VAX–11 DSM User's Guide

AA–H800B–TE

December 1982

This document describes how to use DIGITAL Standard MUMPS operating under VAX/VMS (VAX–11 DSM). It also describes how to install the VAX–11 DSM system software, and how to operate and manage VAX–11 DSM in a multiuser environment.

This revised document supersedes the *VAX–11 DSM User's Guide* for version 1.0 of VAX–11 DSM.

OPERATING SYSTEM: VAX/VMS, V3

SOFTWARE: VAX–11 DSM, V2.0

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061
Telephone:(603)884–6660

Central Region

Digital Equipment Corporation
Accessories and Supplies Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone:(312)640–5612

Western Region

Digital Equipment Corporation
Accessories and Supplies Center
632 Caribbean Drive
Sunnyvale, California 94086
Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, October 1980
Revised, December 1982

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

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

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

The following are trademarks of Digital Equipment Corporation:

digital™

DEC	MASSBUS	UNIBUS
DECmate	PDP	VAX
DECsystem–10	P/OS	VMS
DECSYSTEM–20	Professional	VT
DECUS	Rainbow	Work Processor
DECwriter	RSTS	
DIBOL	RSX	

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

Contents

Chapter 2 Accessing the System

Chapter 3 Using VAX/VMS

Part 2: Using VAX–11 DSM

Chapter 4 VAX–11 DSM Command Line Options

Chapter 5 Developing and Maintaining Application Routines

Chapter 6 Input/Output Processing

Chapter 7 VAX–11 DSM Utilities

Chapter 8 Procedure Calling

Chapter 9 The VAX–11 DSM Data Base

Part 3: Operating VAX–11 DSM

Chapter 10 Installing VAX–11 DSM

Chapter 11 Managing VAX–11 DSM

Chapter 12 Running the VAX–11 DSM Job Controller

Chapter 13 Running the VAX–11 DSM Journal Process

Appendix A VAX–11 DSM Error Messages

Figures

Tables

Acknowledgement

DIGITAL Standard MUMPS is an extension of the ANSI Standard Specification (X11.1–1977) for the Massachusetts General Hospital Utility Multi–Programming System (MUMPS). MUMPS was originally developed at the Laboratory of Computer Science at Massachusetts General Hospital and was supported by grant HS00240 from the National Center for Health Services Research and Development.

This manual reflects all of the enhancements approved by the MUMPS Development Committee up to July 1982.

Preface

MANUAL OBJECTIVES

This manual describes how to develop and maintain VAX–11 DSM application routines using features of the VAX–11 DSM system and VAX/VMS. It also describes how to install the VAX–11 DSM system software, and how to operate and manage VAX–11 DSM in a multiuser environment. This manual does not describe the language elements or syntax of the VAX–11 DSM language; such material is described in the *VAX–11 DSM Language Reference Manual.*

INTENDED AUDIENCE

This manual is intended for programmers with a working knowledge of the VAX–11 DSM language and either limited or extensive knowledge of VAX/VMS. This manual is also intended for the VAX–11 DSM System Manager and/or operator who has the responsibility for installing the VAX–11 DSM system software, establishing VAX–11 DSM user accounts, and performing other privileged or "sensitive" system operations.

DOCUMENT PREREQUISITES

The *VAX/VMS Summary Description and Glossary* introduces the basic concepts of the VAX/VMS operating system. All VAX–11 DSM users should be familiar with the information in the Summary Description before reading this document.

