM$B_ESL
ESS	byte	Expanded string area size	NAM$B_ESS
FID[1]	3 words	File identification	NAM$W_FID
FNB[1]	longword	File name status bits	NAM$L_FNB
RLF	longword	Related file NAM block address	NAM$L_RLF
RSA	longword	Resultant string area address	NAM$L_RSA
RSL[1]	byte	Resultant string length	NAM$B_RSL
RSS	byte	Resultant string area size	NAM$B_RSS
WCC[1]	longword	Wildcard context	NAM$L_WCC

[1] Indicates nonuser-initialized field

$NAM

7.2 NAM BLOCK ALLOCATION

The $NAM macro instruction allocates and initializes storage for a NAM block. You cannot use this macro instruction within a sequence of executable instructions.

FORMAT

OPERATION	PARAMETERS
label: $NAM	ESA=address
	ESS=size
	RLF=nam-address
	RSA=address
	RSS=size

label: $NAM

7.2.1 Label

The label for the $NAM macro instruction assigns a name for a particular NAM block, and thus provides a symbolic address to be stored in the name block address field of the FAB.

For example, if a label of NMBLK is used for a NAM block, the syntax is:

```
$FAB    MRS=512,MRN=1000,NAM=NMBLK,ORG=REL
```

A label must be separated from the $NAM macro name by a colon (:).

$NAM ESA

7.2.2 Expanded String Area Address

The ESA parameter initializes the expanded string area address field of the NAM block, which contains the symbolic address of a user-allocated buffer. This buffer receives the file specification string resulting from the translation of logical names and the application of default file specification information to the original file string (file specification string of the FAB). The default file specification information consists of the default file specification string of the FAB, the related file resultant specification string, and the process defaults.

You must specify this field for wildcard processing.

FORMAT

```
ESA=address
```

address
The symbolic address of a buffer in your program to receive the expanded file specification string.

For example, if the buffer in your program has a symbolic address of NAMBUF, the syntax is:

```
$NAM ESA=NAMBUF,ESS=32
```

The symbolic offset for this field is:

```
NAM$L_ESA
```

$NAM ESS

7.2.3 Expanded String Area Size

The ESS parameter initializes the expanded string area size field. This field contains the size of the user-allocated buffer whose address is stored in the expanded string area address field (see Section 7.2.2).

FORMAT

```
ESS=size
```

size
A numeric value representing the size, in bytes, of the user buffer that contains the file specification string, in the range of 0 to 255.

For example, if the user buffer is 32 bytes long, the syntax is:

```
$NAM ESS=32,ESA=NAMBUF
```

The offset for this field is:

```
NAM$B_ESS
```

The symbolic value NAM$C_MAXRSS defines the maximum possible length of an expanded file specification string.

$NAM RLF

7.2.4 Related File NAM Block Address

The RLF parameter sets the related file NAM block address field to indicate the address of the NAM block for the related file. This field supports the secondary file concept of the command language (DCL), giving an extra default level in processing file specifications. See Chapter 8 for a description of file specification string parsing.

FORMAT

RLF=nam-address

nam-address
The symbolic address of the NAM block for the related file.

For example, if the $NAM macro instruction for the related file NAM block has the label INNAM, the syntax is:

```
$NAM    RLF=INNAM
```

The offset for this field is:

NAM$L_RLF

$NAM RSA

7.2.5 Resultant String Area Address

The RSA parameter sets the resultant string area address field. This field contains the address of a user-allocated buffer that will receive a copy of the resultant file specification string. This string results from the resolution of all system defaults, including version numbers and wildcard substitutions. You must specify this field for wildcard processing or when you select the SPL (spool) or SCF (submit) options in the FAB.

FORMAT

RSA=address

address
The symbolic address of a buffer in your program that will receive the resultant file specification string.

For example, if the buffer has a label of STRING defining its starting address, the syntax is:

```
$NAM RSA=STRING,RSS=48
```

The offset for this field is:

NAM$L_RSA

$NAM RSS

7.2.6 Resultant String Area Size

The RSS parameter sets the resultant string area size field. This field defines the length of the user-allocated buffer whose address is contained in the resultant string area address field (see Section 7.2.5).

FORMAT

RSS=size

size
A numeric value representing the size, in bytes, of the buffer that will receive the copy of the file specification string, in the range of 0 to 255.

For example, if the label STRING defines the starting address of a buffer 48 bytes long, the syntax is:

```
$NAM  RSA=STRING,RSS=48
```

The offset for this field is:

```
NAM$B_RSS
```

NOTE

The symbolic value NAM$C_MAXRSS defines the maximum possible length of a resultant file specification string.

7.3 NONINITIALIZABLE NAM BLOCK FIELDS

The following list describes the NAM block fields that you cannot initialize at assembly time. Either they are static or VAX-11 RMS sets them for you.

BID
Block identifier field; identifies the block as a NAM block to VAX-11 RMS. The $NAM macro instruction sets this field to the symbolic value NAM$C_BID; you cannot alter this field.

BLN
Block length field; defines the length of the NAM block, in bytes. The $NAM macro instruction sets this field to the symbolic value NAM$C_BLN; you cannot alter this field.

DID
Directory identification field; identifies the directory for the file. VAX-11 RMS outputs this field as part of the $OPEN, $CREATE, and $PARSE macro instructions. If, once you open the file, you want to refer to this directory again, you can do so more quickly by specifying that the NAM block has a valid directory identifier (see Chapter 8).

The offset to this 3-word field is NAM$W_DID.

DVI
Device identification field; defines the device for the file. VAX-11 RMS outputs this field as part of the $OPEN, $CREATE, and $PARSE macro instructions. You can use this field with the file identification field to reopen the file by referring to the NAM block (see Chapter 8).

The offset to this field is NAM$T_DVI. The symbolic value NAM$C_DVI gives the length of this field, in bytes.

ESL
Expanded string length field; VAX-11 RMS sets this field as part of the $OPEN, $CREATE, and $PARSE macro instructions. This field is set to the length, in bytes, of the file specification string returned in the buffer whose address is in the expanded string area address field (see Section 7.2.2).

The offset to this field is NAM$B_ESL.

FID

File identification field; provides the identifier of the file. VAX-11 RMS sets this field on a normal open or create operation. You can also set this field before opening the file if you are going to open by file identifier (see Chapter 8).

The offset to this 3-word field is NAM$W_FID.

FNB

File name status bits field; is set by VAX-11 RMS to indicate status information about the file as determined by the file specification parsing routine. Each bit within this field denotes a specific status relative to the various components of the file specification. The bits, and the conditions they express, are described in Table 7-2.

Each status bit has its own offset and mask value. The bit offset for each is formed by prefixing the bit name with NAM$V_. The mask value is formed by prefixing the bit name with NAM$M_.

The offset to this field is NAM$L_FNB.

RSL

Resultant string length field; VAX-11 RMS sets this field as part of the $OPEN, $SEARCH, and $CREATE macro instructions. This field is set to the length, in bytes, of the file specification string returned in the buffer whose address is in the resultant string area address field (see Section 7.2.5).

The offset to this field is NAM$B_RSL.

WCC

Wildcard context field; contains information required for using wildcards in place of the various file specification components. In particular, this field restarts a directory search to find the next matching file name, type, and/or version number.

The offset to this field is NAM$L_WCC.

Table 7-2
File Name Status Bits

Bit Names	Description
DIR_LVLS	Number of sub-directory levels (value is 0 if there is a user file directory only); 3-bit field
EXP_DEV	Device type was explicit
EXP_DIR	Directory specification was explicit
EXP_NAME	File name was explicit
EXP_TYPE	File type was explicit
EXP_VER	Version number was explicit
GRP_MBR	Directory specification is of the group/member number format
HIGHVER	A higher-numbered version (or versions) of the file exists (output from create and enter)
LOWVER	A lower-numbered version (or versions) of the file exists (output from create and enter)
NODE	File specification includes a node name
PPF	File is indirectly accessed process permanent file
QUOTED	File specification includes a quoted string
WILDCARD	File specification string included a wildcard; (this value is returned whenever any of the other wildcard bits are set)
WILD_DIR	Directory specification includes a wildcard
WILD_GRP	Group number contains a wildcard
WILD_MBR	Member number contains a wildcard
WILD_NAME	File name contained a wildcard
WILD_SFD1 through WILD_SFD7	Sub-file directory 1 through 7 specification includes a wildcard
WILD_TYPE	File type contained a wildcard
WILD_UFD	User file directory specification includes a wildcard
WILD_VER	Version number contained a wildcard

CHAPTER 8

RUN-TIME PROCESSING INTERFACE

This chapter describes the interface that VAX-11 RMS uses to access and manipulate files and records within files.

As outlined in Chapter 3, the run-time macro instructions work with the various control blocks to form the record management environment. The file-processing macro instructions deal with the file access block (FAB), and the record-processing macro instructions deal with the record access block (RAB).

The sections that follow discuss the run-time processing interface:

- VAX-11 RMS calling sequence and macro instruction general format
- The path to a file
- Control block usage
- Completion status codes

8.1 THE VAX-11 RMS CALLING SEQUENCE

VAX-11 RMS uses the standard VAX-11/780 calling sequence and conventions, and preserves all general registers across a call, with the exception of R0 and R1. When the routine completes execution, it returns control to the calling program, passing a return status code in R0. You should analyze the return code to determine the success or failure of the routine, and to alter the flow of execution, if necessary.

When you call a VAX-11 RMS routine, you must provide an argument list to define the associated control block (FAB or RAB) and, optionally, any completion routines. The argument list is from two to four longwords in length, as shown in Figure 8-1. (The rename service, however, uses a 5-longword argument list; see Section 13.4.)

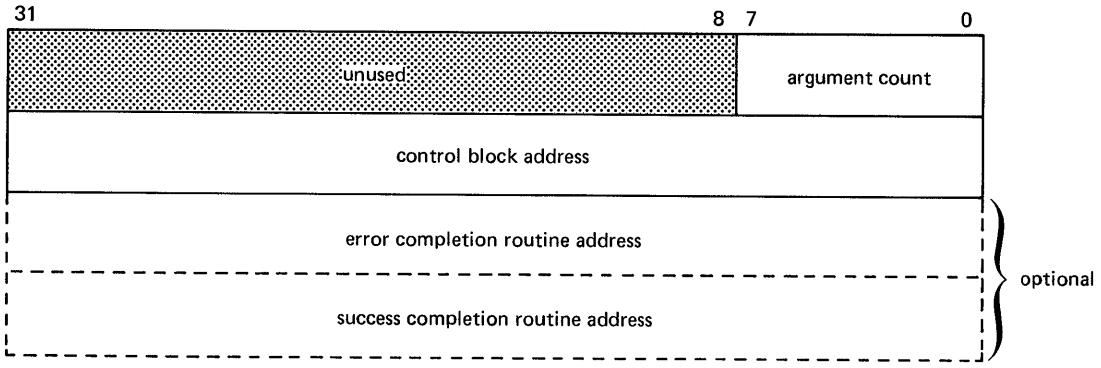


Figure 8-1 Argument List Format

VAX-11 RMS interprets the fields in the argument list as follows:

- Argument count -- contains a binary value, from 1 to 3, representing the number of arguments in the argument list
- Control block address -- contains the address of either the FAB (for file operations) or the RAB (for record operations)
- Error completion routine address -- contains the address of a user-written completion routine to be called if the requested operation fails
- Success completion routine address -- contains the address of a user-writen completion routine to be called if the requested operation completes successfully

The run-time macro instructions use two generalized formats, as follows:

```
1   label: macro-name

                          FAB=fab-address
2   label: macro-name                          ERR=entry  SUC=entry
                          RAB=rab-address
```

Chapters 9 through 13, which deal with the specific macro instructions, provide the exact format for each individual run-time processing macro instruction, with a capsule explanation of the parameters.

The remainder of this section provides an overview of the parameters, and lists the conventions that are followed during calls on success or error completion routines.

The first format above takes no parameters. You supply the argument list within your program, and the argument pointer register (AP) is assumed to contain the address of the argument list.

In the second format, you supply parameters that automatically generate an argument list on the stack according to the values you supplied. You specify these parameters through keywords, which can be in any order. You must separate each keyword by a comma, a blank space, or tabs. The only parameter required when using the second format is the control block address (FAB=fab-address or

RAB=rab-address). This parameter must be either a general register (R0 through R11) containing the control block address, or a suitable address for a PUSHAL instruction. If you omit this parameter, no other parameters are allowed; that is, you must use the first format.

The ERR=entry and SUC=entry parameters are optional and, if used, provide the addresses of completion routine entry points. VAX-11 RMS places the values you supply into the argument list on the stack during execution of the expanded macro instruction. These values must be addresses that can be used by a PUSHAL instruction.

When the argument list contains a completion routine argument, the following conventions are used:

- An asynchronous system trap (AST) is queued for the routine when the specified condition (error or success) occurs.
- General registers R0 through R11 are undefined. The argument pointer register (AP) contains the address of the AST argument list (see the VAX/VMS System Services Reference Manual); the AST parameter value in the AST argument list specifies the address of the associated control block (FAB or RAB). The status must be retrieved from the completion status code field of the associated control block.
- You can modify any general registers saved by an entry mask, in addition to R0 and R1.
- You can issue additional macro instructions for VAX-11 RMS routines within the completion routines.
- To exit from a completion routine, you must perform any necessary clean-up operations and execute a RET instruction.

8.2 THE PATH TO A FILE

Before you can perform operations on a file, you must provide input to the $OPEN, $CREATE, $PARSE, and $ERASE macro instructions to establish a path to the file. You do this by setting the file specification string address and size fields (and possibly the default file specification string address and size fields) of the FAB to describe an ASCII string within the program. In this ASCII string, you can have a concatenation of the network node name; a logical or device name; the directory name; and the file name, type, and version number. The following sections describe the processes that resolve all logical names to provide the required file specification components.

8.2.1 Interpretation of the File Specification

To establish a path to a file, VAX-11 RMS first calls an internal file specification parse routine. The parse routine forms a fully qualified file specification. If the NAM block specifies an expanded string buffer (see Section 7.2.2), this specification is returned to the user program as the expanded file specification string.

In forming a fully qualified file specification, VAX-11 RMS goes through the following steps:

1. If you specify an open by NAM block (see Section 8.2.3), VAX-11 RMS checks the NAM block fields for a fully qualified file specification.

2. If you do not specify an open by NAM block, or if the open fails to provide all the components of a fully qualified file specification, VAX 11 RMS processes the string specified by the file specification string address and string size fields (FNA and FNS) of the FAB. This string may have one of three different forms, which are treated as follows:

   a. If the file specification string has the form

      node-spec::"quoted-string"

      VAX-11 RMS copies the file specification string without modification to the expanded file specification string, and starts network processing to locate the file.

   b. If the file specification contains only a file name, VAX-11 RMS attempts to translate the string as a logical name. If this attempt succeeds, the equivalence string replaces the original file specification string, and the parse routine restarts. If the attempt fails, the file specification string is taken as the file name and default processing begins (see Section 8.2.2).

If the file name string is neither of the two forms (a and b) discussed above, processing proceeds as follows:

- VAX-11 RMS isolates the various components of the file specification, checks them for correct syntax, and copies them to the expanded file specification string. If the file specification does not include a device name component, default processing begins. If there is a device name component, the VAX-11 RMS parse routine attempts to translate it as a logical name. If a node name has been seen, only user-entered logical names are considered for translation. If the translation attempt fails, the component is treated as a device name.

  However, if the translation attempt succeeds, the equivalence string is checked to determine whether it refers to a process permanent file (see the VAX/VMS Command Language User's Guide). If the equivalence string does not refer to a process permanent file, the parse routine restarts, using the equivalence string as its input. If, however, the equivalence string indicates that this is an indirect reference to a process permanent file, the indicated file is therefore the target file resulting from the parse routine, and the logical name is copied to the expanded file specification string.

8.2.2 File Specification Default Application

If the file specification contains any missing components after VAX-11 RMS completely parses the primary file specification string (specified by the file specification string address and string size fields of the FAB), defaults are applied until either:

1. No more components are missing in the specification, or
2. No more defaults can be applied.

When VAX-11 RMS applies defaults, program defaults are applied first, in the following order:

1. The default file specification string specified by the contents of the default file specification string address and string size fields (DNA and DNS) of the FAB can supply any of the components necessary to form a full file specification. VAX-11 RMS parses and copies the default file specification string components in the same manner that it does for the primary file specification string (see Section 8.2.1). However, a duplicate field will not cause an error, because VAX-11 RMS ignores any attempt to fill a field that is already occupied.
2. If a NAM block is specified in the FAB, and if a related file NAM block has been specified in the related file field of the NAM block, defaulting can occur as follows. If the related file NAM block has a resultant file specification string, components of the related resultant file specification can be used depending on the state of the OFP bit in the file options field of the FAB. If the OFP bit is set, VAX-11 RMS parses the output file specification, and only the file name and file type components can be defaulted from the related file (also the file version if the output file version is an explicit wildcard). If the OFP bit is clear (indicating an input file parse), all file specification components, except the file version, are defaulted from the related resultant file specification string.

After program defaults, unless a node specification has been seen, system defaults apply in the following order:

1. If the device name component of the expanded file specification is missing, VAX-11 RMS translates the logical name SYS$DISK and parses the equivalence string; any expanding components are merged into the expanded file specification string. If the translation yields duplicate components, an error occurs. If the equivalence string includes a logical name, recursion may occur. This step must generate a device name; otherwise, an error occurs.
2. If the directory specification is missing from the expanded file specification, VAX-11 RMS uses the current default directory string from the process I/O control page.

After VAX-11 RMS applies the program and system defaults, the expanded name string is complete.

Chapter 13 describes the VAX-11 RMS file specification processing macro instructions. Among the services provided by these macro instructions, the parse service lets you call to parse a file specification string independent of the services of the $OPEN, $CREATE, or $ERASE macro instructions.

8.2.3 Opening and Creating a File by Name Block

When VAX-11 RMS successfully opens a file, the device identification, file identification, and directory identification fields of the NAM block (if present) are filled with the values pertaining to that file.

If you want to reopen the file after it is closed, you can specify the filled-in NAM block by setting the name block address field of the FAB to indicate the address of the NAM block, and setting the NAM option of the file-processing options field (FOP) of the FAB.

If the device identification and file identification fields are nonzero, the file is a fully qualified file specification. However, if the device identification field is nonzero, but the file identification field is 0, VAX-11 RMS uses the normal file specification string parsing routine to supply any missing portions of the full file specification. In this case, the directory specification may come from a nonzero directory identification field of the NAM block. If either the file identification or directory identification field is used, the directory and/or file name, type, and version number of the expanded and resultant file specification strings may be null.

You can create a file the same way that you open a file (above), except that VAX-11 RMS does not use the file identification field as input.

8.3 CONTROL BLOCK USAGE

The control block fields accessed by any run-time macro instruction provide VAX-11 RMS with the means to define or qualify the file and record operations. Depending on the operation, VAX-11 RMS uses one or more of these control blocks with one or more fields being used as input or output to or from the operation. In the chapters that follow, a list of each field being used is provided in the explanation of each macro instruction. Although not individually listed, the block identification (BID) and block length (BLN) fields of every control block used are always inputs to every VAX-11 RMS service.

Before your program calls for the execution of the macro instruction, you must ensure that all the appropriate control block fields used as input contain the necessary values. There are three methods of setting the values in the control block fields:

1. Explicit assembly time initialization
2. Implicit assembly time initialization
3. Run-time initialization

At assembly time, you explicitly initialize the fields by the use of parameters in the macro instruction for the particular control block (Chapters 4 through 7). You can initialize a field implicitly if VAX-11 RMS has defined a default value for the field. In this case, no action is required on your part. You simply allow the assembly time expansion of the control block allocation macro instruction to set the default value in the field.

At run time, you can initialize or alter the contents of a control block through the use of the various control block $xxx_STORE macro instructions or directly through instructions that use the defined symbolic offsets associated with the fields (these methods do not

provide defaults at run time). If you do not appropriately set a field that is defined as an input field to a particular operation, the operation may fail. VAX-11 RMS assumes that every value found in an input field was placed there for use by the current operation.

8.4 COMPLETION STATUS CODES

Before returning to your program from a file or record operation, VAX-11 RMS indicates the success or failure of the operation by setting a value in the completion status code field (STS) of the associated control block (FAB or RAB).

When first returning to your program after a call to an operation, VAX-11 RMS also sets general register 0 to the value in the status code field. In the case of asynchronous operations, register 0 may simply indicate that the operation is under way.

In the chapters that follow, the discussion of each run-time macro instruction includes a list of the possible nonsevere error and success status codes that you can receive. See Appendix A for a complete list of all VAX-11 RMS status codes.

In general, you may receive one of many error or success codes from an operation. You should thus test for success by checking only the low-order bit of the status code for a true condition (bit is set). The low-order three bits returned in the status code, when taken together, indicate the severity of the code. The severity codes are:

001 (1) -- Success (low-order bit set)

000 (0) -- Warning; indicates a nonstandard condition. The operation may have performed some, but not all, of the requested function.

010 (2) -- Error; you must recognize that a problem exists and provide a contingency plan in your program for such a condition.

100 (4) -- Severe error; normally caused by program logic or other unrecoverable condition.

Certain error status codes result in a value being set in the status value field (STV) of the control block. The description of the codes in Appendix A indicate the instances when the status value field contains such information.

Note that VAX-11 RMS services are considered system services for the purpose of generating system service exceptions on errors (see the VAX/VMS System Services Reference Manual). If you test for error conditions in your program, you should be sure to disable any unwanted system service exception generation.

8.5 PROCESS PERMANENT FILES

A process permanent file is one that is opened (or created) through VAX-11 RMS by supervisor or executive mode code having the PPF bit set in the file processing options (FOP) field of the FAB. This causes the VAX-11 RMS-maintained internal data structures to be allocated in an area of memory in the process control region that remains allocated for the life of the process. Thus, process permanent files can remain

open across image activations. You cannot directly access process permanent files by user mode code; you can, however, indirectly access them. VAX-11 RMS provides a subset of the total available operations to the indirect accessor.

Indirect accessors gain access to process permanent files through the logical name mechanism, as follows:

1. The LOGIN command image, or at a later point the command interpreter, opens or creates a file corresponding to the process' input, output, and error message streams. A logical name is created in the process logical name table for SYS$INPUT, SYS$OUTPUT, and SYS$ERROR, respectively. The equivalence string for the logical name has a special format that indicates the correspondence between the logical name and the related process permanent file. For example, for an interactive user, a single process permanent file is opened for the terminal and all three logical names refer to the one file.

