digital

VAX–11 Record Management Services User's Guide

Order No. AA-D781C-TE

VAX11

March 1980

This document contains detailed information on using the capabilities of VAX-11 Record Management Services efficiently. Typical examples are provided to illustrate programming concepts.

VAX-11 Record Management Services User's Guide

Order No. AA-D781C-TE

SUPERSESSION/UPDATE INFORMATION: This document supersedes the document of the same name, Order No. AA-D781B-TE, published February 1979.

OPERATING SYSTEM AND VERSION: VAX/VMS V2.0

SOFTWARE VERSION: VAX/VMS V2.0

digital equipment corporation · maynard, massachusetts

First Printing, August 1978
Revised, January 1979
Revised, March 1980

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

PREFACE

MANUAL OBJECTIVES

The intent of this manual is to present some of the different uses of the VAX-11 Record Management Services (VAX-11 RMS), so you can tailor the various components and routines to suit your record management and record processing needs.

INTENDED AUDIENCE

This manual is intended for VAX/VMS users who want to develop a basic understanding of how to use VAX-11 RMS I/O routines within their programs. VAX-11 MACRO programmers generally use the VAX-11 RMS routines directly within their programs. High-level language programmers normally use the I/O facilities of their particular language to utilize a subset of VAX-11 RMS facilities. However, they may also use VAX-11 RMS directly through a call facility within their language.

This manual is aimed at VAX-11 MACRO programmers. It is assumed that you are familiar with and understand the VAX-11 MACRO conventions for constructing symbols and the use of numbers, operators, and expressions.

STRUCTURE OF THIS MANUAL

The information in this document is structured as follows:

Chapter 1 provides an overview of the salient features of the data record file organizations that can be created, displayed, and maintained by using VAX-11 RMS. This information will help you to determine the type of file organization best suited to your data record management requirements.

Chapter 2 describes the VAX-11 RMS routines and the user control blocks defined within your program, which are used to communicate between your program and the VAX-11 RMS routines.

Chapter 3 describes file specification syntax and the file specification defaults.

Chapter 4 describes how you create and process data record files by sequential access mode with three file organizations.

Chapter 5 describes how you create and process ,data record files by using random access mode.

Appendix A provides additional programming examples.

Appendix B describes the RMS File Analyzer.

ASSOCIATED DOCUMENTS

A prerequisite to this manual is the _Introduction to VAX-11 Record Management Services Manual_, which describes in detail the concepts of file organization, record access modes, record formats, and other concepts required for your understanding of VAX-11 RMS file construction. You should have available a copy of the _VAX-11 Record Management Services Reference Manual_. This document contains the complete description of the components of VAX-11 RMS, and therefore constitutes a source reference for the materials presented in this user's guide.

Other manuals allied to this document are:

- _VAX/VMS Primer_
- _VAX/VMS System Services Reference Manual_
- _VAX/VMS Command Language User's Guide_
- _VAX-11 MACRO Language Reference Manual_
- _VAX-11 BLISS Language Reference Manual_

SUMMARY OF TECHNICAL CHANGES

This manual has been revised to reflect VAX-11 RMS support for wild card characters and uppercase translation of logical names.

CHAPTER 1

FILE GUIDELINES: DETERMINE YOUR NEEDS

The VAX-11 Record Management Services (VAX-11 RMS) are system routines that provide an efficient and flexible means of accessing files and their records. The VAX-11 RMS routines speed up and simplify the task of program development.

1.1 THE RATIONALE FOR RECORD MANAGEMENT

As a user writing application programs, you need to create programs that will (1) accept new input, (2) read or modify data, and/or (3) produce output in some meaningful form. These programs can be, at times, somewhat difficult to produce, because the operations required in handling the data can be complex. However, many of these operations are basically the same, with only minor modifications needed depending on the operation. Therefore, generalized routines that encompass a wide variety of functions can be very useful to you in dealing with your file and record management programming needs. VAX-11 RMS provides such generalized routines.

VAX-11 RMS routines are an integral part of the operating system; they are always there. You need not perform any special linking or declaring of global entry points to access the routines since a simple reference to a routine generates the appropriate call. Calls to VAX-11 RMS routines are consistent with the VAX/VMS calling standard; arguments are passed and results and errors are returned in the standard VAX/VMS fashion.

Because the file organization is fixed for the life of the file, it is very important that you decide, before you begin to write your program, which file organization best meets your requirements. The following questions should help you determine your file organization requirements.

- How will the records be accessed? Will the whole file or only selected records be processed? Will the records be accessed randomly? Will the records be accessed by other nodes in a network?

- What kind of record maintenance is needed? Must records be updated, added, or deleted?

- What is the record format? How large are the records; are they all the same size? What is their maximum size?

- What is the total size of the file? Is this size fixed or can it be extended?

- Where will the file reside? Will the medium be tape, disk, or cards. Will the file be written to a line printer or terminal?

As these questions indicate, many issues affect your choice of file organization. Often, the choice is not clear-cut. Table 1-1 lists some of the advantages and disadvantages of the three types of file organizations: sequential, relative, and indexed.

Table 1-1
File Organizations: Advantages and Disadvantages

File Organization	Advantages	Disadvantages
Sequential	Uses disk and memory efficiently: minimum disk overhead, block-boundary crossing Provides optimal usage if the application accesses all records sequentially on each run Provides flexible record format Allows data to be stored on many different types of media, in a device-independent manner Allows easy file extension	Allows sequential access only for some high-level languages Allows records to be added only to end of file Allows sharing by multiple, concurrent users, but only with user's implemented synchronization. (The exception is 512-byte fixed-length records; VAX-11/RMS manages the synchronization for such files).
Relative	Allows sequential and random access by record number for all languages Allows random record deletion and insertion Allows records to be read- and write-shared	Allows data to be stored on disk only Requires that programs contain a record cell for each relative record number allocated; therefore, files may be sparsely populated Requires that record cells be the same size Allows record insertion only to empty cells (or at the end of the file)

(continued on next page)

Table 1-1 (Cont.)
File Organizations: Advantages and Disadvantages

File Organization	Advantages	Disadvantages
Indexed	Allows sequential and random access by key value for all languages Allows random record deletion and insertion Allows records to be read- and write-shared Allows variable-length records to change length on update Allows easy file extension	Allows data to be stored on disk only Requires more disk space Uses more of the central processing unit to process records. Generally requires mulitple disk accesses to prrocess a record.

CHAPTER 2

VAX-11 RMS STRUCTURES AND INTERFACE

The facilities of VAX-11 Record Management Services (VAX-11 RMS) are available at run time through the calling of record management procedures. Communication with the VAX-11 RMS routines is by means of user control blocks defined within your program. This chapter provides an introduction to these routines and control blocks, and the macro instructions that facilitate their use.

2.1 USER CONTROL BLOCKS

VAX-11 RMS uses data structures called control blocks to communicate between your program and the VAX-11 RMS routines.

The VAX-11 RMS routines also create their own internal data structures, reflecting the information in your control blocks. These internal data structures reside in the process control region, in what is called the I/O segment.

You set up fields in the control blocks to reflect exactly what operations you want to perform, and then call the routine. The routine uses these fields as input to perform the requested action and, as necessary, uses these fields again to return status and other related information. The amount of information your program exchanges with VAX-11 RMS (both as input and output) depends on the nature of your request and the file attributes.

Table 2-1 lists the control blocks that are part of your program interface with VAX-11 RMS.

You must allocate space for these control blocks within your program. You can do this either at assembly time or run time. VAX-11 RMS provides macro instructions for the assembly-time allocation and initialization of the control blocks, shown in the Macro Name column of Table 2-1. At run time, you can directly manipulate the control blocks through either the defined symbolic offsets or the "store" macro instructions. For efficiency, and to prevent a warning message from the assembler, align each control block on a longword boundary.

In general, you must allocate one File Access Block (FAB) for every open file in your program, and one Record Access Block (RAB) for each individual record stream connected to a FAB. (More than one RAB can be connected to each FAB simultaneously.) The Extended Attribute Blocks (XABs) and the Name Block (NAM) are optional, depending on whether you need the information they provide and the functions they perform.

Table 2-1
Control Blocks

Structure	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)	Contain 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. The variable xxx is a 3-character XAB-type specification.

2.2 VAX-11 RMS ROUTINES

The VAX-11 RMS routines execute in executive mode. VAX-11 RMS protects its internal data structures and buffers from destruction by user programs, and ensures that files will be left in an orderly state. When your program exits, an I/O rundown routine closes all files, writing buffers and file attributes as required, even when the exit is the result of a severe error.

VAX-11 RMS routines are integrated in a straightforward manner. Within your program, you place a call to the appropriate routines. Generally you make these calls with run-time macro instructions. At run time, the expanded code of these macro instructions causes calls to be made to the appropriate routines, which refer to the appropriate control blocks. These calls are consistent with the VAX-11 calling standard. You can specify the parameters with keywords; you can list them in any order or omit the keywords entirely.

When you call a routine, you set up an argument list to define the associated control block (FAB or RAB) and any optional completion routines to be called if an error occurs.

The operations performed by VAX-11 RMS routines are classified as either file oriented or record oriented, requiring the address of a FAB and RAB respectively as the control block argument in a call to any of them.

Table 2-2 summarizes the essential macro instructions for run-time processing.

Table 2-2
Macro Instructions for Run-Time Processing

Category	Macro Name	Service
File Processing	$CREATE	Creates and opens a new file of any organization (sequential, relative, or indexed)
	$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 indexed file
	$FIND	Locates and positions to a record and returns its RFA
	$CONNECT	Connects record stream to a file
	$DISCONNECT	Disconnects a record stream from a file
	$RELEASE	Unlocks a record by its RFA
	$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 3

SPECIFYING THE FILE TO BE PROCESSED

A file is a logically related collection of records. All the information that the operating system reads and writes on behalf of users' requests is defined in terms of files and records.

File processing is influenced by the hardware device that performs the actual data transfer (reading or writing). Devices are classified as:

- Mass storage devices
- Record-oriented devices

Mass storage devices provide a way to save the contents of files on a magnetic medium, called a volume. Files that are thus saved can be accessed at any time and updated, modified, or reused. Disks and tapes are mass storage devices.

Record-oriented devices read and/or write only single physical units of data at a time, and do not provide for permanent storage of the data. Terminals, printers, and card readers are record-oriented devices. Printers and card readers are also called unit record devices. In certain cases, magnetic tapes are treated as record oriented devices.

3.1 FILE SPECIFICATIONS

File specifications provide the system with all the information it needs to identify a unique file or device.

File specifications have one of the following formats:

- node::device:[directory]filename.type;version
- node::"foreign-file-spec"
- node::"task-spec"

You must use the punctuation marks and brackets to separate the fields of the file specification. Either matching square brackets or angle brackets may delimit the directory specification. The type and version specifications may be separated by either a period (.) or a semi-colon (;). The fields and their contents are listed below.

Field	**Contents**
node	Node name and optional access control string
device	Device name
directory	Directory name and optional subdirectory names
filename	File name
type	File type
version	File version number
"..."	Designates a program to communicate with on a remote node or designates a file specification that is not to be parsed locally.

Directory names, file names, file types, and version numbers apply only to files on disk or tape devices. For record-oriented devices (terminals, printers, and card readers), only the device name field of the file specification is required; fields following it are ignored. Blanks, tabs, and null characters are accepted but ignored in file specifications.

You may use wild card characters in file specifications. These are more fully discussed in Section 3.1.5. The ellipsis [...] and minus sign [-] wild card characters can be used only in the directory name field of a file specification. The asterisk (*) and percent sign (%) wild card characters can be used in the following fields of a file specification:

- Directory name
- File name
- File type
- File version number

Appendix C of the VAX-11 Record Management Services Reference Manual contains a rigorous explanation of the entire syntax for file specifications. The following sections, however, provide sufficient information for you to have a basic understanding of how to supply file specifications.

3.1.1 Network Nodes

Each computer system in a DECnet network is uniquely identified by a 1- through 6-alphanumeric character node name. Optionally, a node name may be followed by an access control string enclosed in quotes (") and the entire node specification is identified by two colons (::). An access control string consists of a username, password, and optional account name separated from each other by one or more spaces and/or tabs. Its total length is 3 through 42 characters. You include an access control string in a node specification when you want to login at the remote node as a specific user for the file access operation. If you omit the access control string, the default DECnet account (if established) is used. The following are examples of node specifications.

```
BOSTON::
BOSTON"COWENS CELTICS"::
BOSTON"COWENS CELTICS NBA"::
```

In addition, you may define a logical name for a node specification and then use it in file specifications. Logical names are described in detail in Section 3.3.

For complete details on the use of node name specifications, see the DECnet-VAX User's Guide.

3.1.2 Devices

Each physical hardware device in the system has a unique identification, in the format:

```
     devcu:
```

In this format, dev is a mnemonic for the device type, c is a controller designation and u is a unit number.

Table 3-1 lists the valid device types and their mnemonics.

The controller and unit number identify the location of the actual device within the hardware configuration of the system. Controllers are designated with alphabetic letters A through Z. Unit numbers are decimal numbers from 0 through 65535.

The maximum length of the device name field, including controller and unit number, is 15 characters. You must follow a device name with a colon (:).

A complete device name specification is called a physical device name. You can specify physical device names to indicate an input or output device for a program. Or, you can equate a physical device name to a logical name and use a logical name to refer to a device. Logical names are described in detail in Section 3.3.

Table 3-1
Device Names

Mnemonic	Device Type
CR	Card Reader
CS	Console Storage Device
DB	RP04, RP05, RP06 Disk
DD	TU58, Cassette Tape
DL	RL02, Cartridge Disk
DM	RK06, RK07 Cartridge Disk
DR	RM03, RM05 Disk
DY	RX02 Floppy Diskette
LA	LPA11-K Laboratory Peripheral Accelerator
LP	Line Printer
MB	Mailbox
MS	TS-11 Magnetic Tape
MT	TE16, TU45, TU77 Magnetic Tape
NET	Network Communications Logical Device
OP	Operator's Console
RT	Remote Terminal
TT	Interactive Terminal
XA	DR11-W General Purpose DMA Interface
XF	DR32 Interface Adapter
XJ	DUP11 Synchronous Communications Line
XM	DMC11 Synchronous Communications Line

3.1.3 Directories

A user file directory (UFD) is a file that lists the identifications and locations of files on a disk device that belong to a particular user. The UFD is listed in the volume's master file directory (MFD). The MFD is the root of the volume's directory structure, and also lists the reserved files for the volume.

Directory names apply to files on magnetic tape and disk devices. They are expressed in one of three formats where each format requires that you enclose the directory name in either square brackets ([and]) or angle brackets (< and >). The closing bracket must match the opening bracket. The formats for specifying directory names are as follows:

- As a 1- through 9-alphanumeric character string representing a UFD name.
- As a two-part number separated by a comma (,) in the format of a user identification code (UIC).
- As a UFD name followed by one or more subdirectory names, each preceded by a period (.). Each subdirectory name represents a unique subdirectory level of the UFD and has the same syntax as a UFD name.

3.1.3.1 **Alphanumeric Character String Format** - The character string used to specify a UFD 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 you specify a directory as [010PAY] the directory 010PAY.DIR;1 is searched. (DIR is the file type for a directory, and 1 is the version number.)

3.1.3.2 **UIC Format** - You can refer to a UFD in a format similar to that for a UIC: for example, [abc,xyz], where "abc" is a group number and "xyz" is a member number. To specify a UFD in this format, separate the group number from the member number with a comma. If you specify less than three characters for either "abc" or "xyz", they are left zero-filled. Therefore, if you specify a UFD in a UIC format as [26,1], the directory searched is 026001.DIR;1.

UIC directories have corresponding names in alphanumeric format. The group and member numbers are each left zero-filled (if necessary). For example:

```
     [122001]
```

The directory name for the UFD specified in this command is equivalent to the specification [122,1].

A directory in this format is usually owned by a user with a corresponding UIC. However, this may not always be the case, as UIC and directory ownership are independent.

3.1.3.3 **Subdirectories** - When UFDs are referenced 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.

There is no maximum 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.[1]

3.1.4 File Names, File Types, and Version Numbers

File names, file types, and version numbers uniquely identify files within directories.

A file name is a 1- through 9-alphanumeric character string that identifies a file. When you create a file, you can assign it a file name that is meaningful to you.

A file type is a 1- through 3-alphanumeric character string that extends a file name. Usually, a file type name is chosen to suggest the contents of the file.

File types must be preceded with a period (.).

The system uses a set of standard file types, by convention, to identify various classifications of files, and to provide default file types in many commands. Table 3-2 is a list of file types.

Table 3-2
Default File Types

File Type	Contents
ANL	Output file for the ANALYZE command
BAS	Input source file for the VAX-11 BASIC compiler
B2S	Input source file for the PDP-11 BASIC-PLUS-2/VAX compiler
B32 or BLI	Input source file for the VAX-11 BLISS-32 compiler
CBL	Input file containing source statements for the PDP-11 COBOL-74/VAX compiler

(continued on next page)

1. See the VAX/VMS Command Language User's Guide for an explanation of this command and any others that appear throughout this manual.

Table 3-2 (Cont.)
Default File Types

File Type	Contents
CMD	Compatibility mode indirect command file
COB	Input file containing source statements for the VAX-11 COBOL-74 compiler
COR	Input source file for the PDP-11 CORAL 66/VAX compiler
COM	Command procedure file to be executed with the @ (execute procedure) command, or to be submitted for batch execution with the SUBMIT command
DAT	Input or Output data file
DIF	Output listing created by the DIFFERENCES command
DIR	Directory File
DIS	Distribution list for the MAIL command
DMP	Output form the DUMP command
EDT	Initialization command input file for EDT
EXE	Executable program image created by the linker
FOR	Input file containing source statements for the VAX-11 FORTRAN compiler
FTN	Compatibility Mode FORTRAN IV PLUS source file
HLB	Help text library file
HLP	Help text source file
JNL	Journal file output form PATCH utility
JOU	Journal file/audit trail from EDT
L32	Precompiled Librrary for VAX-11 Bliss-32
LIB	Input file containing VAX-11 COBOL-74 source Statements to be copied into another file during compilation
LIS	Listing file created by a language compiler or assembler; default input file type for PRINT and TYPE commands
LOG	Batch job output file
LST	Compatibility mode listing file
MAC	MACRO-11 source file

(continued on next page)

Table 3-2 (Cont.)
Default File Types

File Type	Contents
MAI	Mail message file
MAP	Memory allocation map created by the linker, invoked by the LINK command
MAR	VAX-11 MACRO source file
MDL	Maynard Definition Language (Language-independent structure definitions)
MLB	Macro library
NEW	Any new source file
OBJ	Object file created by a language compiler or assembler
ODL	Overlay descriptor file
OLB	Object module library
OLD	Any old source file
OPT	Options for input to the LINK command
PAR	A SYSGEN parameter file
PAS	Input file containing source statements for the VAX-11 PASCAL compiler
R32 or REQ	VAX-11 BLISS-32 source file required for compilation
STB	Symbol table file created by the linker
SYS	System image
TEC	TECO indirect command input file
TLB	Text library
TMP	Temporary file
TMx	SOS temporary file ("x" is a digit)
TXT	Input file for text libraries or output file for mail command
UPD	Update file of changes for a VAX-11 source program; also input to the SUMSLP editor

Version numbers are decimal numbers from 1 through 32767 that differentiate between versions of a file. When you update or modify a file, the system saves the original version for backup and increments the version number of the modified file by 1.

Version numbers must be preceded with a semicolon (;) or a period (.)

3.1.5 Wild Card Characters

As noted in the VAX/VMS Record Management Services Reference Manual, wild card characters can be used in the directory name, file name, file type, and file version number fields of a file specification, when given to a program designed to accept them. One purpose of wild card characters is to refer to a group of files by a more general file specification, rather than by each of the specific file specifications. There are four characters (or strings of characters) that can be used as wild card characters. These are the asterisk (*), the percent sign (%), the ellipsis (...), and the minus sign (-).

An asterisk is used to match the missing component of a file specification with an alphanumeric character string of any length (including the null string). A percent sign is used to match any single alphanumeric character in that particular position (the null string does not match). The asterisk and the percent sign can be combined in many ways. For example, the sequence:

```
     A*E%B*.B*;*
```

matches a group of file specifications in which the file name starts with an "A" followed by a string of zero to "n" characters, followed by an "E", followed by a single character, followed by a "B", followed by a string of zero to "n" characters. The file type begins with a "B" and is followed by a string of zero to two characters. Finally, the version number in this group will be any and all versions of that file, beginning with the highest version number.

The ellipsis and minus sign wild card characters are aids to searching, or traversing, directory hierarchies. Both the ellipsis and the minus sign allow you to refer to directories in a relative positional sense, rather than by an absolute name for the first directory or group of directories. The ellipsis enables you to select files from all directory levels from a specified level downward. The minus sign, on the other hand, enables you to search up the hierarchy, rather than down. A single minus sign will send the search back up one level from the current default directory level.