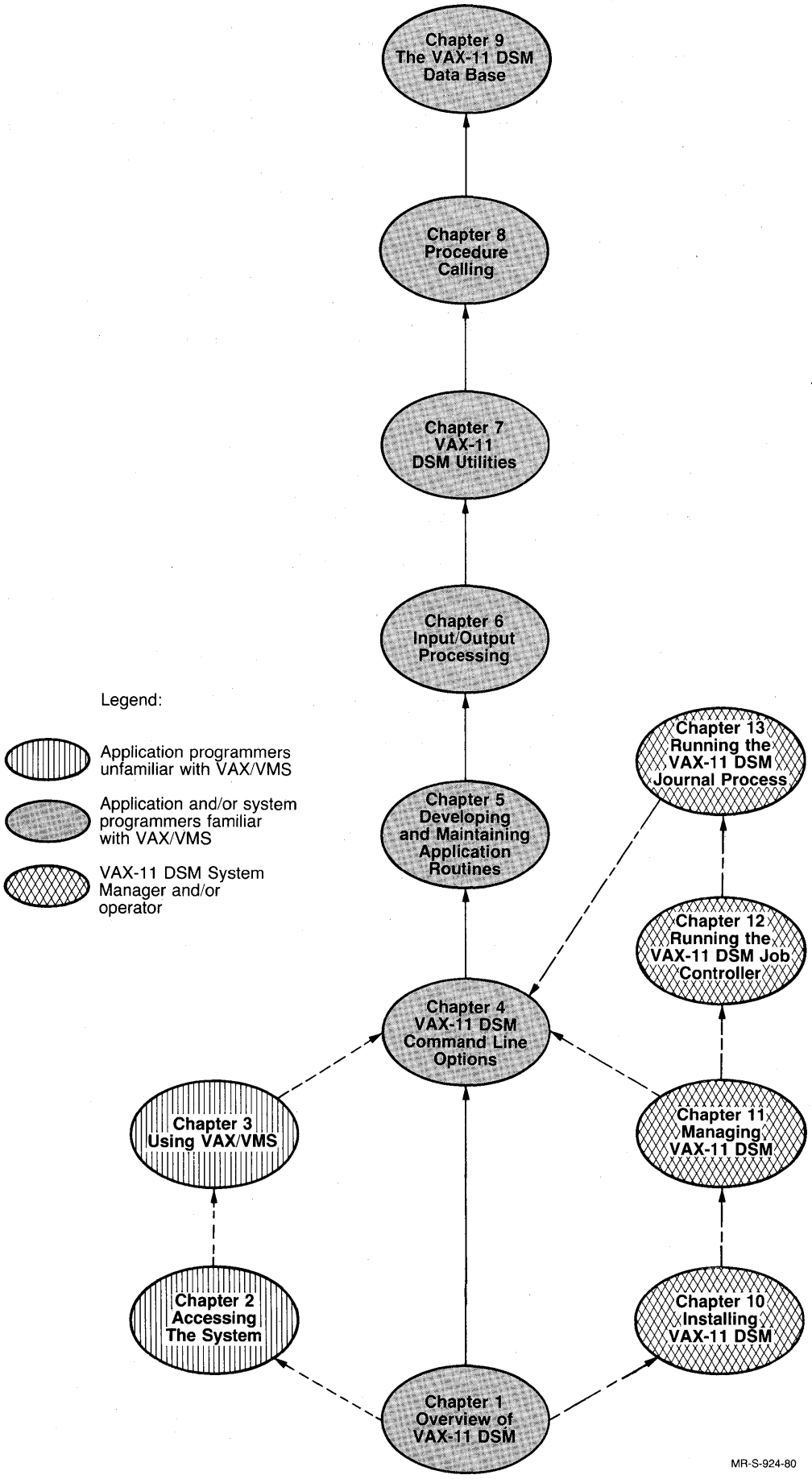
The *VAX/VMS System Management and Operations Guide* provides information for people who have the overall responsibility for controlling the operations of a VAX/VMS installation. People who will be installing the VAX–11 DSM system software and establishing VAX–11 DSM user accounts should be familiar with the information in the *VAX/VMS System Management and Operations Guide* before reading this document.