2. When an indirect accessor opens or creates a file specifying a logical name that has one of these special equivalence strings, VAX-11 RMS recognizes this and therefore does not open or create a new file; instead, the returned value for the internal file identifier (and later the value for the internal stream identifier from a connect service) is set to indicate that access to the associated process permanent file is with the indirect subset of allowable functions.

Some of the implications for the indirect accessor are:

- A create service for a process permanent file becomes an open service; the fields of the FAB are output according to the description of the open, not the create.

- The open or create service requires no I/O operations.

- Any number of indirect opens and creates are allowed.

- There is only one position context for the file; that is each sequence of the open/create service accesses the same record stream, not an independent stream.

- If the process permanent file was initially opened with the SQO bit set in the file-processing options field, neither random access nor the rewind service is permitted. This is the case for SYS$INPUT, SYS$OUTPUT, and SYS$ERROR.

- Certain options to various services produce errors. For example, you cannot set the NFS, PPF, and UFO bits of the file processing options field for the open and create services. Other options are ignored, such as the SPL, SCF, and DLT bits of the file processing options field for the close service, the ASY bit of the record-processing options field, and both the multiblock count and multibuffer count fields.

- If a NAM block is used and either an expanded or resultant file specification string is returned, it consists solely of the process logical name followed by a colon, such as SYS$INPUT:

- The file access field is ignored on an open service; instead, operations are checked against the file access field specified for the original open or create service.

- Information from the record attributes field is saved on each open service (and subsequent connect service) in the value returned in the internal file identifier (and internal stream identifier) field. If the output file is a print file (variable with fixed control record format and the PRN bit is set in the record attributes field), mapping is performed for each put service from the user-specified carriage control to the print file carriage control format. Thus, different carriage control types from different indirect open services all work correctly.

- You cannot use the erase service.

- Checking is performed for $DECK, $EOD, and other dollar sign ($) records on the SYS$INPUT stream (see the VAX/VMS Command Language User's Guide).

- At image exit time the VAX-11 RMS Rundown control routine insures that the indirect I/O on process permanent files terminates; the process permanent files are not closed.

- You can use only sequential files in this manner.

CHAPTER 9

FILE-PROCESSING MACRO INSTRUCTIONS

VAX-11 RMS provides file-processing macro instructions that you can insert into your programs. At run time, the expanded code of these macro instructions causes calls to be made to corresponding VAX-11 RMS services.

The file-processing macro instructions cause VAX-11 RMS to perform some operation related to the file as a whole. These macro instructions, therefore, deal with fields in the file access block (FAB). See Chapter 4 for a description of the effect of these fields.

In most cases, you use a file-processing macro instruction with parameters to indicate the symbolic address of the FAB and the address of any optional error or success completion routine you may have provided. You can also use the macro instruction without parameters, but you must then create an argument list in your program to define the values for these addresses (see Section 8.1).

Table 3-2 summarizes all the run-time processing macro instructions. This chapter deals only with the following macro instructions, which pertain to file processing:

- $CREATE
- $OPEN
- $DISPLAY
- $EXTEND
- $CLOSE
- $ERASE

To facilitate your reference, the macro instructions are presented in alphabetical order.

$CLOSE

9.1 TERMINATING FILE PROCESSING

The $CLOSE macro instruction invokes the close service, which terminates file processing and closes the file.

You can issue a $CLOSE macro instruction only when no operation is under way for the file, that is, when all record access blocks (RABs) associated with the file are inactive. Otherwise, the file will not be closed nor will the internal file-identifier field be set to 0. When the close service operates normally, VAX-11 RMS disconnects all RABs for you, performs the various clean-up procedures (including file option and XAB processing), and closes the file. The only types of XABs that the close service processes are the file protection and revision date and time, and then only if the file is write-accessed.

FORMAT

OPERATION	PARAMETERS
label: $CLOSE	FAB=fab-address
	ERR=entry
	SUC=entry

label
An optional, user-defined symbolic address for the $CLOSE macro instruction.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-1 lists the FAB fields that VAX-11 RMS uses as input and output for the close service.

Table 9-1
Close FAB Fields

Usage	Field Name	Description
Input	FOP	File-processing options (DLT, NAM, RWC, SCF, SPL, and TEF only)
	IFI	Internal file identifier
	NAM	Name block address (used only if NAM is set in file-processing options)
	XAB	Extended attribute block address
Output	IFI	Internal file identifier (zeroed)
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A (along with the nonsevere). However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the close service are listed below. Note that even though a failure may be indicated by the completion status code value, if the internal file-identifier value was zeroed by VAX-11 RMS, the file was indeed closed nonetheless.

Success:

RMS$_NORMAL Operation successful

RMS$_OK_NOP Key XAB not filled in when file opened for flock I/O.

Failure:

RMS$_ACT File activity precludes operation

RMS$_DAC File deaccess error

RMS$_DNR Device not ready

RMS$_EXP Expiration date not yet reached

RMS$_MKD Files-11 ACP could not mark file for deletion

RMS$_PRV Privilege violation; access denied

RMS$_WLK Device write-locked

$CREATE

9.2 CREATING A FILE

The $CREATE macro instruction invokes the create service, which constructs a new file according to the attributes you specify in the FAB. If any extended attribute blocks (XABs) are chained to the FAB, then the qualities described in the XABs are applied to the file. If an allocation control XAB is present, its allocation quantity (ALQ), allocation options (AOP -- only for the CTG and CBT bits), bucket size (BKZ), and default extension quantity (DEQ) fields are used instead of the corresponding fields of the FAB. When either key definition or allocation XABs are present, they must be grouped in ascending order (by REF or AID, respectively) but they need not be dense. No other types of XABs may intervene. If a name block (NAM) is also connected to the FAB, VAX-11 RMS fills in its fields with information about the created file.

The $CREATE macro instruction leaves the file opened. Therefore, you must close the file when processing is completed, even if no record operations were performed.

The create service implies PUT access; that is, you need not specify PUT in the file access field of the FAB.

FORMAT

OPERATION	PARAMETERS
label: $CREATE	FAB=fab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $CREATE macro instruction.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-2 lists the FAB fields that VAX-11 RMS uses as input and output for the create service. See Chapter 4 for information regarding the contents of the various input and output fields.

Table 9-2
Create FAB Fields

Usage	Field Name	Description
Input	ALQ	Allocation quantity. This field is ignored if an allocation XAB is present.
	BKS	Bucket size. This field is ignored if an allocation XAB is present.
	BLS	Block size (sequential organization only)
	DEQ	Default file extension quantity. This field is ignored if an allocation XAB is present.
	DNA	Default file specification string address
	DNS	Default file specification string size
	FAC	File access
	FNA	File specification string address
	FNS	File specification string size
	FOP	File-processing options
	FSZ	Fixed control area size
	IFI	Internal file identifier (must be 0)
	MRN	Maximum record number (relative organization only)
	MRS	Maximum record size
	NAM	Name block address
	ORG	File organization
	RAT	Record attributes
	RFM	Record format
	RTV	Retrieval window size
	SHR	File sharing
	XAB	Extended attribute block address
Output	ALQ	Allocation quantity (contains actual number of blocks allocated)
	BLS	Block size (sequential organization only)
	DEV	Device characteristics
	IFI	Internal file identifier
	SDC	Spooling device characteristics
	STS	Completion status code (also returned in Register 0)
	STV	Status value (contains the I/O channel number if the operation is successful)

Table 9-3 lists the NAM block fields that VAX-11 RMS uses as input and output for the create service if the name block address field is specified in the FAB. See Chapter 7 for information regarding the contents of the various input and output fields of the NAM block.

Table 9-3
Create NAM Block Fields

Usage	Field Name	Description
Input	DID	Directory identification (input only if NAM bit is set in the file processing options (FOP) field of FAB)
	DVI	Device identification (input only if NAM bit is set in the FOP field of the FAB)
	ESA	Expanded string area address
	ESS	Expanded string area size
	RLF	Related file NAM block address (if nonzero, RSA and RSL are input from related file NAM block)
	RSA	Resultant string area address
	RSS	Resultant string area size
Output	DID	Directory identification
	DVI	Device identification
	ESL	Expanded string length (if, on input, both the ESA and ESS are nonzero, and if the NAM bit of the FOP field of the FAB is clear or DID is 0, the expanded file specification string is copied to the buffer specified by the input ESA field)
	FID	File identification
	FNB	File name status bits (FNB is output only if NAM bit in FOP field of FAB is clear, or if DID field was 0 on input)
	RSL	Resultant string length (if RSA and RSS are both nonzero on input, the resultant file specification is copied to the buffer specified by RSA)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the create service are listed below. If a failure is indicated, the file may indeed have been created, but will not be opened for processing, depending on the nature of the failure.

Success:

RMS$_CREATED	File was created, not opened. This status is returned when the CIF option is used and the file must be created. If the file is opened, RMS$_NORMAL is returned.
RMS$_NORMAL	Operation successful
RMS$_SUPERSEDE	Created file supersedes an existing file

Failure:

RMS$_ACT	File activity precludes operation
RMS$_CRE	File create error
RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_EXP	Expiration date not yet reached
RMS$_FEX	File already exists
RMS$_FLK	File locked; not available
RMS$_PRV	Privilege violation; access denied
RMS$_WLK	Device write-locked

$DISPLAY

9.3 OBTAINING ATTRIBUTES OF A FILE

The $DISPLAY macro instruction invokes the display service, which retrieves file attribute information about a file and places this information in fields in the XABs chained to the FAB. VAX-11 RMS determines the type of file attribute information needed by the type of XABs present.

FORMAT

OPERATION	PARAMETERS
label: $DISPLAY	FAB=fab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $DISPLAY macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-4 lists the FAB fields that VAX-11 RMS uses as input and output for the display service.

Table 9-4
Display FAB Fields

Usage	Field Name	Description
Input	IFI	Internal file identifier
	XAB	Extended attribute block address
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value; contains the address of the XAB that caused error.

VAX-11 RMS places the attribute values in the corresponding fields of the appropriate XAB.

Note that the open service performs an implicit display service (see Section 9.6).

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the display service are listed below.

Success:

RMS$_NORMAL Operation successful

RMS$_OK_NOP Key XAB not filled in when file opened for block I/O.

Failure:

RMS$_ACT File activity precludes operation

RMS$_DNR Device not ready

RMS$_PRV Privilege violation; access denied

$ERASE

9.4 DELETING A FILE

The $ERASE macro instruction invokes the erase service, which deletes a VAX-11 RMS disk file and removes the file's directory entry as specified in the path to the file (see Section 8.2). You must use the $REMOVE macro instruction to delete additional directory entries, if any (see Chapter 13).

Deleting a file releases the file's allocated space for use by another file; the deletion does not physically remove the data (as does overwriting or zeroing). Only files that are closed can be deleted; an open file cannot be deleted with the erase service, but may be deleted by the $CLOSE macro instruction by setting the DLT bit in the file processing options field of the FAB. Furthermore, you cannot delete files from magnetic tape volumes.

FORMAT

OPERATION	PARAMETERS
label: $ERASE	FAB=fab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $ERASE macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-5 lists the FAB fields that VAX-11 RMS uses as input and output for the erase service.

Table 9-5
Erase FAB Fields

Usage	Field Name	Description
Input	DNA	Default file specification string address
	DNS	Default file specification string size
	FNA	File specification string address
	FNS	File specification string size
	FOP	File-processing options (NAM bit only)
	IFI	Internal file identifier (must be 0)
	NAM	Name block address
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

Table 9-6 lists the NAM block fields that VAX-11 RMS uses as input and output for the erase service if the name block address field is specified in the FAB.

Table 9-6
Erase NAM Block Fields

Usage	Field Name	Description
Input	DID	Directory identification (input only if NAM bit is set in the file processing options (FOP) field of FAB)
	DVI	Device identification (input only if NAM bit is set in the FOP field of the FAB)
	ESA	Expanded string area address
	ESS	Expanded string area size
	FID	File identification (input only if NAM bit is set in the FOP field of the FAB)
	RLF	Related file NAM block address (if nonzero, RSA and RSL are from related file NAM block)
	RSA	Resultant string area address
	RSS	Resultant string area size
Output	DID	Directory identification
	DVI	Device identification
	ESL	Expanded string length (if, on input, both the ESA and ESS are nonzero, and if the NAM bit of the FOP field of the FAB is clear or DID is 0, the expanded file specification string is copied to the buffer specified by the input ESA field)
	FNB	File name status bits
	RSL	Resultant string length (if RSA and RSS are both nonzero on input, the resultant file specification is copied to the buffer specified by RSA)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the erase service are listed below.

Success:

RMS$_NORMAL Operation successful

Failure:

RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_MKD	Files-11 ACP could not mark file for deletion
RMS$_PRV	Privilege violation; access denied
RMS$_WLK	Device write-locked

$EXTEND

9.5 EXTENDING A FILE'S ALLOCATED SPACE

The $EXTEND macro instruction invokes the extend service, which increases the amount of space allocated to a VAX-11 RMS disk file. You can only extend open files; otherwise, an error occurs.

The allocation quantity field of the FAB (or the allocation XAB, if used) must contain the number of blocks that VAX-11 RMS is to add to the file. Furthermore, you can indicate other attributes regarding the manner and location for allocation. For example, you can indicate that the additional blocks must be allocated contiguously. If you do, however, and not enough contiguous space is available, the operation will fail. (This extension does not have to occur contiguous to the initial file space.)

If an allocation control XAB is present, its allocation quantity (ALQ) and allocation options (AOP -- the CBT and CTG bits only) fields are used instead of the corresponding fields in the FAB. The allocation quantity field of the XAB is set to the actual extension size.

FORMAT

OPERATION	PARAMETERS
label: $EXTEND	FAB=fab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $EXTEND macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-7 lists the FAB fields that VAX-11 RMS uses as input and output for the extend service.

Table 9-7
Extend FAB Fields

Usage	Field Name	Description
Input	ALQ	Allocation quantity. This field is ignored if an allocation XAB is present.
	FOP	File-processing options. Checked to see if the CTG or CBT bit is set to indicate contiguous allocation; ignored if allocation XAB is present.
	IFI	Internal file identifier
	XAB	Extended attribute block address. Only the allocation type of XAB will be processed.
Output	ALQ	Allocation quantity (contains the actual extension allocation value if no allocation XAB is present)
	STS	Completion status code (also returned in Register 0)
	STV	Status value (contains the total of blocks allocated, totaled across all allocation XABs)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the extend service are listed below.

Success:

RMS$_NORMAL Operation successful

Failure:

RMS$_ACT File activity precludes operation

RMS$_DNR Device not ready

RMS$_EXT File extend error

RMS$_WLK Device write-locked

$OPEN

9.6 OPENING AN EXISTING FILE

The $OPEN macro instruction invokes the open service, which makes an existing file available for processing by your program. This macro instruction implements the type of access desired, and sets the degree to which the file can be shared. You must open a file before you perform any record operations. If any XABs are chained to the FAB, VAX-11 RMS places the attribute values in the fields of the appropriate XAB. If you specify a NAM block in the FAB, the contents of the device, directory, and file identification fields can be used to perform an open by NAM block (see Section 8.2.3). In addition, the various fields of this NAM block are filled in with auxiliary file specification information.

FORMAT

OPERATION	PARAMETERS
label: $OPEN	FAB=fab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $OPEN macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 9-8 lists the FAB fields used as input and output for the open service. See Chapter 4 for information regarding the contents of the various input and output fields.

Table 9-8
Open FAB Fields

Usage	Field Name	Description
Input	DEQ	Default file extension quantity. If a nonzero value is present in this field, it applies to this open of the file only.
	DNA	Default file specification string address
	DNS	Default file specification string size
	FAC	File access
	FNA	File specification string address
	FNS	File specification string size
	FOP	File-processing options
	FSZ	Fixed control area size; unit record devices only.
	IFI	Internal file identifier (must be 0)
	NAM	Name block address
	RAT	Record attributes; unit record devices only
	RFM	Record format; unit record devices only
	RTV	Retrieval window size
	SHR	File sharing
	XAB	Extended attribute block address
Output	ALQ	Allocation quantity; contains the highest numbered block allocated to the file.
	BKS	Bucket size; not used for sequential files
	BLS	Block size; for sequential files only
	DEQ	Default file extension quantity
	DEV	Device characteristics
	FOP	File-processing options; the bits CTG, CBT, RCK, and WCK are set or cleared individually according to the file attributes
	FSZ	Fixed control area size; only applies to variable with fixed length control records
	IFI	Internal file identifier
	MRN	Maximum record number; for relative files only
	MRS	Maximum record size
	ORG	File organization
	RAT	Record attributes
	RFM	Record format
	SDC	Spooling device characteristics
	STS	Completion status code (also returned in Register 0)
	STV	Status value (contains the I/O channel number if the operation is successful)

Table 9-9 lists the NAM block fields that VAX-11 RMS uses as input and output for the open service if the name block address field is specified in the FAB. See Chapter 7 for information regarding the contents of the fields used as input and output.

Table 9-9
Open NAM Block Fields

Usage	Field Name	Description
Input	DID	Directory identification (input only if NAM bit is set in the file processing options (FOP) field of FAB)
	DVI	Device identification (input only if NAM bit is set in the FOP field of the FAB)
	ESA	Expanded string area address
	ESS	Expanded string area size
	FID	File identification (input only if NAM bit set in FOP field of FAB)
	RLF	Related file NAM block address (if non-zero, RSA and RSL are from related file NAM block)
	RSA	Resultant string area address
	RSS	Resultant string area size
Output	DID	Directory identification
	DVI	Device identification
	ESL	Expanded string length (if, on input, both the ESA and ESS are nonzero, and if NAM bit of the FOP field of the FAB is clear or DID and FID are 0, the expanded file specification string is copied to the buffer specified by RSA)
	FID	File identification
	FNB	File name status bits
	RSL	Resultant string length (if RSA and RSS are both nonzero and if NAM bit is clear or FID is 0, the resultant file specification is copied to the buffer specified by RSA)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the open service are listed below.

Success:

RMS$_NORMAL Operation successful

Failure:

RMS$_ACC	File access error
RMS$_ACT	File activity precludes operation
RMS$_DEV	Bad device or in appropriate device type for operation
RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_FLK	File is locked and therefore unavailable
RMS$_FNF	No such file exists
RMS$_PRV	Privilege violation
RMS$_WLK	Device is write-locked
RMS$_OK_NOP	Key XAB not filled in when file opened for block I/O

CHAPTER 10

RECORD OPERATION PERFORMANCE

Some of the key concepts that you must understand in relation to record operations are:

- Record access
- Current record context
- Record streams
- Synchronous and asynchronous operations
- Record locking

Sections 10.1 through 10.5 discuss these concepts; then, Chapter 11 describes each record-processing macro instruction in detail.

10.1 RECORD ACCESS

To process a record, you must identify the record and specify the record access mode you are going to use. Once the record is identified, you have two different record transfer modes available to manipulate it. The following sections describe how you specify the record access mode and the transfer mode.

10.1.1 Specifying the Record Access Mode

The value that you set in the record access mode field of the RAB tells VAX-11 RMS what type of record access to use for the particular record operation. During program execution, you can switch the record access mode by changing the contents of this field. This is known as dynamic access.

VAX-11 RMS lets you set any one of the following three values:

1. SEQ -- this value indicates the sequential record access mode. When you use this record access mode, the access will be a function of the Next Record (see Section 10.2), and no additional record specification is necessary. This record access mode is valid for any file organization.

2. KEY -- this value indicates random access by key. This record access mode is used with relative files (and sequential files on disk with fixed-length records) to denote random access by relative record number and with indexed files to denote random access by key value. The key value

for the record to be found or retrieved is placed in the key buffer, which is described by the values set in the key buffer address and key size fields of the RAB. When accessing an indexed file, the particular key of reference (index to search on) must be specified in the KRF field of the RAB.

3. RFA -- this value indicates that access is random by the record's file address (RFA). This record access mode is limited to retrieval operations for disk files.

   To use this access mode, you must save the RFA that VAX-11 RMS returned from a previous operation. Then, before you initiate a new operation, you specify access by RFA mode in the record access mode field of the RAB, and restore the RFA. The RFA does not change when you close a file and later reopen it.

   The format of the RFA is known internally to VAX-11 RMS.

VAX-11 RMS examines the contents of the record access mode field of the RAB during the execution of a $GET, $FIND, or $PUT macro instruction. You need not specify a record access mode for operations with a $UPDATE, $DELETE, or $TRUNCATE macro instruction. However, you cannot request these operations until you have first accessed the target record with a $GET or $FIND macro instruction.

10.1.2 Specifying the Record Transfer Mode

The record-processing option field of the RAB lets you specify the record transfer mode. There are two record transfer modes -- locate and move -- which tell VAX-11 RMS how to access the target record for the get service ($GET macro instruction) once the record is in memory. You can switch the record transfer mode while your program is executing by changing the contents of the record-processing option field.

In the record-processing option field you indicate locate mode by setting the LOC bit. If you do not set this bit, VAX-11 RMS uses move mode, by default.

In locate mode, your program accesses records directly in an I/O buffer. Therefore, VAX-11 RMS normally does not need to move records between I/O buffers and a user program buffer. Also, VAX-11 RMS does not support locate mode for operations involving the $PUT or $UPDATE macro instructions. However, the $GET macro instruction supports locate mode operations on files of all organizations. Note that locate mode, even if specified, may not actually be used due to the occurrence of any of the following:

1. Records crossing block boundaries
2. The file access field of the FAB being set to UPD
3. Multiple record streams

In move mode, VAX-11 RMS transfers individual records between I/O buffers and your program buffer. For the $GET macro instruction, VAX-11 RMS reads a block (for sequential files) or a bucket (for relative and indexed files) into an I/O buffer. VAX-11 RMS then selects the desired record from the buffer and moves it to a program-specified location.

When writing records to the file ($PUT and $UPDATE), your program first builds a record in any desired program location, stores its address and size in the RAB, and calls the appropriate VAX-11 RMS routine as specified by the particular macro instruction. VAX-11 RMS moves the record from its specified location into an I/O buffer. Depending upon the file organization and options, the buffer may be written immediately or only when it is filled.

10.2 CURRENT RECORD CONTEXT

For each RAB connected to a file access block (FAB), VAX-11 RMS maintains current context information, identifying where each RAB is positioned at any point in time. VAX-11 RMS modifies the current context as your program performs record operations.