3.2 DEFAULT FILE SPECIFICATIONS

Defaults are valuable because they are easy to use, and they let you enter as short a file specification as possible. The less you enter, the less chance you have of making a syntax error, or an incorrect or invalid specification. The default values were selected because they conform to the most applicable and frequently used practices.

When you enter a file specification and omit fields in it, the system supplies values for these fields.

The node name defaults to your local node. The device and directory names, if omitted, default to your current default disk and directory name. These are initially established when you log in to the system, based on an entry under your user name in the system authorization file.

You can find your default disk and directory name by using the SHOW DEFAULT command. For example:

```
     $ SHOW DEFAULT

       DBA1:[PAY01]
```

The response to the command indicates that the current default disk is DBA1, and the directory name is PAY01.

You can change the disk and directory name defaults with the SET DEFAULT command.

System defaults also apply for fields other than the device and directory name. Table 3-3 summarizes the defaults that apply to each field in the file specification.

Table 3-3
File Specification Defaults

Field	Defaults
node	Local system
device	Default device established at login, or by the SET DEFAULT command; almost always a disk device If a controller designation is omitted, it defaults to A. If a unit number is omitted, it defaults to 0. (The ALLOCATE and SHOW DEVICES commands, however, treat a device name that does not contain controller and/or unit numbers as a generic device name.)
directory	Directory name established at login or by the SET DEFAULT command, or next higher level in a subdirectory
file name	No defaults are applied to file names in input file specifications, except for those commands accepting multiple input file specifications, where, for specifications other than the first, the file name (as well as node, device, directory, and file type) is often defaulted from the previous input file specification. Most commands default output file names based on the file name of an input file
file type	Various commands apply defaults for file types, based on the standard file type conventions summarized in Table 3-2
file version	For input files, the system assumes the most recent version (that is, the highest number) For output files, the system increases the version number by 1 for existing files, and supplies a version number of 1 for new files

File specification defaults can be applied in other ways as well. Chapter 8 of the VAX-11 Record Management Services Reference Manual describes an advanced method for applying defaults to file specifications. This method involves the use of defaults built into your program, the default file specification string address and size fields of the FAB, and the related file NAM block.

3.3 LOGICAL NAMES

The use of logical names is an effective technique for achieving device independence within a program. The logical names provide a convenient shorthand method for specifying files that you refer to frequently.

The ASSIGN command equates a file specification to a logical name. For example, assume that, external to your program code, you specify the following:

```
$ ASSIGN DBA0:[PAYROLL]MASTER.DAT OLD_MASTER:
$ ASSIGN DBA1:[PAYROLL]MASTER.DAT NEW_MASTER:
```

The ASSIGN command equates the logical name OLD_MASTER to file MASTER.DAT on disk device DBA0 in the directory PAYROLL. The logical name NEW_MASTER equates to file MASTER.DAT on disk device DBA1 in the directory PAYROLL on that device. (This file specification is known as the equivalence string for the logical name.) Subsequently, within your program, you can specify these files as follows:

```
INFILE:   $FAB FNM=<OLD_MASTER:>
OUTFILE:  $FAB FNM=<NEW_MASTER:>
```

Alternatively, you can make the following external assignments:

```
$ ASSIGN  INDEVICE:[PAYROLL]  OLD_MASTER:
$ ASSIGN  OUTDEVICE:[PAYROLL] NEW_MASTER:
$ ASSIGN  DBA0: INDEVICE:
$ ASSIGN  DBA1: OUTDEVICE:
```

Note in the example above that logical name equivalence strings are not always full file specifications. Furthermore, note that the use of logical names is recursive; that is, the equivalence string for a given logical name may contain a further logical name. This assignment would require a slight modification to the program to specify the same files. You would have to indicate the file name and file type in the FAB file specification. For example:

```
INFILE:  $FAB FNM=<OLD_MASTER:MASTER.DAT>
OUTFILE:  $FAB FNM=<NEW_MASTER:MASTER.DAT>
```

Depending on the degree of flexibility you need, numerous other alternatives are possible in assigning logical names. The best alternative is determined according to individual circumstance.

Logical names and their equivalence name strings can each have a maximum of 63 characters, and can be used to form all or part of a file specification. If only part of a file specification is a logical name, specify the logical name in place of the device name in subsequent file specifications.

For example, a logical name can be assigned to a device name, as follows:

```
$ ASSIGN DMA1:  BACKUP
```

After this ASSIGN command, you can use the logical name BACKUP in place of the device name field when referring to files on the disk.

You may also create a logical name for a node name or node specification. This is useful for reducing the length of a long node specification and for protecting the password field of an access control string. For example:

```
$ DEFINE DAVE "BOSTON""COWENS CELTICS""::"
$ TYPE DAVE::DBB2:[REPORT]JAN80.DOC
```

The logical node name DAVE, defined above, has an equivalence string of BOSTON"COWENS CELTICS":: which is substituted for the node name DAVE in the TYPE command.

RMS does not allow the use of lowercase logical names in file specifications. If you try to use a lowercase logical name, RMS will convert to uppercase the entire string prior to attempting translation and will continue to do so on each successful translation thereafter. RMS will accept and ignore the use of blanks, tabs, and null characters in file specifications and logical name assignments. Such characters will be ignored by RMS, unless they are enclosed in quotes.

3.3.1 Logical Name Tables

Logical names and their equivalence names are maintained in three logical name tables:

- Process logical name table -- contains entries that are local to a particular process. When you equate a file specification to a logical name with the ASSIGN or DEFINE command, the logical name, by default, is placed in this table.
- Group logical name table -- contains entries that are qualified by a group number. These entries can be accessed only by processes that execute within the same group number in their UIC. To make an entry in the group logical name table, you use the /GROUP qualifier with the ASSIGN or DEFINE command.
- System logical name table -- contains entries that can be accessed by any process in the system. To make any entry in this table, use the /SYSTEM qualifier with the ASSIGN or DEFINE command.

You must have user privileges to place entries in the group or system logical name tables.

3.3.2 Logical Name Translation and Recursion

When the system reads a file specification, it examines the file specification to see if the left-most component is a logical name. If it is, the system substitutes the equivalence name in the file specification. This is called logical name translation.

When the system translates logical names, it searches the process, group, and system tables, in that order, and uses the first match that it finds.

When RMS translates logical names in file specifications, the logical name translation is recursive. This means that after RMS translates a logical name in a file specification, it repeats the process of translating the file specification. For VAX-11 RMS, the parse routine will perform up to 10 logical name translations in an effort to identify the actual file name. For example, consider logical name table entries made with ASSIGN commands as follows:

```
$ ASSIGN DBA1:  DISK
$ ASSIGN DISK:WEATHER.SUM REPORT
```

The first ASSIGN command equates the logical name DISK to device DBA1. The second ASSIGN command equates the logical name REPORT to the file specification DISK:WEATHER.SUM. In subsequent requests for this file, you can refer to the logical name REPORT. In translating the logical name REPORT, the system finds the equivalence name DISK:WEATHER.SUM. It then checks to see if the portion on the left of the colon in this file specification is a logical name; if it is (as DISK is in this example) it translates that logical name also. When the logical name translation is complete, the translated file specification is:

```
DBA1:WEATHER.SUM
```

Note that when you assign one logical name to another logical name, you must terminate the equivalence name with a colon (:) if you are going to use the logical name in a file specification in place of a device name. For example:

```
$ ASSIGN DBA1:  TEST
$ ASSIGN TEST:  GO
```

Logical node name translation is also recursive to 10 levels. The equivalence string produced from a logical node name must be another node specification. That is, it cannot supply other missing elements of a file specification.

3.3.3 Defaults for File Names

When the system completes the translation of a logical name, it must use defaults to fill in the still-unspecified fields in the file specification.

Many system commands create output files automatically and provide default file types for the output files. When you use a logical name to specify the input file for a command, the command uses the logical name to assign a file specification to the output file as well. Thus, if the equivalence name contains a file name and file type, the output file is given the same file name and file type as the input file.

For example, the LINK command creates, by default, an executable image file that has the same file name as the input file and a default file type of EXE. However, if you make a logical name assignment and invoke the LINK command as shown below, the results are not as you would expect:

```
$ ASSIGN RANDOM.OBJ TESTIT
$ LINK TESTIT
```

The linker translates the logical name TESTIT and links the file RANDOM.OBJ. When it creates the output file, it also uses the same logical name for the output file. Because the equivalence name includes a file type, the LINK command does not use the default file type of EXE. The executable image is named RANDOM.OBJ and has a version number one higher than the version number of the input file.

3.3.4 Bypassing Logical Name Translations

The system always checks a file specification to see if it contains a logical name. When you enter a device name or file specification, you can request that no translation is to take place. You do this by preceding the device name or file specification with an underscore character (_). (If the file specification contains a node name, then both the node name and device name may be prefixed with an underscore.) For example, if you do not want the system to check whether DMA2 is a logical name on an ALLOCATE command, you would enter the following:

```
$ ALLOCATE _DMA2:
```

3.3.5 Default Process Logical Names

When you log in to the system, the system creates logical name table entries for your process. The logical names, which all have a prefix of SYS, are listed in Table 3-4.

Table 3-4
Default Process Logical Names

Logical Name	Equivalence Name
SYS$INPUT	Default input stream for the process. For an interactive user, SYS$INPUT is equated to the terminal. In a batch job, SYS$INPUT is equated to the batch input stream
SYS$OUTPUT	Default output stream for the process. For an interactive user, SYS$OUTPUT is equated to the terminal. In a batch job, SYS$OUTPUT is equated to the batch job log file
SYS$ERROR	Default device to which the system writes messages For an interactive user, SYS$ERROR is equated to the terminal. In a batch job, SYS$ERROR is equated to the batch job log file
SYS$COMMAND	Original SYS$INPUT device for an interactive user or batch job
SYS$DISK	Default disk device most recently established by the SET DEFAULT command
SYS$SYSDISK	System disk used to boot VMS
SYS$LOGIN	Default disk and directory established at login
SYS$NET	Is defined only for the target process in DECnet task-to-task communication. The equivalence string for SYS$NET identifies the source process that invoked the target process. SYS$NET, when opened, represents the logical link over which the target process can exchange data with its partner. (For additional information, see the DECnet-VAX User's Guide)
SYS$NODE	Identifies the local node name on which your system is running, if DECnet is installed

3.4 PROCESS-PERMANENT FILES

Process-permanent files are an important feature of the VAX/VMS operating system. They exist over the life of a process; hence the term process permanent. In contrast, most files accessed from an image are closed when the image exits, and any control blocks that describe them are deallocated.

You can use VAX-11 RMS to open or create a process-permanent file of your own definition only in supervisor or executive mode. You set the PPF bit in the file processing options field (FOP) of the FAB. This allocates internal data structures, maintained by VAX-11 RMS. These structures reside in the process control region until the end of the process.

You cannot directly access a process-permanent file in user mode. However, you can gain indirect access to a subset of all the available functions of process-permanent files by use of the logical name mechanism. When you log in to the system, a process-permanent file corresponding to the process's input, output, and error message streams is opened. (This means that the most commonly accessed files need not be reopened by each image that executes in the context of a process.) These process-permanent files have a logical name created for them in the process logical name table (see Table 3-4). The specific format of the names in the process logical name table indicates a correspondence between the logical name and the related process-permanent file. VAX-11 RMS recognizes these names and thus provides easy access to the process-permanent files.

CHAPTER 4

PROCESSING FILES WITH SEQUENTIAL RECORD ACCESS MODE

The sequential record access mode is the way to retrieve or store records by starting at a designated point in the file and continuing to the end of the desired area. Records are accessed in the order in which they logically appear in the file.

Section 4.1 deals with sequential access to the sequential file organization. Section 4.2 deals with sequential access to the relative file organization. Section 4.3 deals with sequential access to the indexed file organization.

4.1 THE USE OF SEQUENTIAL FILE ORGANIZATION

This section explores various ways to use sequential file organization with sequential record access mode. Some basic programming examples will be used to illustrate this simple, flexible, and easy-to-use file organization. Once you understand sequential file organization, you can use it where it best suits your needs, and build on the techniques described in this chapter to use this file organization to its fullest capabilities.

4.1.1 Reading Records

This section describes a sample program that illustrates how records are read from a sequentially organized file. Each record is a fixed-length, 50-byte record, as follows:

Byte	**Contents**
0-4	Part number
5	Discount type code
6-25	Part description
26-29	Quantity on hand
30-33	Reorder quantity
34-42	Last reorder date (dd mon yy)
43-49	List price

The purpose of this program is to count the records that have the character A as the fifth byte of the record (discount type code).

Assume that, external to the program, the following assignment will be made:

```
     $ ASSIGN     18SEP78.INV     INFILE:
```

First, you need a FAB to describe the file. You thus issue a $FAB macro call, using parameters to set values in the FAB fields. In some cases, the fields you use for a file can have the value applied by default, so you need not specify these fields.

For example, the file access field indicates the type of operation you want to perform on the file. In this example, you want to open the file for read access (with a $GET macro instruction). Normally, you do so by setting FAC=GET on the $FAB macro instruction. However, FAC=GET is the default when you are opening a file, so you need not specify it. If you were going to perform some other type of operation when you opened the file, such as delete, you must specify that operation explicitly. In addition, defaults can change depending on the operation (see Section 4.1.2; the default is write access when you create a file).

In this example, the file has no special characteristics, such as file processing options. In any case, most FAB fields used for an open operation are only returned as output. Therefore, the only field you need specify as input is the file specification. In the external assignment, the logical name INFILE: is equated to 18SEP78.INV. Therefore, with the FNM parameter, you can indicate the file as follows:

```
     INFAB:  $FAB FNM=<INFILE:>
```

Note that the label field contains INFAB. This lets you refer to this FAB in the $RAB macro instruction, to connect the record stream, and define the address of the FAB for the run-time macro instructions in your program.

Next, you need a RAB to describe the records and how you intend to access the file. You must associate the RAB with the FAB (using the FAB parameter) and set up a buffer area (UBF and USZ parameters). Access to this file will be sequential, which is the default record access mode, and therefore need not be specified. The $RAB macro instruction would be as follows:

```
     INRAB: $RAB  FAB=INFAB,-
                  UBF=REC_BUFFER,-
                  USZ=REC_BUFFER_SIZE
```

The label field contains the value INRAB, giving you a means of referring to this RAB in your run-time macro instructions. Note also the use of the continuation hyphen (-) to continue the instruction on the next line.

To process this file, you need certain VAX-11 RMS run-time processing macro instructions to perform the operations. First, because this is an existing file, you must open it for access with a $OPEN macro instruction and specify the FAB that describes the file, as follows:

```
     $OPEN    FAB=INFAB
```

Next, you must establish the record stream for this file with a $CONNECT macro instruction indicating the RAB, as follows:

```
     $CONNECT RAB=INRAB
```

Once you open the file and connect the record stream, you must indicate what operations you are going to perform. In this application, you want to retrieve records from a file. The $GET macro instruction performs this function. This macro instruction uses the RAB.

```
    $GET   RAB=INRAB
```

After you have read all the records, and processing is finished, you must close the file with the $CLOSE macro instruction indicating the FAB for the file, as follows:

```
    $CLOSE   FAB=INFAB
```

The $CLOSE macro instruction also disconnects the record stream for all RABs. If you want to disconnect the record stream for a particular RAB connected to a FAB (more than one RAB can be connected to a single FAB), you can use the $DISCONNECT macro instruction, specifying the RAB to disconnect.

Figure 4-1 lists the program code to count the discount type code A records. The VAX-11 RMS macro instructions are shown in red. Note that this program, in effect, produces no worthwhile result because the program does not communicate the record count to you.

```
        .TITLE  COUNT - COUNTS TYPE A DISCOUNT RECORDS
;
;  PROGRAM TO READ INVENTORY FILE COUNTING
;  TYPE 'A' DISCOUNT RECORDS
;
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_BUFFER_SIZE
REC_BUFFER:  .BLKB 50                          ;  USER RECORD BUFFER
REC_BUFFER_SIZE=.- REC_BUFFER
COUNT:  .WORD 0                                ;  COUNT OF TYPE 'A' RECORDS
;
;  OPEN FILE, CONNECT STREAM
;
        .PSECT  CODE
BEGIN:  .WORD   0
         $OPEN   FAB=INFAB                     ;  OPEN INPUT FILE
        BLBC    R0,EXIT                        ;  BRANCH ON ERROR
        $CONNECT        RAB=INRAB              ;  CONNECT STREAM
        BLBC    R0,EXIT                        ;  BRANCH ON ERROR
;
;  READ RECORDS, COUNTING TYPE 'A' RECORDS
;
READ:   $GET    RAB=INRAB                      ;  READ A RECORD
        BLBC    R0,DONE                        ;  BRANCH ON ERROR
                                               ;  (ERROR MAY BE EOF)
        CMPB    REC_BUFFER+5,#^A/A/            ;  IS DISCOUNT TYPE = 'A'?
        BNEQ    READ                           ;  BRANCH IF NOT
        INCW    COUNT                          ;  COUNT TYPE 'A' RECORD
        BRB     READ                           ;  GO GET THE NEXT RECORD

;
;  ALL DONE, CLOSE FILE AND EXIT.
;
DONE:   $CLOSE  FAB=INFAB                      ;  CLOSE THE FILE
EXIT:   $EXIT_S R0                             ;  EXIT WITH STATUS

        .END BEGIN
```

Figure 4-1 Program to Count Records in a Sequential File

4.1.2 Creating a Sequential File

This section describes a sample program that illustrates how you can use the sequential file organization to create a new file by copying an existing file. The format and contents of the records in the file are the same as those described for the example in Section 4.1.1.

Assume that, external to the program, the following assignments will be made:

```
$ ASSIGN   18SEP78.INV   INFILE:
$ ASSIGN   18SEP78.CPY   OUTFILE:
```

Because this program uses two files, one for input and one for output, two separate FABs are required to describe the files. For the input file, you need only define the file specification. In the external assignment, it was equated to INFILE:. Therefore, with the FNM parameter, you indicate the file as follows:

```
INFAB:   $FAB FNM=<INFILE:>
```

For the output file, you must also define the file specification. In the external assignment, it was equated to OUTFILE:. Because you are creating this file, you use the $PUT macro instruction to write records to the new file. The default is write access when creating a file; therefore, you need not specify FAC=PUT. When you create a file, you must indicate the record format. In this file, the records are fixed length, so the specification is RFM=FIX. You also must specify the maximum record size. For fixed-length records, the maximum record size indicates the actual length of each record in the file. The records for this file are each 50 bytes long. You can specify this record size either by indicating MRS=50, or by defining a record size within your program and referring to this definition, for example, REC_SIZE=50 and MRS=REC_SIZE. Defining the record size in your program also lets you make other references to this record size within your program, for example, in defining the size of the buffer areas for the RAB.

As an option, you can indicate that each record is to be preceded by a line feed and followed by a carriage return whenever the record is output to a line printer or terminal. Set the record attributes field with RAT=CR. The FAB for the output file is then defined as follows:

```
OUTFAB:   $FAB   FNM=<OUTFILE:>,-
                 RFM=FIX,-
                 MRS=REC_SIZE,-
                 RAT=CR
```

You must also define RABs for both files. The FAB parameter associates a RAB with the appropriate FAB. Because the sequential record access mode is the default, you can omit the RAC parameter. Both files also need a buffer area. In fact, they both can use the same buffer area, since you will read a record into a buffer, and then write it from the buffer before you read another record into the buffer. The output RAB, however, uses the RBF and RSZ parameters to define the buffer, rather than the UBF and USZ parameters. The reason is that the $PUT macro instruction does not use UBF and USZ as input; it uses RBF and RSZ. The $RAB macro instructions would be as follows, with the input RAB shown first.

```
INRAB:  $RAB   FAB=INFAB,-
               UBF=REC_BUFFER,-
               USZ=REC_SIZE

OUTRAB:  $RAB  FAB=OUTFAB,-
               RBF=REC_BUFFER,-
               RSZ=REC_SIZE
```

The run-time processing macro calls for the input file consist of a $OPEN, a $CONNECT, a $GET, and a $CLOSE. For the output file, you must specify a $CREATE macro instruction (rather than an $OPEN), which opens and constructs a new file. In this macro instruction, you indicate the FAB that contains the attributes for the new file, as follows:

```
$CREATE FAB=OUTFAB
```

As with the input file, you must also specify the $CONNECT macro instruction to connect the record stream and the $CLOSE macro instruction to close the file. However, before the file is closed, it must be processed. In the case of a copy operation, records must be written to the new file. Use the $PUT macro instruction, specifying the RAB, as follows:

```
$PUT RAB=OUTRAB
```

Figure 4-2 lists the program code to copy a file. The VAX-11 RMS macro instructions appear in red.

4.1.2.1 **Dynamically Creating a Sequential File** - The example in this section produces results identical to the results of the program listed in Figure 4-2. The difference between the two, however, is that the allocation and initialization of the control blocks for the output file (FAB and RAB) is dynamic, performed at run time rather than at assembly time. The "store" macro instructions let you dynamically set fields.

The values you supply with the "store" macro instructions expand into code that affects the contents of data fields during the execution of your program.