HOW TO USE THIS DOCUMENT

The *VAX–11 DSM User's Guide* is divided into four parts, each of which is intended for a VAX–11 DSM user with a different background or job. The following chart summarizes the contents of each part and indicates who should read them. Figure P–1 illustrates the order in which this document should be read by various VAX–11 DSM users.

Part	Intended Audience	Contents
I		
Chapter 1	All VAX–11 DSM users	• Overview of VAX–11 DSM
Chapters 2–3	Application programmers unfamiliar with VAX/VMS	• How to access VAX/VMS from the terminal and begin programming in the VAX–11 DSM language • Information about VAX/VMS required for using VAX–11 DSM.
II		
Chapters 4–9	Application and system programmers familiar with VAX/VMS; System Manager and/or Operator	• Detailed reference information about the VAX–11 DSM system and its interaction with VAX/VMS
III		
Chapters 10–13	VAX–11 DSM System Manager/Operator	• VAX–11 DSM software installation procedure • Establishing VAX–11 DSM user accounts • Running the VAX–11 DSM Job Controller and Journal Process
Appendixes		
Appendixes A–C	Application and system programmers; System Manager and/or operator	• Error messages • DSM Editor • Data structures

Figure P–1: How to Read the VAX–11 DSM User's Guide



ASSOCIATED DOCUMENTS

This manual refers to the following documents that contain supplemental information relevant for VAX–11 DSM programming and system operations:

- *VAX–11 DSM Language Reference Manual*
- *Introduction to DSM*
- *VAX/VMS Summary Description and Glossary*
- *VAX/VMS Command Language User's Guide*
- *VAX–11 Run–Time Library Reference Manual*
- *VAX/VMS System Services Reference Manual*
- *VAX/VMS System Management and Operations Guide*
- *VAX–11 Utilities Reference Manual*
- *VAX/VMS System Messages and Recovery Procedures Manual*
- *VAX/VMS I/O User's Guide*
- *Introduction to VAX–11 Record Management Services*
- *VAX–11 Record Management Services Tuning Guide*
- *VAX–11 Record Management Services Reference Manual*
- *VAX–11 Record Management Services Utilities Manual*
- *VAX–11 Symbolic Debugger Reference Manual*
- *VAX–11 Architecture Handbook*

CONVENTIONS USED IN THIS DOCUMENT

This manual uses the following documentation conventions and symbols:

Convention	Meaning
CTRL/x	Indicates that you press the CTRL key on the terminal keyboard while simultaneously pressing some other key (represented here by x).
RET	Indicates that you press the carriage return key on the terminal keyboard.
SP	Indicates a space must separate components of a command or command line.
[ ]	Square brackets indicate the enclosed item is optional. (Square brackets are not, however, optional in the syntax of a directory name in a VAX/VMS file specification.)
...	A horizontal ellipsis indicates that additional command parameters can be added to the command line.
⋮	A vertical ellipsis indicates a break between two illustrated lines of user input and that all user input is not shown.
file-spec	Indicates a VAX/VMS file specification.
Contrasting Colors	Black — Indicates system output. Red — Indicates user input.
Italics	Indicates a new term, defined in the sentences that follow.

Part 1
Introduction

Chapter 1
Overview of VAX–11 DSM

This chapter introduces the basic features of the VAX–11 DSM data management system.

1.1 System Overview

VAX–11 DSM stands for DIGITAL Standard MUMPS for VAX–11 processors. VAX–11 DSM is a multiuser data management system that runs under the VAX/VMS operating system. Its basic features include:

- An extended version of the ANSI Standard MUMPS language
- A language precompiler
- A subset of the I/O options of VAX/VMS
- Support of the VAX–11 Procedure Calling Standard
- Code and data sharing through VAX/VMS global sections
- A journaling facility for monitoring and recording changes to the DSM data base
- A lock manager and centralized control capability
- Utilities for system maintenance, status information, routine backup, and journaling control

1.1.1 The DSM Language

The DSM language is a high-level *interpretive* language. Interpretive processing of the language means that each line of a DSM routine is translated into machine-readable form and executed immediately. You do not have to compile or link routines written in an interpretive language. Object modules and execution files are not generated.