At any point in time, the current context is represented by, at most, two records:

1. The Current Record
2. The Next Record

The context of these two records is internal to VAX-11 RMS; you have no direct contact with them. However, an explanation of their purpose and importance can aid in your understanding of how VAX-11 RMS works.

10.2.1 Current Record

The Current Record represents the target record for an $UPDATE, $DELETE, and $TRUNCATE macro instructions. The Current Record also facilitates sequential processing on disk devices for a stream. VAX-11 RMS rejects any request to update, delete, or truncate that does not have a Current Record. In addition, an operation with a $GET macro instruction using sequential record access mode and immediately preceded by a $FIND macro instruction operates on the record specified by the Current Record. If the find service did not lock the record (for relative and indexed file organizations) and the current record has been deleted, the get service will access the next existing record.

When a RAB is first connected to a FAB, the Current Record is undefined. Furthermore, any unsuccessful record operation, or successful execution of a macro instruction other than $GET or $FIND, causes the Current Record to be undefined.

The Current Record is set to the RFA of the record upon which an operation is performed with a $GET or $FIND macro instruction. VAX-11 RMS also places this address in the record's file address field of the RAB. This means that:

1. After initialization, the Current Record always contains the record's file address of the most recent successful operation with a $GET or $FIND macro instruction (unless failure occurs or a macro instruction other than $GET or $FIND executes).
2. The record's file address field of the RAB, unless you modify it, always contains the address of the target record (if the operation fails, the record's file address is undefined).

Table 10-1 summarizes the effect that each successful record operation has on the context of the Current Record.

10.2.2 Next Record

VAX-11 RMS uses the Next Record for operations involving sequential record access mode. When the record access mode field of the RAB indicates sequential processing, the Next Record represents the target record for the next operation involving:

- The $FIND macro instruction
- The $PUT macro instruction
- The $GET macro instruction (if the immediately preceding operation was not a $FIND macro instruction); if the next record cell in a relative file organization does not contain a record, the target record is the next existing record.

This "look-ahead" ability significantly decreases access time for sequential processing. VAX-11 RMS uses its internal knowledge of file organization and structures to determine the Next Record as follows:

- Operations with the $CONNECT macro instruction initialize the Next Record to:
  - The first record or cell in a file of sequential or relative organization, respectively
  - The first record in the collating sequence of the specified key of reference in an indexed file.
  - The end of a sequential file on disk if the record processing options field of the RAB has the EOF option bit set.
  - The end of a write-accessed magnetic tape file unless the file processing options field of the FAB has the NEF bit set.
- Operations with the $GET macro instruction in any record access mode and the $FIND macro instruction in sequential record access mode cause the Next Record to indicate the next record or cell in the file.
- Operations with a $TRUNCATE macro instruction cause the Next Record to indicate the end of file. Therefore, you need only use $PUT macro instructions after truncation to extend the file. You can truncate only sequential files.
- Operations with the $FIND or $PUT macro instructions in random access mode have no effect on the Next Record.
- Operations with the $PUT macro instruction in sequential access mode initialize the Next Record to:
  - The end of file in a sequential file.
  - The next record or cell in a relative file.
- Operations with the $PUT macro instruction in sequential access mode in an indexed file cause the Next Record to be undefined.

- Operations with the $DELETE, $UPDATE, $FREE, or $RELEASE macro instructions in any record access mode have no effect on the Next Record.

- Operations with the $REWIND macro instruction in any record access mode cause the Next Record to indicate the first record or cell in the file.

- Any unsuccessful record operation has no effect on the Next Record.

Table 10-1 summarizes the effect that each successful record operation has on the Next Record.

Table 10-1
Record Access Stream Context

Record Operation	Record Access Mode	Current Record	Next Record
Connect	does not apply	none	first record
Connect with EOF bit set in record-processing options field	does not apply	none	end of file
Get last operation not a find	sequential	new	new Current Record+1
Get last operation was a find	sequential	unchanged	Current Record+1
Get	random	new	new Current Record+1
Put	sequential	none	1. sequential file—end of file 2. relative file—next record position 3. indexed file—undefined
Put	random	none	unchanged
Find	sequential	new	new Current Record+1
Find	random	new	unchanged
Update	does not apply	none	unchanged
Delete	does not apply	none	unchanged
Truncate	does not apply	none	end of file
Rewind	does not apply	none	first record
Free	does not apply	none	unchanged
Release	does not apply	none	unchanged

NOTES:

1. Except for the truncate operation, VAX-11 RMS establishes the Current Record before establishing the identity of the Next Record.
2. The notation "+1" indicates the next sequential record as determined by the file organization. For indexed files, the current key of reference is part of this determination.
3. The correct operation on an indexed file establishes the Next Record to be the first record in the index represented by the RAB key of reference (KRF) field.
4. The connect operation leaves the Next Record as the end of file for a magnetic tape file opened for put operations (unless the NEF bit is set).

10.3 RECORD STREAMS

Before you can process the records in a file, you must first establish a record stream to that file. A record stream is the logical association of a RAB with a FAB. Once you have established this association, you can issue requests for operations on the records in the file that the FAB represents.

For all but the sequential file organization, there can be any number of RABs associated with a single FAB, and each RAB represents an independent record stream. If you establish a single record stream, your program uses the stream to issue a sequence of record operations, which are executed serially. Therefore, you can process only one record at a time. However, when you establish multiple record streams for a file, you can process a record from each stream in parallel. Therefore, multiple record streams provide concurrently active sequences of record operations to the same file.

After you open a file by issuing a $OPEN (or $CREATE) macro instruction, you establish the record stream by placing the address of the FAB in the file access block field of the appropriate RAB or RABs. Then, you issue a $CONNECT macro instruction. Once you have completed the desired sequence of operations, you terminate the association by issuing a $DISCONNECT macro instruction.

Chapter 11 describes the $CONNECT and $DISCONNECT macro instructions.

10.4 SYNCHRONOUS AND ASYNCHRONOUS OPERATIONS

Within each record stream, VAX-11 RMS lets you perform operations either synchronously or asynchronously. In synchronous operations, VAX-11 RMS returns control to your program only after the record operation request is satisfied.

In asynchronous operations, VAX-11 RMS may return control to your program before the operation is satisfied. In this way, your program can use the time required to transfer data between the file and memory to perform other computations. Note that in asynchronous operations, the operation may complete before control is returned to your program. This is due to several factors. For example, the required record may already reside in an I/O buffer, or the operating system may schedule another program, thus possibly allowing a necessary operation to complete before the original program is rescheduled.

The following sections describe how you declare synchronous and asynchronous operations.

10.4.1 Synchronous Operations

To declare a synchronous operation, you must clear the ASY bit in the record-processing options field of the RAB. Since by default this bit is off at assembly time, you normally do not have to set it off unless you had set it on previously.

Normally, you would not use success and error routines with synchronous operations. Instead, you would test the completion status code for an error and change the program's flow accordingly. However, if you use these routines, they will be executed as asynchronous system traps (ASTs) before the in-line return to your program (unless ASTs are disabled).

10.4.2 Asynchronous Operations

To declare an asynchronous record operation, you must set the ASY bit in the record-processing options field of the RAB. You can switch between synchronous and asynchronous operations during processing of a record stream by setting or clearing the ASY bit on a per-operation basis.

You can specify completion routines to be executed as ASTs if success or error conditions occur. Within such routines, you can issue additional operations, but they too should be asynchronous. Otherwise, all other currently active asynchronous requests in your program cannot have their completion routines executed until the synchronous operation completes.

If an asynchronous operation is not yet complete at the time of return from a call to a VAX-11 RMS service, the completion status field of the RAB will be 0, and a success status code of RMS$_PENDING will be returned in Register 0. This status code indicates that the operation was initiated but is not yet complete. You must never modify the contents of a RAB when an operation is in progress.

If you issue a second record operation request for the same stream before a prior request is complete, you will receive an error status code of RMS$_RSA, indicating that the record stream is still active. This can also occur when an AST level routine attempts to use an active record stream; the original I/O request may be synchronous or asynchronous. In either case it is the caller's responsibility to recognize the possibility and prevent the problem by issuing a $WAIT macro instruction (see Chapter 11). If you are going to reuse the RAB of the original operation, the $WAIT macro instruction must occur before you attempt the new operation. If you use a different RAB, however, it is possible to simply issue the new operation and handle the error by waiting and retrying.

Upon completion of the operation, your program receives control at the point following the $WAIT macro instruction.

10.5 RECORD LOCKING

VAX-11 RMS provides a record locking capability for relative and indexed files. This capability affords control over operations when more than one stream or process is simultaneously accessing the file. Record locking makes certain that when a program is adding, deleting, or modifying a record on a given stream, another stream or process cannot access the same record.

VAX-11 RMS does not support record locking for sequential files. These files can be write-shared, however, as long as the user provides the necessary logic to handle the simultaneous reading and writing. This is specified by the setting of the UPI bit in combination with the other shared access bits in the file sharing field (SHR) of the FAB. The UPI bit, when set, indicates that one or more writers can access the file, but the user assumes the responsibility for any required interlocking.

Record locking occurs on a file accessed for some form of writing (FAC is set to either PUT, UPD, or DEL) only if the file sharing field (SHR) of the FAB is set to some form of writing or the MSE fit is set.

There are two types of record locking: automatic and manual. VAX-11 RMS handles automatic record locking transparently. You use it when you are dealing with a lock on a single record at a time. Manual record locking requires additional effort on your part. You use it when dealing with locks on multiple records at a time.

A record can be in any of three states: unlocked, automatically locked, or manually locked. When a record is initially locked, it is in either the manually or automatically locked state. It will remain in that state until the lock is released; that is, it cannot move directly from the automatically to manually locked state, or vice versa. Therefore, you make an initial decision based on your needs to use automatic or manual locking for a given record, and continue to deal with that record using the same type of locking until the lock is released.

The following sections describe the two types of record locking.

10.5.1 Automatic Record Locking

For automatic record locking, the lock occurs on every execution of a $FIND or $GET macro instruction (unless the NLK bit is set in the record-processing options field). The lock is released on the next operation on the stream; that is, the lock is released when the Next Record is accessed, the Current Record is updated or deleted, the record stream is disconnected, the file is closed, or an operation causing an error occurs. Therefore, the record is freed when you issue any of the following macro instructions:

- $FIND
- $GET
- $PUT
- $UPDATE
- $DELETE
- $REWIND
- $DISCONNECT
- $CLOSE
- $FREE
- $RELEASE

The $FREE and $RELEASE macro instructions let you explicitly unlock the record.

If you place a record in an empty cell in a relative file with a $PUT macro instruction, the cell is, in effect, locked by the put service. It is unlocked when the service completes.

One exception to the automatic unlocking exists: On a sequential get service following a find service that caused the record to be locked, the record remains automatically locked.

10.5.2 Manual Record Locking

For manual record locking, you have explicit control -- and therefore deadlock prevention responsibility -- over the unlocking of records. Thus, manual record locking lets you control operations that must be done together.

Manual record locking occurs when the ULK bit is set in the record-processing options field on the execution of a $GET, $FIND, or $PUT macro instruction. (These three macro instructions will also unlock any record that was locked with automatic record locking.) Once the record is manually locked, it will remain in that state until explicitly unlocked by either the free or release service, or until the stream terminates (by a disconnect or close service). Other operations on the record or stream, including operations that result in errors, do not cause the record to be unlocked. For example, you can only issue the $UPDATE and $DELETE macro instructions if the record is already locked. When the service completes, the record (or record cell in a relative file in the case of the delete service) remains locked. If the target record cell is already manually locked when you issue a $PUT macro instruction (by using the NXR bit in the record-processing options field), the put service is performed and the record remains locked.

10.5.3 Controlling Record Locking

Three of the bits in the record-processing options field (ROP) of the RAB control manual record locking and unlocking:

1. ULK
2. NLK
3. RLK

The ULK bit selects manual (as opposed to automatic) locking and unlocking. This bit is input to a $GET, $FIND, or $PUT macro instruction, and specifies that the record locked as a result of that operation cannot be unlocked automatically by VAX-11 RMS. Once a record is locked, you must explicitly unlock it with a $FREE or $RELEASE macro instruction.

The NLK bit specifies that the record accessed with either a $GET or $FIND macro instruction is not to be locked. This bit takes precedence over the ULK bit.

The RLK bit specifies that other readers can access a record locked with a $GET or $FIND macro instruction, but the record is to remain locked. This bit is used primarily when the user who locks the record does not mind other users reading the record, but does not want any other user to modify the record.

If a get, find, or put service fails because a record is already locked by another stream, a status code of RMS$_RLK returns. If the get or find service succeeds, but the record is already locked by this stream, a success status code of RMS$_OK_ALK returns. In this case, the locked state (either manual or automatic) is not changed by the current operation. If a get or find service with the NLK bit set accesses a record locked with the RLK bit set, a success status code of RMS$_OK_RLK returns.

CHAPTER 11

RECORD-PROCESSING MACRO INSTRUCTIONS

After you open a file for processing with a $OPEN or $CREATE macro instruction, you can perform operations on the records in the file.

The record-processing macro instructions cause VAX-11 RMS to perform some operation related to records in the file, rather than on the file itself. The macro instructions, therefore, deal with fields in the record access block (RAB). See Chapter 5 for a description of the effect these fields have on the record operation.

In most cases, you use a record-processing macro instruction with parameters indicating the symbolic address of the associated RAB and the address of any optional error or success completion routine you may have provided. You can also use the macro instruction without parameters, but you must then create an argument list in your program to define the values for these addresses (see Section 8.1).

Table 3-2 summarizes all the run-time processing macro instructions. This chapter deals only with the following macro instructions for record processing:

- Record access and current record context
  - $GET
  - $DELETE
  - $PUT
  - $FIND
  - $UPDATE
- Record streams
  - $CONNECT
  - $DISCONNECT
- Synchronization with asynchronous operations
  - $WAIT
- Miscellaneous operations
  - $FLUSH
  - $RELEASE
  - $FREE
  - $REWIND
  - $NXTVOL
  - $TRUNCATE

Chapter 10 discussed some of the general concepts involved in performing record operations. This chapter presents the details of the particular macro instructions that perform record operations, in alphabetical order.

$CONNECT

11.1 ESTABLISHING A RECORD STREAM

The $CONNECT macro instruction invokes the connect service, which establishes a record stream by associating and connecting a RAB with a FAB. For sequential files, only one RAB can be connected to a FAB. For relative or indexed files, any number of RABs can be connected to a FAB, if the MSE bit was set in the file-sharing field (SHR) of the FAB when the file was opened or created. Each RAB represents an independent record stream.

When you issue a $CONNECT macro instruction, VAX-11 RMS allocates an internal counterpart for the RAB. This counterpart consists of the necessary internal controls needed to support the stream, such as record pointers and request status information. All required I/O buffers are also allocated at this time, and can be locked in the working set (see Section 5.2.8). $CONNECT also initializes the next record pointer to the first record. In indexed files, the key of reference establishes the index of the next record pointer.

You can issue a $CONNECT macro instruction only to files that are already open.

FORMAT

OPERATION	PARAMETERS
label: $CONNECT	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $CONNECT macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-1 lists the RAB fields that the connect service uses for input and output.

Table 11-1
Connect RAB Fields

Usage	Field Name	Description
Input	FAB	File access block address (used to access only the internal file identifier field of the FAB)
	KRF	Key of reference (used only with indexed files)
	MBC	Multiblock count (sequential disk files only)
	MBF	Multibuffer count
	ROP	Record-processing options (ASY, BIO, EOF, RAH, and WBH only)
Output	ISI	Internal stream identifier
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the connect service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation

$DELETE

11.2 DELETING A RECORD

The $DELETE macro instruction invokes the delete service, which removes an existing record from a relative or indexed file (you cannot use this macro instruction with sequential files). A record delete operation always applies to the current record. Therefore, immediately before you issue the $DELETE macro instruction, you must lock the record by issuing a $FIND or $GET macro instruction.

FORMAT

OPERATION	PARAMETERS
label: $DELETE	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $DELETE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-2 lists the RAB fields that the delete service uses for input and output.

Table 11-2
Delete RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	ROP	Record-processing options (ASY bit only)
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the $DELETE macro instruction are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_EXP	Expiration date not yet reached
RMS$_RNL	Warning; record not locked
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_WLK	Device write-locked

$DISCONNECT

11.3 TERMINATING A RECORD STREAM

The $DISCONNECT macro instruction invokes the disconnect service, which breaks the connection between a RAB and a FAB, thereby terminating a record stream. All system resources, such as I/O buffers and data structure space, are deallocated.

The close service (see Section 9.1) performs an implied disconnect for all record streams connected to the FAB.

FORMAT

OPERATION	PARAMETERS
label: $DISCONNECT	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $DISCONNECT macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-3 lists the RAB fields that the disconnect service uses for input and output.

Table 11-3
Disconnect RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	ROP	Record-processing options (ASY bit only)
Output	ISI	Internal stream identifier (zeroed)
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the disconnect service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_RSA	Record stream still active
RMS$_WLK	Device write-locked

$FIND

11.4 LOCATING A RECORD

The $FIND macro instruction invokes the find service, which locates a specified record in a file and returns its record's file address in the RFA field of the RAB. This applies to all file organizations.

The main uses of the find service are:

- Skipping records when you are using the sequential record access mode (by issuing successive requests for find operations)
- Locking, but not retrieving, a record, thereby establishing a current record for an operation with a $UPDATE, $DELETE, or $TRUNCATE macro instruction
- Establishing a random accessed starting point in a file for subsequent sequential access

FORMAT

OPERATION	PARAMETERS
label: $FIND	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $FIND macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-4 lists the RAB fields that the find service uses for input and output.

Table 11-4
Find RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	KBF	Key buffer address (used only if RAC=KEY or if RAC=SEQ and the LIM option is selected in the ROP)
	KRF	Key of reference (used only with indexed files)
	KSZ	Key size (used only if RAC=KEY or if RAC=SEQ and the LIM option is selected in the ROP)
	RAC	Record access
	RFA	Record's file address (used only if RAC=RFA)
	ROP	Record-processing options
Output	BKT	Bucket code; set to the relative record number for relative files accessed sequentially
	RFA	Record's file address
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The record address (RBF) and record size (RSZ) fields are undefined after a find service.

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the find service are listed below.

Success:

RMS$_CONTROLC	Operation completed under Control C
RMS$_CONTROLY	Operation completed under Control Y
RMS$_OK_ALK	Record already locked
RMS$_OK_DEL	Deleted record accessed correctly
RMS$_NORMAL	Operation successful
RMS$_OK_RLK	Record locked but read anyway
RMS$_PENDING	Asynchronous operation not yet complete
RMS$_OK_LIM	Retrieved record exceeds specified key value

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DEL	Record accessed by the RFA record access mode has been deleted
RMS$_DNR	Device not ready
RMS$_EOF	File is at end of file
RMS$_RLK	Record locked by another task
RMS$_RSA	Record stream still active
RMS$_RTB	Warning; record too large for user buffer
RMS$_TMO	Warning; time-out period expired
RMS$_WLK	Device write-locked

$FLUSH

11.5 WRITING OUT MODIFIED I/O BUFFERS

The $FLUSH macro instruction invokes the flush service, which writes out all modified I/O buffers and file attributes associated with the file. This ensures that all record activity up to the point at which this macro instruction executes is actually reflected in the file.

The flush service is not required at any time. In addition, you should not use it before issuing a $CLOSE macro instruction because the close service implicitly performs the flush functions.

During asynchronous operations, you must wait for the completion of any I/O activity before issuing a $FLUSH macro instruction. This wait can be accomplished by issuing a $WAIT macro instruction. You may also issue a $FLUSH macro instruction after having received notification of completion through an asynchronous system trap (AST).

FORMAT

OPERATION	PARAMETERS
label: $FLUSH	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $FLUSH macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-5 lists the RAB fields that the flush service uses for input and output.

Table 11-5
Flush RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	ROP	Record-processing options (ASY bit only)
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the flush service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet completed

Failure:

RMS$_ACT	File active; operation not performed
RMS$_DNR	Device not ready
RMS$_RSA	Record stream still active

$FREE

11.6 UNLOCKING ALL RECORDS

The $FREE macro instruction invokes the free service, which unlocks all records that were previously locked for the record stream (see also the $RELEASE macro instruction, Section 11.10). If no records are locked for the record stream, VAX-11 RMS returns a status code of RMS$_RNL.

Section 10.5 describes the record-locking action.

FORMAT

OPERATION	PARAMETERS
label: $FREE	RAB=rab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $FREE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-6 lists the RAB fields that the free service uses for input and output.

Table 11-6
Free RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the free service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity procedures operation
RMS$_RNL	Record not locked by the record stream
RMS$_RSA	Record stream still active (asynchronous operations)

$GET

11.7 RETRIEVING A RECORD

The $GET macro instruction invokes the get service, which causes a record to be retrieved from a file. For a file of sequential organization, you can use either sequential or random by record's file address (RFA) record access mode. RFA record access mode, however, applies to disk files only. For relative or indexed file organization (and sequential files on disk with fixed-length records), you can use any of the record access modes: sequenital, random by key (relative record number for relative and sequential files and key value for indexed files), or random by RFA.

You can have the record placed in your own buffer through move mode, or have the record pointed to in a system I/O buffer area through locate mode. However, note that locate mode is only partial locate; in some cases, the record may have to be moved to your buffer. This partial locate can occur, for example, when records cross block boundaries in sequential files, or whenever the file is accessed for update. To handle all such cases, you must always supply a user record buffer for record retrieval operations. In addition, during

asynchronous operations, you must include either a completion routine or a $WAIT macro instruction to guarantee operation completion.
FORMAT

OPERATION	PARAMETERS
label: $GET	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $GET macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-7 lists the RAB fields that the get service uses for input and output.

Table 11-7
Get RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	KBF	Key buffer address (used only if RAC=KEY or if RAC=SEQ and the LIM option is selected in the ROP)
	KRF	Key of reference (used only with indexed files)
	KSZ	Key buffer size (used only if RAC=KEY or if RAC=SEQ and the LIM option is selected in the ROP)
	PBF	Prompt buffer address; applies to terminals only
	PSZ	Prompt buffer size; applies to terminals only
	RAC	Record access mode
	RFA	Record's address (used only if RAC=RFA)
	RHB	Record header buffer; used for variable with fixed control records
	ROP	Record-processing options
	TMO	Time-out period
	UBF	User record area address
	USZ	User record area size
Output	BKT	Bucket code; set to the relative record number for relative files when the record access mode is sequential
	RBF	Record address
	RFA	Record's file address
	RSZ	Record size
	STS	Completion status code (also returned in Register 0)
	STV	Status value (contains a terminator character for terminal input or the record length if the requested record is too large for the user buffer area)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the get service are listed below.