Figure 4-3 lists the program code for this example. Note that only minor changes have been made to the program listed in Figure 4-2. Lines 11 through 19 in Figure 4-2 have been replaced in Figure 4-3 with lines 12, 13, and 14 to begin the definition of the output FAB and RAB and to provide a .ASCIC directive to specify the character string for the file specification.

```
OUTFAB:  $FAB
OUTRAB:  $RAB  FAB=OUTFAB
OUT_FILESPEC: .ASCIC      /OUTFILE:/
```

```
        .TITLE  COPYFILE - MAKE COPY OF INPUT FILE
;
;  PROGRAM TO MAKE A COPY OF THE INPUT FILE
;
REC_SIZE=50                                     ; RECORD SIZE
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_SIZE
OUTFAB: $FAB    FNM=<OUTFILE:>,-                ; OUTPUT FILE HAS FIXED
                RFM=FIX,-                       ; LENGTH RECORDS, 50 BYTES
                MRS=REC_SIZE,-                  ; IN LENGTH, WITH IMPLIED
                RAT=CR                          ; NEW LINE CARRIAGE CONTROL
OUTRAB: $RAB    FAB=OUTFAB,-
                RBF=REC_BUFFER,-
                RSZ=REC_SIZE
                                                ; NOTE: OUTPUT RAB USES
                                                ; SAME RECORD BUFFER AS INPUT RAB
REC_BUFFER:     .BLKB   REC_SIZE
        .PSECT CODE,NOWRT
;
;  INITIALIZATION - OPEN INPUT AND OUTPUT FILES AND CONNECT STREAMS
;
START:  .WORD   0
        $OPEN   FAB=INFAB                       ; OPEN INPUT FILE
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $CREATE FAB=OUTFAB                      ; OPEN OUTPUT FILE
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $CONNECT        RAB=INRAB               ; CONNECT INPUT RAB
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $CONNECT        RAB=OUTRAB              ; CONNECT OUTPUT RAB
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
;
;  COPY RECORDS
;
READ:   $GET    RAB=INRAB                       ; READ A RECORD
        BLBC    R0,DONE                         ; BRANCH ON ERROR
        $PUT    RAB=OUTRAB                      ; WRITE THE RECORD TO
                                                ; THE OUTPUT FILE
        BLBS    R0,READ                         ; BRANCH ON SUCCESS
EXIT1:  BRB     EXIT                            ; GET OUT ON ERROR
;
;  ALL SET - CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB                       ; CLOSE INPUT FILE
        $CLOSE  FAB=OUTFAB                      ; CLOSE OUTPUT FILE

EXIT:   $EXIT_S R0                              ; EXIT WITH STATUS
        .END START
```

Figure 4-2 Program to Copy a Sequential File

A $FAB_STORE macro instruction has been inserted in lines 23 through 28 of Figure 4-3 to initialize the output FAB and set the needed values. (Note that the FNM parameter has been replaced by two parameters: FNA and FNS. This is because you cannot use the FNM parameter to provide the file specification dynamically; you must use the FNA and FNS parameters.)

```
    $FAB_STORE    FAB=OUTFAB,-
                  FNA=OUT_FILESPEC+1,-
                  FNS=OUT_FILESPEC,-
                  RFM=FIX,-
                  MRS=#REC_SIZE,-
                  RAT=CR
```

The $CREATE macro instruction (line 28 in Figure 4-2) has been replaced in Figure 4-3 with a new $CREATE macro instruction (now on line 30). This opens and constructs the output file, indicating the register containing the address of the FAB--R0. (Note that the FAB_STORE macro instruction loaded the FAB address into register 0 by default.)

```
    $CREATE    FAB=R0
```

A $RAB_STORE macro has been inserted in lines 34, 35, and 36 of Figure 4-3 to initialize the output RAB and set the needed values.

```
    $RAB_STORE    RAB=OUTRAB,-
                  RBF=REC_BUFFER,-
                  RSZ=#REC_SIZE
```

The $CONNECT macro instruction (line 32 in Figure 4-2) has been replaced with a new $CONNECT macro instruction (now on line 38). This instruction establishes the record stream for the output file, indicating the register of the RAB--R0.

```
    $CONNECT   RAB=R0
```

4.2 THE USE OF RELATIVE FILE ORGANIZATION

Relative file organization is available for use on disk devices only. This organization affords more capabilities than the sequential file organization, but, in most cases, requires additional planning and coding to implement (see Chapter 1).

Relative file organization uses a fixed-length cell for each record in the file (or as a space for a record to be inserted). However, while all the cells are fixed-length, the individual records need not be; they can be variable length, fixed length, or variable with fixed-length control.

The relative file organization allows random retrieval of records by means of keys (a key in a relative file is the relative record number assigned to each record). The fixed-length cell allows for a direct calculation of the record's actual position.

4.2.1 Reading a Relative File

The program described in this section produces the same result as the program listed in Figure 4-1. The program counts discount type code A records in the file. The record contents are the same, and so are the external assignments. The only difference is that the file is a relative file.

You need not specify a file organization in the FAB for the file when you open it because the file organization already is assigned. In addition, you do not need to specify sequential file organization for a create; since it is the default. Therefore, the program code would be identical to the one for a sequential file (Figure 4-1).

```
        .TITLE  COPYFILE1 - MAKE COPY OF INPUT FILE
;
;  PROGRAM TO MAKE A COPY OF THE INPUT FILE
;
REC_SIZE=50                                         ;  RECORD SIZE
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_SIZE
;
OUTFAB: $FAB                                        ;  OUTPUT FILE FAB
OUTRAB: $RAB    FAB=OUTFAB                          ;  OUTPUT FILE RAB
OUT_FILESPEC:   .ASCIC  /OUTFILE:/

REC_BUFFER:     .BLKB   REC_SIZE                    ;  RECORD BUFFER
        .PSECT CODE,NOWRT
;
;  INITIALIZATION - OPEN INPUT AND OUTPUT FILES AND CONNECT STREAMS
;
START:  .WORD   0
        $OPEN   FAB=INFAB                           ;  OPEN INPUT FILE
        BLBC    R0,EXIT1                            ;  BRANCH ON ERROR
        $FAB_STORE      FAB=OUTFAB,-                ;  INITIALIZE OUTPUT FAB
                FNA=OUT_FILESPEC+1,-                ;  SET OUT FILE SPEC ADDRESS
                FNS=OUT_FILESPEC,-                  ;  SET OUT FILE SPEC LENGTH
                RFM=FIX,-                           ;  SET RECORD FORMAT
                MRS=#REC_SIZE,-                     ;  SET MAXIMUM RECORD SIZE
                RAT=CR                              ;  NEW LINE CARRIAGE CONTROL

        $CREATE FAB=R0                              ;  OPEN OUTPUT FILE
        BLBC    R0,EXIT1                            ;  BRANCH ON ERROR
        $CONNECT        RAB=INRAB                   ;  CONNECT INPUT RAB
        BLBC    R0,EXIT1                            ;  BRANCH ON ERROR
        $RAB_STORE      RAB=OUTRAB,-                ;  INITIALIZE OUTPUT FILE RAB
                RBF=REC_BUFFER,-                    ;  SET USER BUFFER ADDRESS
                RSZ=#REC_SIZE                       ;  SET USER BUFFER SIZE

        $CONNECT        RAB=R0                      ;  CONNECT OUTPUT RAB
        BLBC    R0,EXIT1                            ;  BRANCH ON ERROR
;
;  COPY RECORDS
;
READ:   $GET    RAB=INRAB                           ;  READ A RECORD
        BLBC    R0,DONE                             ;  BRANCH ON ERROR
        $PUT    RAB=OUTRAB                          ;  WRITE THE RECORD TO
                                                    ;  THE OUTPUT FILE
        BLBS    R0,READ                             ;  BRANCH ON SUCCESS
EXIT1:  BRB     EXIT                                ;  GET OUT ON ERROR
;
;  ALL SET - CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB                           ;  CLOSE INPUT FILE
        $CLOSE  FAB=OUTFAB                          ;  CLOSE OUTPUT FILE

EXIT:   $EXIT_S R0                                  ;  EXIT WITH STATUS
        .END START
```

Figure 4-3 Program to Copy a Sequential File, Setting the Output Control Blocks Dynamically

4.2.2 Creating a Relative File

When you create a file, you must specify the type of file organization you want, either by default for sequential or by an explicit specification for relative.

You indicate that you want the relative file organization assigned to the file by specifying ORG=REL on the $FAB macro call that applies to the file.

If you use the same example as in Section 4.1.2 (and Figure 4-2), but create a relative file rather than a sequential file, only the output file $FAB macro instruction changes, as indicated by an arrow in the portion of code shown in Figure 4-4. Everything else in the program remains the same.

```
REC_SIZE=50                                        ;  RECORD SIZE
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_SIZE
;
OUTFAB: $FAB    FNM=<OUTFILE:>,-                   ;  OUTPUT FILE HAS FIXED
                RFM=FIX,-                          ;  LENGTH RECORDS, 50 BYTES
                MRS=REC_SIZE,-                     ;  IN LENGTH, WITH IMPLIED
                RAT=CR,-                           ;  NEW LINE CARRIAGE CONTROL
                ORG=REL  <-------
OUTRAB: $RAB    FAB=OUTFAB,-
                RBF=REC_BUFFER,-
                RSZ=REC_SIZE
                                                   ;  NOTE:  OUTPUT RAB USES
                                                   ;  SAME RECORD BUFFER AS INPUT RAB
REC_BUFFER:     .BLKB   REC_SIZE
        .PSECT CODE,NOWRT
```

Figure 4-4 Creating a Relative File

4.2.2.1 **Dynamically Creating a Relative File** - Section 4.1.2.1 described how to dynamically specify the parameters to create a file with the sequential file organization. Section 4.2.2 described how to create a file with the relative file organization specified at assembly time. By combining what was discussed about the output FAB in both of these sections, you can specify dynamically, at run time, the parameters to create a relative file.

At assembly time, the $FAB macro instruction included the specification of ORG=REL (see Figure 4-4). By adding this same specification to the $FAB_STORE macro instruction (see Figure 4-3), you specify the parameters dynamically, at run time.

Figure 4-5 lists a section of code, showing the inclusion of ORG=REL to the $FAB_STORE macro instruction.

Appendix A contains an additional example of the use of sequential record access mode.

```
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_SIZE
;
OUTFAB: $FAB                                    ; OUTPUT FILE FAB
OUTRAB: $RAB    FAB=OUTFAB                      ; OUTPUT FILE RAB
OUT_FILESPEC:   .ASCIC  /OUTFILE:/

REC_BUFFER:     .BLKB   REC_SIZE                ; RECORD BUFFER
        .PSECT CODE,NOWRT
;
;  INITIALIZATION - OPEN INPUT AND OUTPUT FILES AND CONNECT STREAMS
;
START:  .WORD   0
        $OPEN   FAB=INFAB                       ; OPEN INPUT FILE
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $FAB_STORE      FAB=OUTFAB,-            ; INITIALIZE OUTPUT FAB
                FNA=OUT_FILESPEC+1,-            ; SET OUT FILE SPEC ADDRESS
                FNS=OUT_FILESPEC,-              ; SET OUT FILE SPEC LENGTH
                RFM=FIX,-                       ; SET RECORD FORMAT
                MRS=#REC_SIZE,-                 ; SET MAXIMUM RECORD SIZE
                RAT=CR,-                        ; SET IMPLIED CARRIAGE CONTROL
                ORG=REL                         ; RELATIVE FILE ORGANIZATION

        $CREATE FAB=R0                          ; OPEN OUTPUT FILE
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $CONNECT        RAB=INRAB               ; CONNECT INPUT RAB
        BLBC    R0,EXIT1                        ; BRANCH ON ERROR
        $RAB_STORE      RAB=OUTRAB,-            ; INITIALIZE OUTPUT FILE RAB
                RBF=REC_BUFFER,-                ; SET USER BUFFER ADDRESS
                RSZ=#REC_SIZE                   ; SET USER BUFFER SIZE

        $CONNECT        RAB=R0                  ; CONNECT OUTPUT RAB
```

Figure 4-5 Creating a Relative File Dynamically

4.3 THE USE OF INDEXED FILE ORGANIZATION

Indexed file organization is available for use on disk devices only. This organization affords more capabilities than the sequential or relative file organization.

The indexed file allows the use of truly variable-length records. Their lengths are limited only by the size of the bucket or by a maximum record size that you establish. Since variable-length records may change size on an update, there is no need to pad records to their maximum size. The record size may be increased or decreased later with an update operation.

Indexed files allow random access to either fixed- or variable-length data records by a key value. A key in an indexed file can be a character string, a packed decimal number, a 2- or 4-byte signed integer, or a 2- or a 4-byte unsigned binary number within the record. This type of file organization stores the records by ascending key value. These records can then be retrieved sequentially in ascending order or randomly by supplying a specific key value to retrieve.

When an indexed file is created, a key is defined by its location and length within each record. At least one key, called a primary key, must be defined for an indexed file. Optionally, additional keys referred to as alternate keys, may be defined.

As your program puts records into an indexed file, VAX-11 RMS uses the values of the primary and alternate keys to build indexes. An index is the structure which allows the records to be retrieved randomly. Each data record is placed in the file in sorted order by primary key. In alternate indexes, the sort sequence is established by pointers to the actual data record. These mechanisms enable the data records to be read sequentially in sorted order by any key.

Because VAX-11 RMS completely controls the placement of records in an indexed file, location of the records in the file is transparent to your program.

4.3.1 Reading an Indexed File

The program described in this section produces the same result as the program listed in Figure 4-1 and described in Section 4.1.1. The program counts discount type code A records in the file. The record contents are the same and so are the external assignments. The difference is that the file is an indexed file. In this example, the discount type field within the record has been defined as the first alternate key. This will allow random access to the first record containing discount type code A and sequential access to all succeeding type A records. This eliminates the need to read all of the records in the file and, in fact, simplifies the program logic. Though some of the program code is identical to that for sequential files, some is unique to indexed files (see Figure 4-6).

Assume that, external to the program, the following assignment will be made:

```
     $ ASSIGN       18SEP78.INV       INFILE:
```

First, you need a FAB to describe the file. You therefore issue a $FAB macro instruction, using arguments to set values in the FAB fields.

For example, the file access field indicates the type of operations allowed when the file is opened. You want to open the file for read access only. Normally, you do so by setting FAC=GET on the $FAB macro instruction. However, FAC=GET is the default when you are opening a file, so you need not specify it. If you were going to perform some other type of operation when you opened the file, such as delete, you would have to specify that operation explicitly.

The only field you need specify as input is the file specification. In the external assignment, the logical name INFILE: is equated to 18SEP78.INV. Therefore, with the FNM parameter, you can indicate the file as follows:

```
     INFAB:  $FAB FNM=<INFILE:>
```

Note that the label field contains INFAB. This lets you refer to this FAB in the $RAB macro instruction, to connect the record stream, and define the address of the FAB for the run-time macro instructions in your program.

Next, you need a RAB to describe the access to the records and to the file. You must associate the RAB with the FAB (using the FAB parameter) and set up a buffer area (UBF and USZ parameters). You must also specify the buffers for the key value, and the size of the key value (KBF and KSZ parameters). Specifying KRF=1 causes the first alternate index to be used when retrieving records from the file.

Then you specify the record processing options ROP=LIM to compare the key value described by the KBF and KSZ fields with the value in the record accessed on sequential get operations. When the key value in the record exceeds that value in the key buffer on a sequential get operation, a success code of RMS$_OK_LIM will be returned. Finally, the initial record access mode is to be by key (RAC=KEY). The $RAB macro instruction would be as follows.

```
INRAB:  $RAB  FAB=INFAB,-
              UBF=REC_BUFFER,-
              USZ=REC_BUFFER_SIZE,-
              KRF=1,-
              KBF=KEY_BUFF,-
              KSZ=KEY_BUFF_SIZE,-
              ROP=LIM,-
              RAC=KEY
```

The label field contains the value INRAB, giving you a means of referring to this RAB in your run-time macro instructions.

Then you must set up the user buffer and the key buffer as follows:

```
REC_BUFFER:     .BLKB    50
REC_BUFFER_SIZE=.-REC_BUFFER
KEY_BUFF:      .BLKB     1
KEY_BUFF_SIZE=.-KEY_BUFF
```

To process this file, you need certain VAX-11 RMS run-time processing macro instructions. First, because this is an existing file, you must open it with a $OPEN macro instruction and specify the FAB that describes the file, as follows:

```
$OPEN   FAB=INFAB
```

Next, you must establish the record stream for this file with a $CONNECT macro instruction indicating the RAB, as follows:

```
$CONNECT RAB=INRAB
```

Now you specify that the key you want is the first record containing discount type code A. To position to the first record with discount type code A, you issue a $FIND macro instruction (with RAC=KEY set by the $RAB macro instruction); then you change the record access mode to sequential with the record access mode parameter option (RAC=SEQ on the $RAB_STORE macro instruction).

Now that you have established the logical starting point in the file (the first record with discount type A), you want to retrieve that record and all succeeding records with discount type A. The $GET macro instruction performs that function. This macro instruction uses the RAB.

```
$GET   RAB=INRAB
```

When the success code RMS$_OK_LIM is returned from a $GET macro instruction, you will have retrieved all records in the file with a discount type A. The current record and any succeeding records (if not at the end of file) will have a higher key value, such as B. After record processing is finished, you must close the file with a $CLOSE macro instruction, indicating the FAB for the file, as follows:

```
$CLOSE   FAB=INFAB
```

The $CLOSE macro instruction also disconnects the record stream for all RABs. If you want to disconnect the record stream for a particular RAB connected to a FAB (more than one RAB can be connected to a single FAB), you can use the $DISCONNECT macro instruction, specifying which RAB to disconnect.

Figure 4-6 lists the program code to count the discount type code A records in an indexed file. The VAX-11 RMS macro instructions are shown in red. Note that this program, in effect, produces no worthwhile result, because the program does not communicate the record count to you; the program serves only as an example.

```
        .TITLE  COUNT - COUNTS TYPE A DISCOUNT RECORDS
;
;  PROGRAM TO READ INVENTORY FILE COUNTING
;  TYPE 'A' DISCOUNT RECORDS
;
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_BUFFER_SIZE,-
                KRF=1,-                              ;  KEY TO SEARCH ON
                KBF=KEY_BUFF,-                       ;  BUFFER TO HOLD KEY VALUE
                KSZ=KEY_BUFF_SIZE,-                  ;  SIZE OF KEY VALUE
                ROP=LIM,-
                RAC=KEY
REC_BUFFER:     .BLKB   50
REC_BUFFER_SIZE=.-REC_BUFFER
KEY_BUFF:       .BLKB
KEY_BUFF_SIZE=.-KEY_BUFF
COUNT:  .WORD   0
;
;  OPEN FILE, CONNECT STREAM
;
        .PSECT  CODE
BEGIN:  .WORD   0
        $OPEN   FAB=INFAB
        BLBC    R0,EXIT                              ;  BRANCH ON ERROR
        $CONNECT        RAB=INRAB                    ;  CONNECT STREAM
        BLBC    R0,EXIT                              ;  BRANCH ON ERROR
;
;  READ RECORDS, COUNTING TYPE 'A' RECORDS
;
        MOVB    #^A/A/,KEY_BUFF                      ;  SPECIFY KEY WE'RE SEARCHING FOR
        $FIND   RAB=INRAB                            ;  POSITION TO FIRST TYPE 'A' REC
                                                     ;  NOTE:  THIS IS THE RECORD THAT
                                                     ;  WILL BE ACCESSED ON FIRST GET
        BLBC    R0,EXIT                              ;  BRANCH ON ERROR
        $RAB_STORE      RAB=INRAB,-                  ;  CHANGE RECORD ACCESS MODE TO SEQ.
                RAC=SEQ
READ:   $GET    RAB=INRAB                            ;  READ A RECORD
        BLBC    R0,DONE                              ;  BRANCH ON ERROR
                                                     ;  (ERROR MAY BE EOF)
        CMPL    R0,#RMS$_OK_LIM                      ;  IS RETREIVED RECORD'S KEY
                                                     ;  > THAN KEY VALUE IN KEY BUFF
        BEQL    DONE                                 ;  ALL DONE
        INCW    COUNT                                ;  COUNT TYPE 'A' RECORD
        BRB     READ                                 ;  GO GET THE NEXT RECORD
;
;  ALL DONE.  CLOSE FILE AND EXIT.
;
DONE:   $CLOSE  FAB=INFAB                            ;  CLOSE THE FILE
EXIT    $EXIT_S R0                                   ;  EXIT WITH STATUS

        .END    BEGIN
```

Figure 4-6 Program to Count Records in an Indexed File

4.3.2 Creating an Indexed File

The sample program in this section illustrates how to create a new indexed file by copying an existing file of any organization. The format and contents of the records in the file are the same as those described in Section 4.1.1.