Because the DSM language is interpretive, you can write, debug, edit, and run a routine in one interactive session. As you enter source code (that is, lines of a routine written in the DSM language) from the terminal, the interpreter examines and analyzes each DSM statement and executes the specified operation. It performs error checking during routine execution and reports most errors at the terminal. This reduces problem-solving time, the computer time required to check the routine, and, most importantly, the time required to obtain a final running application.

The DSM language has many capabilities. However, its basic orientation is procedural. The language is directed primarily toward the processing of variable-length string data, making interactive data base systems easier to implement and maintain.

1.1.2 Data Management

The DSM language allows you to reference data symbolically through variables. A variable represents either a numeric value or an alphanumeric string of variable length.

VAX–11 DSM utilizes two types of data: local data and global data, and thus two types of variables: local variables and global variables.

Local data is defined solely for the routine or routines stored (as ASCII character strings) in your source routine buffer area. This type of data is not intended for permanent storage, but only for the duration of the current process. VAX–11 DSM assigns a fixed amount of space to store such data in an internal data structure called the local symbol table. You can allot additional space in the symbol table to store local data as required.

Global data is stored on disk and is referenced through variables called global variables, or simply globals. Global variable names are similar to local variable names, but must begin with a circumflex (^). Subscripted global variables form a system of arrays stored on disk. The data in these arrays forms a common data base that can be made available simultaneously to one or more processes.

Global arrays are mapped dynamically into the disk structures that store them. The process is handled entirely by VAX–11 DSM and is transparent to the user.

1.1.3 The Precompiler

VAX–11 DSM provides a language precompiler to optimize routine execution in an application environment. The precompiler interacts with the interpreter to produce code in precompiled format. Precompiled format is a more efficient form of source code that is faster and simpler to interpret.

Each time you terminate a DSM routine line with a carriage return, the precompiler transforms the source code into precompiled format by:

- Stripping comments.
- Checking syntax.
- Setting up an internal table for line labels, which optimizes GOTO statements and DO statements that transfer control to other routine lines.
- Evaluating numeric constants and transforming them into an internal representation (that is, packed decimal or longword).
- Converting arithmetic expressions into Reverse Polish Notation.
- Restricting the evaluation of a series of postconditionals to the occurrence of the first false condition. To do this, the precompiler generates code that specifies the appropriate offset to a given instruction.

When you terminate a command line in direct mode, VAX–11 DSM precompiles the line and executes it immediately.

When you execute a routine (with the DO command, for example), the interpreter executes the precompiled code and reports syntax errors.

When you store a routine on disk, the system places both the source and precompiled versions of the routine in your DSM routine directory (see Section 5.2 for details about VAX–11 DSM directories). Thus, for a given version of a routine, the precompilation procedure occurs only once. When you execute a routine from your directory, VAX–11 DSM loads the precompiled version of the routine and executes it.

You can always load, edit, and test DSM routines interactively, however, because the system saves both forms of your routines. If you edit a routine, VAX–11 DSM repeats the precompilation procedure. Therefore, when you save an updated routine, an updated version of the precompiled routine replaces the previous version.

1.1.4 I/O Options

VAX–11 DSM provides a subset of the input/output (I/O) options of VAX/VMS. VAX–11 DSM supports a number of I/O device types, as described in the following paragraphs. Each device type can be accessed through commands in the DSM language. You can submit an I/O request to any supported device that is available for use.

VAX–11 DSM responds to all I/O requests by providing an interface to the appropriate VAX/VMS I/O handler. Terminal I/O and interprocess communication through mailboxes are handled by the VAX/VMS Queue I/O Request system service ($QIO); $QIO provides the interface between a process and the device I/O drivers.