Success:

RMS$_CONTROLC	Operation completed under Control C
RMS$_CONTROLY	Operation completed under Control Y
RMS$_NORMAL	Operation successful
RMS$_OK_RNF	Nonexistent record accessed correctly
RMS$_PENDING	Asynchronous operation not yet complete
RMS$_OK_RLK	Record locked but read anyway
RMS$_OK_LIM	Retrieved record exceeds specified key value

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DEL	Record accessed by the RFA record access mode has been deleted
RMS$_DNR	Device not ready
RMS$_EOF	End of file
RMS$_RLK	Record locked by another task
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_RTB	Warning; illegal record size
RMS$_TMO	Warning; time-out period expired
RMS$_WLK	Device write-locked

$NXTVOL

11.8 CONTINUE PROCESSING ON NEXT VOLUME

The $NXTVOL macro instruction invokes the next volume service. This service applies only to files on magnetic tape volumes. Use this macro instruction when you want to proceed to the next volume in the set before the end of the current volume (EOV label) is reached on input, or before the end of tape (EOT mark) is reached on output. VAX-11 RMS will then position to the first file section on the next volume. File sections occur when a file is written on more than one volume, the portion of the file on each of the volumes constituting a file section.

For input files, the following occurs:

- If the current volume is the last volume of the set, VAX 11 RMS reports end of file.

- If another file section exists, the next volume is mounted. When necessary, the current volume is rewound and a request to mount the next volume is issued to the operator.

- The header label (HDR1) of the file section on the newly mounted volume is read. If it is not the volume being sought, the operator is requested to mount the correct volume.

For output files, the following occurs:

- The file section on the current volume is closed with the appropriate end-of-volume labels, and the volume is rewound.

- The next volume is mounted.

- A file with the same file name and the next higher file section number is opened for output, and processing continues.

If operating asynchronously, you must wait for the completion of any I/O activity on this volume before issuing a $NXTVOL macro instruction.

The next volume service performs a flush operation for write-accessed volumes (see Section 11.5), thus writing the I/O buffers on the current volume before creating the next file section. If this is an input-only file, all records currently contained in the I/O buffers are lost, and the next get call will return the first record on the next volume.

FORMAT

OPERATION	PARAMETERS
label: $NXTVOL	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $NXTVOL instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-8 lists the RAB fields that the next volume service uses for input and output.

Table 11-8
Next Volume RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	ROP	Record-processing options (ASY bit only)
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the next volume service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File active, operation not performed
RMS$_DNR	Device not ready
RMS$_DPE	Device positioning error
RMS$_RSA	Record stream still active (asynchronous operations)

$PUT

11.9 WRITING A RECORD TO A FILE

The $PUT macro instruction invokes the put service, which inserts a record into a file. Note that only new records can be placed in the file; existing records can only be updated if their contents are to be modified. The new records can be placed either at the end of the file (sequential and relative file organizations) or in empty record cells in place of deleted records (relative file organization). Location of new records in an indexed file is controlled by VAX-11 RMS, which examines the contents of the primary key field of the record to determine where to write the record into the file.

The put service moves the record to be put from the user buffer to the VAX-11 RMS I/O buffer; VAX-11 RMS does not support locate mode because the I/O buffers are in memory that is protected against writing at the user level. In addition, you cannot issue a $PUT macro instruction when using random by RFA record access mode. The put service works only in combination with sequential or random by key record access mode.

When using sequential files, you can only write records at the end of the file, and only with the sequential record access mode. These records cannot have a length greater than the maximum you specified when you created the file. You can use random by key record access mode to write fixed-length records in a sequentially organized disk file.

In a relative file, you can use either sequential or random by key record access mode. However, records cannot be larger than the size set when the file was created, the target record cell cannot already contain a record, and the record's relative record number must not exceed the maximum record number established for the file.

In an indexed file, you can use either sequential or random by key record access mode. When sequential access is used to put (insert) records, the primary key value of the record to be put must be equal to or greater than the primary key value of the preceding record. The records cannot be larger than the size established (if a maximum length was specified) when the file was created. Each record written must contain a complete primary key, but the records do not have to contain all alternate keys. If alternate keys are partially or completely missing because of record length, VAX-11 RMS will not make an entry for that new record in the associated alternate index(es). Put operations to an indexed file do not require a key value or key of reference. VAX-11 RMS determines where to write the record by examining the contents of the primary key in the record. When inserting the records in the file, VAX-11 RMS compares the key values (primary and alternate) in the record with the key values of records already existing in the file. This comparison determines if the writing of the record would result in the presence of duplicate key values among records of the file. If duplicates would occur, VAX-11 RMS verifies that duplicates are allowed. If duplicates are not allowed for a particular key, VAX-11 RMS rejects the operation with an RMS$_DUP error code. However, if duplicates are allowed, VAX-11 RMS performs the operation. Subsequent sequential operations on a given index will always retrieve records with identical key values in the order in which the records were written.

FORMAT

OPERATION	PARAMETERS
label: $PUT	RAB=rab-address ERR=entry SUC=entry

label

A user-defined symbolic address for the $PUT macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-9 lists the RAB fields that the put service uses for input and output.

Table 11-9
Put RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	KBF	Key buffer address (used only if RAC=KEY and the file is a relative file)
	KSZ	Key size (used only if RAC=KEY and the file is a relative file)
	RAC	Record access mode
	RBF	Record address
	RHB	Record header buffer; only applies to variable with fixed control records
	RSZ	Record size
	ROP	Record-processing options (ASY, CCO, RLK, TPT, UIF, ULK and WBH only)
Output	BKT	Bucket code; set to the relative record number for sequential access to relative files
	RFA	Record's file address
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the put service are listed below.

Success:

RMS$_CONTROLC	Operation completed under Control C
RMS$_CONTROLO	Operation completed under Control O
RMS$_CONTROLY	Operation completed under Control Y
RMS$_NORMAL	Operation successful
RMS$_OK_ALK	Record already locked
RMS$_PENDING	Asynchronous operation not yet complete
RMS$_OK_IDX	Record was inserted, but error occurred on index update which could cause slow access

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_EXT	File extend error
RMS$_PRV	Privilege violation; access denied
RMS$_REX	Record already exists in target record cell
RMS$_RLK	Record locked by another task
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_RVU	Error updating RRVs
RMS$_WLK	Device write-locked

$RELEASE

11.10 UNLOCKING A RECORD

The $RELEASE macro instruction invokes the release service, which unlocks the record pointed to by the contents of the record's file address RFA field of the RAB (see also the $FREE macro instruction, Section 11.6). If the named record is not locked, VAX-11 RMS returns a status code of RMS$_RNL.

Section 10.5 describes record locking.

FORMAT

OPERATION	PARAMETERS
label: $RELEASE	RAB=rab-address
	ERR=entry
	SUC=entry

label
A user-defined symbolic address for the $RELEASE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-10 lists the RAB fields that the release service uses for input and output.

Table 11-10
Release RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	RFA	Record's file address
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the release service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_RNL	Warning; record not locked
RMS$_RSA	Record stream still active (asynchronous operations)

$REWIND

11.11 POSITIONING TO THE FIRST RECORD

The $REWIND macro instruction invokes the rewind service, which sets the current context of a stream to the first record in the file. VAX-11 RMS alters the context of the Next Record to indicate the first record as being the next record. The rewind service implicitly performs the flush and free services, writing out all I/O buffers and releasing all locked records. The service is valid for all file organizations on disk volumes and for sequential files on tape volumes. For indexed files, the KRF field establishes the index to be used. You cannot, however, rewind a unit record device (card reader or printer) or a terminal.

FORMAT

OPERATION	PARAMETERS
label: $REWIND	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $REWIND macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-11 lists the RAB fields that the rewind service uses for input and output.

Table 11-11
Rewind RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	KRF	Key of reference (used only with indexed files)
	ROP	Record-processing options (ASY bit only)
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_BOF	Warning; file is already at beginning of file
RMS$_DNR	Device not ready
RMS$_DPE	Device-positioning error
RMS$_EOF	End of file
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_WLK	Device write-locked

$TRUNCATE

11.12 TRUNCATING A SEQUENTIAL FILE

The $TRUNCATE macro instruction invokes the truncate service, which truncates records from the end of a sequential file. Note that you can only truncate a sequential file (you cannot use this service for a relative or indexed file) and the file must be open for exclusive

access (file-sharing field of the FAB set or defaulted to NIL). The truncate service deletes the record indicated as the Current Record, and all following records. You can only use this service immediately after successful execution of a $GET, $FIND, or $UPDATE macro instruction (thereby setting the context of the Current Record).

VAX-11 RMS declares an end of file at the starting record position for the truncation, and then causes the context of the Next Record to be set to this end of file. Therefore, you can now add records to the file by issuing successive $PUT macro instructions.

FORMAT

OPERATION	PARAMETERS
label: $TRUNCATE	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $TRUNCATE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-12 lists the RAB fields that the truncate service uses for input and output.

Table 11-12
Truncate RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	ROP	Record-processing options (ASY only)
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the truncate service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_DPE	Device-positioning error
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_WLK	Device write-locked

$UPDATE

11.13 UPDATING AN EXISTING RECORD

The $UPDATE macro instruction invokes the update service, which modifies (updates) the contents of an existing record in a disk file only. The record to be updated must first be locked by this stream, either by a $FIND or $GET macro instruction. You cannot use locate mode; you must supply a buffer.

For sequential files, the record length cannot change. For relative files with variable-length or variable with fixed control records, the length of the replacement record can differ from the length of the original record, but cannot be larger than the maximum size you set when you created the file.

For indexed files, the length of the replacement (updated) record written by the $UPDATE macro instruction may be different from the original record; restrictions, however, apply to the replacement record in an indexed file:

- The length of the replacement record cannot exceed the maximum size defined at file creation.
- Each replacement record must be large enough to contain a complete primary key, but the replacement record does not have to contain all alternate keys. If an alternate key is partially or completely missing in the replacement record, the key must have the characteristic that the values can change; this is true also if the replacement record contains a key that was not present in the original record.

Update operations to an indexed file do not require a key value or key of reference. Before writing the record, VAX-11 RMS compares the key values (primary and alternate) in the replacement record with the key values of original record already existing in the file. This comparison takes into account the defined characteristics of each key. For example, if a particular key is not allowed to change, VAX-11 RMS rejects the operation with an RMS$_CHG error code if the replacement record contains an altered value in the associated key. Similarly, this comparison determines if the replacement record would result in the presence of duplicate key values among records of the file. If duplicates would occur, VAX-11 RMS verifies the defined characteristics for the keys being duplicated. If duplicates are not allowed for a particular key, VAX-11 RMS rejects the operation with an RMS$_DUP error code. However, if duplicates are allowed, VAX-11 RMS performs the operation.

Subsequent sequential operations on a given index will always retrieve records with identical key values in the order in which the records were written.

FORMAT

OPERATION	PARAMETERS
label: $UPDATE	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $UPDATE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 11-13 lists the RAB fields that the update service uses for input and output.

Table 11-13
Update RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	RBF	Record address
	RHB	Record header buffer; applies only to variable with fixed control records
	ROP	Record-processing options (ASY and WBH only)
	RSZ	Record size
Output	RFA	Record's file address
	STS	Completion status code (also returned in Register 0)
	STV	Status value

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the update service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete
RMS$_OK_IDX	Record was inserted, but error occurred on index update which could cause slow access

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_RNL	Warning; record not locked
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_RRV	Error updating RRVs
RMS$_WLK	Device write-locked

$WAIT

11.14 STALL FOR I/O COMPLETION

The $WAIT macro instruction invokes the wait service, which determines when an asynchronous record operation completes. Upon completion of the operation, VAX-11 RMS returns control to your program at the point following the $WAIT macro instruction. Any completion routines specified on the operation being awaited are also executed before VAX-11 RMS returns control (unless ASTs are disabled).

The $WAIT macro instruction takes no parameters to define entry points for user-written completion routines; the completion routines are specified by the operation being awaited.

FORMAT

OPERATION	PARAMETERS
label: $WAIT	RAB=rab-address

label
A user-defined symbolic address for the $WAIT macro instruction; optional.

RAB=rab-address
The only parameter allowed if parameters are used; rab-address defines the symbolic address of either the record access block having an I/O request in progress, or some other record access block.

Table 11-14 lists the RAB fields that the wait service uses for input and output.

Table 11-14
Wait RAB Fields

Usage	Field Name	Description
Input	ISI	Internal stream identifier
	STS	Completion status code
Output	STS	Completion status code (also returned in Register 0)

The VAX-11 RMS completion status codes for the wait service are determined by the operation being awaited, unless the address of the RAB specified for the wait is not the same as that specified for the awaited operation. In this case, RMS$_NORMAL is returned.

CHAPTER 12

PERFORMING BLOCK I/O

Besides the record access provided by the sequential, random by key, and random by record's file address record access modes, VAX-11 RMS also provides another means to access data in a file: block I/O.

Block I/O operations let you directly read or write the blocks of a file. These operations are provided for users who must keep system overhead to a minimum and need no interpretation of file data as logical records, yet still want to take advantage of the easy file access of VAX-11 RMS. Block I/O is an intermediate step between the VAX-11 RMS record operations and direct use of VAX/VMS input/output system services.

You specify block I/O for a record stream by setting the BIO bit in the file access field of the file access block (FAB; see Section 4.2.10) as input to the $OPEN or $CREATE macro instructions. If you intend to write to the file, you must set the PUT bit in the file access field. If you want to read from the file, you must set the GET bit in the file access field.

You cannot perform block I/O operations on files on which record operations are already being performed. Conversely, you cannot perform record operations on files on which block I/O operations are being performed. However, VAX-11 RMS allows you to set the BRO bit in the file access field of the FAB, indicating that operations can switch from block I/O to record operation and vice versa when an operation is completed (but not using both at the same time). Only the sequential file organization allows this switching. For other file organizations, setting of the BRO bit of the file access field merely allows the decision about performing block or record operations to be delayed until the first RAB is connected. If the BIO bit is set in the record options field of the RAB, only block I/O operations will be permitted; if the BIO bit is clear, only record operations will be permitted. All connected record streams must be connected in the same manner; that is, there can be no mixing of block and record I/O.

If you do mix modes of operation for sequential files, you must exercise caution, as the context of the Current Record, Next Record, and the next block pointer (see NOTES below) are all undefined when you switch operations on disk devices. Therefore, the operation that initiates the switch must not use sequential record access mode. For magnetic tape devices, the context of the Next Record or next block indicates the start of the following block on the tape for the operation initiating the switch.

NOTES

1. If you set the BRO bit in the file access field of the FAB for the sequential file organization, you indicate that you want to mix block I/O and record operations. If, once the file is open, you want only to perform block I/O, you can set the BIO bit in the record-processing options field of the RAB. This overrides the setting of the BRO bit for this record stream, and acts as a flag to the $CONNECT macro instruction, indicating that no VAX-11 RMS I/O buffers need be allocated (but you must still allocate buffers in the user program for block I/O operations).

2. If you set the BRO bit when creating an indexed file, the key definition XABs for that file must be present. For a create service to the relative or indexed file organization, specifying the BIO bit in the file access field of the FAB causes VAX-11 RMS to omit prologue processing and initial space pre-zeroing in relative files. Allocated space pre-zeroing is also omitted for the extend service when connected for block I/O.

3. For files of unknown organization or undefined record format, block I/O is the only form of processing allowed. Processing proceeds identically to that for block I/O to the relative file organization.

Three macro instructions are provided for performing block I/O.

- $READ -- transfers a specified number of bytes into memory
- $SPACE -- positions a file forward or backward a specified number of blocks
- $WRITE -- writes a specified number of bytes to a file

In addition, you can use the following macro instructions on a record stream connected for block I/O operations:

- $DISCONNECT
- $FLUSH
- $NXTVOL
- $REWIND

These instructions, which are described in Chapter 11, perform miscellaneous operations or disconnect the record stream, and do not work on the contents of the records themselves.

For sequential block I/O operations to disk files, VAX-11 RMS maintains an internal next block pointer (NBP) that:

- Points to the beginning of the file after execution of a $CONNECT macro instruction if the EOF bit is cleared in the record-processing options field of the record access block (RAB), or if the EOF bit is set, NBP points to the block following the end of file. For indexed files, setting EOF is an illegal procedure
- Points to the block following the highest numbered block transferred by a read or write service ($READ or $WRITE macro instructions)
- Points to the next block after an operation with the $SPACE macro instruction

The block I/O macro instructions deal with fields in the RAB; Chapter 5 describes the effect of these fields on the operations.

You indicate the symbolic address of the associated RAB through a parameter on each block I/O macro instruction you are using, and the address of any optional error or success completion routine you may have provided. However, you can also use the macro instruction without parameters, but you must then create an argument list in your program to define the values for these addresses (see Section 8.1).

$READ

12.1 TRANSFER TO MEMORY

The $READ macro instruction invokes the read service, which retrieves a specified number of bytes from a file (on a block boundary) and transfers them to memory. A read operation using block I/O can be performed on any file organization.

To use this macro instruction, you must:

1. Supply a buffer area into which VAX-11 RMS is to transfer data (user record area address field).
2. Indicate the number of bytes to be transferred (user record area size field).
3. Indicate the first virtual block number (VBN) for the transfer (bucket number field). If the value for the VBN is zero, the transfer will start with the block indicated by the NBP.

For the read service to operate, you must first declare the file open for block I/O operations by setting the BIO (or BRO) bit in the file access field of the FAB.

FORMAT

OPERATION	PARAMETERS
label: $READ	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $READ macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 12-1 lists the RAB fields that the read service uses for block I/O.

Table 12-1
Read RAB Fields

Usage	Field Name	Description
Input	BKT	Bucket number; must contain the virtual block number of the first block to be read
	ISI	Internal stream identifier
	ROP	Record-processing options; checked to see if ASY is set for asynchronous operations
	UBF	User record area address
	USZ	User record area size; indicates the length of transfer, in bytes
Output	RBF	Record address
	RFA	Record's file address; contains the virtual block number of the first block transferred
	RSZ	Record size; indicates the actual number of bytes transferred
	STS	Completion status code (also returned in Register 0)
	STV	Status value (contains terminator character for terminal input)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the $READ macro instruction are listed below.

Success:

RMS$_CONTROLC	Operation completed under Control C
RMS$_CONTROLO	Operation completed under Control O
RMS$_CONTROLY	Operation completed under Control Y
RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_EOF	End of file; checking for the logical end of file is performed for the sequential file organization only. If an end-of-file error occurs, it implies that the first virtual block number specified was at or past the end of the file. If the end-of-file pointer occurs during a transfer, the record size field is set to the number of bytes before the logical end of file. For the relative file organization, this status code indicates an attempt to read past the end of the currently allocated space.
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_TMO	Warning; time-out period expired
RMS$_WLK	Device write-locked

$SPACE

12.2 POSITIONING TO A BLOCK

The $SPACE macro instruction invokes the space service, which lets you position a file forward or backward a specified number of blocks.

This macro instruction is intended primarily for use with magnetic tape files; the tape is spaced the number of blocks specified in the bucket number field. If the value in this field is positive, the tape spaces forward; if the value is negative, the tape spaces backward. For disk files, the NBP is updated to reflect the new sequential operation position.

To use the $SPACE macro instruction, you must first declare the file open for block I/O operations by setting the BIO (or BRO) bit in the file access field of the FAB.

FORMAT

OPERATION	PARAMETERS
label: $SPACE	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $SPACE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written success completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 12-2 lists the RAB fields that the space service uses as input and output.

Table 12-2
Space RAB Fields

Usage	Field Name	Description
Input	BKT	Bucket number; indicates the number of blocks to space forward (positive value) or backward (negative value)
	ISI	Internal stream identifier
	ROP	Record-processing options; checked to see if ASY bit is set to indicate an asynchronous operation
Output	STS	Completion status code (also returned in Register 0)
	STV	Status value (set to number of blocks actually spaced; positive value always)

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the space service are listed below.

Success:

RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete

Failure:

RMS$_ACT	File activity precludes operation
RMS$_BOF	File is at beginning of file (backspace operations)
RMS$_DNR	Device not ready
RMS$_DPE	Device-positioning error
RMS$_EOF	File is at end-of-file position (forward space operations)
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_WLK	Device write-locked

$WRITE

12.3 WRITE TO A FILE

The $WRITE macro instruction invokes the write service, which transfers a user-specified number of bytes, beginning on a block boundary, to a VAX-11 RMS file of any file organization.

You indicate the number of bytes to be written in the record size field of the RAB, and indicate the address of the buffer for the transfer in the record address field. In the bucket number field, you indicate the virtual block number of the first block to be written; if this number is 0, the transfer starts with the block indicated by the NBP.

For sequential files, the file is automatically extended if you write a block past the end of the currently allocated space. For the relative file organization, you must use the $EXTEND macro instruction.

VAX-11 RMS maintains a logical end of file to correspond to the last block and highest byte written within the block.

To use the $WRITE macro instruction, you must first declare the file open for block I/O operations by setting the BIO (or BRO) bit in the file access field of the FAB.

FORMAT

OPERATION	PARAMETERS
label: $WRITE	RAB=rab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $WRITE macro instruction; optional.

RAB=rab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the RAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 12-3 lists the RAB fields that the write service uses as input and output.

Table 12-3
Write RAB Fields

Usage	Field Name	Description
Input	BKT	Bucket number; must contain the virtual block number of the first block to be written
	ISI	Internal stream identifier
	RBF	Record address
	ROP	Record-processing options (ASY and TPT bits only)
	RSZ	Record size; indicates the transfer length, in bytes.
Output	RFA	Record's file address; contains the virtual block number of the first block transferred.
	STS	Completion status code (also returned in Register 0)
	STV	Status value; contains the actual number of bytes transferred if an end-of-file error occurs.

The VAX-11 RMS completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the write service are listed below.

Success:

RMS$_CONTROLC	Operation completed under Control C
RMS$_CONTROLO	Operation completed under Control O
RMS$_CONTROLY	Operation completed under Control Y
RMS$_NORMAL	Operation successful
RMS$_PENDING	Asynchronous operation not yet complete.