Assume that, external to the program, the following assignments will be made:

```
     $ ASSIGN    18SEP78.INV    INFILE:
     $ ASSIGN    18SEP78.CPY    OUTFILE:
```

Because this program uses two files, one for input and one for output, two separate FABs are required to describe the files. For the input file, you need only define the file specification. In the external assignment, it was equated to INFILE:. Therefore, with the FNM parameter, you indicate the file as follows:

```
     INFAB:     $FAB FNM=<INFILE:>
```

For the output file, you must also define the file specification. In the external assignment, it was equated to OUTFILE:. Because you are creating this file, you use the $PUT macro instruction to write records to the new file. The default is write access when creating a file; therefore, you need not specify FAC=PUT. When you create a file, you must indicate the record format. In this file, the records are variable length, so the specification is RFM=VAR.

You also must specify the maximum record size. For fixed-length records, the maximum record size indicates the actual length of each record in the file. For variable-length records, the maximum record size specifies the size limit for a record being written initially into the file, or an existing record being updated. If you do not specify the maximum record size, it is limited only by bucket size. In this example, the maximum record size and record size are identical. The records for this file are each 50 bytes long. You can specify this limit either by indicating MRS=50 or by defining a record size within your program, for example, REC_SIZE=50 and MRS=REC_SIZE, and referring to this definition defining the record size in your program also lets you make other references to this record size within your program, for example, in defining the size of the buffer areas for the RAB.

You must specify that the file is an indexed file and you must specify the initial extended attribute blocks of the chain, so the specifications are ORG=IDX and XAB=KEY0.

As an option, you can indicate that each record is to be preceded by a line feed and followed by a carriage return whenever the record is output to a line printer or terminal. Set the record attributes field with RAT=CR. The FAB for the output file is then defined as follows:

```
     OUTFAB:  $FAB  FMN=<OUTFILE:>,-
                    RFM=VAR,-
                    MRS=REC_SIZE,-
                    ORG=IDX,-
                    XAB=KEY0,-
                    RAT=CR
```

You must also define RABs for both files. The FAB parameter associates a RAB with the appropriate FAB. Because the sequential record access mode is the default, you can omit the RAC parameter.

Both files also need a buffer area. In fact, they both can use the same buffer area, since you're going to read a record into a buffer, and then write it from the buffer before you read another record into the buffer. The output RAB, however, uses the RBF and RSZ parameter to define the buffer, rather than the UBF and USZ parameters. The reason is that the $PUT macro instruction does not use UBF and USZ as input; it uses RBF and RSZ. The $RAB macro instructions would be as follows, with the input RAB shown first.

```
     INRAB:    $RAB  FAB=INFAB,-
                     UBF=REC_BUFFER,-
                     USZ=REC_SIZE

     OUTRAB:   $RAB  FAB=OUTFAB,-
                     RBF=REC_BUFFER,-
                     RSZ=REC_SIZE
```

Since you are creating an indexed file, you must specify the primary key and the alternate keys, if any. In this example the primary key (key 0) and two alternate keys (key 1 and key 2) are defined. They are defined by the key definition extended attribute blocks $XABKEY REF=0, $XABKEY REF=1, and $XABKEY REF=2 macro instructions respectively. The position of the keys within each record and the length of key must be specified with the POS and SIZ parameters.

In the sample program, the primary and alternate keys are simple keys (that is, not segmented); hence, only one position parameter value and one size parameter value is defined for each key. Simple keys consist of a single string of contiguous bytes. You should note that if segmented keys are specified, the key position and key size fields must define an equal quantity of key position values and key size values. The key position value is the starting (byte) position of the key within each record (with the first byte being byte 0, the second being 1, etc.). The key size value is the length (in bytes) of the key; in the sample program, the primary key is a simple key, starting in the first byte of the record and is five bytes long; this is defined as follows:

```
     KEY0:  $XABKEY  REF=0,-
                     POS=0,-
                     SIZ=5,-
                     NXT=KEY1
```

Note that the NXT parameter points to the next XAB in the chain, which has a label of KEY1.

The alternate keys (key 1 and key 2) likewise are defined as being in byte positions 6 and 7, respectively, and as being 1 and 20 bytes in length, respectively. They are defined as follows:

```
     KEY1:  $XABKEY  REF=1,-
                     POS=5,-
                     SIZ=1,-
                     NXT=KEY2
```

and

```
     KEY2:  $XABKEY  REF=2,-
                     POS=6,-
                     SIZ=20
```

Note that the NXT parameter is omitted from the XAB with a label of KEY2; therefore the default is 0, which indicates there are no more XABs in the chain.

In the sample program, the alternate keys may change values (on an update) and there may be duplicate alternate keys. Changes and duplications can be defined by FLG=<DUP,CHG>; this is also the default for alternate keys and, therefore it is not necessary to actually define this parameter.

The default for the primary key is no duplicates allowed. The primary key is never allowed to change key value on update.

The run-time processing macro instructions for the input file consist of a $OPEN, a $CONNECT, a $GET, and a $CLOSE. For the output file, you must specify a $CREATE macro instruction (rather than an $OPEN), which opens and constructs a new file. In this macro instruction, you indicate the FAB that contains the attributes for the new file, as follows:

```
     $CREATE FAB=OUTFAB
```

As with the input file, you must also specify the $CONNECT macro instruction to connect the record stream and the $CLOSE macro instruction to close the file. However, before the file is closed, it must be processed. In the case of a copy operation, records must be written to the new file. Use the $PUT macro instruction, specifying the RAB, as follows:

```
     $PUT RAB=OUTRAB
```

Figure 4-7 lists the program code to copy a file. The VAX-11 RMS macro instructions appear in red.

```
        .TITLE  COPYFILE  -  MAKE COPY OF INPUT FILE
;
;  PROGRAM TO MAKE A COPY OF THE INPUT FILE
;
REC_SIZE=50                                     ;  RECORD SIZE
        .PSECT  DATA,LONG
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_SIZE
OUTFAB: $FAB    FNM=<OUTFILE:>,-                ;  OUTPUT FILE HAS FIXED
                RFM=VAR,-                       ;  LENGTH RECORDS, 50 BYTES
                MRS=REC_SIZE,-                  ;  IN LENGTH, WITH IMPLIED
                ORG=IDX,-                       ;  NEW LINE CARRIAGE CONTROL,
                XAB=KEY0,-                      ;  WITH INDEXED FILE ORG.,
                RAT=CR                          ;  AND A CHAIN OF KEY XABS
OUTRAB: $RAB    FAB=OUTFAB,-
                RBF=REC_BUFFER,-                ;  NOTE:  OUTPUT RAB USES
                RSZ=REC_SIZE                    ;  SAME RECORD BUFFER AS INPUT RAB
;
;  CREATE NEW FILE WITH PRIMARY KEY=PART#, AND TWO ALTERNATE KEYS
;
KEY0:   $XABKEY REF=0,-
                POS=0,-
                SIZ=5,-
                NXT=KEY1
KEY1:   $XABKEY REF=1,-
                POS=5,-
                SIZ=1,-
                NXT=KEY2
KEY2:   $XABKEY REF=2,-
                POS=6,-
                SIZ=20
;
REC_BUFFER:     .BLKB   REC_SIZE
;
        .PSECT  CODE,NOWRT
;
;  INITIALIZATION  -  OPEN INPUT AND OUTPUT FILES AND CONNECT STREAMS
;
START:  .WORD   0
        $OPEN   FAB=INFAB                       ;  OPEN INPUT FILE
        BLBC    R0,EXIT1                        ;  BRANCH ON ERROR
        $CREATE FAB=OUTFAB                      ;  OPEN OUTPUT FILE
        BLBC    R0,EXIT1                        ;  BRANCH ON ERROR
        $CONNECT        RAB=INRAB               ;  CONNECT INPUT RAB
        BLBC    R0,EXIT1                        ;  BRANCH ON ERROR
        $CONNECT        RAB=OUTRAB              ;  CONNECT OUTPUT RAB
        BLBC    R0,EXIT1                        ;  BRANCH ON ERROR
;
;  COPY RECORDS
;
READ:   $GET    RAB=INRAB                       ;  READ A RECORD
        BLBC    R0,DONE                         ;  BRANCH ON ERROR
        $PUT    RAB=OUTRAB                      ;  WRITE THE RECORD TO
                                                ;  THE OUTPUT FILE
        BLBS    R0,READ                         ;  BRANCH ON SUCCESS
EXIT1:  BRB     EXIT                            ;  GET OUT ON ERROR
;
;  ALL SET  -  CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB                       ;  CLOSE INPUT FILE
        $CLOSE  FAB=OUTFAB                      ;  CLOSE OUTPUT FILE

EXIT:   $EXIT_S R0                              ;  EXIT WITH STATUS
        .END    START
```

Figure 4-7 Program to Create an Indexed File
by Copying an Existing File

CHAPTER 5

PROCESSING FILES WITH RANDOM RECORD ACCESS

Two different modes provide random access to records:

- Random by key
- Random by record's file address

In the random by key access mode, you retrieve or store a record by specifying a key value. In the random by record's file address access mode, the retrieval or storage of the record is based on a unique address returned to the user by VAX-11 RMS.

Section 5.1 deals with random access to the sequential file organization. Section 5.2 deals with random access to the relative file organization. Section 5.3 deals with random access to the indexed file organization.

5.1 RANDOM ACCESS TO SEQUENTIAL FILE ORGANIZATION

The sequential file organization provides for random access to records only if the file containing the records is on a disk device.

The sequential file organization allows random retrieval of fixed-length records by means of keys only (a key in a sequential file is the relative record number assigned to each record). To gain random access to variable-length records in a sequential file, you must use the random by record's file address mode.

5.1.1 Random Read of a Record

This section describes a sample program that accepts the key (relative record number) from the operator, finds the requested record in a file, and then displays the contents of the record.

Assume that the following external assignment will be made:

```
$ ASSIGN      18SEP78.INV    INFILE:
```

You must provide this program with definitions for three files: an output file, a file to accept the request, and an input file (where you define that the record access mode is random, since the input file is the one you search for the records).

OUTPUT FILE

The first file that must be defined is the output file, SYS$OUTPUT:, which is a process logical name assigned for the output stream. For an interactive user, SYS$OUTPUT is a terminal. The FAB for this file only need provide this name, and also an optional record attribute that induces a line feed before and a carriage return after printing the record at the terminal.

```
     TYPE_FAB:      $FAB            FNM=<SYS$OUTPUT>,-
                                    RAT=CR
```

At assembly time, the $RAB macro instruction only need associate the RAB with the FAB.

```
     TYPE_RAB:      $RAB            FAB=TYPE_FAB
```

The actual contents of the RAB are defined dynamically, at run time rather than assembly time with a $RAB_STORE macro instruction. The reason for this is that the record to be output varies. On the one hand, records from the input file are displayed (see lines 83 through 86 of Figure 5-1), while on the other hand, a number of fixed strings are output using the "TYPE" macro (see lines 82, 92, and 94; the macro definition itself appears on lines 7 through 17). Each of the different outputs requires that the RSZ and RBF parameters be set dynamically to indicate the record to be written.

The $RAB_STORE macro instruction indicates the symbolic address of the RAB allocated at assembly time. It must also define the location and size of the buffer that contains the record to be printed on SYS$OUTPUT. When displaying records read from the input file, the location and size are at the address of INRAB (the input RAB) plus the offset to each field (RAB$L_RBF for the address and RAB$W_RSZ for the size).

```
     $RAB_STORE           RAB=TYPE_RAB,-
                          RBF=@INRAB+RAB$L_RBF,-
                          RSZ=INRAB+RAB$W_RSZ
```

REQUEST FILE

The second file that must be defined is the request file, which prompts a message to solicit information from the operator and accepts the requested record number from the terminal. This file is SYS$INPUT:, which is a process logical name. Note that for an interactive process, SYS$INPUT and SYS$OUTPUT both refer to a terminal. In this example, it would be possible to use the same file (either SYS$INPUT or SYS$OUTPUT) to accept requests and display output. In so doing, however, you would lose the ability to run the program within a batch stream. (As the program currently stands, you could do this.)

```
     PROMPT_FAB:    $FAB            FNM=<SYS$INPUT:>
```

The RAB you connect to this FAB defines a buffer area and associates the RAB with the FAB. The RAB also defines a record processing option of ROP=PMT. This option indicates that the contents of the specified prompt buffer (filled as part of the expansion of the "PROMPT" macro), are to be output to the terminal operator in order to indicate what data is being requested for output.

```
     PROMPT_RAB:    $RAB            FAB=PROMPT_FAB,-
                                    UBF=PROMPT_BUFF,-
                                    USZ=132,-
                                    ROP=PMT
```

INPUT FILE

The third file that must be defined is the input file, which must provide the file specification. The external assignment equates 18SEP78.INV to INFILE:.

```
     INFAB:         $FAB            FNM=<INFILE:>
```

The RAB associated with this file must name its FAB and define a buffer area. The record stream of this RAB will deal with records by their relative record number, so you must set a value in the key buffer address field. This value points to a buffer you set up to contain the relative record number of the record you want. In the program listed in Figure 5-1, the address of the buffer is KEY; therefore you set KBF=KEY. Access to the records in this file is through the random by key mode (the relative record number is the key for sequential files). You indicate this by setting RAC=KEY. (The specification of KEY in this case should not be confused with KBF=KEY, explained previously. The specification of KEY for the record access mode is defined by VAX-11 RMS to indicate key value, which is the relative record number. In KBF=KEY, the KEY specification is user-defined.)

```
     INRAB:         $RAB            FAB=INFAB,-
                                    UBF=REC_BUFFER,-
                                    USZ=REC_BUFFER_SIZE,-
                                    KBF=KEY,-
                                    RAC=KEY
```

When the three files are defined, you must use run-time macro instructions to call the routines that act on these files.

You must open the input file (INFILE) and the request file (SYS$INPUT) with $OPEN macro instructions. The output file for the terminal (SYS$OUTPUT) uses a $CREATE macro instruction, since this is an output file to be created. However, since SYS$OUTPUT is a logical name, the file was created for you when you logged into the system. Therefore, this $CREATE macro instruction acts as a $OPEN macro instruction, so you could, in fact, use the $OPEN macro instruction for SYS$OUTPUT in this program.

Each file you open in the program must have a RAB connected to the appropriate FAB with a $CONNECT macro instruction.

For the input file, use a $GET macro instruction to retrieve the record. For the output file, use a $PUT macro instruction to place the record in SYS$OUTPUT so it can be printed at the terminal.

All open files must be closed when you finish processing. Therefore, you must use three $CLOSE macro instructions.

Figure 5-1 lists the program code that accepts the key (relative record number) from the operator and displays the contents of that record on the terminal. Note that in this program, two macro definitions appear. The first builds the string that is displayed on the terminal. The second macro definition accepts input from SYS$INPUT and prompts with the string specified as its argument. Notice that both of these macro definitions make use of run-time macro instructions ($PUT and $GET) in their construction.

You will also note that this program is written in subroutines. Therefore, for some files, the $CLOSE macro instruction appears before the $OPEN or $CREATE macro instruction.

```
        .TITLE  DISPLAY - DISPLAY SPECIFIED RECORD
;
;  PROGRAM TO ACCEPT RECORD NUMBER FROM OPERATOR AND DISPLAY
;  CORRESPONDING RECORD
;

.MACRO  TYPE    STRING                               ;  MACRO TO TYPE "STRING"
        .SAVE                                        ;  SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS, NOWRT                  ;  CHANGE TO TYPE STRINGS PSECT
        ...TMPA=.                                    ;  NOTE ADDRESS
        .ASCII  \STRING\                             ;  STORE STRING
        ...TMPL=. -...TMPA                           ;  NOTE LENGTH
        .RESTORE                                     ;  BACK TO ORIGINAL PSECT
        MOVL    #...TMPA, TYPE_RAB+RAB$L_RBF         ;  SET STRING ADDRESS
        MOVW    #...TMPL, TYPE_RAB+RAB$W_RSZ         ;  SET STRING LENGTH
        $PUT    RAB=TYPE_RAB                         ;  WRITE THE RECORD
.ENDM
;
.MACRO  PROMPT  STRING                               ;  MACRO TO ACCEPT INPUT
                                                     ;  FROM SYS$INPUT, PROMPTING
                                                     ;  WITH "STRING"
        .SAVE                                        ;  SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS, NOWRT                  ;  CHANGE TO TYPE STRINGS PSECT
        ...TMPA=.                                    ;  NOTE ADDRESS
        .BYTE   13, 10                               ;  CARRIAGE RETURN, LINE FEED
        .ASCII  \STRING\                             ;  STORE STRING
        ...TMPL=. -...TMPA                           ;  NOTE LENGTH
        .RESTORE                                     ;  BACK TO ORIGINAL PSECT
        MOVL    #...TMPA, PROMPT_RAB+RAB$L_PBF       ;  SET PROMPT BUFFER ADDRESS
        MOVB    #...TMPL, PROMPT_RAB+RAB$B_PSZ       ;  SET PROMPT BUFFER SIZE
        $GET    RAB = PROMPT_RAB                     ;  GET THE INPUT
        MOVZWL  PROMPT_RAB+RAB$W_RSZ,R1              ;  GET INPUT LENGTH
        MOVL    PROMPT_RAB+RAB$L_RBF,R2              ;  GET INPUT ADDRESS
.ENDM
;
        .PSECT  DATA,LONG
TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-
                        RAT=CR
TYPE_RAB:       $RAB    FAB=TYPE_FAB
PROMPT_FAB:     $FAB    FNM=<SYS$INPUT:>
PROMPT_RAB:     $RAB    FAB=PROMPT_FAB,-
                        UBF=PROMPT_BUFF,-
                        USZ=132,-
                        ROP=PMT
PROMPT_BUFF:    .BLKB   132
;
INFAB:  $FAB    FNM=<INFILE:>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_BUFFER_SIZE,-
                KBF=KEY,-
                RAC=KEY
REC_BUFFER:     .BLKB   50                           ;  USER RECORD BUFFER
REC_BUFFER_SIZE=.-REC_BUFFER
        .ALIGN  LONG
KEY:    .BLKL   1                                    ;  RECORD NUMBER TO RETRIEVE
;
;
;  OPEN FILE,CONNECT STREAM
;
        .PSECT  CODE,NOWRT
BEGIN:  .WORD   0
        $OPEN   FAB=INFAB                            ;  OPEN INPUT FILE
        BLBC    R0,EXIT1                             ;  BRANCH ON ERROR
        $CONNECT        RAB=INRAB                    ;  CONNECT STREAM
        BLBS    R0,CONT1                             ;  BRANCH ON SUCCESS
EXIT1:  BRW     EXIT                                 ;  BRANCH ON ERROR
CONT1:  BSBW    INIT_TYPE                            ;  INITIALIZE TYPE AND PROMPT FILES
;
;  ACCEPT NUMBER OF RECORD TO BE DISPLAYED
;
GET_REC_NO:
        PROMPT  <ENTER RECORD NUMBER:>               ;  GET RECORD NUMBER
        BLBS    R0,CONT2                             ;  BRANCH ON SUCCESS
        BRW     DONE                                 ;  BRANCH ON ERROR
```

Figure 5-1 Random Read of a Sequential File

```
CONT2:  BSBW    CONVERT_KEY                     ; CONVERT KEY TO BINARY
        BLBC    R0,BAD_KEY                      ; BRANCH IF BAD
        MOVL    R3,KEY                          ; SET RECORD NUMBER
        $GET    RAB=INRAB                       ; GET RECORD FOR PART
        BLBC    R0,BAD_PART                     ; BRANCH ON ERROR
                                                ;
        $RAB_STORE      RAB=TYPE_RAB,-
                RBF=@INRAB+RAB$L_RBF,-
                RSZ=INRAB+RAB$W_RSZ
        $PUT    RAB=R0                          ; PRINT RECORD
        BLBC    R0,EXIT                         ; BRANCH ON ERROR
        BRW     GET_REC_NO                      ; LOOP
;
;  REPORT ERRORS
;
BAD_KEY:        TYPE    <BAD KEY VALUE!>
        BRW     GET_REC_NO
BAD_PART:       TYPE    <RECORD DOES NOT EXIST.>
        BRW     GET_REC_NO

;
;  ALL DONE  - CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB
        $CLOSE  FAB=TYPE_FAB
        $CLOSE  FAB=PROMPT_FAB
EXIT:   $EXIT_S R0
;++
;
;  SUBROUTINE TO CONVERT ASCII INPUT STRING TO BINARY
;
;  INPUTS:   R1, R2 = LENGTH AND ADDRESS OF INPUT STRING
;
;  OUTPUTS:  R0 - STATUS CODE
;            R3 - BINARY VALUE
;            R1, R2, R4 DESTROYED
;--
CONVERT_KEY:
        CLRQ    R3                              ; INITIALIZE OUTPUT VALUE
        BRB     20$                             ; GO CHECK IF ANY CHARACTERS
10$:    MULL2   #10, R3                         ; SHIFT PARTIAL RESULT
        BVS     30$                             ; BRANCH ON OVERFLOW
        SUBB3   #^A/0/, (R2)+, R4               ; GET BINARY VALUE FOR CHARACTER

        BLSS    30$                             ; BRANCH IF BAD
        CMPB    R4,#^A/9/-^A/0/                 ; CHARACTER > 9 ?
        BGTRU   30$                             ; BRANCH IF BAD
        ADDL2   R4, R3                          ; ADD IN CHARACTER TO PARTIAL RESUL
20$:    DECL    R1                              ; ANY MORE INPUT?
        BGEQ    10$                             ; BRANCH IF MORE
        MOVL    #1, R0                          ; SHOW SUCCESS
        RSB
30$:    CLRL    R0                              ; SHOW FAILURE
        RSB
;++
;
;  SUBROUTINE TO INITIALIZE THE TYPE AND PROMPT FILES
;
;--
INIT_TYPE:
        $CREATE FAB=TYPE_FAB
        $OPEN   FAB=PROMPT_FAB
        $CONNECT        RAB=TYPE_RAB
        $CONNECT        RAB = PROMPT_RAB
        RSB

        .END BEGIN
```

Figure 5-1 (Cont.) Random Read of a Sequential File

5.2 RELATIVE FILE ORGANIZATION

Random access to the relative file organization, like any access to the relative file organization, is available on disk devices only.