VAX–11 DSM supports a subset of the file-handling capabilities of VAX–11 Record Management Services (RMS). VAX–11 RMS is the file and record access subsystem of the VAX/VMS operating system. VAX–11 DSM uses RMS primarily to process file I/O requests to disk and magnetic tape. However, VAX–11 DSM can access any device to which VAX–11 RMS provides the interface (such as card readers or line printers), provided no device-specific options are required.

All disk I/O requests are to Files–11 disk volumes. (Files–11 is the name of the directoried disk volume structure used by the VAX/VMS operating system.)

1.1.5 Procedure Calls

Procedures are service routines that are not part of the DSM language or the VAX–11 DSM system. You can access procedures through a DIGITAL–implemented extension to Standard MUMPS called the $ZCALL function. Through $ZCALL, you can call VAX/VMS system services, routines in the VAX–11 Common Run–Time Library, or routines written in languages other than DSM. With $ZCALL, you can create and define functions that are not supplied with VAX–11 DSM and call them directly from your DSM application routines.

1.1.6 Shared Areas of Memory

VAX–11 DSM supports a high degree of code and data sharing through the use of VAX/VMS virtual memory sections. Mapping a DSM application in a virtual memory section improves its performance because the system does not have to perform I/O to access DSM routines from disk storage. Instead, it can execute the routines directly from virtual memory.

Virtual memory sections can be either private or shared. If shared, they are called *global sections*. Global sections can be created dynamically by a process, or can be permanently present in the system. Permanent global sections are generally created from routines to which a number of users require access. When a group of routines or an application is installed in a global section, all users of that application share the same copy of the code. At run time, a copy of this code is mapped into the virtual address space of each requesting process.

All users can create private virtual memory sections or map to an existing global section. However, you must have sufficient VAX/VMS privileges to create and install a global section.

1.1.7 Journaling

Journaling is a means of keeping a record on secondary storage (disk or magnetic tape) of transactions that alter the data base, that is, global variable SETs and KILLs. In VAX–11 DSM, journaling is handled by a VAX/VMS detached process called a *Journal Process*.

VAX–11 DSM provides a number of journaling options to meet the needs of a system running multiple applications. Depending on the options you select, there can be one or more Journal Processes. You can run one Journal Process for each group in the system (where "group" is determined by the group number of a process's UIC) or for the entire system.

Each Journal Process monitors data base transactions through *mailboxes*, which are VAX/VMS pseudodevices used for interprocess communication. When a DSM user process performs a SET or KILL on a global variable, the Journal Process reads the transaction from a mailbox and makes a record of it in one of many possible journal files. If the data base is corrupted, you can use these files to restore it.

1.1.8 The DSM Job Controller

VAX–11 DSM uses another detached process to manage interlocks and provide centralized control for DSM applications. This process is called the *DSM Job Controller*. Although VAX/VMS has its own Job Controller to coordinate requests by VMS processes, VAX–11 DSM uses the DSM Job Controller to provide special services for DSM user processes. Specifically, the DSM Job Controller provides the following services:

- It enables and disables DSM application mode start-up. (See Section 1.2 for an overview of DSM's programmer and application modes.)
- It manages interlock requests (through the DSM LOCK command) by multiple DSM user processes.
- It enables and disables journaling on a group-by-group basis.

As with journaling, communication between a VAX–11 DSM user process and the DSM Job Controller takes place through mailboxes.

VAX–11 DSM lets you either use or bypass the DSM Job Controller at DSM start-up time. For work that does not affect the common data base (for example, application development), you can bypass the Job Controller. However, when you run a DSM application, journaling and interlocking common variables between DSM user processes become necessary, so you must use the Job Controller.

1.1.9 System and Library Utilities

The VAX–11 DSM software package includes a number of utility routines. These *utilities* are written in the DSM language and are provided:

- To help DSM application programmers develop and maintain the software and data for their applications.
- To help the DSM system manager to control the operation of the system running DSM applications.