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNR	Device not ready
RMS$_EOF	End of file; for the sequential file organization, this error implies that the file could not be extended
RMS$_EXT	File extend error
RMS$_PRV	Privilege violation; access denied
RMS$_RSA	Record stream still active (asynchronous operations)
RMS$_WLK	Device write-locked

12.4 NONFILE-STRUCTURED OPERATIONS

VAX-11 RMS lets you perform nonfile-structured operations, that is, operations that deal with magnetic tape or disk volumes directly rather than through the Files-11 structure.

Nonfile-structured operations also are known as logical I/O operations. Logical I/O is the reading and writing of data in blocks. For magnetic tape, each block of the tape is read or written with no interpretation of labels. For disk, the starting block for a transfer is identified by a logical block number (LBN). Since LBNs are volume-relative (see Appendix B), no file-relative translation is required to determine the blocks to transfer.

You can perform nonfile-structured operations under the following conditions.

1. For file devices that have been mounted as Files-11 volumes, you must set the NFS bit in the file-processing options field (FOP) of the FAB as input to the create or open service.

2. For file devices mounted as foreign (i.e. nonfile-structured), VAX-11 RMS performs nonfile-structured operations regardless of the state of the NFS bit.

3. For nonfile devices, nonfile-structured operations occur always.

4. If the NFS bit is set, the I/O channel is assigned in the mode of the caller, thus allowing I/O calls to be performed directly, if desired.

5. You must have the appropriate privileges to perform nonfile-structured operations (logical I/O privilege).

6. Either block I/O or the get and put services are allowed. For magnetic tape, blocking information must be specified on the MOUNT command (see the VAX/VMS Command Language User's Guide), using the /RECORD qualifier; this allows the blocking and unblocking of fixed-length records, with records not crossing block boundaries. For disk, each block is read as a fixed-length record of 512 bytes.

7. The file specification only needs the device and unit number.

If the above conditions have been met, VAX-11 RMS will change its operations to include the following:

1. The block I/O services including space are permitted, even if not in block I/O mode.

2. The rewind service rewinds the entire magnetic tape.

3. If the close service is performed to a write-accessed magnetic tape, two tape marks are output, followed by a backspace. This allows the creation of multiple files. On input, end-of-file errors cause the tape mark to be skipped.

4. For disk, the normal input of the bucket code field (BKT) of the RAB for read and write services specifies the logical block number (LBN) rather than the virtual block number (VBN). Since logical block numbers start at 0 and virtual block numbers start at 1, a problem may arise when you want to access LBN 0 (a 0 in the bucket code field indicating sequential operations). However, you can access LBN 0 by setting the bucket code field to 0 immediately after a connect or rewind service (or by issuing an appropriate space service to backspace to the beginning of the volume).

5. For the get and put service, random access by key (RAC=KEY), set the key buffer pointed to by the key buffer address field to the starting LBN.

CHAPTER 13

FILE SPECIFICATION PROCESSING MACRO INSTRUCTIONS

VAX-11 RMS provides macro instructions that perform actions related to the file specification. These macro instructions are used only for relatively complex operations, such as wildcard processing, as their functions are normally performed by the open and create services.

These macro instructions, therefore, deal with fields in the file access block (FAB), and the name (NAM) block. Chapters 4 and 7 describe the effects of these fields for the FAB and NAM block, respectively. The file specification processing macro instructions are:

- $ENTER
- $PARSE
- $REMOVE
- $RENAME
- $SEARCH

You indicate the symbolic address of the associated FAB through a parameter on the file specification processing macro instructions. You do not indicate the NAM block on the macro instructions; rather, you associate this NAM block with the FAB through the name block address field of the FAB.

On the file specification processing macro instructions, you can also use a parameter to indicate the address of any optional error or success completion routine you may have provided. You can use the macro instruction without parameters, but you must then create an argument list in your program to define the values for these addresses (see Section 8.1).

$ENTER

13.1 ENTER A FILE NAME

The enter service, which you invoke with the $ENTER macro instruction, inserts a file name into a directory. This is performed automatically by the create service (unless either the TMP or TMD bit is set in the file-processing options field of the FAB). The enter service, however, allows you to perform this step separately.

When you enter a file name into a directory, no file can already be open with the FAB, and no wildcard specifications are allowed.

The enter service requires many NAM block fields as input. You normally precede the enter service with an open, create, or parse service (see Section 13.2), and a search service (see Section 13.5), specifying the same FAB and NAM block for each service.

FORMAT

OPERATION	PARAMETERS
label: $ENTER	FAB=fab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $ENTER macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 13-1 lists the fields in both the FAB and NAM block that the enter service uses as input and output.

Table 13-1
Enter Fields

Usage	Control Block	Field Name	Description
Input	FAB	IFI	Internal file identifier (must be zero)
		NAM	Name block address
	NAM	DID	Directory identification; file name and identifier are entered into this directory
		DVI	Device identification of the device containing directory where file name is to be entered
		ESA	Expanded string area address; contains file name, type, and version to be entered
		ESL	Expanded string length
		FID	File identification of file to be entered into directory
		RSA	Resultant string area address
		RSS	Resultant string size
Output	FAB	STS	Completion status code (also returned in Register 0)
		STV	Status value
	NAM	RSL	Resultant string length

The optional resultant string is moved to the buffer described by the resultant string area address (RSA) and size (RSS) fields of the NAM block (only if both these fields are nonzero).

If the file version number of the name string described by the expanded string length and area address fields of the NAM block is either not present or 0, the enter service scans the entire directory. It assigns a version number one higher than the highest found (or 1 if none is found).

The completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the enter service are listed below.

Success:

RMS$_NORMAL Operation successful

Failure:

RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_ENT	Files-11 ACP enter function failed
RMS$_FNF	File not found
RMS$_PRV	Privilege violation
RMS$_WLK	Device write-locked

$PARSE

13.2 PARSE A FILE SPECIFICATION STRING

The $PARSE macro instruction invokes the parse service, which analyzes the file specification string (as described in Section 8.2) and fills in various NAM block fields. The functions of the parse service are performed automatically as part of the open, create, and erase services.

When you parse a file name string, there must be no file already open in conjunction with the FAB.

FORMAT

OPERATION	PARAMETERS
label: $PARSE	FAB=fab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $PARSE macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 13-2 lists the fields in both the EAB and NAM block that the parse service uses as input and output.

Table 13-2
Parse Fields

Usage	Control Block	Field Name	Description
Input	FAB	DNA	Default file specification string address
		DNS	Default file specification string size
		FNA	File specification string address
		FNS	File specification string size
		FOP	File-processing options (OFP bit only)
		IFI	Internal file identifier (must be zero)
		NAM	Name block address
	NAM	ESA	Expanded string area address
		ESS	Expanded string area size
		RLF	Related file NAM block address
	Related file NAM block (if any)	RSA	Resultant string area address
		RSL	Resultant string length
Output	FAB	STS	Completion status code (also returned in Register 0)
		STV	Status value
	NAM	DID	Directory identification
		DVI	Device identification
		ESL	Expanded string length
		FID	File identification (zeroed)
		FNB	File name status bits; contains information about the parse results
		WCC	Wildcard context (zeroed to initialize the wildcard context for subsequent directory searches)

The expanded file specification string is moved to the buffer described by the expanded string area address (ESA) and size (ESS) fields of the NAM block (only if both fields are nonzero). The ESA and ESS NAM block parameters must be specified (nonzero) for wildcard processing (see Sections 7.2.2 and 7.2.3).

The completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the parse service are listed below:

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_DNF	Directory not found
RMS$_DNR	Device not ready

$REMOVE

13.3 REMOVE A FILE NAME

The $REMOVE macro instruction invokes the remove service, which deletes a file name from a directory. (This service does not delete the file itself. This is done with the erase service; see Section 9.4). The functions of the remove service are performed automatically as part of an erase service that specifies a directory.

When you remove a file name from a directory, no file can already be open for the FAB. In addition, you normally call the parse service to set the NAM block contents before you call the remove service.

Each removal deletes the next directory entry whose file name, type, and version number matches those specified in the expanded string length and expanded string area address fields of the NAM block.

FORMAT

OPERATION	PARAMETERS
label: $REMOVE	FAB=fab-address ERR=entry SUC=entry

label

A user-defined symbolic address for the $REMOVE macro instruction; optional.

FAB=fab-address

Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry

The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 13-3 lists the fields in both the FAB and NAM blocks that the remove service uses as input and output.

Table 13-3
Remove Fields

Usage	Control Block	Field Name	Description
Input	FAB	FOP	File-processing options (NAM bit only)
		IFI	Internal file identifier (must be zero)
		NAM	Name block address
	NAM	DID	Directory identification of directory cataloging file to be removed
		DVI	Device identification of device containing directory from which file is to be removed
		ESA	Expanded string area address specifying the name, type, and version of file to be removed
		ESL	Expanded string length
		FID	File identification; if nonzero and NAM bit is set in file-processing options field of input FAB, the first file in the directory with this file identification is removed
		FNB	File name status bits (wildcard bits only)
		RSA	Resultant string area address specifying the name, type, and version number of last file removed (required for wildcard processing)
		RSL	Resultant string length
		RSS	Resultant string area size
		WCC	Wildcard context
Output	FAB	STS	Completion status code (also returned in Register 0)
		STV	Status value
	NAM	RSL	Resultant string length
		WCC	Wildcard context

The resultant string is moved to the buffer described by the resultant string area address (RSA) and size (RSS) fields of the NAM block (only if both fields are nonzero).

The completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the remove service are listed below.

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_FNF	File not found
RMS$_PRV	Privilege violation
RMS$_WLK	Device write-locked

$RENAME

13.4 RENAME A FILE

The $RENAME macro instruction invokes the rename service, which changes the name of a file in a directory. This service performs the equivalent of two parse services (old and new name), a search service for the old directory, an enter service to insert the new file name into the new directory, and a remove service to delete the old file name from the old directory.

This service affects directory entries only. It does not alter the file name stored in the file header block.

When you change the name of the file in a directory, no file can already be open for the FAB, and no wild card specifications are allowed. You can rename a file from one directory to another, but both directories must be on the same disk device.

If the rename service is successful, the new directory entry is created and the old entry is deleted. If the service fails, the old entry remains, and the new entry, depending on when the error occurs, may or may not be created.

FORMAT

OPERATION	PARAMETERS
label: $RENAME	OLDFAB=fab-address ERR=entry SUC=entry NEWFAB=new-fab-address

label
A user-defined symbolic address for the $RENAME macro instruction; optional.

OLDFAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB that specifies the old file name. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

NEWFAB=new-fab-address
Required if you use prameters in the macro instruction. This parameter defines the symbolic address of the FAB that specifies the new file name. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Secion 8.1).

NOTE

If you issue this macro instruction without parameters, you must construct an additional field within your argument list to contain the address of the FAB that specifies the new file name. This additional field is placed in the argument list following the field for the success completion routine (see Section 8.1), and the argument count is set to 4.

Table 13-4 lists the fields in two FABs and two NAM blocks that the rename service uses as input and output. In the table these blocks are called FAB#1 and NAM#1 for the old entry, and FAB#2 and NAM#2 for the new entry. For output, FAB#2 is not used, although it must be in writable memory.

The resultant file specification string for each of the names (old and new) is placed in the buffer described by the resultant string area address (RSA) and size (RSS) fields of the separate NAM blocks (only if both fields are nonzero).

The completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the rename service are listed below.

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_FNF	File not found
RMS$_PRV	Privilege violation

Table 13-4
Rename Fields

Usage	Control Block	Field Name	Description
Input	FAB#1 and FAB#2	DNA	Default file specification string address
		DNS	Default file specification string size
		FNA	File specification string address
		FNS	File specification string size
		IFI	Internal file identifier (must be zero)
		NAM	Name block address
	NAM#1 and NAM#2	ESA	Expanded string area address (must be nonzero)
		ESS	Expanded string area size (must be non-zero)
		RLF	Related file NAM block address
		RSA	Resultant string area address
		RSS	Resultant string area size
	Related file NAM blocks	RSA	Resultant string area address
		RSL	Resultant string length
Output	FAB#1	STS	Completion status code (also returned in Register 0)
		STV	Status value
	NAM#1 and NAM#2	DID	Directory identification
		DVI	Device identification
		ESL	Expanded string length
		FID	File identification
		FNB	File name status bits
		RSL	Resultant string length
		WCC	Wildcard context

$SEARCH

13.5 SEARCH FOR FILE NAME

The $SEARCH macro instruction invokes the search service, which scans a directory file and fills in various NAM block fields. Normally, you precede the search service with the parse service to initialize the NAM block appropriately. The basic functions of the search service are performed automatically as part of the open, create, and erase service.

When you scan a directory file, no file can already be open for the FAB.

When called, the search service scans the directory file specified by the directory identification field of the NAM block. It looks for an entry that matches the file name, type, and version number specified by the expanded string area address and expanded string length fields. Upon finding a match, VAX-11 RMS returns the file name, type, and version number in the buffer described by the resultant string area address and size fields, and the file identification field is filled in, thereby allowing a subsequent open by NAM block (see Section 8.2.3).

FORMAT

OPERATION	PARAMETERS
label: $SEARCH	FAB=fab-address ERR=entry SUC=entry

label
A user-defined symbolic address for the $SEARCH macro instruction; optional.

FAB=fab-address
Required if you use parameters in the macro instruction. This parameter defines the symbolic address of the FAB for the file. If you omit this parameter, no other parameters are permitted; you must supply the argument list within your program (see Section 8.1).

ERR=entry
The symbolic address of a user-written error completion routine; optional.

SUC=entry
The symbolic address of a user-written success completion routine; optional.

Table 13-5 lists the fields in both the FAB and NAM block that the search service uses as input and output.

Table 13-5
Search Fields

Usage	Control Block	Field Name	Description
Input	FAB	IFI	Internal file identifier (must be zero)
		NAM	Name block address
	NAM	DID	Directory identification of directory to be searched
		DVI	Device identification of device containing directory to be searched
		ESA	Expanded string area address, specifying file name, type, and version of file
		ESL	Expanded string length
		FNB	File name status bits (wildcard bits only)
		RSA	Resultant string area address, specifying name, type and version of last file found (required for wildcard processing)
		RSL	Resultant string length
		RSS	Resultant string area size
		WCC	Wildcard context
Output	FAB	STS	Completion status code (also returned in Register 0)
		STV	Status value
	NAM	FID	File identification
		RSL	Resultant string length
		WCC	Wildcard context

The resultant file specification string is placed in the buffer described by the resultant string area address (RSA) and size (RSS) fields of the NAM block (only if both fields are nonzero). The RSA and RSS NAM block parameters must be specified (nonzero) for wildcard processing (see Sections 7.2.4 and 7.2.5).

The completion status codes categorized as severe errors are contained in Appendix A. However, to help you foresee and possibly circumvent any nonsevere conditions that can arise, any error or warning completion status codes that can cause a failure for the search service are listed below.

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_ACT	File activity precludes operation
RMS$_DNF	Directory not found
RMS$_DNR	Device not ready
RMS$_FND	Files-11 ACP find function failed
RMS$_FNF	File not found
RMS$_NMF	No more files to be searched
RMS$_PRV	Privilege violation

CHAPTER 14

RUN-TIME CONTROL BLOCK INITIALIZATION

VAX-11 RMS provides run-time equivalents of the assembly-time macro instructions that allocate and initialize control blocks. These run-time instructions are the "store" macro instructions.

The store macro instructions copy either the contents of a location or a value into data fields in the designated control block. Regardless of field size, you can access a data field with these macro instructions.

14.1 THE STORE MACRO INSTRUCTIONS

You form the name for each store macro instruction by adding _STORE to the name of each assembly-time macro instruction.

For example, the run-time equivalent of the $FAB macro instruction is:

```
$FAB_STORE
```

A run-time equivalent exists for each of the following macro instructions:

- $FAB
- $RAB
- $NAM
- $XABDAT
- $XABALL
- $XABKEY
- $XABPRO
- $XABFHC
- $XABRDT

FORMAT

OPERATION	PARAMETERS
label: macro-name	{FAB / RAB / NAM / XAB} =pointer keyword-1=value-1, . . . , keyword-n=value-n

label
A user specified symbolic address referring to the store macro instruction; optional.

macro-name
The name of the control block (FAB, RAB, NAM, XABDAT, XABALL, XABKEY, XABPRO, XABFHC, or XABRDT). The control block name is prefixed with a dollar sign ($) and followed by _STORE.

An optional pointer to the control block; the keyword to the left of the equal sign indicates the type of control block you are using. A keyword of XAB is used for all the different XABs. If the value of "pointer" is a register, the register must contain the address of the control block. If the value of "pointer" is not a register, the address that "pointer" represents is moved to Register 0; Register 0 is then used as the pointer to the control block.

If you omit this parameter, VAX-11 RMS assumes that you have already stored the address of the control block in Register 0.

keyword-1=value-1,..., keyword-n=value-n
A variable number of keywords that correspond to the data fields of the control block, and the values to be placed in these data fields. These values can be either keywords for options, as in the assembly-time macro instructions; or can be run-time addressing expressions. If the value is an addressing expression, the following restrictions apply.

1. For any address field -- such as the extended attribute block field (XAB) of the FAB, the file access block field (FAB) of the RAB, or the expanded string area address (ESA) and resultant string area address (RSA) fields of the NAM block -- a MOVAL instruction is generated rather than a MOVL instruction.

2. For a quadword field, whose source is a register, two successive registers are accessed. Therefore, the source register should not be greater than Register 11.

3. For a field that is any one of the following, and whose source is a register, two successive registers are accessed:

   - Directory identification (DID)
   - File identification (FID)
   - Record's file address (RFA)

Therefore, the source register should not be greater than Register 11. In addition, you cannot use the byte, word, or longword displacements for an offset, or any indexed or deferred addressing.

4. If you specify the device identification field (DVI), the source cannot be a register, since four registers would have to be accessed. In addition, you cannot use the byte, word, or longword displacements for an offset, or any indexed or deferred addressing.

5. The file protection (PRO) and group/member number (UIC) fields can be expressed in either of two ways:

   a. Individually -- in a manner similar to the assembly-time macro instructions. For the file protection field (PRO), the values must still be the keywords R, W, E, D. For the group/member number (UIC) fields, the values must be either run-time values or constants. The radix for constants is octal.

   b. Together -- filled in as one entity, by specifying one run-time address.

An example of a store macro instruction follows:

```
$FAB_STORE FAB=R1,ORG=SEQ,RFM=VFC,MRS=10(R2),FSZ=#30,FOP=#0,NAM=NBLK
```

In this example, Register 1 contains the address of the FAB; the file organization is sequential; the record format is variable with fixed control; and the maximum record size is to be taken from the contents of the location specified by 10(R2). In addition, the fixed size of the record is 30 bytes, the file-processing options field is to be cleared, and the address of NBLK is to be moved into the NAM block address field of the FAB.

CHAPTER 15

CONTROL ROUTINES

VAX-11 RMS provides three control routines, as follows:

- Rundown control routine
- Default Directory control routine
- Default File Protection control routine

These control routines all operate synchronously; therefore, no $WAIT macro instruction is needed.

You do not call a control routine with a macro instruction. Rather, you provide an argument list and call VAX-11 RMS at the entry point for the routine.

15.1 HALT I/O AND CLOSE FILES

The Rundown control routine closes all files opened by VAX-11 RMS for the image or process, and halts I/O activity. This is not the same as closing the files with a close service, which guarantees that all I/O will be completed (see Section 9.1). Each call made to a Rundown control routine closes at least one file. Therefore, you should continue to call Rundown control routines until you receive the success completion status code of RMS$_NORMAL.

The entry point for this control routine is:

SYS$RMSRUNDWN

There are two arguments for this control routine. The first is the address of a 22-byte buffer to receive the device identification (16 bytes) and file identification (6 bytes) of an improperly closed output file.

The second argument is a single byte code specifying the type of rundown to be performed. This type code has the following values and meanings:

0 - rundown of image and indirect I/O for process permanent files

1 - rundown of image and process permanent files; the caller's mode must be other than user

2 - abort VAX-11 RMS I/O; the caller's mode must be either executive or kernel

The Rundown control routine does not reference fields in the user control blocks.

The completion status codes are listed below.

Success:

RMS$_NORMAL	All files closed

Failure:

RMS$_CCF	An output file could not be closed successfully; user buffer identifies the file
RMS$_IAL	An output file could not be closed successfully, and the user buffer cannot be written

15.2 SET DEFAULT DIRECTORY

The Default Directory control routine informs you of and/or changes the default directory for the process. The entry point for this control routine is:

SYS$SETDDIR

The argument list consists of three parameters, all optional. The first is the address of the descriptor for the new default directory, or 0 if it is not to be changed. The second parameter is the address of a word to receive the length of the current default directory (or 0 if not wanted). The third is the address of the descriptor of a buffer to receive the current default directory string (or 0 if it is not wanted).

The new directory name string is checked for correct syntax.

You should restore the old default directory string to its original status unless you want the changed default directory string to last beyond the exit of your image.

The Default Directory control routine does not refer to fields in the user control blocks.

The completion status codes are listed below.

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_DIR	Directory string invalid
RMS$_IAL	Invalid argument list

15.3 SET DEFAULT FILE PROTECTION

The Default File Protection control routine informs you of and/or changes the default file protection for the process. The entry point for this control routine is:

SYS$SETDFPROT

The argument list consists of two parameters, both optional. The first is the address of a word giving the new default file protection specification (Section 6.4 describes the file protection specification), or 0 if it is not to be changed. The second parameter is the address of a word to receive the current default file protection specification, or 0 if it is not wanted.

You should restore the old default file protection specification unless you want the changed default to last beyond the exit of your image.

The Default File Protection control routine does not refer to fields in the user control blocks.

The completion status codes are described below.

Success:

RMS$_NORMAL	Operation successful

Failure:

RMS$_IAL	Invalid argument list

APPENDIX A

COMPLETION STATUS CODES

This appendix lists, in alphabetical order, the completion status codes that VAX-11 RMS can return, cross-referenced to any applicable service in which they can occur. The error codes are listed in the first part of this appendix and the success codes are listed at the end.

NOTES:

1. The errors that apply to the close service do not include errors that can arise due to setting of the SCF and SPL bits in the file-processing options field of the FAB.

2. The wait service has its own unique errors. This service can also return any status code of the awaited operation.

3. For completion status codes marked with an asterisk (*), the error can be the result of the buffer being written because it is needed now, even though the buffer needs to be written because of a previous put, update, or delete service.