Relative file organization, unlike sequential file organization, does not require that records be fixed-length in order to use random access. Therefore, the relative file organization provides more flexibility for random access than does the sequential file organization. However, it does cost more in space requirements, since all record cells are the same size, and some (or all) may not be completely filled.

5.2.1 Random Read of a Record in the Relative File Organization

This section describes a sample program illustrated in Figure 5-2 that builds on the program listed in Figure 5-1. The only difference between the programs is that the input file in this program uses the relative file organization. Since it is an input file, you do not have to indicate the file organization when you open a file and you do not have to change the FAB to indicate the relative file organization. (Note, however, that you do have to change the input file FAB when you specify the $DELETE macro instruction. See the following discussion.)

This program, besides accepting the key (relative record number) from the operator and displaying the contents of the record on the terminal, also queries the operator as to whether or not the record should be deleted. Therefore, you must use a $DELETE macro instruction within the code that handles record deletion (lines 93 through 101 of Figure 5-2).

```
     $DELETE        RAB=INRAB
```

This $DELETE macro instruction points to the RAB for the input file. The relative file organization lets you delete a record from anywhere in the file, thereby leaving the record cell free to accept another record. You do not have to create a new file; the input file, in effect, is also the output file. (You cannot use the $DELETE macro instruction with the sequential file organization. To remove a record from a sequential file, you must use the $TRUNCATE macro instruction, but it is limited to removing a record, and any succeeding records, from the end of a file. There cannot be empty space in the sequential file organization, because it does not use the concept of record cells.)

When you specify the $DELETE macro instruction, you also must make a change to the input file FAB to indicate, in the file access field, that a delete operation can occur. Do this by adding FAC=<DEL> to the $FAB macro instruction. You can omit the angle brackets from DEL; you only need them if more than one operation applies. (In reality, more than one operation does apply to this file. For example, since you are also going to retrieve records, you could specify FAC=<DEL,GET>, to indicate the get operation. However, GET is implied by DEL, so you can omit it.)

```
     INFAB:         $FAB           FNM=<INFILE>,-
                                   FAC=<DEL>
```

Figure 5-2 lists the program code that accepts the key (relative record number) from you and displays the contents of that record on the terminal, with the option to delete the record.

Appendix A contains additional examples of random access to the relative file organization.

```
        .TITLE  DISPLAY - DISPLAY SPECIFIED RECORD
;
;  PROGRAM TO ACCEPT RECORD NUMBER FROM OPERATOR AND DISPLAY
;  CORRESPONDING RECORD
;
;
.MACRO  TYPE    STRING                          ; MACRO TO TYPE "STRING"
        .SAVE                                   ; SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS, NOWRT             ; CHANGE TO TYPE STRINGS PSECT
        ...TMPA=.                               ; NOTE ADDRESS
        .ASCII  \STRING\                        ; STORE STRING
        ...TMPL=. -...TMPA                      ; NOTE LENGTH
        .RESTORE                                ; BACK TO ORIGINAL PSECT
        MOVL    #...TMPA, TYPE_RAB+RAB$L_RBF    ; SET STRING ADDRESS
        MOVW    #...TMPL, TYPE_RAB+RAB$W_RSZ    ; SET STRING LENGTH
        $PUT    RAB=TYPE_RAB                    ; WRITE THE RECORD
.ENDM
;
.MACRO  PROMPT  STRING                          ; MACRO TO ACCEPT INPUT
                                                ; FROM SYS$INPUT, PROMPTING
                                                ; WITH "STRING"
        .SAVE                                   ; SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS, NOWRT             ; CHANGE TO TYPESTRINGS PSECT
        ...TMPA=.                               ; NOTE ADDRESS
        .BYTE   13, 10                          ; CARRIAGE RETURN, LINE FEED
        .ASCII  \STRING\                        ; STORE STRING
        ...TMPL=. -...TMPA                      ; NOTE LENGTH
        .RESTORE                                ; BACK TO ORIGINAL PSECT
        MOVL    #...TMPA, PROMPT_RAB+RAB$L_PBF  ; SET PROMPT BUFFER ADDRESS
        MOVB    #...TMPL, PROMPT_RAB+RAB$B_PSZ  ; SET PROMPT BUFFER SIZE
        $GET    RAB = PROMPT_RAB                ; GET THE INPUT
        MOVZWL  PROMPT_RAB+RAB$W_RSZ, R1        ; GET INPUT LENGTH
        MOVL    PROMPT_RAB+RAB$L_RBF,R2         ; GET INPUT ADDRESS
.ENDM
;
.MACRO  ON_ERROR        DEST,?L                 ; MACRO TO BRANCH ON ERROR
        BLBS    R0,L                            ; BRANCH ON SUCCESS
        BRW     DEST                            ; LONG FORM OF BRANCH

L:
.ENDM
;
        .PSECT  DATA,LONG
TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-
                        RAT=CR
TYPE_RAB:       $RAB    FAB=TYPE_FAB
PROMPT_FAB:     $FAB    FNM=<SYS$INPUT:>
PROMPT_RAB:     $RAB    FAB=PROMPT_FAB,-
                        UBF=PROMPT_BUFF,-
                        USZ=132,-
                        ROP=PMT
;
INFAB:  $FAB    FNM=<INFILE:>,-
                FAC=<DEL>
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_BUFFER_SIZE,-
                KBF=KEY,-
                RAC=KEY
;
PROMPT_BUFF:    .BLKB   132
REC_BUFFER:     .BLKB   50                      ; USER RECORD BUFFER
REC_BUFFER_SIZE=.-REC_BUFFER
        .ALIGN  LONG
KEY:    .BLKL   1                               ; RECORD NUMBER TO RETRIEVE
;
;  OPEN FILE,CONNECT STREAM
;
        .PSECT  CODE,NOWRT
BEGIN:  .WORD   0
        $OPEN   FAB=INFAB                       ; OPEN INPUT FILE
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        $CONNECT        RAB=INRAB               ; CONNECT STREAM
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
```

Figure 5-2 Random Read of a Relative File

```
        BSBW    INIT_TYPE                       ; INITIALIZE TYPE AND PROMPT FILES
;
;  ACCEPT NUMBER OF RECORD TO BE DISPLAYED
;
GET_REC_NO:
        PROMPT  <ENTER RECORD NUMBER:>          ; GET RECORD NUMBER
        ON_ERROR        DONE                    ; BRANCH ON ERROR (E.G., EOF)
        BSBW    CONVERT_KEY                     ; CONVERT KEY TO BINARY
        ON_ERROR        BAD_KEY                 ; BRANCH IF BAD
        MOVL    R3,KEY                          ; SET RECORD NUMBER
        $GET    RAB=INRAB                       ; GET RECORD FOR PART
        ON_ERROR        BAD_PART                ; BRANCH ON ERROR
        TYPE    <RECORD IS:>                    ;
        $RAB_STORE      RAB=TYPE_RAB,-
                RBF=@INRAB+RAB$L_RBF,-
                RSZ=INRAB+RAB$W_RSZ
        $PUT    RAB=R0                          ; PRINT RECORD
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        PROMPT  <DELETE RECORD (Y/N)?>          ; ASK IF RECORD SHOULD BE DELETED
        ON_ERROR        DONE                    ; BRANCH ON ERROR
        TSTW    R1                              ; ZERO LENGTH INPUT?
        BEQL    GETNXT                          ; BRANCH IF YES
        CMPB    (R2),#^A/Y/                     ; ANSWER START WITH 'Y'?
        BNEQ    GETNXT                          ; BRANCH IF NOT
        $DELETE RAB=INRAB                       ; DELETE RECORD
        ON_ERROR        EXIT                    ; BRANCH ON FAILURE
        TYPE    <RECORD DELETED.>
GETNXT:
        BRW     GET_REC_NO                      ; LOOP
;
;  REPORT ERRORS
;
BAD_KEY:        TYPE    <BAD KEY VALUE!>
        BRW     GET_REC_NO
BAD_PART:       TYPE    <RECORD DOES NOT EXIST.>
        BRW     GET_REC_NO

;
;  ALL DONE  - CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB
        $CLOSE  FAB=TYPE_FAB
        $CLOSE  FAB=PROMPT_FAB
EXIT:   $EXIT_S R0
;++
;
;  SUBROUTINE TO CONVERT ASCII INPUT STRING TO BINARY
;
;  INPUTS:   R1, R2 = LENGTH AND ADDRESS OF INPUT STRING
;
;  OUTPUTS:  R0 - STATUS CODE
;            R3 - BINARY VALUE
;            R1, R2, R4 DESTROYED
;--
CONVERT_KEY:
        CLRQ    R3                              ; INITIALIZE OUTPUT VALUE
        BRB     20$                             ; GO CHECK IF ANY CHARACTERS
10$:    MULL2   #10, R3                         ; SHIFT PARTIAL RESULT
        BVS     30$                             ; BRANCH ON OVERFLOW
        SUBB3   #^A/0/, (R2)+, R4               ; GET BINARY VALUE FOR CHARACTER

        BLSS    30$                             ; BRANCH IF BAD
        CMPB    R4,#^A/9/-^A/0/                 ; CHARACTER > 9 ?
        BGTRU   30$                             ; BRANCH IF BAD
        ADDL2   R4, R3                          ; ADD IN CHARACTER TO PARTIAL RESUL
20$:    DECL    R1                              ; ANY MORE INPUT?
        BGEQ    10$                             ; BRANCH IF MORE
        MOVL    #1, R0                          ; SHOW SUCCESS
        RSB
30$:    CLRL    R0                              ; SHOW FAILURE
        RSB
;++
;
;  SUBROUTINE TO INITIALIZE THE TYPE AND PROMPT FILES
```

Figure 5-2 (Cont.) Random Read of a Relative File

```
;
;--
INIT_TYPE:
        $CREATE FAB=TYPE_FAB
        $OPEN   FAB=PROMPT_FAB
        $CONNECT        RAB=TYPE_RAB
        $CONNECT        RAB = PROMPT_RAB
        RSB

        .END BEGIN
```

Figure 5-2 (Cont.) Random Read of a Relative File

5.3 INDEXED FILE ORGANIZATION

Random access to the indexed file organization, like any access to the indexed file organization, is available on disk devices only.

In an indexed file, random access by key is independent of the record format (either fixed or variable). Therefore, the indexed file provides more flexibility for random access than does the relative or sequential file organizations.

5.3.1 Random Read of a Record in the Indexed File Organization

This section describes a sample program, illustrated in Figure 5-3, that builds upon the program listed in Figure 5-1. The major difference between the programs is that the input file in this program uses the indexed file organization. Since it is an input file, you do not have to indicate the file organization when you open a file.

This program, besides accepting the key (the part number) from the operator and displaying the contents of the record on the terminal, also modifies the discount type field of that record to contain an A. Then this program sequentially accesses and displays any subsequent records containing part numbers in which the first four characters match those of the first record accessed. Therefore, you must use a $UPDATE macro instruction within the code that handles record updating (lines 94 through 103 of Figure 5-3).

```
     $UPDATE    RAB=INRAB
```

This $UPDATE macro instruction points to the RAB for the input file.

Assume that the following external assignment will be made:

```
     $ ASSIGN   18SEP78.INV     INFILE:
```

You must provide this program with definitions for three files: an output file, a file to accept the request, and an input file (where you define that the record access mode is random, since the input file is the one you search for the records).

OUTPUT FILE

The first file that must be defined is the output file, SYS$OUTPUT:, which is a process logical name assigned for the output stream. For an interactive user, SYS$OUTPUT is a terminal. The FAB for this file only has to provide this name and an optional record attribute that induces a line feed before and a carriage return after printing the record at the terminal.

```
     TYPE_FAB:        $FAB          FNM=<SYS$OUTPUT:>,-
                                    RAT=CR
```

At assembly time, the $RAB macro instruction only has to associate the RAB with the FAB.

```
     TYPE_RAB:        $RAB          FAB=TYPE_FAB
```

The actual contents of the RAB are defined dynamically, at run time rather than at assembly time, with a $RAB_STORE macro instruction. The reason for this is that the record to be output varies. On one hand, records from the input file are displayed (see lines 111 through 114 of Figure 5-3), while on the other hand, a number of fixed strings are output using the "TYPE" macro (see lines 124,128, and 134; the macro definition itself appears on lines 11 through 22). Each of the different outputs require that the RSZ and RBF parameters be set dynamically to indicate the record to be written.

The $RAB_STORE macro instruction (see line 111) indicates the symbolic address of the RAB allocated at assembly time. It must also define the location and size of the buffer that contains the record to be printed on SYS$OUTPUT. When displaying records read from the input file, the location and size are at the address of INRAB (the input RAB) plus the offset to each field (RAB$L_RBF for the address and RAB$W_RSZ for the size).

```
     $RAB_STORE          RAB=TYPE_RAB,-
                         RBF=@INRAB+RAB$L_RBF,-
                         RSZ=INRAB+RAB$W_RSZ
```

REQUEST FILE

The second file that must be defined is the request file, which prompts a message to solicit information from the operator and accepts the requested record number from the terminal. This file (see line 52) is SYS$INPUT:, which is a process logical name. Note that for an interactive process, SYS$INPUT and SYS$OUTPUT both refer to a terminal. In this case, it would be possible to use the same file name (either SYS$INPUT or SYS$OUTPUT) to accept requests and display output. In so doing, however, you would lose the ability to run the program within a batch stream.

```
     PROMPT_FAB:       $FAB          FNM=<SYS$INPUT:>
```

The RAB you connect to this FAB defines a buffer area and associates the RAB with the FAB. The RAB also defines a record processing option of ROP=PMT. This option indicates that the contents of the specified prompt buffer (filled as part of the expansion of the "PROMPT" macro) are to be output to the terminal operator in order to indicate what data is being requested for output.

```
     PROMPT_RAB:       $RAB          FAB=PROMPT_FAB,-
                                     UBF=PROMPT_BUFF,-
                                     USZ=132,-
                                     ROP=PMT
```

INPUT FILE

The third file that must be defined is the input file (see line 60), which must provide the file specification. The external assignment equates 18SEP78.INV to INFILE:.

When you specify the $UPDATE macro instruction, you also must make a change to the input file FAB to indicate, in the file access field, that an update operation can occur. Do this by adding FAC=<UPD> to the $FAB macro instruction. You can omit the angle brackets from UPD; you need them only if more than one operation applies. (In reality, more than one operation does apply to this file. For example, since you are also going to retrieve records, you could specify FAC=<UPD,GET> to indicate the get operation. However, GET is implied by UPD, so you can omit it.)

```
INFAB:      $FAB      FNM=<INFILE:>,-
                      FAC=UPD
```

When the three files are defined, you must use run-time macro instructions to call the routines that act on these files the same as described in Section 5.1.1 for the program listed in Figure 5-1.

Each file you open in the program must have a RAB connected to the appropriate FAB with a $CONNECT macro instruction.

For the input file, use a $GET macro instruction to retrieve the record. For the output file, use a $PUT macro instruction to place the record in SYS$OUTPUT so it can be printed at the terminal.

All open files must be closed when you finish processing. Therefore, you must use three $CLOSE macro instructions.

You switch from random to sequential access mode (see line 116, Figure 5-3) in order to access and display any subsequent records containing part numbers (the primary key) in which the first four characters match those of the first record accessed as follows:

```
$RAB_STORE            RAB=INRAB,-
                      RAC=SEQ
```

Since you are accessing an existing indexed file, you do not have to specify the position or size of the key. However you must specify the key to search on. In this example, the primary key (key 0) is specified by default.

Figure 5-3 lists the code for this program.

Appendix A contains additional examples of random access to an indexed file.

```
        .TITLE  DISPLAY  -  DISPLAY RELATED RECORDS

;
;  PROGRAM TO ACCEPT PART # FROM OPERATOR AND DISPLAY
;  CORRESPONDING RECORD AS WELL AS ALL SUBSEQUENT RECORDS THAT
;  MATCH THE FIRST FOUR CHARACTERS OF THE PART NUMBER.
;  MODIFY THE DISCOUNT TYPE FIELD OF THE FIRST RECORD ACCESSED
;  TO CONTAIN AN "A".
;

.MACRO  TYPE    STRING                          ;  MACRO TO TYPE "STRING"

        .SAVE                                   ;  SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS,NOWRT              ;  CHANGE TO TYPE STRING
        ...TMPA=.                               ;  NOTE ADDRESS
        .ASCII  \STRING\                        ;  STORE STRING
        ...TMPL=.-...TMPA                       ;  NOTE LENGTH
        .RESTORE                                ;  BACK TO ORIGINAL PSECT
        MOVL    #...TMPA,TYPE_RAB+RAB$L_RBF     ;  SET STRING ADDRESS
        MOVW    #...TMPL,TYPE_RAB+RAB$W_RSZ     ;  SET STRING LENGTH
        $PUT    RAB=TYPE_RAB                    ;  WRITE THE RECORD
.ENDM
;
.MACRO  PROMPT  STRING                          ;  MACRO TO ACCEPT INPUT
                                                ;  FROM SYS$INPUT, PROMPTING
                                                ;  WITH "STRING"
        .SAVE                                   ;  SAVE CURRENT PSECT
        .PSECT  TYPE_STRINGS,NOWRT              ;  CHANGE TO TYPE STRINGS PSECT
        ...TMPA=.                               ;  NOTE ADDRESS
        .BYTE   13,10                           ;  CARRIAGE RETURN,LINE FEED
        .ASCII  \STRING\                        ;  STORE STRING
        ...TMPL=.-...TMPA                       ;  NOTE LENGTH
        .RESTORE                                ;  BACK TO ORIGINAL PSECT
        MOVL    #...TMPA,PROMPT_RAB+RAB$L_PBF   ;  SET PROMPT BUFFER ADDRESS
        MOVB    #...TMPL,PROMPT_RAB+RAB$B_PSZ   ;  SET PROMPT BUFFER SIZE
        $GET    RAB=PROMPT_RAB
        MOVZWL  PROMPT_RAB+RAB$W_RSZ,R1
        MOVL    PROMPT_RAB+RAB$L_RBF,R2
.ENDM
;
.MACRO  ON_ERROR        DEST,?L                 ;  MACRO TO BRANCH ON ERROR

        BLBS    R0,L                            ;  CONTINUE ON SUCCESS
        BRW     DEST                            ;  BRANCH LONG ON ERROR
L:
.ENDM
;
        .PSECT  DATA,LONG
TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-
                        RAT=CR
TYPE_RAB:       $RAB    FAB=TYPE_FAB
PROMPT_FAB:     $FAB    FNM=<SYS$INPUT:>
PROMPT_RAB:     $RAB    FAB=PROMPT_FAB,-
                        UBF=PROMPT_BUFF,-
                        USZ=132,-
                        ROP=PMT
;
;  INPUT FILE FAB AND RAB AND XABS
;
INFAB:  $FAB    FNM=<INFILE:>,-
                FAC=UPD
;
INRAB:  $RAB    FAB=INFAB,-
                UBF=REC_BUFFER,-
                USZ=REC_BUFFER_SIZE,-
                KBF=KEY_BUFF,-
                KSZ=KEY_BUFF_SIZE
;
;
PROMPT_BUFF:    .BLKB   132
REC_BUFFER:     .BLKB   50
REC_BUFFER_SIZE=.-REC_BUFFER
DISCOUNT_TYPE=REC_BUFFER+5

```

Figure 5-3 Random Read of an Indexed File

```
        .ALIGN  LONG
KEY_BUFF:       .BLKB   5                       ; PART # OF RECORD TO RETRIEVE
KEY_BUFF_SIZE=.-KEY_BUFF
MATCH_PART_NO:  .BLKL   1                       ; FIRST 4 CHARACTERS OF THE PART #
MATCH_FLAG:     .BLKB   0                       ; SET TO 1 IF RELATED RECORD SEEN
;
;  OPEN FILE, CONNECT STREAM
;
        .PSECT  CODE,NOWRT
BEGIN:  .WORD   0
        $OPEN   FAB=INFAB                       ; OPEN INPUT FILE
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        $CONNECT        RAB=INRAB               ; CONNECT STREAM
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        BSBW    INIT_TYPE                       ; INITIALIZE TYPE AND PROMPT FILES
;
;  ACCEPT PART NUMBER OF RECORD TO BE DISPLAYED
;
GET_PART_NO:
        PROMPT  <ENTER PART NUMBER:>            ; GET PART NUMBER
        ON_ERROR        DONE                    ; BRANCH IF DONE
        MOVC5   R1,(R2),#^A/0/,-                ; MOVE PART NUMBER INTO THE
                #5,KEY_BUFF                     ; KEY BUFFER, ZERO FILLING
        $RAB_STORE      RAB=INRAB,-             ; KEY ACCESS TO ACCESS RECORD
                RAC=KEY                         ; WITH SPECIFIED PART #
        $GET    RAB=INRAB                       ; GET RECORD WITH PART#=KEY
        ON_ERROR        BAD_PART                ; BRANCH IF RECORD NOT FOUND
        MOVB    #^A/A/,DISCOUNT_TYPE            ; MODIFY DISCOUNT TYPE TO 'A'
        $UPDATE RAB=INRAB                       ; UPDATE RECORD, WITH NEW
                                                ; DISCOUNT TYPE
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        TYPE    <RECORD CHANGED TO:>
        CLRB    MATCH_FLAG                      ; SAY NO RELATED RECORDS SEEN
        MOVL    @INRAB+RAB$L_RBF,MATCH_PART_NO  ; SAVE FIRST 4 CHARACTERS OF
                                                ; PART # TO MATCH
DISPLAY:
        $RAB_STORE      RAB=TYPE_RAB,-
                RBF=@INRAB+RAB$L_RBF,-
                RSZ=INRAB+RAB$W_RSZ
        $PUT    RAB=R0                          ; PRINT RECORD
        ON_ERROR        EXIT                    ; BRANCH ON ERROR
        $RAB_STORE      RAB=INRAB,-
                RAC=SEQ                         ; SWITCH TO SEQUENTIAL ACCESS
        $GET    RAB=R0                          ; GET NEXT RECORD
        BLBC    R0,CHECK_RELATED                ; END OF FILE?
        CMPL    @INRAB+RAB$L_RBF,MATCH_PART_NO  ; IS THIS A MATCH?
        BNEQ    CHECK_RELATED                   ; ALL DONE MATCHING
        BBSS    #1,MATCH_FLAG,DISPLAY           ; BRANCH IF HEADER HAS ALREADY
                                                ; BEEN PRINTED
        TYPE    <RELATED RECORD(S):>
        BRB     DISPLAY                         ; LOOP TO GET NEXT MATCH
CHECK_RELATED:
        BBS     #1,MATCH_FLAG,GETNEXT           ; BRANCH IF RELATED RECORDS PRINTED
        TYPE    <NO RELATED RECORDS.>
GETNEXT:
        BRW     GET_PART_NO                     ; LOOP TO GET NEXT PART #
;  REPORT ERRORS
;
BAD_PART:
        TYPE    <RECORD DOES NOT EXIST.>
        BRW     GET_PART_NO                     ; LOOP TO GET NEXT PART #
;
;  ALL DONE - CLOSE FILES AND EXIT
;
DONE:   $CLOSE  FAB=INFAB
        $CLOSE  FAB=TYPE_FAB
        $CLOSE  FAB=PROMPT_FAB

EXIT:   $EXIT_S R0
;
;++
```

Figure 5-3 (Cont.) Random Read of an Indexed File

```
;
;  SUBROUTINE TO INITIALIZE THE TYPE AND PROMPT FILES
;
;--
INIT_TYPE:
        $CREATE FAB=TYPE_FAB
        $OPEN   FAB=PROMPT_FAB
        $CONNECT        RAB=TYPE_RAB
        $CONNECT        RAB=PROMPT_RAB
        RSB

        .END    BEGIN
```