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_ABO 000183EC Severe error	Operation aborted														•															
RMS$_ACC 0001C002 Error	File access error; the status value field (STV) contains an ACP error code														•															
RMS$_ACT 0001825A Error	File activity precludes operation	•	•	•	•	•	•			•	•	•	•	•	•	•		•	•	•			•		•	•	•			•
RMS$_AID 000183F4 Severe error	Bad area identification number field in allocation XAB			•			•			•					•															
RMS$_ALN 000183FC Severe error	Invalid alignment boundary type in allocation XAB			•						•																				
RMS$_ALQ 00018404 Severe error	Incorrect allocation quantity in allocation XAB; the value either exceeds the maximum allowed, or is equal to zero for the extend service			•						•																				
RMS$_ANI 0001840C Severe error	Records in a magnetic tape file are not ANSI D format										•			•																
RMS$_AOP 00018414 Severe error	Invalid allocation option in allocation XAB			•						•																				

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_ATR ⊛ 0001C0CC Severe error	Read error on file header; the status value field (STV) contains an ACP error code	•		•		•	•	•	•	•	•	•		•		•	•	•	•		•	•	•	•	•	•	•			•
RMS$_ATW ⊛ 0001C0D4 Severe error	Write error on file header; the status value field contains an ACP error code	•		•		•		•	•	•	•	•		•				•	•		•	•	•		•	•	•			•
RMS$_BKS 0001841C Severe error	Invalid bucket size in FAB			•																										
RMS$_BKZ 00018424 Severe error	Invalid bucket size (greater than 32) in the allocation XAB			•																										
RMS$_BLN 0001842C Severe error	Invalid value in block length field	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_BOF 00018198 Warning	File is already at beginning of the file (backspace operation)																					•		•						
RMS$_BUG 00018434 Severe error	Internal VAX-11 RMS error detected — contact a Software Specialist			•	•																									
RMS$_BUG_DAP 00018444 Severe error	DAP protocol violation — contact a Software Specialist	•	•	•	•	•			•		•			•		•		•	•					•		•			•	

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																											
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_BUG_DDI 0001843C Severe error	Invalid default directory. Internal VAX-11 RMS error; no recovery possible — contact a Software Specialist			•				•							•	•					•								
RMS$_CAA 0001848C Severe error	Cannot access argument list	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_CCF 0001C0DC Severe error	Cannot close file; the status value field (STV) contains an error code	•																											
RMS$_CCR 00018494 Severe error	Cannot connect RAB (only one record stream permitted for sequential files or MSE not set for indexed file)		•																										
RMS$_CDA 0001C0E4 Severe error	Cannot deliver AST; the status value field (STV) contains error code	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_CHG 0001849C Severe error	Attempt to change a key value when that attribute not set by the key definition XAB key option flag																									•			
RMS$_CHK 000184A4 Severe error	Index file bucket check — byte mismatch. The bucket has been corrupted. STV contains VBN of bucket.				•					•			•				•									•			
RMS$_CHN 0001C0EC Severe error	Channel assignment failure; the status value field (STV) contains an error code			•				•	•						•	•				•	•		•						

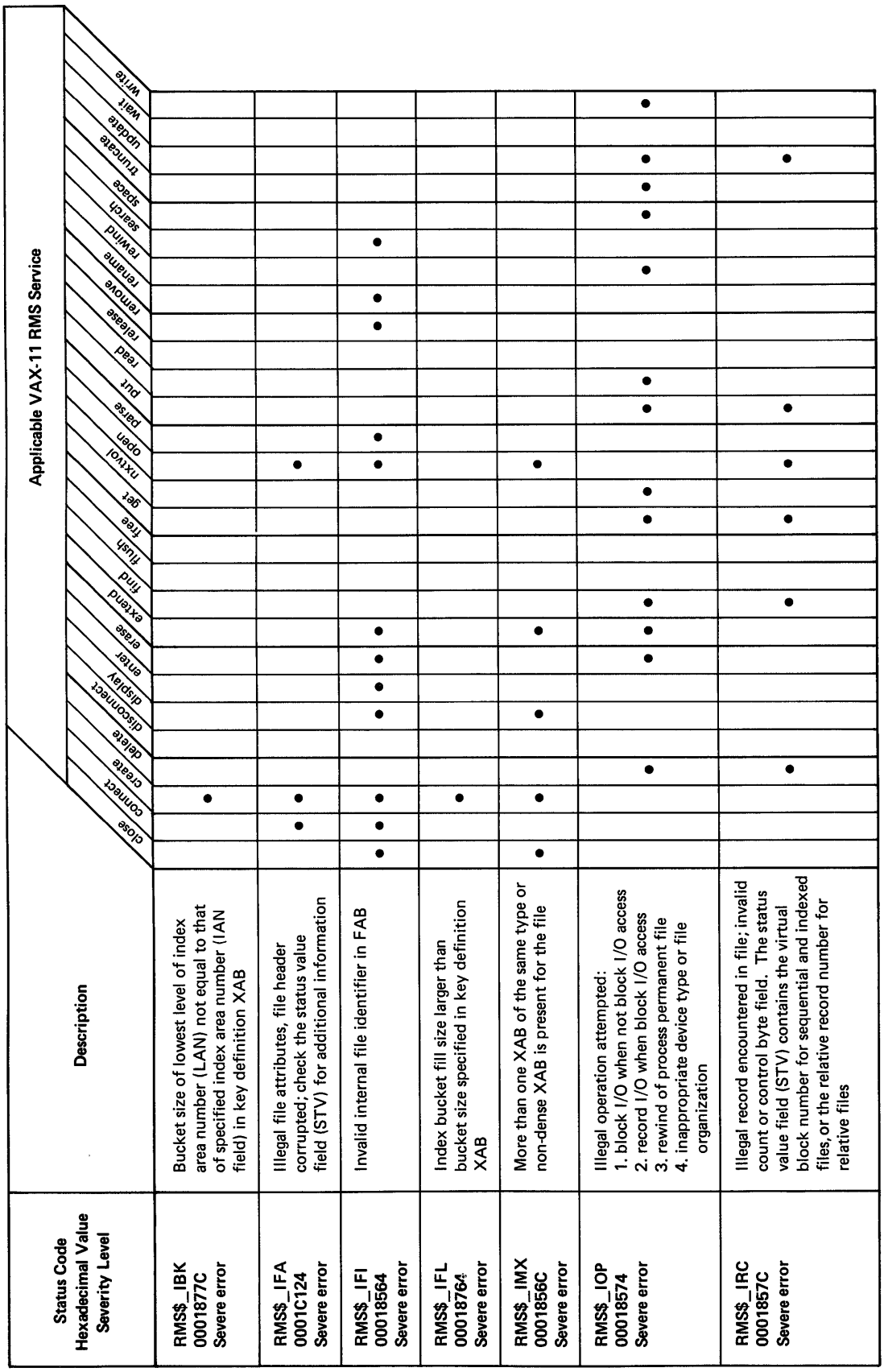
Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_COD 000184AC Severe error	Invalid type code in XAB	•		•			•			•						•													
RMS$_CRE 0001C00A Error	File create error; the status value field (STV) contains an ACP error code			•																									
RMS$_CUR 000184B4 Severe error	No current record; operation not immediately preceded by a successful get or find service				•																				•	•			
RMS$_DAC 0001C012 Error	File deaccess error during a close service; the status value field contains an ACP error code	•																											
RMS$_DAN 000184BC Severe error	Invalid data area number in key definition XAB			•																									
RMS$_DEL 00018262 Error	Record accessed by RFA record access mode has been deleted									•			•																
RMS$_DEV 000184C4 Severe error	Bad device or inappropriate device type for operation			•				•	•						•	•				•	•		•						
RMS$_DFL 0001876C Severe error	Data bucket fill size larger than bucket size specified in key definition XAB			•																									

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																											
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_DIR 000184CC Severe error	Error in directory name			•				•							•	•					•								
RMS$_DME 000184D4 Severe error	Dynamic memory exhausted; occurs only if the related I/O segment in the control region is full and the file is either a direct access process permanent file or the user has disallowed the use of the program region for I/O buffers to VAX-11 RMS	•	•	•	•	•	•	•	•	•	•	•		•	•	•	•	•	•		•	•	•	•	•	•	•		•
RMS$_DNA 000184DC Severe error	Invalid default file specification string address			•				•							•	•					•								
RMS$_DNF 0001C04A Error	Directory not found; the status value field (STV) contains an ACP error code			•				•							•	•				•	•		•						
RMS$_DNR 00018272 Error	Device not ready	•		•	•	•	•	•	•	•	•	•		•	•	•	•	•	•		•	•	•	•	•	•	•		•
RMS$_DPE 0001C03A Error	Device positioning error (applies only to magnetic tape); the status value field (STV) contains an ACP error code													•								•		•	•				
RMS$_DTP 000184E4 Severe error	Invalid data type in key definition XAB			•																									
RMS$_DUP 000184EC Severe Error	Duplicate key detected, key definition XAB key option flag not set to allow duplicate key values																•									•			

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_DVI 000184F4 Severe error	Invalid device identification in NAM block		•				•	•							•					•	•		•						
RMS$_ENT 0001C01A Error	Error entering file name in directory; the status value field (STV) contains an ACP error code						•														•								
RMS$_ENV 00018724 Severe error	Environment error; the code necessary to support the file organization or facility was not selected at system generation.		•												•														
RMS$_EOF 0001827A Error	End of file									•			•					•				•		•					•
RMS$_ESA 000184FC Severe error	Invalid expanded string area address in NAM block		•				•	•							•	•				•	•		•						
RMS$_ESL 00018714 Severe error	Invalid expanded string length in NAM block						•													•			•						
RMS$_ESS 00018504 Severe error	Expanded string area too short		•					•							•	•					•								
RMS$_EXP 000182C2 Error	File expiration date not yet reached	•	•																										

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_EXT 0001C022 Error	File extend error; the status value field (STV) contains an ACP error code								•								•											•	
RMS$_FAB 0001850C Severe error	Invalid FAB; block identifier field incorrect	•	•	•			•	•	•	•					•	•				•	•		•						
RMS$_FAC 00018514 Severe error	Operation not allowed by the value set in the file access field of the FAB				•				•	•			•				•	•							•	•		•	
RMS$_FEX 00018282 Error	File already exists			•																										
RMS$_FLG 0001851C Severe error	Invalid combination of values in key XAB FLG field; example: CHG or NUL for primary key			•																										
RMS$_FLK 0001828A Error	File is locked and therefore not available			•											•															
RMS$_FNA 00018524 Severe error	Invalid file specification string address in FAB			•				•							•	•					•									
RMS$_FND 0001C02A Error	Files-11 find function failed; the status value field (STV) contains an ACP error code																						•							

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_FNF 00018292 Error	File not found							•	•						•					•	•		•						
RMS$_FNM 0001852C Severe error	Syntax error in file name			•					•						•	•					•								
RMS$_FOP 0001853C Severe error	Invalid file processing options			•											•														
RMS$_FSZ 00018534 Severe error	Invalid fixed control area size in FAB (equal to 1 for print files)			•																									
RMS$_FUL 00018544 Severe error	Device full; cannot create or extend file			•					•								•												•
RMS$_IAL 0001854C Severe error	Invalid argument list	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_IAN 00018554 Severe error	Invalid index area number in key definition XAB			•																									
RMS$_IBF 00018754 Severe error	Illegal bucket format, STV contains bucket VBN				•					•			•				•									•			



Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_IBK 0001877C Severe error	Bucket size of lowest level of index area number (LAN) not equal to that of specified index area number (IAN field) in key definition XAB			•																									
RMS$_IFA 0001C124 Severe error	Illegal file attributes, file header corrupted; check the status value field (STV) for additional information		•	•												•					•	•							
RMS$_IFI 00018564 Severe error	Invalid internal file identifier in FAB	•	•	•		•	•	•	•	•						•	•				•	•							
RMS$_IFL 00018764 Severe error	Index bucket fill size larger than bucket size specified in key definition XAB			•																									
RMS$_IMX 0001856C Severe error	More than one XAB of the same type or non-dense XAB is present for the file	•		•		•			•							•													
RMS$_IOP 00018574 Severe error	Illegal operation attempted: 1. block I/O when not block I/O access 2. record I/O when block I/O access 3. rewind of process permanent file 4. inappropriate device type or file organization				•			•	•	•				•				•					•		•	•		•		
RMS$_IRC 0001857C Severe error	Illegal record encountered in file; invalid count or control byte field. The status value field (STV) contains the virtual block number for sequential and indexed files, or the relative record number for relative files				•					•				•				•										•		

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_ISI **00018584** **Severe error**	Invalid internal stream identifier in RAB				•	•					•	•	•	•	•			•	•	•			•		•	•	•	•	•	•
RMS$_KBF **0001858C** **Severe error**	Invalid key buffer address; not in access limits										•			•				•												
RMS$_KEY **00018594** **Severe error**	Invalid record key for random operation to a relative or indexed file										•			•				•												
RMS$_KNM **00018774** **Severe error**	Invalid key name buffer address in key definition XAB			•			•									•														
RMS$_KRF **0001859C** **Severe error**	Invalid key of reference in KRF field		•								•			•									•							
RMS$_KSI **00018784** **Severe error**	Key size too large to permit two keys in index bucket, STV value is key of reference for index			•												•														
RMS$_KSZ **000185A4** **Severe error**	Key size not equal to 4 (relative file) or key size too large (indexed file)										•			•				•												
RMS$_LAN **000185AC** **Severe error**	Invalid index lowest-level-bucket area number in key definition XAB			•																										

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																											
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_LBL **000185B4** **Severe error**	Tape is not ANSI labeled														•														
RMS$_LEX **0001878C** **Severe error**	Attempt to extend area containing an unused extent								•																				
RMS$_LNE **000185BC** **Severe error**	Logical name error; resulted in recursion or invalid process permanent file equivalence string			•				•							•	•					•								
RMS$_LOC **000185C4** **Severe error**	Invalid allocation location in allocation control XAB; this error status code is applicable only when allocation control XAB alignment option cylinder (CYL) or logical block number (LBN) is also specified			•					•																				
RMS$_MBC **00018734** **Severe error**	Invalid multi-block count; must not be greater than 127		•																										
RMS$_MKD **0001C032** **Error**	Files-11 could not mark file for deletion; the status value field (STV) contains an ACP error code	•						•																					
RMS$_MRN **000185CC** **Severe error**	Illegal value for maximum record number (negative) or relative key greater than maximum record number			•						•			•				•												
RMS$_MRS **000185D4** **Severe error**	Illegal value for maximum record size			•																									

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_NAM 000185DC Severe error	Invalid NAM block	•		•				•	•							•	•				•	•		•						
RMS$_NEF 000185E4 Severe error	Attempt to use the put service to a sequential file when not positioned to end of file																•													
RMS$_NET 0001874C Severe error	Network operation failed; the status value field contains DAP code	•	•	•	•	•			•		•		•		•		•	•						•		•			•	
RMS$_NID 000185EC Severe error	Cannot allocate internal index description: Insufficient room in space pool while attempting to open an indexed file		•	•						•			•		•														
RMS$_NMF 000182CA Error	No more files for a search or remove operation																			•			•							
RMS$_NOD 000185F4 Severe error	Node name error			•				•							•	•					•									
RMS$_NPK 000185FC Severe error	No primary key defined in key definition XAB when creating an indexed file			•																										
RMS$_ORD 00018604 Severe error	Chained XABs not in correct (ascending) order, not dense (sequential) when required, or different types of XABs are interleaved in the same XAB sub-chain			•																										

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service: close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_ORG 0001860C Severe error	Illegal file organization			●												●													
RMS$_PBF 00018614 Severe error	Invalid prompt buffer address										●			●					●										
RMS$_PLG 0001861C Severe error	Error in file prologue; file is corrupted		●				●			●	●			●		●		●									●		
RMS$_PLV 0001872C Severe error	Prologue version unsupported															●													
RMS$_POS 00018624 Severe error	Invalid key position (greater than MRS) in key definition XAB			●																									
RMS$_PRM 0001862C Severe error	Invalid file date field retrieved from XAB															●													
RMS$_PRV 0001829A Error	Privilege violation; access denied	●		●			●	●	●							●		●			●	●		●					●
RMS$_QUO 00018634 Severe error	Error in quoted string			●					●							●	●					●							

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_RAB 0001863C Severe error	Not a valid RAB; block identifier field incorrect		•		•	•					•	•	•	•	•			•	•	•			•		•	•	•	•	•	•
RMS$_RAC 00018644 Severe error	Illegal value in record access mode field of RAB										•			•				•												
RMS$_RAT 0001864C Severe error	Record attributes invalid in FAB			•												•														
RMS$_RBF 00018654 Severe error	Invalid record address																	•									•			•
RMS$_REF 0001875C Severe error	Invalid key of reference in XAB, greater than number in file, equal to 255			•			•				•			•		•														
RMS$_RER 0001C0F4 Severe error	File read error; the status value field (STV) contains an ACP error code										•			•				•	•								•	•		
RMS$_REX 000182A2 Error	Record already exists; in a random record access mode operation to a relative file a record was found in the target record cell																	•												
RMS$_RFA 0001865C Severe error	Invalid record's file address contained in FAB										•			•																

Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_RFM 00018664 Severe error	Illegal record format		•	•																									
RMS$_RHB 0001866C Severe error	Invalid record header buffer										•			•				•									•		
RMS$_RLF 00018674 Severe error	Invalid related file			•					•							•	•					•							
RMS$_RLK 000182AA Error	Record locked by another task										•			•				•											
RMS$_RMV 0001C0FC Severe error	Files-11 remove function failed; the status value field (STV) contains an ACP error code																				•	•							
RMS$_RNF 000182B2 Error	Record not found										•			•															
RMS$_RNL 000181A0 Warning	Record not locked				•								•							•							•		
RMS$_ROP 0001867C Severe error	Invalid record option		•								•			•															

(Columns: Applicable VAX-11 RMS Service)

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																													
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_RPL 0001C104 Severe error	Error while reading prologue; the status value field contains an ACP error code		•	•	•		•			•	•			•		•		•									•			
RMS$_RRV 00018684 Severe error	Invalid RRV record encountered in indexed file, file may be corrupted				•						•			•				•									•			
RMS$_RSA 000182DA Error	Record stream active; an attempt was made to issue a record operation request in an asynchronous environment to a record stream that has a request outstanding				•	•					•	•	•	•	•			•	•	•			•			•	•	•			•
RMS$_RSL 0001873C Severe error	Resultant string length field of NAM block invalid							•													•	•		•						
RMS$_RSS 00018694 Severe error	Resultant string area size is too small			•				•	•						•						•	•		•						
RMS$_RST 0001869C Severe error	Invalid resultant string area address in NAM block			•				•	•						•						•	•		•						
RMS$_RSZ 000186A4 Severe error	Illegal record size																•										•			•
RMS$_RTB 000181A8 Warning	Record too large for user buffer												•																		

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																												
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_RVU 0001868C Severe error	Error while updating RRVs, some paths to data may be lost																•									•			
RMS$_SEG 00018794 Severe error	Segmented key for key data type other than string			•																										
RMS$_SEQ 000186AC Severe error	Primary key of record to be written is not equal to or greater than key of previous record and RAC field set to SEQ																•													
RMS$_SHR 000186B4 Severe error	Invalid value in the file sharing field of FAB			•											•															
RMS$_SIZ 000186BC Severe error	Invalid key size specified in key definition XAB SIZ field; i.e., specified size exceeds maximum record size, not equal to defined length on binary and integer key data types, greater than 16 for packed decimal key data type, or equal to 0 for string or packed decimal			•																										
RMS$_SNE 0001879C Severe error	File sharing not enabled			•											•															
RMS$_SPE 000187A4 Severe error	File sharing data base page count exceeded		•	•											•															

Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_SPL 0001C042 Error	Spool or submit command file option to a close service failed	●																											
RMS$_SQO 000186C4 Severe error	Operation not sequential										●			●				●											
RMS$_SUP 000182D2 Error	Operation not supported; status value field contains DAP code	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●		●
RMS$_SYN 000186D4 Severe error	Syntax error in file specification			●					●							●	●					●							
RMS$_SYS 0001C10C Severe error	Error in system QIO directive; the status value field (STV) contains the directive or QIO status code	●		●	●	●		●		●	●	●		●	●	●		●	●		●	●	●	●	●	●	●		●
RMS$_TMO 000181B0 Warning	Time-out period expired										●			●					●										
RMS$_TRE 000186DC Severe error	Index tree error: file is corrupted				●						●			●				●									●		
RMS$_TYP 000186E4 Severe error	Error in file type			●					●							●	●					●							

Applicable VAX-11 RMS Service

Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
		Applicable VAX-11 RMS Service																											
RMS$_UBF 000186EC Severe error	Invalid user record area address												•					•											
RMS$_UPI 000187AC Severe error	SHR bit UPI not set when file sharing with FOP = BIO or FOP = BRO			•											•														
RMS$_USZ 000186F4 Severe error	Invalid user record area size												•					•											
RMS$_VER 000186FC Severe error	Error in version number			•				•							•	•					•								
RMS$_VOL 00018704 Severe error	Invalid volume number in allocation control XAB			•					•																				
RMS$_WBE 0001C12C Severe error	Error writing behind; the status value field (STV) contains an ACP error code	•			•				•	•	•		•				•	•				•		•	•	•			•
RMS$_WER ⊛ 0001C114 Severe error	File write error; the status value field (STV) contains an ACP error code	•			•				•	•	•		•				•	•				•		•	•	•			•
RMS$_WLD 00018744 Severe error	Invalid wildcard operation						•									•				•	•		•						

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																													
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_WLK **000182BA** **Error**	Device is write-locked	•		•	•	•		•	•	•	•	•		•				•	•		•		•		•	•	•		•
RMS$_WPL **0001C11C** **Severe error**	Error while writing prologue; the status value field (STV) contains an ACP error code			•						•								•									•		
RMS$_WSF **0001871C** **Severe error**	Working set full		•																												
RMS$_XAB **0001870C** **Severe error**	Not a valid XAB, not readable or writable, invalid code or length	•		•			•			•					•															
RMS$_CONTROLC **00010651** **Success**	Operation completed under Control C; terminal I/O may have been truncated										•			•				•	•											•
RMS$_CONTROLO **00010609** **Success**	Operation completed under Control O; terminal output may have been truncated																	•												•
RMS$_CONTROLY **00010611** **Success**	Operation completed under Control Y; terminal I/O may have been truncated										•			•				•	•											•
RMS$_CREATED **00010619** **Success**	File was created, not opened; used in conjunction with the CIF option			•																										

Status Code Hexadecimal Value Severity Level	Description	Applicable VAX-11 RMS Service																											
		close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_KFF 00018031 Success	Known file found														•														
RMS$_NORMAL 00010001 Success	Operation successful (synonym for RMS$_SUC)	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_OK_ALK 00018039 Success	Record already locked									•			•				•												
RMS$_OK_DEL 00018041 Success	Deleted record accessed successfully (NXR bit set in ROP field)									•			•																
RMS$_OK_DUP 00018011 Success	Record inserted has duplicate already on file																•												
RMS$_OK_IDX 00018019 Success	Record inserted, but error occurred on index update which could cause slow access																•									•			
RMS$_OK_LIM 00018051 Success	Retrieved record exceeds specified key value									•			•																
RMS$_OK_NOP 00018059 Success	Key XAB not filled in because file opened for block I/O						•								•														

Status Code Hexadecimal Value Severity Level	Description	close	connect	create	delete	disconnect	display	enter	erase	extend	find	flush	free	get	nxtvol	open	parse	put	read	release	remove	rename	rewind	search	space	truncate	update	wait	write
RMS$_OK_RLK 00018021 Success	Record locked but read anyway; locker set RLK bit in ROP field									•			•																
RMS$_OK_RNF 00018049 Success	Non-existent record accessed successfully (NXR bit set in ROP field)									•			•																
RMS$_PENDING 00018009 Success	Asynchronous operation not yet completed		•		•	•				•	•		•	•			•	•				•		•	•	•		•	
RMS$_SUC 00010001 Success	Operation successful (synonym for RMS$_NORMAL)	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
RMS$_SUPERSEDE 00010631 Success	Created file superseded an existing version			•																									

APPENDIX B

FILE/RECORD CONCEPTS AND FORMATS

VAX-11 RMS supports a variety of file organizations, record access modes, and record formats. The specific use of the file determines which one is best. The sections that follow outline the capabilities of each of the above items. Moreover, the Introduction to VAX-11 Record Management Services manual provides a complete description of these concepts.