Figure 5-3 (Cont.) Random Read of an Indexed File

APPENDIX A

PROGRAM EXAMPLES

This appendix contains additional program examples that you can examine to gain a better understanding of VAX-11 RMS. They are somewhat more detailed than the examples in Chapters 4 and 5; but you may find that a study of their construction, in conjunction with the VAX-11 Record Management Services Reference Manual, is quite beneficial.

```
REORDER         - INDICATE REORDERED ITEMS               21-JUL-1978  14:36:23   VAX-11 MACRO X0.3-9          Page   1
                                                                                                                      (1)

                      0000          .TITLE REORDER - INDICATE REORDERED ITEMS
                      0000
                      0000   ;
                      0000   ;  PROGRAM TO READ THE OLD INVENTORY MASTER FILE AND CREATE A
                      0000   ;  NEW MASTER FILE, RECOGNIZING THOSE ITEMS WITH AN ON-HAND
                      0000   ;  QUANTITY LESS THAN THE REORDER QUANTITY, AND SETTING THE REORDER
                      0000   ;  DATE IN THE NEW MASTER FILE TO TODAY'S DATE, AND LISTING THE
                      0000   ;  RECORD ON SYS$OUTPUT.
                      0000   ;
                      0000   .MACRO  TYPE    STRING                          ;  MACRO TO TYPE "STRING"
                      0000           .SAVE                                   ;  SAVE CURRENT PSECT
                      0000           .PSECT  TYPE_STRINGS,NOWRT              ;  CHANGE TO TYPE STRINGS PSECT
                      0000           ...TMPA=.                               ;  NOTE ADDRESS
                      0000           .ASCII  \STRING\                        ;  STORE STRING
                      0000           ...TMPL=.-...TMPA                       ;  NOTE LENGTH
                      0000           .RESTORE                                ;  BACK TO ORIGINAL PSECT
                      0000           MOVL    #...TMPA,TYPE_RAB+RAB$L_RBF     ;  SET STRING ADDRESS
                      0000           MOVW    #...TMPL,TYPE_RAB+RAB$W_RSZ     ;  SET STRING LENGTH
                      0000           $PUT    RAB=TYPE_RAB                    ;  WRITE THE RECORD
                      0000   .ENDM
                      0000   ;
            00000032  0000   REC_SIZE=50                                     ;  RECORD LENGTH
                  00000000          .PSECT  DATA,LONG
                      0000   TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-     ;  FAB FOR USE WITH TYPE MACRO
                      0000                           RAT=CR
                      0050   TYPE_RAB:       $RAB    FAB=TYPE_FAB            ;  RAB FOR USE WITH TYPE MACRO
                      0094   ;
                      0094   INFAB:  $FAB    FNM=<INFILE:>
                      00E4   INRAB:  $RAB    FAB=INFAB,-
                      00E4                   UBF=REC_BUFFER,-
                      00E4                   USZ=REC_SIZE
                      0128   OUTFAB: $FAB    FNM=<OUTFILE:>
                      0178   OUTRAB: $RAB    FAB=OUTFAB
                      018C   ;
                      018C   ;  DEFINE FIELDS OF RECORD
                      018C   ;
            00000005  018C   PART_NO_LEN=5
            00000014  018C   PART_DESC_LEN=20
            00000004  018C   QTY_LEN=4
            00000009  018C   DATE_LEN=9
            00000007  018C   PRICE_LEN=7
                      018C   ;
                      018C   REC_BUFFER:
            000001C1  018C   PART_NUMBER:    .BLKB   PART_NO_LEN
            000001C2  01C1   DISCOUNT_TYPE:  .BLKB   1
            000001D6  01C2   PART_DESCRIPT:  .BLKB   PART_DESC_LEN
            000001DA  01D6   QTY_ON_HAND:    .BLKB   QTY_LEN
            000001DE  01DA   REORDER_QTY:    .BLKB   QTY_LEN
            000001E7  01DE   REORDER_DATE:   .BLKB   DATE_LEN
            000001EE  01E7   LIST_PRICE:     .BLKB   PRICE_LEN
                      01EE   ;
```

```
                    01EE   ;
                    01EE   ;  BUFFER TO FORMAT PRINT RECORD
                    01EE   ;
              20    01EE   TYPE_BUF:       .ASCII  / /
        000001F4    01EF   TYPE_PART:      .BLKB   PART_NO_LEN
              20    01F4                   .ASCII  / /
        00000209    01F5   TYPE_DESC:      .BLKB   PART_DESC_LEN
  20 20 20 20 20    0209                   .ASCII  /     /
        00000212    020E   ON_HAND:        .BLKB   QTY_LEN
  20 20 20 20 20    0212                   .ASCII  /     /
        0000021B    0217   REORDER:        .BLKB   QTY_LEN
        0000002D    021B   TYPE_LEN=.-TYPE_BUF
              00    021B   HEADING:        .BYTE 0
                    021C           .ALIGN LONG
                    021C   ;  BUFFER TO GET CURRENT DATE
        0000000B    021C   DATE_BUF:       .LONG   11                        ;  LENGTH OF BUFFER
        00000224'   0220                   .LONG   TODAYS_DATE               ;  ADDRESS OF BUFFER
        0000022B    0224   TODAYS_DATE:    .BLKB   7                         ;  DD-MON-
        0000022D    022B   YR_CENTURY:     .BLKB   2                         ;  YY
        0000022F    022D   YEAR:           .BLKB   2                         ;  YY
            00000000       .PSECT  CODE,NOWRT
                    0000   ;
                    0000   ;  INITIALIZATION - OPEN INPUT AND OUTPUT FILES, CONNECT STREAMS, AND
                    0000   ;                  GET TODAY'S DATE
                    0000   ;
                    0000
```

```
REORDER         - INDICATE REORDERED ITEMS              21-JUL-1978  14:36:23   VAX-11 MACRO X0.3-9            Page   3
                                                                                                                      (1)
                              0000  0000     80 START:   .WORD
                                    0002     81          $OPEN    FAB=INFAB                         ; OPEN INPUT FILE
          3B          50        E9  000F     82          BLBC     R0,EXIT1                          ; BRANCH ON ERROR
                                    0012     83          $FAB_STORE       FAB=OUTFAB,-              ; INITIALIZE OUTPUT FAB FROM INPUT
                                    0012     84                   RFM=FAB$B_RFM+INFAB,-             ; SET RECORD FORMAT
                                    0012     85                   MRS=FAB$W_MRS+INFAB,-             ; SET RECORD SIZE
                                    0012     86                   RAT=FAB$B_RAT+INFAB               ; SET RECORD ATTRIBUTE
                                    0031     87          $CREATE FAB=R0                             ; OPEN OUTPUT FILE
          10          50        E9  003A     88          BLBC     R0,EXIT1                          ; BRANCH ON ERROR
                                    003D     89          $CONNECT         RAB=INRAB                 ; CONNECT INPUT RAB
          03          50        E8  004A     90          BLBS     R0,CONT1                          ; BRANCH ON SUCCESS
                      01AF      31  004D     91 EXIT1:   BRW      EXIT                              ; BRANCH ON ERROR
                                    0050     92
                                    0050     93 CONT1:   $CONNECT         RAB=OUTRAB                ; CONNECT OUTPUT RAB
          ED          50        E9  005D     94          BLBC     R0,EXIT1                          ; BRANCH ON ERROR
                                    0060     95          $ASCTIM_S        TIMBUF=DATE_BUF           ; GET CURRENT DATE
 0000022B'EF   0000022D'EF      B0  0073     96          MOVW     YEAR,YR_CENTURY                   ; MAKE INTO "YY" FORMAT
                                    007E     97                                                     ; (RATHER THAN "YYYY")
                                    007E     98          $OPEN    FAB=TYPE_FAB                      ; OPEN REPORT FILE
          BF          50        E9  0089     99          BLBC     R0,EXIT1                          ; BRANCH ON ERROR
                                    008E    100          $CONNECT         RAB=TYPE_RAB              ; CONNECT REPORT RAB
          AF          50        E9  0098    101          BLBC     R0,EXIT1                          ; BRANCH ON ERROR
                                    009E    102
                                    009E    103          TYPE     <LIST OF INVENTORY ITEMS BELOW REORDER POINT>
                                    00BD    104          TYPE
                                    00DC    105
                                    00DC    106 ;
                                    00DC    107 ;   COPY RECORDS FROM OLD MASTER TO NEW MASTER CHECKING QUANTITY
                                    00DC    108 ;   ON HAND VS. REORDER QUANTITY
                                    00DC    109 ;
                                    00DC    110 READ:    $GET     RAB=INRAB                         ; READ A RECORD
          03          50        E8  00E9    111          BLBS     R0,10$                            ; BRANCH ON SUCCESS
                      00C2      31  00EC    112          BRW      DONE                              ; FINISH BRANCH ON ERROR
 000001D6'EF          04        29  00EF    113 10$:     CMPC3    #QTY_LEN,QTY_ON_HAND,REORDER_QTY
               000001DA'EF          00F6
                                    00FB    114                                                     ; ON-HAND LESS THAN REORDER QTY?
                      03        19  00FB    115          BLSS     20$                               ; BRANCH IF YES
                      009C      31  00FD    116          BRW      WRITE                             ; OMIT REORDER PROCESSING IF NOT
 00000224'EF          09        28  0100    117 20$:     MOVC3    #DATE_LEN,TODAYS_DATE,REORDER_DATE
               000001DE'EF          0107
                                    010C    118                                                     ; SET REORDER DATE TO TODAY'S DATE
 0000021B'EF          01        E2  010C    119          BBSS     #1,HEADING,REPORT_ITEM            ; BRANCH IF HEADING ALREADY PRINTED
                      3E            0113
                                    0114    120          TYPE     <PART #  PART DESCRIPTION   ON HAND REORDER PT.>
                                    0133    121          TYPE
                                    0152    122 REPORT_ITEM:                                        ; BUILD REPORT RECORD
 000001BC'EF          05        28  0152    123          MOVC3    #PART_NO_LEN,PART_NUMBER,TYPE_PART
               000001EF'EF          0159
 000001C2'EF          14        28  015E    124          MOVC3    #PART_DESC_LEN,PART_DESCRIPT,TYPE_DESC
               000001F5'EF          0165
 0000020E'EF   000001D6'EF      D0  016A    125          MOVL     QTY_ON_HAND,ON_HAND
```

```
REORDER         - INDICATE REORDERED ITEMS                    21-JUL-1978  14:36:23   VAX-11 MACRO X0.3-9            Page   4
                                                                                                                               (1)

   00000217'EF   000001DA'EF     D0  0175     126          MOVL     REORDER_QTY,REORDER
                                     0180     127          $RAB_STORE      RAB=TYPE_RAB,-
                                     0180     128                   RBF=TYPE_BUF,-
                                     0180     129                   RSZ=#TYPE_LEN
                                     0193     130          $PUT     RAB=R0                        ;  PRINT REPORT RECORD
                                     019C     131 WRITE:   $PUT     RAB=OUTRAB                    ;  WRITE NEW MASTER RECORD
            02            50     E8  01A9     132          BLBS     R0,READ1                      ;  BRANCH TO READ
                          51     11  01AC     133          BRB      EXIT                          ;  BRANCH ON ERROR
                          FF28   31  01AE     134 READ1:   BRW      READ                          ;  BRANCH ON SUCCESS
                                     01B1     135
                                     01B1     136 ;
                                     01B1     137 ;  ALL SET - CLOSE FILES AND EXIT
                                     01B1     138 ;
                                     01B1     139 DONE:    $CLOSE   FAB=INFAB
                                     01BE     140          $CLOSE   FAB=OUTFAB
   0000021B'EF            01     E0  01CB     141          BBS      #1,HEADING,CLOSE_TYPE         ;  BRANCH IF HEADING PRINTED
                          1F         01D2
                                     01D3     142          TYPE     <NONE>                        ;  INDICATE NO ITEMS REORDERED
                                     01F2     143 CLOSE_TYPE:                                     ;  INDICATE NO ITEMS REORDERED
                                     01F2     144          $CLOSE   FAB=TYPE_FAB
                                     01FF     145 EXIT:    $EXIT_S R0
                                     0208     146
                                     0208     147          .END START
```

```
DISPLAY         - DISPLAY RELATED RECORDS             14-JUL-1978  12:53:13   VAX-11 MACRO X0.3-11           Page  1
                                                                                                                  (1)

    0000          .TITLE  DISPLAY - DISPLAY RELATED RECORDS
    0000  ;
    0000  ;  PROGRAM TO ACCEPT RECORD NUMBER FROM OPERATOR AND DISPLAY
    0000  ;  CORRESPONDING RECORD AS WELL AS ALL SUBSEQUENT RECORDS THAT
    0000  ;  MATCH THE FIRST FOUR CHARACTERS OF THE PART NUMBER.
    0000  ;  MODIFY THE DISCOUNT TYPE FIELD OF THE FIRST RECORD ACCESSED
    0000  ;  TO CONTAIN AN 'A'.
    0000  ;
    0000
    0000  .MACRO  TYPE     STRING                                  ;  MACRO TO TYPE "STRING"
    0000          .SAVE                                            ;  SAVE CURRENT PSECT
    0000          .PSECT  TYPE_STRINGS, NOWRT                      ;  CHANGE TO TYPE STRINGS PSECT
    0000          ...TMPA=.                                        ;  NOTE ADDRESS
    0000          .ASCII  \STRING\                                 ;  STORE STRING
    0000          ...TMPL=. -...TMPA                               ;  NOTE LENGTH
    0000          .RESTORE                                         ;  BACK TO ORIGINAL PSECT
    0000          MOVL    #...TMPA, TYPE_RAB+RAB$L_RBF             ;  SET STRING ADDRESS
    0000          MOVW    #...TMPL, TYPE_RAB+RAB$W_RSZ             ;  SET STRING LENGTH
    0000          $PUT    RAB=TYPE_RAB                             ;  WRITE THE RECORD
    0000  .ENDM
    0000  ;
    0000  .MACRO  PROMPT   STRING                                  ;  MACRO TO ACCEPT INPUT
    0000                                                           ;  FROM SYS$INPUT, PROMPTING
    0000                                                           ;  WITH "STRING"
    0000          .SAVE                                            ;  SAVE CURRENT PSECT
    0000          .PSECT  TYPE_STRINGS, NOWRT                      ;  CHANGE TO TYPESTRINGS PSECT
    0000          ...TMPA=.                                        ;  NOTE ADDRESS
    0000          .BYTE   13, 10                                   ;  CARRIAGE RETURN, LINE FEED
    0000          .ASCII  \STRING\                                 ;  STORE STRING
    0000          ...TMPL=. -...TMPA                               ;  NOTE LENGTH
    0000          .RESTORE                                         ;  BACK TO ORIGINAL PSECT
    0000          MOVL    #...TMPA, PROMPT_RAB+RAB$L_PBF           ;  SET PROMPT BUFFER ADDRESS
    0000          MOVB    #...TMPL, PROMPT_RAB+RAB$B_PSZ           ;  SET PROMPT BUFFER SIZE
    0000          $GET    RAB = PROMPT_RAB                         ;  GET THE INPUT
    0000          MOVZWL  PROMPT_RAB+RAB$W_RSZ, R1                 ;  GET INPUT LENGTH
    0000          MOVL    PROMPT_RAB+RAB$L_RBF,R2                  ;  GET INPUT ADDRESS
    0000  .ENDM
    0000  ;
    0000  .MACRO  ON_ERROR         DEST,?L                         ;  MACRO TO BRANCH ON ERROR
    0000          BLBS    R0,L                                     ;  BRANCH ON SUCCESS
    0000          BRW     DEST                                     ;  LONG FORM OF BRANCH
    0000  L:
    0000  .ENDM
    0000  ;
00000000          .PSECT  DATA,LONG
    0000  TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-
    0000                          RAT=CR
    0050  TYPE_RAB:       $RAB    FAB=TYPE_FAB
    0094  PROMPT_FAB:     $FAB    FNM=<SYS$INPUT:>
    00E4  PROMPT_RAB:     $RAB    FAB=PROMPT_FAB,-
    00E4                          UBF=PROMPT_BUFF,-
    00E4                          USZ=132,-
    00E4                          ROP=PMT
    0128  ;
    0128  ;
    0128  INFAB:  $FAB    FNM=<INFILE:>,-
    0128                  FAC=<UPD>
```

```
DISPLAY         - DISPLAY RELATED RECORDS                     14-JUL-1978  12:53:13   VAX-11 MACRO X0.3-11            Page  2
                                                                                                                           (1)

                                   0178      58 INRAB:   $RAB     FAB=INFAB,-
                                   0178      59                   UBF=REC_BUFFER,-
                                   0178      60                   USZ=REC_BUFFER_SIZE,-
                                   0178      61                   KBF=KEY,-
                                   0178      62                   RAC=KEY
                                   01BC      63 ;
                         00000240  01BC      64 PROMPT_BUFF:      .BLKB   132
                         00000272  0240      65 REC_BUFFER:       .BLKB   50                          ;  USER RECORD BUFFER
                         00000032  0272      66 REC_BUFFER_SIZE=.-REC_BUFFER
                         00000245  0272      67          DISCOUNT_TYPE=REC_BUFFER+5
                                   0272      68          .ALIGN  LONG
                         00000278  0274      69 KEY:     .BLKL   1                                    ;  RECORD NUMBER TO RETRIEVE
                         0000027C  0278      70 MATCH_PART_NO:   .BLKL   1                            ;  FIRST 4 CHARACTERS OF PART NUMBER
                         0000027C  027C      71 MATCH_FLAG:      .BLKB   0                            ;  SET TO 1 IF RELATED RECORD SEEN
                                   027C      72 ;
                                   027C      73 ;  OPEN FILE,CONNECT STREAM
                                   027C      74 ;
                               00000000      75          .PSECT  CODE,NOWRT
                             0000  0000      76 BEGIN:   .WORD   0
                                   0002      77          $OPEN   FAB=INFAB                            ;  OPEN INPUT FILE
                                   000F      78          ON_ERROR        EXIT                         ;  BRANCH ON ERROR
                                   0015      79          $CONNECT        RAB=INRAB                    ;  CONNECT STREAM
                                   0022      80          ON_ERROR        EXIT                         ;  BRANCH ON ERROR
                      0209     30  0028      81          BSBW    INIT_TYPE                            ;  INITIALIZE TYPE AND PROMPT FILES
                                   0028      82 ;
                                   0028      83 ;  ACCEPT NUMBER OF RECORD TO BE DISPLAYED
                                   0028      84 ;
                                   0028      85 GET_REC_NO:
                                   0028      86          PROMPT  <ENTER RECORD NUMBER:>               ;  GET RECORD NUMBER
                                   0058      87          ON_ERROR        DONE                         ;  BRANCH ON ERROR (E.G., EOF)
                      0181     30  005E      88          BSBW    CONVERT_KEY                          ;  CONVERT KEY TO BINARY
                                   0061      89          ON_ERROR        BAD_KEY                      ;  BRANCH IF BAD
  00000274'EF           53     D0  0067      90          MOVL    R3,KEY                               ;  SET RECORD NUMBER
                                   006E      91          $RAB_STORE      RAB=INRAB,-                  ;  SPECIFY KEYED ACCESS
                                   006E      92                  RAC=KEY
                                   0079      93          $GET    RAB=INRAB                            ;  GET RECORD FOR PART
                                   0086      94          ON_ERROR        BAD_PART                     ;  BRANCH ON ERROR
  00000245'EF        41 8F     90  008C      95          MOVB    #^A/A/,DISCOUNT_TYPE                 ;  MODIFY DISCOUNT TYPE
                                   0094      96          $UPDATE RAB=INRAB                            ;  WRITE BACK MODIFIED RECORD
                                   00A1      97          ON_ERROR        EXIT                         ;  BRANCH ON ERROR
                                   00A7      98          TYPE    <RECORD CHANGED TO:>
           0000027C'EF         94  00C6      99          CLRB    MATCH_FLAG                           ;  SAY NO RELATED RECORD SEEN
  00000278'EF   000001A0'FF    D0  00CC     100          MOVL    @INRAB+RAB$L_RBF,MATCH_PART_NO       ;  SAVE PART NUMBER TO MATCH
                                   00D7     101 DISPLAY:
                                   00D7     102          $RAB_STORE      RAB=TYPE_RAB,-
                                   00D7     103                  RBF=@INRAB+RAB$L_RBF,-
                                   00D7     104                  RSZ=INRAB+RAB$W_RSZ
                                   00EE     105          $PUT    RAB=R0                               ;  PRINT RECORD
                                   00F7     106          ON_ERROR        EXIT                         ;  BRANCH ON ERROR
                                   00FD     107          $RAB_STORE      RAB=INRAB,RAC=SEQ            ;  SWITCH TO SEQUENTIAL ACCESS
                                   0108     108 GETSEQ:
                                   0108     109          $GET    RAB=R0                               ;  READ NEXT RECORD
           36           50     E9  0111     110          BLBC    R0,CHECK_DELETED                     ;  BRANCH ON ERROR
  00000278'EF   000001A0'FF    D1  0114     111          CMPL    @INRAB+RAB$L_RBF,MATCH_PART_NO       ;  DO FIRST 4 CHARACTERS
                                   011F     112                                                       ;  OF PART NUMBER MATCH?
                        29     12  011F     113          BNEQ    CHECK_DELETED
  0000027C'EF           01     E2  0121     114          BBSS    #1,MATCH_FLAG,DISPLAY                ;  BRANCH IF HEADER ALREADY PRINTED
```

```
                     AE          0128
                                 0129   115           TYPE    <RELATED RECORD(S):>
                     8D    11    0148   116           BRB     DISPLAY
                                 014A   117 CHECK_DELETED:
      00000000'8F    50    D1    014A   118           CMPL    R0,#RMS$_RNF                 ;  WAS ERROR RECORD FOUND?
                     21    12    0151   119           BNEQ    CHECK_RELATED                ;  BRANCH IF NOT
                                 0153   120           TYPE    <DELETED RECORD SKIPPED>
                     94    11    0172   121           BRB     GETSEQ                       ;  GO GET NEXT RECORD
                                 0174   122 CHECK_RELATED:
      0000027C'EF    01    E0    0174   123           BBS     #1,MATCH_FLAG,GETNEXT        ;  BRANCH IF RELATED RECORDS PRINTED
                     1F          017B
                                 017C   124           TYPE    <NO RELATED RECORDS.>
                                 019B   125 GETNEXT:
                   FE8D    31    019B   126           BRW     GET_REC_NO                   ;  LOOP
                                 019E   127 ;
                                 019E   128 ;  REPORT ERRORS
                                 019E   129 ;
                                 019E   130 BAD_KEY:          TYPE    <BAD KEY VALUE!>
                   FE6B    31    01BD   131           BRW     GET_REC_NO
                                 01C0   132 BAD_PART:         TYPE    <RECORD DOES NOT EXIST.>
                   FE49    31    01DF   133           BRW     GET_REC_NO
                                 01E2   134
                                 01E2   135 ;
                                 01E2   136 ;  ALL DONE  - CLOSE FILES AND EXIT
                                 01E2   137 ;
                                 01E2   138 DONE:   $CLOSE  FAB=INFAB
                                 01EF   139         $CLOSE  FAB=TYPE_FAB
                                 01FC   140         $CLOSE  FAB=PROMPT_FAB
                                 0209   141 EXIT:   $EXIT_S R0
                                 0212   142 ;++
                                 0212   143 ;
                                 0212   144 ;  SUBROUTINE TO CONVERT ASCII INPUT STRING TO BINARY
                                 0212   145 ;
                                 0212   146 ;  INPUTS:   R1, R2 = LENGTH AND ADDRESS OF INPUT STRING
                                 0212   147 ;
                                 0212   148 ;  OUTPUTS:  R0 - STATUS CODE
                                 0212   149 ;            R3 - BINARY VALUE
                                 0212   150 ;            R1, R2, R4 DESTROYED
                                 0212   151 ;--
                                 0212   152 CONVERT_KEY:
                     53    7C    0212   153           CLRQ    R3                           ;  INITIALIZE OUTPUT VALUE
                     13    11    0214   154           BRB     20$                          ;  GO CHECK IF ANY CHARACTERS
               53    0A    C4    0216   155 10$:      MULL2   #10, R3                      ;  SHIFT PARTIAL RESULT
                     16    1D    0219   156           BVS     30$                          ;  BRANCH ON OVERFLOW
               82    30    83    021B   157           SUBB3   #^A/0/, (R2)+, R4            ;  GET BINARY VALUE FOR CHARACTER
                     54          021E
                     10    19    021F   158           BLSS    30$                          ;  BRANCH IF BAD
               09    54    91    0221   159           CMPB    R4,#^A/9/-^A/0/              ;  CHARACTER > 9 ?
                     0B    1A    0224   160           BGTRU   30$                          ;  BRANCH IF BAD
               53    54    C0    0226   161           ADDL2   R4, R3                       ;  ADD IN CHARACTER TO PARTIAL RESUL
                     51    D7    0229   162 20$:      DECL    R1                           ;  ANY MORE INPUT?
                     E9    18    022B   163           BGEQ    10$                          ;  BRANCH IF MORE
               50    01    D0    022D   164           MOVL    #1, R0                       ;  SHOW SUCCESS
                           05    0230   165           RSB
                     50    D4    0231   166 30$:      CLRL    R0                           ;  SHOW FAILURE
                           05    0233   167           RSB
                                 0234   168 ;++
DISPLAY         - DISPLAY RELATED RECORDS             14-JUL-1978  12:53:13   VAX-11 MACRO X0.3-11           Page  4
                                                                                                                    (1)

                                  0234   ;
                                  0234   ;  SUBROUTINE TO INITIALIZE THE TYPE AND PROMPT FILES
                                  0234   ;
                                  0234   ;--
                                  0234   INIT_TYPE:
                                  0234           $CREATE FAB=TYPE_FAB
                                  0241           $OPEN   FAB=PROMPT_FAB
                                  024E           $CONNECT        RAB=TYPE_RAB
                                  025B           $CONNECT        RAB = PROMPT_RAB
                              05  0268           RSB
                                  0269
                                  0269           .END BEGIN
```

```
REORDER        - INDICATE ITEMS TO REORDER              12-DEC-1978 17:27:17    VAX-11 Macro V02.23          Page    1

                   0000           .TITLE  REORDER  -  INDICATE ITEMS TO REORDER
                   0000
                   0000     ;
                   0000     ;  PROGRAM TO READ THE OLD INVENTORY MASTER FILE AND CREATE A
                   0000     ;  NEW MASTER FILE, RECOGNIZING THOSE ITEMS WITH AN ON-HAND
                   0000     ;  QUANTITY LESS THAN THE REORDER QUANTITY, AND SETTING THE REORDER
                   0000     ;  DATE IN THE NEW MASTER FILE TO TODAY'S DATE, AND LISTING THE
                   0000     ;  RECORD ON SYS$OUTPUT.
                   0000     ;
                   0000     .MACRO  TYPE    STRING                              ;  MACRO TO TYPE "STRING"
                   0000             .SAVE                                       ;  SAVE CURRENT PSECT
                   0000             .PSECT  TYPE_STRINGS,NOWRT                  ;  CHANGE TO TYPE STRINGS PSECT
                   0000             ...TMPA=.                                   ;  NOTE ADDRESS
                   0000             .ASCII  \STRING\                            ;  STORE STRING
                   0000             ...TMPL=.-...TMPA                           ;  NOTE LENGTH
                   0000             .RESTORE                                    ;  BACK TO ORIGINAL PSECT
                   0000             MOVL    #...TMPA,TYPE_RAB+RAB$L_RBF         ;  SET STRING ADDRESS
                   0000             MOVW    #...TMPL,TYPE_RAB+RAB$W_RSZ         ;  SET STRING LENGTH
                   0000             $PUT    RAB=TYPE_RAB                        ;  WRITE THE RECORD
                   0000     .ENDM
                   0000     ;
                   0000     .MACRO  ON_ERROR        DEST,?L          ;  MACRO TO BRANCH ON ERROR
                   0000                     BLBS    R0,L                        ;  BRANCH ON SUCCESS
                   0000                     BRW     DEST                        ;  LONG FORM OF BRANCH
                   0000     L:
                   0000     .ENDM
                   0000     ;
  00000032         0000     REC_SIZE=50                                         ;  RECORD LENGTH
        00000000                    .PSECT  DATA,LONG
                   0000     TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT>,-          ;  FAB FOR USE WITH THE TYPE MACRO
                   0000                             RAT=CR
                   0050     TYPE_RAB:       $RAB    FAB=TYPE_FAB                ;  RAB FOR USE WITH TYPE MACRO
                   0094     ;
                   0094     INFAB:  $FAB    FNM=<INFILE:>
                   00E4     INRAB:  $RAB    FAB=INFAB,-
                   00E4                     UBF=REC_BUFFER,-
                   00E4                     USZ=REC_SIZE
                   0128     OUTFAB: $FAB    FNM=<OUTFILE:>,-
                   0128                     ORG=IDX,-
                   0128                     XAB=KEY0
                   0178     OUTRAB: $RAB    FAB=OUTFAB,-
                   0178                     RBF=REC_BUFFER,-
                   0178                     RSZ=REC_SIZE
                   01BC     ;
                   01BC     ;  XAB'S TO ORDER THE KEYS, PART#-PRIMARY, DISCOUNT TYPE-ALT. KEY#1,
                   01BC     ;   DESCRIPTION-ALT.KEY#2
                   01BC     ;
                   01BC     KEY0:   $XABKEY REF=0,-
                   01BC                     POS=0,-
                   01BC                     SIZ=5,-
                   01BC                     NXT=KEY1
                   01FC     KEY1:   $XABKEY REF=1,-
                   01FC                     POS=5,-
                   01FC                     SIZ=1,-
                   01FC                     FLG=<DUP,CHG>,-
                   01FC                     NXT=KEY2
                   023C     KEY2:   $XABKEY REF=2,-
```

```
                     023C    58                   POS=6,-
                     023C    59                   SIZ=20,-
                     023C    60                   FLG=<DUP,CHG>,-
                     023C    61                   NXT=0
                     027C    62 ;
                     027C    63 ;  DEFINE FIELDS OF RECORD
                     027C    64 ;
           00000005  027C    65 PART_NO_LEN=5
           00000014  027C    66 PART_DESC_LEN=20
           00000004  027C    67 QTY_LEN=4
           00000009  027C    68 DATE_LEN=9
           00000007  027C    69 PRICE_LEN=7
                     027C    70 ;
                     027C    71 REC_BUFFER:
           00000281  027C    72 PART_NUMBER:      .BLKB   PART_NO_LEN
           00000282  0281    73 DISCOUNT_TYPE:    .BLKB   1
           00000296  0282    74 PART_DESCRIPT:    .BLKB   PART_DESC_LEN
           0000029A  0296    75 QTY_ON_HAND:      .BLKB   QTY_LEN
           0000029E  029A    76 REORDER_QTY:      .BLKB   QTY_LEN
           000002A7  029E    77 REORDER_DATE:     .BLKB   DATE_LEN
           000002AE  02A7    78 LIST_PRICE:       .BLKB   PRICE_LEN
                     02AE    79 ;
                     02AE    80 ;  BUFFER TO FORMAT AND PRINT RECORD
                     02AE    81 ;
                 20  02AE    82 TYPE_BUF:         .ASCII  / /
           000002B4  02AF    83 TYPE_PART:        .BLKB   PART_NO_LEN
                 20  02B4    84                   .ASCII  / /
           000002C9  02B5    85 TYPE_DESC:        .BLKB   PART_DESC_LEN
        20 20 20 20  02C9    86                   .ASCII  /    /
           000002D1  02CD    87 ON_HAND:          .BLKB   QTY_LEN
        20 20 20 20  02D1    88                   .ASCII  /    /
           000002D9  02D5    89 REORDER:          .BLKB   QTY_LEN
           0000002B  02D9    90 TYPE_LEN=.-TYPE_BUF
                 00  02D9    91 HEADING:          .BYTE   0
                     02DA    92           .ALIGN  LONG
                     02DC    93 ;  BUFFER TO GET CURRENT DATE
           0000000B  02DC    94 DATE_BUF:         .LONG   11                         ;  LENGTH OF BUFFER
          000002E4'  02E0    95                   .LONG   TODAYS_DATE                ;  ADDRESS OF BUFFER
           000002EB  02E4    96 TODAYS_DATE:      .BLKB   7                          ;  DD-MON-
           000002ED  02EB    97 YR_CENTURY:       .BLKB   2                          ;  YY
           000002EF  02ED    98 YEAR:             .BLKB   2                          ;  YY
```

```
                                      00000000          .PSECT  CODE,NOWRT
                                          0000    ;
                                          0000    ;  INITIALIZATION  -  OPEN INPUT AND OUTPUT FILES, CONNECT STREAMS, AND
                                          0000    ;  GET TODAY'S DATE
                                          0000    ;
                                          0000
                                    0000  0000    START:  .WORD   0
                                          0002            $OPEN   FAB=INFAB                          ;  OPEN INPUT FILE
                                          000F            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          0015            $FAB_STORE      FAB=OUTFAB,-               ;  INITIALIZE OUTPUT FAB FROM INPUT
                                          0015                    RFM=FAB$B_RFM+INFAB,-              ;  SET RECORD FORMAT
                                          0015                    MRS=FAB$W_MRS+INFAB,-              ;  SET RECORD SIZE
                                          0015                    RAT=FAB$B_RAT+INFAB                ;  SET RECORD ATTRIBUTE
                                          0034            $CREATE FAB=OUTFAB                         ;  CREATE OUTPUT FILE
                                          0041            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          0047            $CONNECT        RAB=INRAB                  ;  CONNECT INPUT RAB
                                          0054            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          005A            $CONNECT        RAB=OUTRAB                 ;  CONNECT OUTPUT RAB
                                          0067            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          006D            $ASCTIM_S       TIMBUF=DATE_BUF            ;  GET CURRENT DATE
 000002E8'EF  000002ED'EF          B0     0080            MOVW    YEAR,YR_CENTURY                    ;  MAKE INTO YY FORMAT
                                          0088                                                       ;  (RATHER THAN "YYYY")
                                          0088            $OPEN   FAB=TYPE_FAB                       ;  OPEN REPORT FILE
                                          0098            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          009E            $CONNECT        RAB=TYPE_RAB               ;  CONNECT REPORT RAB
                                          00AB            ON_ERROR        EXIT                       ;  BRANCH ON ERROR
                                          00B1
                                          00B1            TYPE    <LIST OF INVENTORY ITEMS BELOW REORDER POINT>
                                          00D0            TYPE
                                          00EF
                                          00EF    ;
                                          00EF    ;  COPY RECORDS FROM OLD MASTER TO NEW MASTER CHECKING QUANTITY
                                          00EF    ;  ON HAND VERSUS REORDER QUANTITY
                                          00EF    ;
                                          00EF    READ:   $GET    RAB=INRAB                          ;  READ A RECORD
                                          00FC            ON_ERROR        DONE                       ;  BRANCH TO DONE, IF FINISHED
 00000296'EF            04         29     0102            CMPC3   #QTY_LEN,QTY_ON_HAND,REORDER_QTY;  ON-HAND LESS THAN REORDER QTY
              0000029A'EF                 0109
                        03         19     010E            BLSS    20$                                ;  BRANCH IF YES
                      009C         31     0110            BRW     WRITE                              ;  OMIT REORDER PROCESSING IF NOT
 000002E4'EF            09         28     0113    20$:    MOVC3   #DATE_LEN,TODAYS_DATE,REORDER_DATE
              0000029E'EF                 011A
                                          011F                                                       ;  SET REORDER DATE TO TODAY'S DATE
 000002D9'EF            01         E2     011F            BBSS    #1,HEADING,REPORT_ITEM             ;  BRANCH IF HEADING ALREADY PRINTED
                        3E                0126
                                          0127            TYPE    <PART #  PART DESCRIPTION   ON HAND REORDER PT.>
                                          0146            TYPE
                                          0165    REPORT_ITEM:                                       ;  BUILD REPORT RECORD
 0000027C'EF            05         28     0165            MOVC3   #PART_NO_LEN,PART_NUMBER,TYPE_PART
              000002AF'EF                 016C
 00000282'EF            14         28     0171            MOVC3   #PART_DESC_LEN,PART_DESCRIPT,TYPE_DESC
              00000285'EF                 0178
 000002CD'EF  00000296'EF          D0     017D            MOVL    QTY_ON_HAND,ON_HAND
 000002D5'EF  0000029A'EF          D0     0188            MOVL    REORDER_QTY,REORDER
                                          0193            $RAB_STORE      RAB=TYPE_RAB,-
                                          0193                    RBF=TYPE_BUF,-
                                          0193                    RSZ=#TYPE_LEN
```

```
REORDER         - INDICATE ITEMS TO REORDER               12-DEC-1978 17:27:17     VAX-11 Macro V02.23              Page   4

                                    01A6             $PUT     RAB=R0                         ; PRINT REPORT RECORD
                                    01AF      WRITE: $PUT     RAB=OUTRAB                     ; WRITE NEW MASTER RECORD
                                    01BC             ON_ERROR        EXIT                    ; BRANCH ON ERROR
                      FF2A    31    01C2             BRW      READ                           ; BRANCH ON SUCCESS
                                    01C5
                                    01C5    ;
                                    01C5    ;  ALL SET  -  CLOSE FILES AND EXIT
                                    01C5    ;
                                    01C5      DONE:  $CLOSE   FAB=INFAB
                                    01D2             $CLOSE   FAB=OUTFAB
   000002D9'EF        01      E0    01DF             BBS      #1,HEADING,CLOSE_TYPE          ; BRANCH IF HEADING PRINTED
                      1F            01E6
                                    01E7             TYPE     <NONE>                         ; INDICATE NO ITEMS REORDERED
                                    0206      CLOSE_TYPE:
                                    0206             $CLOSE   FAB=TYPE_FAB
                                    0213      EXIT:  $EXIT_S  R0
                                    021C
                                    021C             .END     START
```

```
ADDTOFILE       - ADD RECORDS TO FILE              11-DEC-1978 10:32:35    VAX-11 Macro V02.23          Page   1

    0000           .TITLE  ADDTOFILE  -  ADD RECORDS TO FILE
    0000     ;
    0000     ;  THIS PROGRAM ADDS NEW RECORDS TO AN INDEXED FILE, CREATING THE
    0000     ;  FILE INITIALLY, IF IT DOES NOT ALREADY EXIST.
    0000     ;
    0000     ;  IN ADDITION, THE UPDATE IF (UIF) OPTION IS USED ON THE $PUT MACRO.
    0000     ;  IN THIS EXAMPLE, THE PRIMARY KEY IS THE PART NUMBER.  WHEN A RECORD
    0000     ;  WITH A NEW PART NUMBER IS INSERTED, IT WILL SIMPLY BE PUT INTO THE
    0000     ;  FILE.  WHEN A RECORD WITH AN OLD PART NUMBER IS INSERTED, HOWEVER,
    0000     ;  IT WILL UPDATE THE EXISTING RECORD.
    0000     .MACRO  TYPE_STRING                                 ;  MACRO TO TYPE "STRING"
    0000
    0000             .SAVE                                       ;  SAVE CURRENT PSECT
    0000             .PSECT  TYPE_STRINGS,NOWRT                  ;  CHANGE TO TYPE STRING
    0000             ...TMPA=.                                   ;  NOTE ADDRESS
    0000             .ASCII  \STRING\                            ;  STORE STRING
    0000             ...TMPL=.-...TMPA                           ;  NOTE LENGTH
    0000             .RESTORE                                    ;  BACK TO ORIGINAL PSECT
    0000             MOVL    #...TMPA,TYPE_RAB+RAB$L_RBF         ;  SET STRING ADDRESS
    0000             MOVW    #...TMPL,TYPE_RAB+RAB$W_RSZ         ;  SET STRING LENGTH
    0000             $PUT    RAB=TYPE_RAB                        ;  WRITE THE RECORD
    0000     .ENDM
    0000     ;
    0000     .MACRO  PROMPT  STRING                              ;  MACRO TO ACCEPT INPUT
    0000                                                         ;  FROM SYS$INPUT, PROMPTING
    0000                                                         ;  WITH "STRING"
    0000             .SAVE                                       ;  SAVE CURRENT PSECT
    0000             .PSECT  TYPE_STRINGS,NOWRT                  ;  CHANGE TO TYPE STRINGS PSECT
    0000             ...TMPA=.                                   ;  NOTE ADDRESS
    0000             .BYTE   13,10                               ;  CARRIAGE RETURN,LINE FEED
    0000             .ASCII  \STRING\                            ;  STORE STRING
    0000             ...TMPL=.-...TMPA                           ;  NOTE LENGTH
    0000             .RESTORE                                    ;  BACK TO ORIGINAL PSECT
    0000             MOVL    #...TMPA,PROMPT_RAB+RAB$L_PBF       ;  SET PROMPT BUFFER ADDRESS
    0000             MOVB    #...TMPL,PROMPT_RAB+RAB$B_PSZ       ;  SET PROMPT BUFFER SIZE
    0000             $GET    RAB=PROMPT_RAB
    0000             MOVZWL  PROMPT_RAB+RAB$W_RSZ,R1
    0000             MOVL    PROMPT_RAB+RAB$L_RBF,R2
    0000     .ENDM
    0000     ;
    0000     .MACRO  ON_ERROR        DEST,?L                     ;  MACRO TO BRANCH ON ERROR
    0000             BLBS    R0,L                                ;  BRANCH ON SUCCESS
    0000             BRW     DEST                                ;  LONG FORM OF BRANCH
    0000     L:
    0000     .ENDM
    0000     ;
00000000             .PSECT  DATA,LONG
    0000     ;
    0000     ;  FABS AND RABS FOR USE WITH TYPE AND PROMPT MACROS
    0000     ;
    0000     TYPE_FAB:       $FAB    FNM=<SYS$OUTPUT:>,-
    0000                             RAT=CR
    0050     TYPE_RAB:       $RAB    FAB=TYPE_FAB
    0094     PROMPT_FAB:     $FAB    FNM=<SYS$INPUT:>
    00E4     PROMPT_RAB:     $RAB    FAB=PROMPT_FAB,-
    00E4                             UBF=PROMPT_BUFF,-
    00E4                             USZ=132,-
```

```
          00E4                               ROP=PMT
          0128    ;
          0128    ;  INPUT FILE FAB AND RAB AND XABS
          0128    ;
00000032  0128    REC_SIZE=50
          0128    INFAB:  $FAB       FNM=<INFILE:>,-
          0128                       ORG=IDX,-                     ;  FILE ORGANIZATION SPECIFIED
          0128                       RFM=VAR,-                     ;  POSSIBILITY IS PRESENT
          0128                       MRS=REC_SIZE,-                ;  THAT IT MAY NOT EXIST
          0128                       RAT=CR,-                      ;  AND THEREFORE MAY HAVE
          0128                       FAC=<PUT,UPD>,-               ;  TO BE CREATED
          0128                       XAB=KEY0,-
          0128                       FOP=CIF
          0178    ;
          0178    INRAB:  $RAB       FAB=INFAB,-
          0178                       RAC=KEY
          01BC    ;
          01BC    ;  DEFINE KEY XABS, ONE PRIMARY KEY AND TWO ALTERNATES
          01BC    ;
          01BC    KEY0:   $XABKEY REF=0,-
          01BC                       POS=0,-
          01BC                       SIZ=5,-
          01BC                       NXT=KEY1
          01FC    KEY1:   $XABKEY REF=1,-
          01FC                       POS=5,-
          01FC                       SIZ=1,-
          01FC                       FLG=<DUP,CHG>,-
          01FC                       NXT=KEY2
          023C    KEY2:   $XABKEY REF=2,-
          023C                       POS=6,-
          023C                       SIZ=20,-
          023C                       FLG=<DUP,CHG>,-
          023C                       NXT=0
          027C    ;
          027C    ;  DEFINE FIELDS OF RECORD
          027C    ;
00000005  027C    PART_NO_LEN=5
00000014  027C    PART_DESC_LEN=20
00000004  027C    QTY_LEN=4
00000009  027C    DATE_LEN=9
00000007  027C    PRICE_LEN=7
          027C    ;
          027C    REC_BUFFER:
00000281  027C    PART_NUMBER:       .BLKB   PART_NO_LEN
00000282  0281    DISCOUNT_TYPE:     .BLKB   1
00000296  0282    PART_DESCRIPT:     .BLKB   PART_DESC_LEN
0000029A  0296    QTY_ON_HAND:       .BLKB   QTY_LEN
0000029E  029A    REORDER_QTY:       .BLKB   QTY_LEN
000002A7  029E    REORDER_DATE:      .BLKB   DATE_LEN
000002AE  02A7    LIST_PRICE:        .BLKB   PRICE_LEN
          02AE    ;
          02AE            .ALIGN   LONG
00000334  02B0    PROMPT_BUFF:       .BLKB   132
```

```
                         0334   112 ;
                         0334   113 ;   PERFORM INITIALIZATION
                         0334   114 ;
                     00000000   115             .PSECT  CODE,NOWRT
                   0000  0000   116 BEGIN:      .WORD   0
                         0002   117             $CREATE FAB=INFAB                        ;  OPEN FILE IF IT EXISTS
                         000F   118                                                      ;  ELSE CREATE IT
                         000F   119             ON_ERROR        EXIT                     ;  BRANCH ON ERROR
                         0015   120             $CONNECT        RAB=INRAB                ;  CONNECT INPUT RAB
                         0022   121             ON_ERROR        EXIT                     ;  BRANCH ON ERROR
               01EF   30 0028   122             BSBW    INIT_TYPE                        ;  INITIALIZE TYPE AND PROMPT FILES
                         002B   123 ;
                         002B   124 ;   SOLICIT DATA FIELDS INPUT
                         002B   125 ;
                         002B   126 GETNXT:
                         002B   127             PROMPT  <PART #:>                        ;  GET NUMBER OF PART
                         0058   128             ON_ERROR        DONE                     ;  BRANCH IF DONE
               51     D5 005E   129             TSTL    R1                               ;  ANY INPUT?
               03     12 0060   130             BNEQ    10$                              ;  CONTINUE IF YES,
               0185   31 0062   131             BRW     DONE                             ;  ELSE QUIT
     62        51     2C 0065   132 10$:        MOVC5   R1,(R2),#^A/0/,-                 ;  MOVE PART NUMBER TO RECORD BUFFER
               30        0068
0000027C'EF    05        0069   133                     #PART_NO_LEN,PART_NUMBER         ;  ZERO FILLING
                         006F   134             PROMPT  <DISCOUNT TYPE:>                 ;  GET DISCOUNT TYPE
     62        51     2C 009C   135             MOVC5   R1,(R2),#^A/ /,-                 ;  MOVE DISCOUNT CODE TO RECORD BUFF
               20        009F
00000281'EF    01        00A0   136                     #1,DISCOUNT_TYPE                 ;  (BLANK IF NULL)
                         00A6   137             PROMPT  <PART DESCRIPTION:>              ;  GET PART DESCRIPTION
                         00D3   138             ON_ERROR        EXIT
     62        51     2C 00D9   139             MOVC5   R1,(R2),#^A/ /,-                 ;  MOVE PART DESCRIPTION TO RECORD
               20        00DC
00000282'EF    14        00DD   140                     #PART_DESC_LEN,PART_DESCRIPT     ;  BUFF, BLANK FILLING
                         00E3   141             PROMPT  <QUANTITY ON HAND:>              ;  GET NUMBER ON HAND
                         0110   142             ON_ERROR        EXIT
00000296'EF    30202020 8F   D0 0116   143      MOVL    #^A/   0/,QTY_ON_HAND            ;  INITIALIZE BUFFER AREA
     04        51     C3 0121   144             SUBL3   R1,#QTY_LEN,R3                   ;  DETERMINE OFFSET IN BUFFER AREA
               53        0124
               4A     19 0125   145             BLSS    EXIT1                            ;  IF FIELD TOO SMALL, EXIT
     62        51     28 0127   146             MOVC3   R1,(R2),QTY_ON_HAND(R3)          ;  PUT IN VALUE RIGHT ALIGNED
          0296'C3        012A
                         012D   147             PROMPT  <MINIMUM REORDER QUANTITY:>      ;  GET MINIMUM QUANTITY
                         015A   148             ON_ERROR        EXIT
0000029A'EF    30202020 8F   D0 0160   149      MOVL    #^A/   0/,REORDER_QTY            ;  INITIALIZE BUFFER AREA
     04        51     C3 0168   150             SUBL3   R1,#QTY_LEN,R3                   ;  DETERMINE OFFSET
               53        016E
               03     18 016F   151             BGEQ    CONT1                            ;  CONTINUE IF FIELD IS O.K.
               009D   31 0171   152 EXIT1:      BRW     EXIT                             ;  BRANCH LONG TO EXIT
                         0174   153 CONT1:
     62        51     28 0174   154             MOVC3   R1,(R2),REORDER_QTY(R3)          ;  FILL IN BUFFER AREA RIGHT ALIGNED
          029A'C3        0177
     6E        00     2C 017A   155             MOVC5   #0,(SP),#^A/ /,-                 ;  BLANK REORDER DATE
               20        017D
     63        09        017E   156                     #DATE_LEN,(R3)                   ;  (TAKE ADVANTAGE OF ITS
                         0180   157                                                      ;  ADDRESS IN R3)
                         0180   158             PROMPT  <LIST PRICE:>                    ;  GET PRICE
                         01AD   159             ON_ERROR        EXIT
     62        51     2C 01B3   160             MOVC5   R1,(R2),#^A/ /,-                 ;  MOVE PRICE TO RECORD BUFFER
```

```
                    20          01B6
   000002A7'EF      07          01B7                    #PRICE_LEN,LIST_PRICE          ; BLANK FILLING
                                01BD            $RAB_STORE      RAB=INRAB,-            ; SET UP RAB FOR NEW RECORD
                                01BD                    RBF=REC_BUFFER,-
                                01BD                    RSZ=#REC_SIZE,-
                                01BD                    ROP=UIF                        ;; IF PART # ALREADY EXISTS, UPDATE
                                01D4                                                   ; RECORD WITH NEW INFORMATION
                                01D4
                                01D4            $PUT    RAB=INRAB                      ; WRITE NEW RECORD
                                01E1            ON_ERROR        EXIT
                    FE41   31   01E7            BRW     GETNXT                         ; GET NEXT RECORD
                                01EA    ;
                                01EA    ;  ALL SET - CLOSE FILE AND EXIT
                                01EA    ;
                                01EA    DONE:   $CLOSE  FAB=INFAB
                                01F7            $CLOSE  FAB=TYPE_FAB
                                0204            $CLOSE  FAB=PROMPT_FAB
                                0211    EXIT:   $EXIT_S R0
                                021A
                                021A    ;++
                                021A    ;
                                021A    ;  SUBROUTINE TO INITIALIZE THE TYPE AND PROMPT FILES
                                021A    ;
                                021A    ;--
                                021A    INIT_TYPE:
                                021A            $CREATE FAB=TYPE_FAB
                                0227            $OPEN   FAB=PROMPT_FAB
                                0234            $CONNECT        RAB=TYPE_RAB
                                0241            $CONNECT        RAB=PROMPT_RAB
                           05   024E            RSB
                                024F            .END    BEGIN
```