B.1 FILE ORGANIZATIONS

File organization is the physical arrangement of the data in the file. You select the type of file organization you want when you create the file. Once a particular file organization is chosen, it remains fixed for the life of the file; you cannot change it. However, you can copy the file to another area, and in the process convert it to a different file organization (using the CONVERT utility).

VAX-11 RMS currently supports three file organizations:

- Sequential

  In the sequential file organization, records are in physical sequence. Each record, except the first, has another record preceding it, and each record, except the last, has another record following it. The physical order in which records appear is identical to the order in which they are written. A file of sequential organization can contain records of either fixed or variable length.

- Relative

  In the relative file organization, fixed-length positions, or cells, are created in the file beginning at the first record position and continue to the end of the file. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells that contain records. The relative file organization supports records that are either fixed or variable length.

- Indexed

  In the indexed file organization, the location of records is transparent to your program; VAX-11 RMS completely controls the placement of records in an indexed file. Presence of keys in each of the records governs this placement. Records may be fixed-length or variable-length; if the records are variable-length, the maximum record length may be specified and no record can exceed the maximum length when the record is

put in the file or updated. However, if a maximum length is not specified, records may be any length, but a record cannot cross bucket boundaries.

B.2 RECORD ACCESS MODES

The record access mode is the method of retrieving and storing records in a file. In contrast to file organization, which you cannot change once a file is created, you can use a different record access mode each time you process a record.

VAX-11 RMS provides three record access modes:

- Sequential

  VAX-11 RMS supports sequential record access mode for all device types and file organizations.

  When using the sequential record access mode, your program issues a series of requests for the next record. VAX-11 RMS interprets these requests in the context of the file organization. Thus, the organization of the file governs the order in which records are read or written; and the read or write continues, in a serial fashion, until processing of the file is completed. For sequential organization, VAX-11 RMS knows that every record after the first record is followed by another record until the end of the file (last record). For relative organization, VAX-11 RMS recognizes that empty cells can be interspersed among filled record cells and acts accordingly. On a read request, VAX-11 RMS ignores empty cells. For the indexed file, the presence of one or more indexes permits VAX-11 RMS to determine the order in which to process records in sequential access mode. Initially, your program must specify a key of reference (e.g., primary key, first alternate key, second alternate key, etc.) to VAX-11 RMS. Thereafter, VAX-11 RMS uses the index associated with that specified key to access records in the sequence represented by the entries in the index. Each successive record that VAX-11 RMS returns in response to a program read request contains a value in the specified key field that is equal to (when duplicate key values are allowed) or greater than that of the previous record returned.

- Record's File Address (RFA)

  You can use the RFA record access mode with any file organization, but only for disk files and only for read operations.

  The term "record's file address" means that every record in the file has a unique address. The type of file organization assigned to the file determines the format of this address.

  The most important feature of RFA record access mode is that the RFA of any record remains constant while the record remains in the file. VAX-11 RMS returns the RFA to you in the RAB when the record is read or written. (The record must be written using some record access mode other than RFA, since RFA access is available for read operations only. The RFA, however, is returned in the RAB as an output from a write operation.) Your program can then save this RFA for use later during the current execution of the program, or for use at any subsequent time.

- Random by Key

  VAX-11 RMS always supports random access by key for relative and indexed files. VAX-11 RMS also permits random access by relative record number for sequential disk files, but only if the records in the file are of fixed-length.

  In random access by key, your program, not the file organization, determines the order in which record access occurs. Each program request for a record must include the key value (relative record number for relative files and key of reference for indexed files) of the particular record to be accessed. This program randomly identifies, via the key value, any record in the file and VAX-11 RMS accesses that record. Your program can make successive requests for accessing records anywhere within the file.

  Each of your program read requests in random access mode to an indexed file must specify a key value and the index (e.g., primary index, first alternate key index, second alternate key index, etc.) that VAX-11 RMS must search. When the VAX-11 RMS finds the key value in the specified index, it reads the record that the index entry points to and passes the record to your program. Random access can be accomplished on any key by any of the following methods:

  1. Exact match of key values.

  2. Approximate match of key values (e.g., record key value greater than the program-supplied key value, or record key value greater than or equal to the program-supplied key value).

  3. Generic match of key values. Generic match is applicable to string data type keys only. A generic match is defined as a match on some number of leading characters in the key field. You determine the number specifying a search key which is smaller than the entire field.

  4. Combination of approximate and generic match.

  In contrast to read requests, which require a program-specified key value, program requests to write records randomly in an indexed file do not require the separate specification of a key value. All keys (primary and, if any, alternate key values) are in the record itself. When an indexed file is opened, VAX-11 RMS retrieves all key definitions stored in the file. Thus, VAX-11 RMS knows the location and length of each key field in a record. Before writing a record into the file, VAX-11 RMS examines the key values in the records, places the record in the file, and creates new entries in the alternate indexes. In this way, VAX-11 RMS ensures that the record can be retrieved by any of its key values.

Besides the above record access modes, a dynamic access facility is also available. This facility lets you switch the type of access while the file is being processed.

B.3 RECORD FORMATS

The record format is the way a record physically appears on the recording surface of the storage medium. VAX-11 RMS provides three different record formats.

- Fixed-length

  The term fixed-length record format refers to file records that are all equal in size; each record occupies an equal amount of space.

- Variable-length

  The term variable-length record format refers to file records that are not all the same size. VAX-11 RMS prefixes a count field to each record when it is written; this indicates to VAX-11 RMS how many bytes are in each individual record, and therefore the actual size of the record.

  VAX-11 RMS uses two types of variable-length records:

  - Disk files - V format

    Contain a 2-byte binary count field prefixed to each record

  - Tape files - D format

    Contain a 4-byte decimal ASCII count field prefixed to each record

- Variable with Fixed-Length Control (not supported for indexed files)

  This type of record format is similar to V or D format variable-length records, except that a variable with fixed-length control record also contains a fixed control area in addition to the variable-length data portion. A fixed control area lets you construct variable-length records that contain an additional fixed-length piece of data that will always be present and will have a "loose" association with the other contents of the record. The VAX-11 Text Editor (see the VAX-11 Text Editor Reference Manual) uses this type of record, in which a line sequence number is associated with each line of text. This is a "loose" association in that while stored together, they are separate for the purpose of processing.

Table B-1 summarizes the relationship between the VAX-11 RMS file organizations and their permitted record access modes and record formats.

Table B-1
File Organization Relationships
with Record Access Modes and Record Formats

File Organization	Record Access Mode Permitted			Record Format Permitted		
	Sequential	Random by Key	Random by Record's File Address	Fixed	Variable	Variable with Fixed-Length Control
Sequential	Yes	No[1]	Yes[2]	Yes	Yes	Yes
Relative	Yes	Yes[4]	Yes	Yes	Yes[3]	Yes[3]
Indexed	Yes	Yes	Yes	Yes	Yes[5]	No

[1] Random access by key (relative record number) for the sequential file organization is permitted only for the fixed-length record format on disk devices.

[2] Random access by RFA is permitted only on disk devices.

[3] Variable-length records in the relative file organization are stored in fixed-length cells; the size of each cell is the size needed to store the largest record permitted in the file.

[4] The key in relative file records is the relative record number.

[5] A record in an indexed file may not cross bucket boundaries.

B.4 FILES-11 DISK STRUCTURE

Files-11 is the term applied to the logical structure imposed on disk volumes. This structure provides the file access and allocation control mechanism for the volume. A disk volume is defined as an ordered set of blocks, with each block being an array of 512 eight-bit bytes.

In terms of the volume as a whole, the blocks are numbered consecutively in the range of 0 to n-1, where n is the highest number of blocks available on the volume (this depends on the type of disk volume in use). The number assigned to each volume-relative block is the logical block number (LBN). In terms of the individual file on the volume, the blocks are numbered consecutively from 1 to the total number of blocks assigned to the file. The number assigned to each file-relative block is the virtual block number (VBN).

Figure B-1 shows the difference between the scheme of blocks considered at the LBN and VBN levels. Two files, A and B, occupy ten blocks. File A, in relation to the volume, occupies LBNs 10 through 19; but, in relation to a file, this file occupies VBNs 1 through 10. Assume that when file B was created, it was allocated in two different areas, or clusters, with each cluster five blocks in length. The first cluster occupies LBNs 300 through 304, and the second cluster is at LBNs 29 through 33. But when viewed as an individual file, file B occupies consecutive VBNs 1 through 10, just as does file A. Further assume that file B was allocated in two separate extents (this can be done either explicitly at the request of the user, or implicitly by VAX-11 RMS due to the lack of enough contiguous disk space or a default by extent size). Even though files A and B both have the same VBNs, the corresponding blocks are different since the VBNs relate to the block's placement within the individual file, not to the volume as a whole.

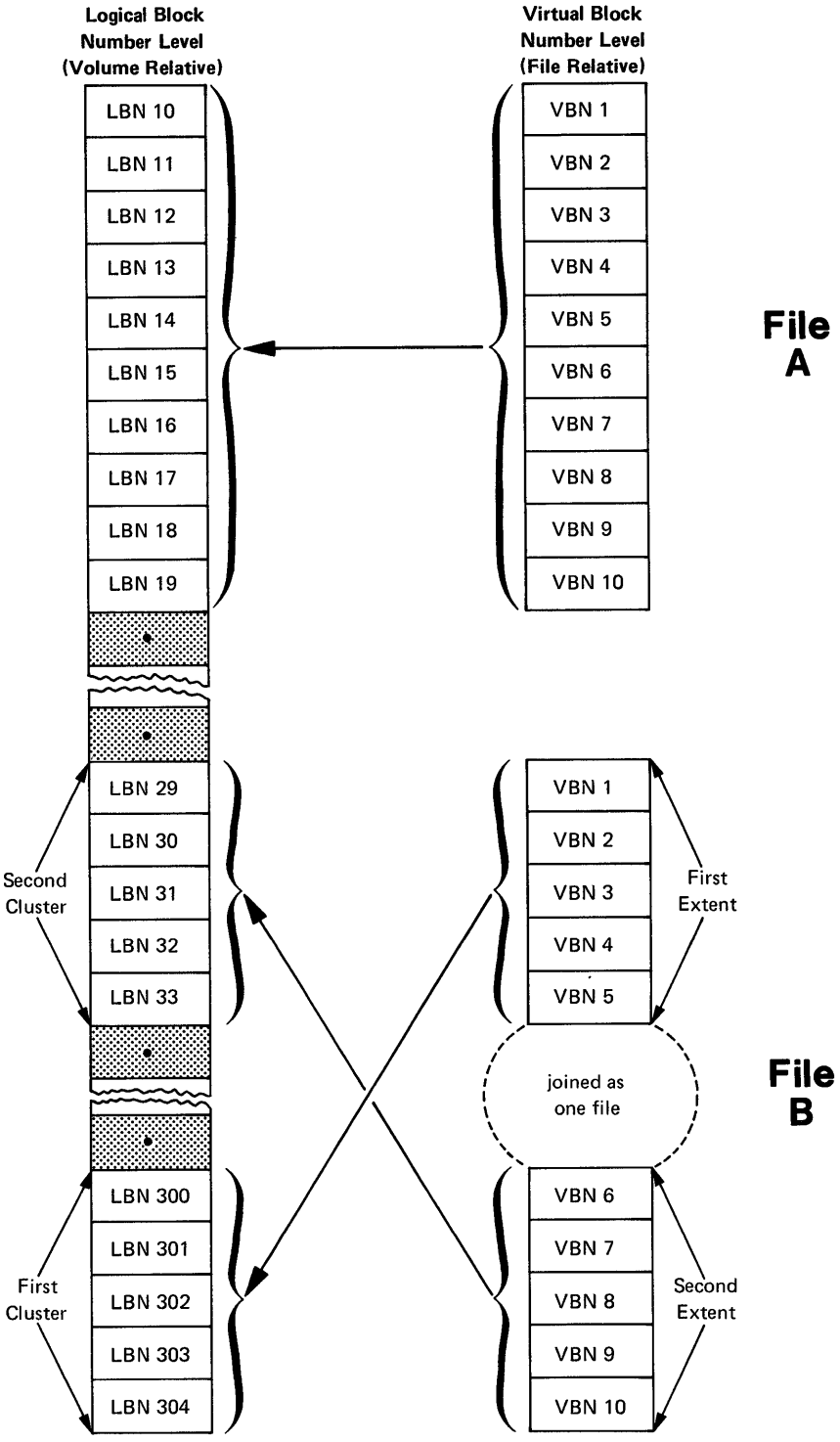


Figure B-1 Logical and Virtual Block Numbers

Every Files-11 volume has an index file, which is created when the volume is initialized. This index file provides the means of identifying, to VAX/VMS, that the volume is a Files-11 structure, and contains the access data for all files on the volume. The index file is listed in the master file directory (MFD) as INDEXF.SYS;1 and contains the following information:

- Bootstrap block

  The volume's bootstrap block is VBN 1 of the index file. Volume relative, it is LBN 0. If the volume is a system device, this block contains a program that loads the operating system into memory. If the volume is not a system device, this block contains a program that displays a message that the volume is not the system device, but rather a device that contains only user files.

- Home block

  The home block identifies the volume as a Files-11 volume, establishes the specific identity of the volume, and serves as the entry point into the volume's file structure. When the volume is part of a volume set, the home block also contains the volume set name and the relative column number. The home block is VBN 2 of the index file. The LBN for the home block is the first good block (physically readable and writeable) on the volume found in the home block search sequence. The search sequence is as follows:

  ```
  1+n * delta
  ```

  n is in the range of 0,1,2,....

  The delta is computed from the geometry of the volume such that if the volume is viewed as a three-dimensional space, the search sequence will travel down the body diagonal of the space. The dimensions included in the search delta are sectors (s), tracks (t), and cylinders (c), according to the rules in Table B-2, to handle the cases in which either one or two dimensions of the volume have a size of 1.

Table B-2
Search Delta Geometry

Geometry			Delta
s	t	c	
–	1	1	1
1	–	1	1
1	1	–	1
–	–	1	s+1
–	1	–	s+1
1	–	–	t+1
–	–	–	(t+1)*s+1

  In most cases, LBN 1 will be a good block, and therefore LBN 1 will be the home block.

- Backup home block

The backup home block is a second copy of the home block. It permits the volume to be used even if the primary home block is destroyed.

The cluster that contains the backup home block maps into the index file at VBN x*2+1 through x*3, where x is the volume cluster factor.

- Index file bit map

  The index file bit map controls (with the information contained in the home block) the allocation of file headers, and thus the number of files on the volume. The bit map contains a bit for each file header that is allowed on the volume. If the value of a bit for a given file header is 0, then a file can be created with this file header. If the value is 1, then the file header is already in use. The index file bit map starts at VBN x*4+1 of the index file and continues through VBN x*4+m, where m is the number of blocks that are necessary to contain the bit map, and x is the storage map cluster factor. The starting LBN for the index file bit map is recorded in the home block.

- File headers

  The major portion of the index file is made up of file headers. A file header exists for each file on the volume and describes the properties of the file, such as file ownership, creation date and time, and file protection. The file header contains all the information necessary for access to the file, including the location of the file's extents.

Besides the index file, Files-11 maintains nine other files to control the volume structure. Just as with the index file, these files are created when a new volume is initialized.

The storage bit map file controls the available space on a volume, and is listed in the MFD as BITMAP.SYS;1. It contains a storage control block, which consists of summary information intended to optimize Files-11 allocation, and the bit map itself, which lists the availability of individual blocks.

The bad block file is listed in the MFD and BADBLK.SYS;1, and is simply a file containing a list of all the bad blocks on the volume.

The master file directory itself (the MFD) is listed in the MFD as 000000.DIR;1. The MFD is the root of the volume's directory structure, and lists the ten files that control the volume structure (these ten files are called the known files) plus any user files on the volume.

The core image file is listed in the MFD as CORIMG.SYS;1, and its use is operating system dependent. In general, it provides a list of the files for the operating system to use, for example, as swap areas or overlay areas.

The free space file is listed in the MFD as FREFIL.SYS;1. This file allows individual Files-11 implementations to use an alternative scheme of space allocation that is more complex than using the storage bit map file alone.

The set list file is listed in the MFD as VOLSET.SYS;1. It is used only on relative volume 1 of a tightly coupled volume set. This file contains a list of the volume labels of the volumes in the volume set.

The backup log file is listed in the MFD as BACKUP.SYS;1. It contains a history log of volume and incremental backups performed on this volume.

The continuation file is listed in the MFD as CONTIN.SYS;1. It is used as the extension for the file identifier when a file crosses from one volume of a loosely coupled volume set to another volume. It allows a multivolume file to be written sequentially with only one volume mounted at a time.

The pending bad block file is listed in the MFD as BADLOG.SYS;1. This file contains a list identifying suspected bad blocks on the volume that are not currently contained in the bad block file (BADBLK.SYS;1).

Each file on the volume, including the ten known files, is uniquely named by a file identifier, which is a 48-bit binary value (three words). The first word provides the file number, which locates the file on the volume. The file number is in the range of 1 through 2^2^4-1. Once a file is deleted, its number can be reused for another file. The file number identifies the file header within the index file associated with the file. The second word is the file sequence number, which identifies the current use of a file number. This prevents any attempt to use a file identifier for a file that has already been deleted and replaced by a file with the same file number. The third word is the relative volume number. It identifies which volume of a multivolume file contains the portion of the file that is of interest.

B.4.1 **Files-11 Directories**

Files-11 provides directory files to allow for accurate access to files on disk devices. A directory is a file that lists the identification and location of files owned by a particular user. Each user allowed access to a VAX/VMS system has an entry in the system authorization file defining the user identification code (UIC) and default user file directory (UFD).

Directory names can take any of three formats. Each format requires that the directory name be enclosed in either square brackets ([and]) or angle brackets (< and >). The closing bracket must match the opening bracket. The formats are as follows:

1. UIC-similar format

   A UFD can be referred to in a format similar to that for a UIC: for example, [abc,xyz], where abc is a group number and xyz is the member number. This refers to a UFD of the name abcxyz.DIR;1 in the MFD. If you specify less than three characters for either abc or xyz, they are left zero-filled. Therefore, if a UFD is specified in a UIC fashion as [26,1], the directory that is searched is 026001.DIR;1 (DIR is the file type for the directory).

   A UFD of this format is usually owned by a user with a corresponding UIC. This, however, is not required, since UIC and UFD ownerships are independent.

2. Alphanumeric character string

   A UFD can also be a 1- to 9-alphanumeric character string. This character string can be the same as your user name or account name, or any valid character string that you request or the system manager assigns you. For example, if a directory is specified as [010PAY], the directory 010PAY.DIR;1 is searched.

3. Subdirectories in addition to the character string UFD

   When UFDs are referred to using the character string format, further hierarchical levels of directories can be expressed as subdirectories. A subdirectory level is expressed by adding a period (.) to the character string for the UFD, followed by the specification for the subdirectory. For example, [010PAY.DED] is the specification for the UFD named 010PAY.DIR;1 and a subdirectory of DED.DIR;1.

   The maximum number of directory levels is eight: one UFD and seven subdirectories. (Combined with the master file directory, this is in effect a 9-level hierarchy.) In the directory specification [010PAY.DED.YTD], 010PAY is the UFD, DED is the first level subdirectory, and YTD is the second level subdirectory.

   No maximum is placed on the number of different hierarchies of directories you can create or access.

   The master file directory is created when the volume is initialized. Subdirectories and UFDs are created with the CREATE command using the DIRECTORY qualifier (see the VAX/VMS Command Language User's Guide).

The maximum number of entries that a single directory can hold ranges from 15000 to approximately 40000, depending on the length of the file specifications. In general, using several subdirectories to list a large number of files results in more efficient access than listing all files in one large directory.

The directory file itself is structured as a contiguous file with sequential organization. The records are variable-length, do not cross block boundaries, and contain no carriage control attributes.

B.5 MAGNETIC TAPE HANDLING

VAX-11 RMS supports the magnetic tape structure defined by American National Standards Institute standard ANSI X3.27-1977, entitled Magnetic Tape Labels and File Structure for Information Interchange. This section describes the processing of magnetic tape files and magnetic tape labeling and file structuring format.

Magnetic tapes containing ANSI labels are coded in ASCII format, and on 9-track tape drives only.

ANSI standard X3.27-1977 allows any of the following combinations:

1. Single file on a single volume
2. Single file on more than one volume
3. Multiple files on a single volume
4. Multiple files on more than one volume

Items 2 and 4 above constitute a volume set.