APPENDIX B

USING THE RMS FILE ANALYZER

The RMS File Analyzer (RMSANLZ), which is not a DIGITAL-supported utility, enables you to inspect the file attributes and index structure of files. With the information provided, you can analyze characteristics of index files such as index tree depth and fill percentages. You can also analyze file corruption problems caused by user program errors and RMS system failures.

You can use RMSANLZ interactively or you can direct the output to a listing file. The following list summarizes the operations you can perform with RMSANLZ:

- Display file attributes, file header characteristics, and prolog information
- Display key description information for any key of an indexed file
- Display, for each index level of a key, the fill percentage, number of buckets, number of records, number of deleted records, number of record reference vectors (RRVs), and the number of deleted RRVs
- Print, for each bucket on each index level of the key, the virtual block number, the number of records and RRVs, and the record IDs of each record
- Display, for any bucket, the bucket control information, record control information, and key values
- Display any bucket in hexadecimal dump format
- Print detailed bucket contents of all buckets

B.1 USES OF RMSANLZ

RMSANLZ has two uses:

- To examine the characteristics of indexed files
- To provide information on file corruption errors caused either by application program errors or by RMS or VMS system failures.

When examining indexed files, RMSANLZ is useful for determining the effects of file activity, file loading, and file definition options. For example, if file size is used in loading an indexed file, RMSANLZ will display the actual fill percentage for further tuning in future file loads.

RMSANLZ can also be useful in determining the need for file reorganization by displaying the number of deleted records and deleted RRVs in the file. If a large fraction of the records is deleted, then file reorganization may be advisable.

Whenever file corruption errors occur and an RMS or VMS system failure is suspected, the complete RMSANLZ analysis of the file should be included with the Software Performance Report (SPR).

B.2 OPERATING RMSANLZ

The RMS File Analyzer (RMSANLZ) is executed by commands obtained from SYS$INPUT (terminal or procedure data). The output, by default, is sent to SYS$OUTPUT or directed to a listing file. You invoke RMSANLZ by typing:

```
$ RUN SYS$SYSTEM:RMSANLZ
```

Control is then passed to RMSANLZ, and RMSANLZ, in turn, displays the following prompt at your terminal:

```
Name of file to analyze:
```

You respond by typing the file specifications of the file to be analyzed.

RMSANLZ then prompts for the file specification to be used for output:

```
Specify output file, default is SYS$OUTPUT:
```

You respond with the listing file specification, or with <RET> to indicate SYS$OUTPUT.

RMSANLZ then displays the file attribute, file header, and file prolog information for the file. This information is in a format similar to a full directory listing, but is more extensive and includes information about file area allocations. An example is shown in Figure B-1.

```
_DBA0:[RMS.ANLZ]ISAM.IDX:1
Organization: Indexed with 2 defined keys
Record Format: Variable          Record Attributes: Carriage return
Maximum Record Size: 200 bytes
File Protection:   System:RWED  Owner:RWED  Group:RWE    World:R
File Owner: [011,122]            File ID: (7214,23,1)
Created: 24-JAN-1980 13:48:57.82
Revised: 24-JAN-1980 13:54:36.43  (3)
Expires: <none specified>
File Allocation: 72              Extension: 0
End-of-file VBN: 52              First_free_byte: 0
Allocation Attributes:
Prolog version: 1                Number of areas: 2

Area ID: 0      Area bucketsize: 3      Area extendsize: 21
  Alignment: CYL        Options:  Contiguous
  Current extent:       Start VBN:      Size: 51        Used: 21

Area ID: 1     Area bucketsize: 2       Area extendsize: 10
  Alignment: None       Options:
  Current extent:       Start VBN: 52   Size: 21        Used: 6
```

Figure B-1 Sample File Attribute Listing

If the file is an indexed file, RMSANLZ then prompts for the key of reference to be analyzed:

```
     Specify key of reference, default is all keys:
```

You respond with a key-of-reference number, or with <RET> to ask RMSANLZ to cycle through all the keys starting with the primary key.

RMSANLZ displays the key description as shown in Figure B-2 and then prompts for the analysis operation to perform for the key:

```
     Operation:
```

You respond with one of the following commands:

HELP or ? or help	- Print this command summary
A(NALYZE)	- Print summary of each index level including fill percentage, number of buckets, records RRVs, deleted records, and deleted RRVs
S(HOW)	- Print detailed bucket contents for specified buckets. The question "Next VBN:" asks for a VBN number until <RET> or EOF is entered
L(IST)	- Print detailed bucket contents for all buckets
D(UMP)	- Print VBNs in hexadecimal dump format for specified buckets. The question "Next VBN:" asks for the VBN number until <RET> or EOF is entered
E(XIT) or <RET>	- Exit from this key and go to command level

```
Key of Reference: 0                 Key Name: PART_NUM_ID
  Total Key Size: 10                Minimum record length: 44
  Number of Key Segments: 2         Key Data Type: String
  Key Attributes: Duplicates        No Changes
  Key Position:      16    42
  Key Size:           8     2
  Area numbers: Data:0  Index:1     Lowest index level:1
  Data Bucketsize:   1536           Data fill size:  1200
  Index Bucketsize:  1024           Index fill size: 600
  Index Depth: 1                    Root VBN: 52
```

Figure B-2 Sample Key Information Listing

During the ANALYZE operation, if you answer yes to the question:

```
     See VBN, #Records, #RRVs for each bucket?  Y/N
```

the VBNs, number of records, and number of RRVs per bucket will be printed in addition to the summary. If you answer yes to the question:

```
     Want to see record IDs for each bucket?  Y/N
```

the record IDs for each bucket for level 0 will be printed. The format of the ANALYZE operation output is shown in Figure B-3.

```
Level Number: 1
        Level 1 Fill Percentage: 6
        Number of buckets on this level: 1
        Number of records on this level: 4


Level Number: 0
  Bucket  VBN  Recs Del_recs RRVs Del_rrvs Fill%           Rec_IDs
     1     4    10      0      3       3       76
                                                   2   3    4   6   9  10  12  13
                                                  14  16    7   8   1   5  11  15
     2    10   11     0     0      2      82
                                                   1   3    4   5   6  8   9  10
                                                  11  12   13   2   7
     3    16   2      1     0      0      23
                                                   1   2    3
     4     7   5      1     7      1      48
                                                   6  1     2  12  15 14   9  11
                                                  10  4     7   8   5 13
     5    13   5      0     4      0      39
                                                  11  1     2   3  14 9   12
                                                  10
     6    19   9      0     0      0      67
                                                   1  2     3   4   5  6   7   8
                                                   9

        Level 0 Fill Percentages: 56
        Number of buckets on this level: 6
        Number of records on this level: 42
        Number of RRVs on this level: 2
        Number of deleted RRVs on this level: 6
```

Figure B-3 Sample Key Analysis Listing

The output format for the SHOW and LIST commands includes:

- Bucket control data including bucket type, index level, area number, and free space.
- For each record in an index bucket, the record pointer and key value.
- For each record in a primary data bucket, the record size and each key value.
- For each record in a secondary data bucket, the key value and all duplicate-record pointers.

If file corruption has occurred or an invalid value is entered to the SHOW command, RMSANLZ will display:

```
***** Invalid Bucket VBN: n *****
```

Using the DUMP command will allow you to examine the corrupted bucket.

If file corruption has occurred or an invalid value is entered to the SHOW command, RMSANLZ will display:

```
***** Invalid Bucket VBN: n *****
```

Using the DUMP command will allow you to examine the corrupted bucket.

INDEX

VAX-11
Record Management Services
User's Guide
AA-D781C-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 cut along this line.

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

— — — 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

BSSG PUBLICATIONS TW/A14
DIGITAL EQUIPMENT CORPORATION
1925 ANDOVER STREET
TEWKSBURY, MASSACHUSETTS 01876

— — Do Not Tear - Fold Here — — — — — — — — — — — — — — — — — — —