Magnetic tape affords sequential access only. Therefore, only one user can have access to a given volume set at any one time, and only one file in the volume set can be open for processing at a time. Access protection is performed on a volume-set basis. For volumes produced by DIGITAL systems, the owner identifier field of the volume label determines access rights (see Section B.5.1).

B.5.1 **Volume Label**

The volume label is always the first label on every tape volume, and serves to uniquely identify the volume and its owner. Figure B-2 presents the form of the volume label, and Table B-3 defines the contents of the fields in this label.

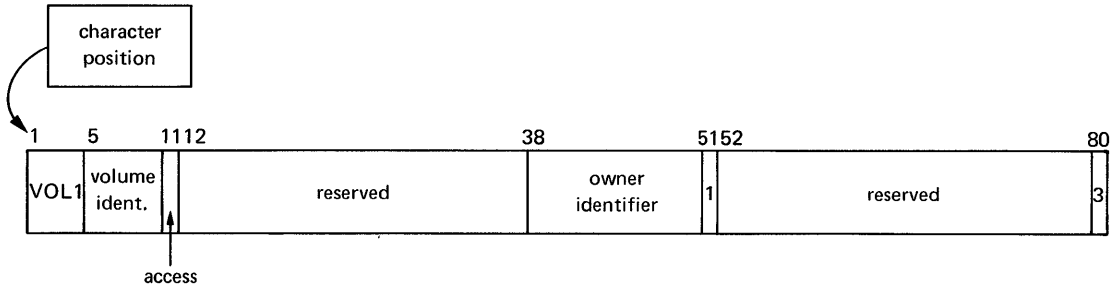


Figure B-2 Volume Label Format

Table B-3
Volume Label Contents

Character Position	Field Name	Length (in bytes)	Contents
1-3	Label identifier	3	Alphabetic characters VOL
4	Label number	1	Numeric character 1
5-10	Volume identifier	6	Volume label; can be any alphanumeric or special character. This field must not be all spaces.
11	Accessibility	1	Volume protection; for the purpose of compatibility with the standards of some non-DIGITAL systems. A space (as used by DIGITAL systems) indicates no restrictions. To achieve other than read-only access to those volumes that contain a non-space character in this field, the override switch must be used at mount time.
12-37	Reserved	26	Spaces
38-50	Owner identifier	13	Volume ownership; the contents of this field are system dependent and are used for volume protection. See details following table for further amplification.
51	DIGITAL standard version	1	Numeric character 1
52-79	Reserved	28	Spaces
80	Label standard version	1	Numeric character 3

Owner identifier field

All magnetic tape volumes produced on DIGITAL systems contain the following in the first three character positions (CP 38-40) of the owner identifier field:

D%m

In the above, D% are both constant, and m represents a machine code, interpreted as follows:

8 - PDP-8
A - PDP-10
B - PDP-11
C - VAX-11/780
F - PDP-15
K - DECSYSTEM-20

If the machine code in character position (CP) 40 is the character C, the meaning of the remainder of the owner identifier field translates as follows:

1. Owner has read and write privileges:

   CP 41-45 group number (ASCII characters)

   CP 46-50 member number (ASCII characters)

2. Owner has read and write privileges; group has read privileges:

   CP 41-45 group number (ASCII characters)

   CP 46 member number high-order digit, zone encoded; therefore, a 0 in the high-order position is the character A, while a 9 is the character J

   CP 47-50 remaining four characters of member number (ASCII)

3. Owner has read and write privileges, world and group have read privileges:

   CP 41 group number high-order digit, zone encoded

   CP 42-45 remaining four characters of group number

   CP 46 member number high-order digit, zone encoded

   CP 47-50 remaining four characters of member number

4. Owner and group have read and write privileges:

   CP 41-45 group number (ASCII characters)

   CP 46-50 blank

5. Owner and group have read and write privileges, system and world have read privileges:

   CP 41 group number high-order digit, zone encoded; therefore, a 0 in the high-order position is the character A, while a 9 is the character J

   CP 42-45 remaining four characters of group number (ASCII)

   CP 46-50 blank

6. All categories have full privileges

   CP 41-50 blank

If the machine code is other than the character C, full privileges are granted unless CP11 is nonblank, in which case you must use the MOUNT command with a qualifier of /OVERRIDE=ACCESSIBILITY.

B.5.2 File Header Label

A file header label precedes every individual file on the tape, and serves to uniquely identify the file and describe its contents. Actually, two different file header labels precede each file; a HDR1 label, for identification, and a HDR2 label, which acts as an extension to the HDR1 label and describes the characteristics of the records in the file. Figure B-3 and Table B-4 present the format and define the contents of the HDR1 label, and Figure B-4 and Table B-5 present the format and define the contents of the HDR2 label.

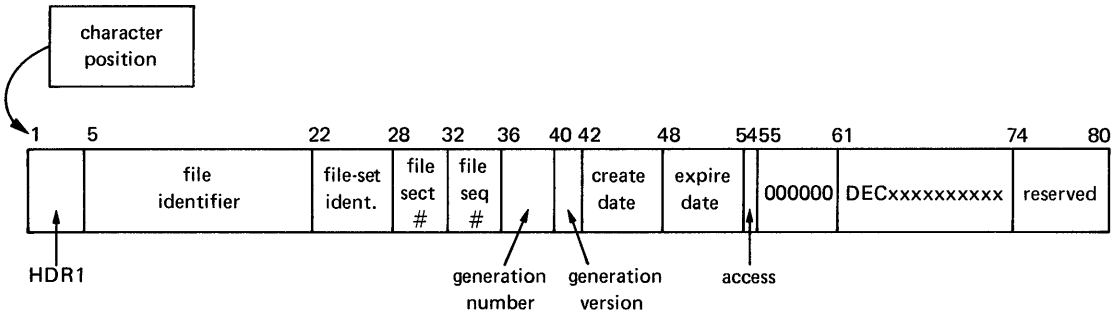


Figure B-3 HDR1 Label Format

Table B-4
HDR1 Label Contents

Character Position	Field Name	Length (in bytes)	Contents
1-3	Label identifier	3	Alphabetic characters HDR to indicate a file header
4	Label number	1	Numeric character 1
5-21	File identifier	17	Any alphanumeric or special characters; see details following table for further amplification
22-27	File-set identifier	6	Same as the volume identifier of the VOL1 label of the first volume of a multi-volume set
28-31	File section number	4	Numeric characters; starts at 0001 and increments by 1 for each additional volume used by the file. This field indicates the positional order of this volume with respect to the first volume on which the file begins.
32-35	File sequence number	4	File number within the volume set for this file; consists of numeric characters, and starts at 0001. This field indicates the position of this file with respect to the first file of the set.
36-39	Generation number	4	Numeric characters; indicate the unique edition of a file. See discussion following table.
40-41	Generation version	2	Numeric characters; indicate the version number of a particular version of a file. See discussion following table.

Table B-4 (Cont.)
HDR1 Label Contents

Character Position	Field Name	Length (in bytes)	Contents
42-47	Creation date	6	Julian date, in the form of yyddd (right-justified with leading space). The creation date is set to the date on which the file is created. If a creation date does not apply to this file, 00000 is used (right-justified with a leading space).
48-53	Expiration date	6	Julian date, in the form of yyddd (right-justified with a leading space). If no expiration date is specified, the value is set to the value of the creation date; therefore, the file immediately is expired.
54	Accessibility	1	File security; for the purpose of compatibility with the standards of some non-DIGITAL systems. A space (as used by DIGITAL systems) indicates no restrictions. A non-space character in this field indicates that the override switch must be used at mount time in order for the user to gain access to the file.
55-60	Block count	6	Always 000000 for the HDR1 label
61-73	System code	13	Identification code of the system that produced the file. The 3-character constant DEC appears in positions 61 through 63, followed by the name of the system. For example, DECFILE112 indicates VAX/VMS, and DECFILE11 indicates a PDP-11. The name is padded with spaces.
74-80	Reserved	7	Spaces

File identifier field

The file identifier field consists of the alphabetic characters A through Z, and the numeric characters 0 through 9. ANSI standard X3.27-1977 allows special characters in this field; however, VAX/RMS translates these characters to Z.

The character preceding a period (.), or a maximum of nine characters if no period is present, constitutes the file name. The three

characters following immediately after the period (or characters 10 through 12 if no period is present) constitute the file type. On output, the file name and file type are automatically separated by a period, and written to the file identifier field left-justified. The version number is generated through the generation number and generation version fields.

Generation number and generation version fields

These two fields are mapped to create the file version number, according to the following formula:

version number=(generation number -1) * 100 + generation version +1

For example, suppose the generation number is 11 and the generation version is 9:

(11 -1) * 100 + 9 + 1

The formula produces a version number of 1010.

At output, the reverse is true. The present version number creates the generation number and generation version, according to the following formula and a remainder produced during the calculation.

$$\text{generation number} = \frac{\text{version number} - 1}{100} + 1$$

In the calculation, any remainder in version number -1 is ignored for the generation number. For example, suppose the version number is 100:

$$\frac{100 - 1}{100} + 1$$

The formula produces a generation number of 1. The remainder of 99 is ignored in the calculation of this generation number, but becomes the generation version.

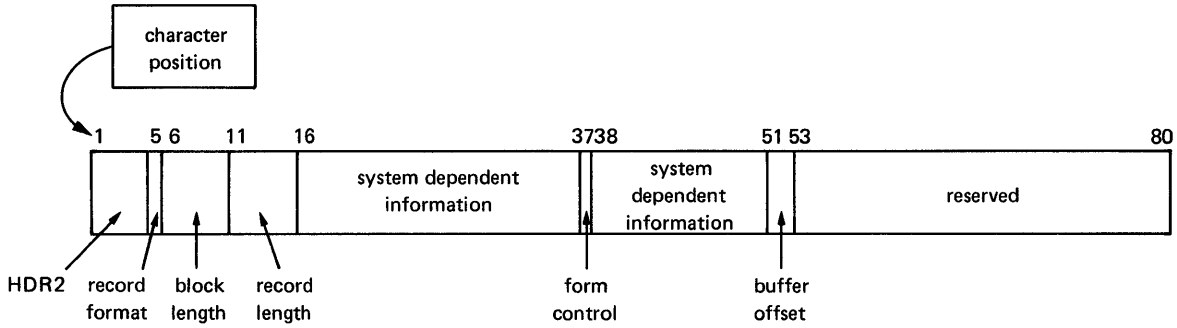


Figure B-4 HDR2 Label Format

Table B-5
HDR2 Label Contents

Character Position	Field Name	Length (in bytes)	Contents
1-3	Label identifier	3	Alphabetic characters HDR to indicate a file header
4	Label number	1	Numeric character 2
5	Record format	1	Character Definition F fixed-length D variable-length U undefined S segmented Undefined record format cannot be used on tapes created for interchange with non-DIGITAL systems. The S for segmented record formats returns as a U (undefined record format).
6-10	Block length	5	Five numeric characters that specify the maximum number of characters per block.
11-15	Record length	5	Numeric characters indicating the record length for fixed-length records.
16-36	System dependent information	21	If this file was created on a VAX/VMS system, then CP 16 through 35 contain 20 bytes of Files-11 attributes that override information in other fields of the HDR2 label; CP 36 contains a space.
37	Form control	1	Defines the carriage control applied to the records in this file, as follows: Character Definition A First byte of record contains FORTRAN control characters

Table B-5 (Cont.)
HDR2 Label Contents

Character Position	Field Name	Length (in bytes)	Contents
			Character Definition M The record contains all form control information. space line feed/ carriage return is to be inserted between records.
38-50	System dependent information	13	If this file was created on a VAX/VMS system, then CP 38 through 49 contain 12 bytes of Files-11 attributes that override information in other fields of the HDR2 label; CP 50 contains a space
51-52	Buffer offset	2	Numeric characters
53-80	Reserved	28	Spaces

B.5.3 End of File and End of Volume Labels

Magnetic tape volumes contain trailer labels, which can be either of two pairs of labels, depending on whether the tape has an end-of-volume or end-of-file condition.

- End of volume

  The end-of-volume label pair consists of an EOV1 label and an EOV2 label. These labels occur only when a file is continued onto another volume. This applies to both of the following categories of magnetic tape volumes:

  - single file, multivolume

  - multifile, multivolume

  The formats of the EOV1 and EOV2 labels are identical to their respective HDR1 and HDR2 labels, except that the label identifier field (CP 1-3) contains EOV and the block count field (CP 55-60) contains the number of data blocks since the last tape mark (a delimiter between labels and file data). This file data recorded since the last tape mark is known as a file section and may, in fact, be only a portion of the entire file (this occurs on a multivolume file). A file section cannot have sections of other files interspersed.

- End of file

  The end-of-file label pair occurs at the end of every file recorded on a magnetic tape volume. The formats of the

end-of-file labels (EOF1 and EOF2) are identical to the formats of the EOV1 and EOV2 labels, except that the label identifier field contains EOF.

B.5.4 Arrangement of Labels and Data

Figures B-5 through B-8 describe the organization of the different volume sets and indicate where the different labels appear. In these figures, the following legends apply:

bot = beginning of tape

* = tape mark

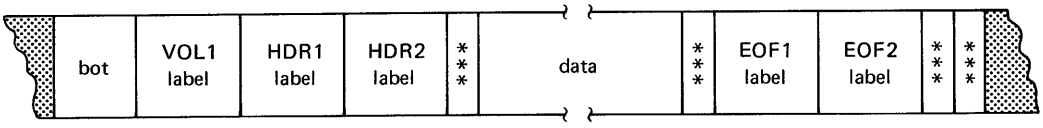


Figure B-5 Single File, Single Volume

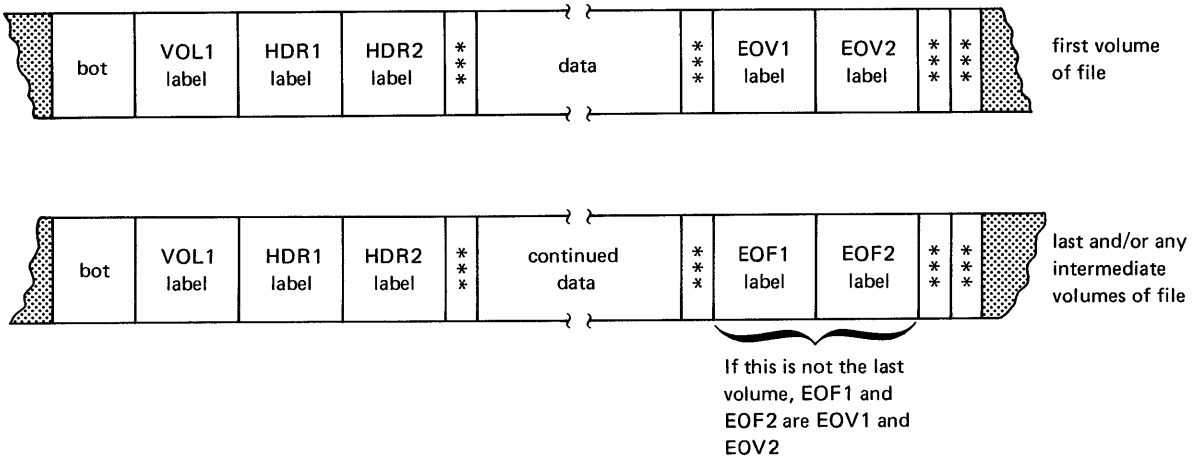


Figure B-6 Single File, Multivolume

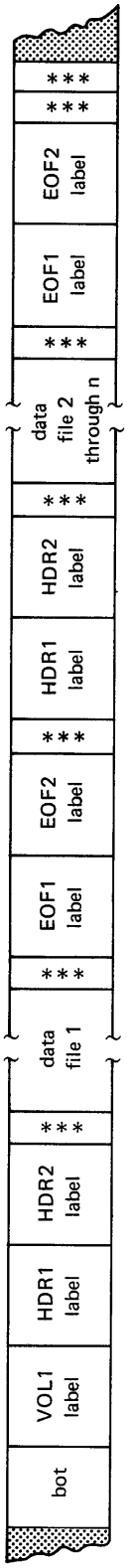
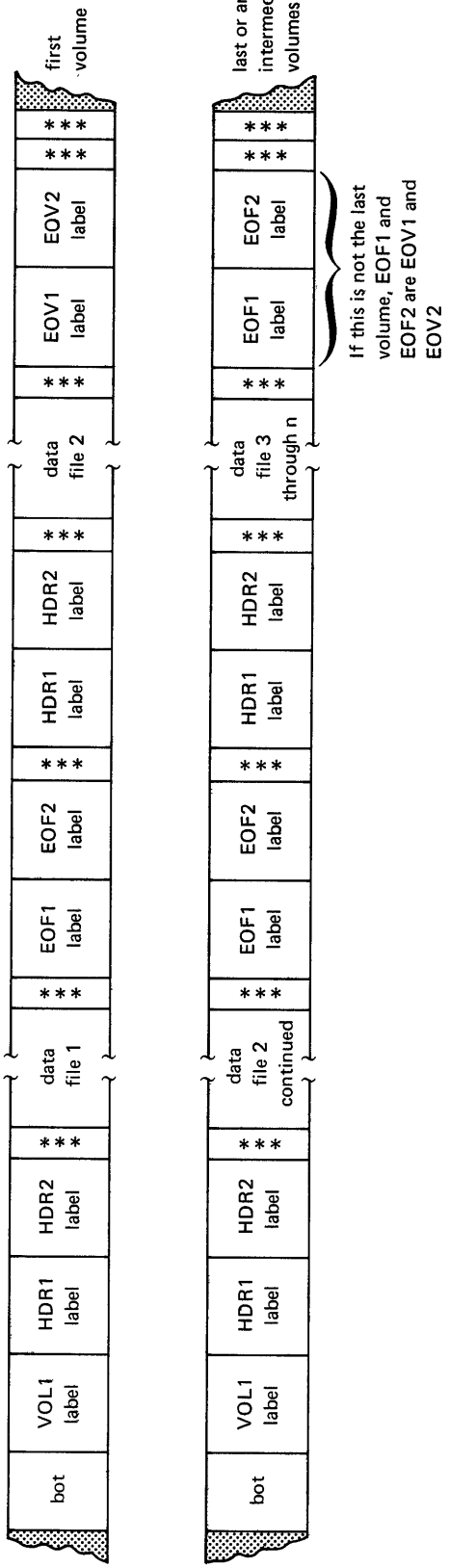


Figure B-7 Multifile, Single Volume



NOTE:
The continuation of a data file between volumes may not actually occur in data file 2; it occurs in any file which happens to be the last file on the particular volume. Data file 2 is an arbitrary choice for this figure.

Figure B-8 Multifile, Multivolume

APPENDIX C

FILE SPECIFICATION PARSING

To obtain a fully qualified file specification, VAX-11 RMS parses the primary file name string and optionally parses the default file name and related file string (if these are provided as input) as described in Section 8.2. Each of these three file name strings must have one of the following syntaxes:

The following sections describe each of these file specification syntaxes in detail.

LOGICAL-NAME-OR-FILE-NAME SYNTAX

```
{ logical-name-or-file-name }
{ quoted-string-specification }
{ full-file-specification }
```

```
logical-name-or-file-name = {logical-name}
                            {file-name   }
```

NOTE: The logical-name takes precedence.

```
                          {alpha-char}
logical-name = {digit     }...
               {dollar-sign}
               {underscore}
```

NOTES:

1. The logical-name is 1 to 63 alphanumeric characters, including the special characters dollar sign ($) and underscore (_).
2. For this to be a logical name, there must be a corresponding entry in the process, group, or system logical name table.
3. If the first character of a potential logical name is an underscore, it will simply be removed by the translation process that replaces a logical name with its equivalence string. The input string, minus the leading underscore, is thus guaranteed not to be a logical name.

```
                    {alpha-char}
file-name = {digit  }...
            {nothing}
```

NOTE: The file-name is 0 to 9 alphanumeric characters. Lowercase alphabetic characters are converted to their uppercase equivalents.

QUOTED-STRING-SPECIFICATION SYNTAX

quoted-string-specification = node-specification quoted-string

node-specification = logical-name-or node-name { access-control-string | nothing } node-delimiter

logical-name-or-node-name = { logical-name | node-name }

NOTE: The logical name takes precedence.

logical-name = same as description in Section C.1 plus the following additional notes.

NOTES:
1. The logical name must translate to an equivalence string that is a node-specification. That is, the equivalence string must terminate with two colons (::) (the node-delimiter) and the access-control-string is optional.
2. If an access-control-string has been found and the logical name translation yields an equivalence string that contains another access-control-string, the new access-control-string is discarded but the new logical-name-or-node-name component of the string is retained. That is, the first access-control-string found during the parse takes precedence over any others that may be produced during logical name translation.

node-name = { upper-case-alpha | digit } ...

NOTE: The node-name is 1 to 6 alphanumeric characters.

access-control-string = string-delimiter ASCII-char string-delimiter

NOTE: The maximum length of the access-control-string is 44 characters. See the Network User Guide for format.

node-delimiter = :: (double colon)

quoted-string = string-delimiter ASCII-char string-delimiter

NOTE: The maximum length of the quoted-string is 127 characters. See the Network User Guide for format.

string-delimiter = " (quotation mark)

ASCII-char = any character from the ASCII character set except a quotation mark.

FULL-FILE-SPECIFICATION SYNTAX

```
full-file-specification = {node-specification}
                          {nothing           }
                          {logical-name-or-device-name}
                          {nothing                    }
                          {directory-specification}
                          {nothing                }
                          {file-name-specification}
                          {nothing                }
                          {file-type-specification}
                          {nothing                }
                          {file-version-specification}
                          {nothing                   }
```

```
node-specification = see previous explanation
```

```
logical-name-or-device-name = {logical-name} device-delimiter
                              {device-name  }

NOTE: The logical-name takes precedence.
```

```
logical-name = see previous explanation
```

```
device-name = device-mnemonic {controller-name} {unit-number}
                              {nothing        } {nothing    }

NOTE: For this to be a valid device name, there must be a
      corresponding entry in the system device data base.
```

```
device-mnemonic = uppercase-alpha uppercase-alpha

NOTE: The device-mnemonic is currently limited to two
      characters.
```

```
controller-name = uppercase-alpha

NOTE: If you omit the controller-name, the default is the character "A".
```

```
unit-number = digit {digit  } ...
                    {nothing}

NOTES: 1. The unit-number is 1 to 5 digits, in the range of
          0 to 65535.

       2. If you omit the unit-number, the default is 0.
```

INDEX

VAX-11 Record Management
Services Reference Manual
AA-D031B-TE

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 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 the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — Do Not Tear - Fold Here —