The utilities are divided into two categories: library utilities and system utilities. Library utilities perform general services in five categories:

- Callable functions which provide date and time and numerical conversion
- Routines for use in debugging DSM routines
- Routines to manipulate globals
- Routines to provide information about your VAX–11 DSM system
- Routines to manipulate DSM routines

System utilities perform services in the following areas:

- Routines to control the use of the DSM Journal Process
- Routines to control the files used by the DSM Journal Process
- A routine that implements auto-login

Generally, you access the system and library utilities through a menu-driven utility package. Most utilities in the package are interactive, that is, they prompt for required user input. In addition, most utilities provide extensive on-line documentation that explains how to use them.

VAX–11 DSM also allows you to call specific utilities directly. Utilities and utility menus are described in Chapter 7.

1.2 The DSM Image

The DSM *image* is the software that allows you to program in the DSM language, execute DSM routines, perform input/output operations, and access and manipulate the data base. When you request the execution of this image, VAX/VMS activates it in the virtual address space of your process. You make this request by issuing the DSM command, described in Section 2.3.

The DSM image includes a set of software components shared among all users running DSM. These components constantly interact with each other. For the purpose of an overview, the following sections describe each component as a separate entity.

1.2.1 The Language Interpreter

The language interpreter implements and controls routines written in the DSM language. It examines and analyzes each language statement, performs precompilation, and executes the specified operation. It also performs error checking and trapping, coordinates the reporting of errors at the terminal, and handles procedure calls.

The interpreter has two operating modes: Programmer Mode and Application Mode. In Programmer Mode, you can create, modify, debug and store routines, and execute commands and routines. In Application Mode, you execute routines.

1.2.2 The I/O Interface

The I/O Interface initiates all input/output activity by coordinating an I/O request with the appropriate VAX/VMS I/O handler, either the $QIO system service or VAX–11 RMS. The I/O interface handles all requests involving the following DSM commands and special variables:

OPEN	CLOSE	USE
READ	WRITE	ZPRINT
ZWRITE	ZUSE	argumentless ZLOAD
$X	$Y	$IO
$ZIO	$ZA	$ZB
$ZCONTROLC		

1.2.3 The Data Base Supervisor

The Data Base Supervisor coordinates the dynamic mapping of global arrays into the disk structures that store them. The Data Base Supervisor interacts with VAX–11 RMS, which performs the physical and logical allocation of records into RMS indexed files. (VAX–11 DSM bases its implementation of globals on indexed files. Chapter 9 describes this relationship in detail.) The Data Base Supervisor handles all requests involving the following DSM commands and functions (when used with global variables):

SET	KILL	$DATA
$NEXT	$ORDER	$ZNEXT
$ZORDER	$ZSORT	

1.2.4 The Routine Handler

The Routine Handler implements all requests to load routines from disk and save routines to disk. This module maps routines and global sections and handles requests involving the following DSM commands:

DO	ZSAVE	$ZDIRECTORY
ZLOAD with arguments		

1.2.5 User Data Structures

In addition to the five components described above, which are shared among all users if the DSM image is installed as a known image, each process running a DSM image has a private set of data structures in its virtual address space. These include:

- The DSM local symbol table
- Buffers to store DSM routine lines. These include the source routine buffer and the precompiled routine buffer.
- The DSM call stack
- Other miscellaneous DSM data structures, such as control structures for open DSM devices and for open globals
- VMS image and process structures

Chapter 2
Accessing the System

This chapter provides the information you need to operate the computer terminal, access VAX/VMS, and run the DSM image.

2.1 Using the Computer Terminal

All terminals have a keyboard configuration that resembles a conventional typewriter. The keyboard contains alphabetic and numeric keys on which you type the commands you want the computer to process.

Terminal keyboards also contain a number of keys that allow you to signal the computer to perform special operations. These keys are called function keys and control characters. The following sections describe the behavior of these keys.

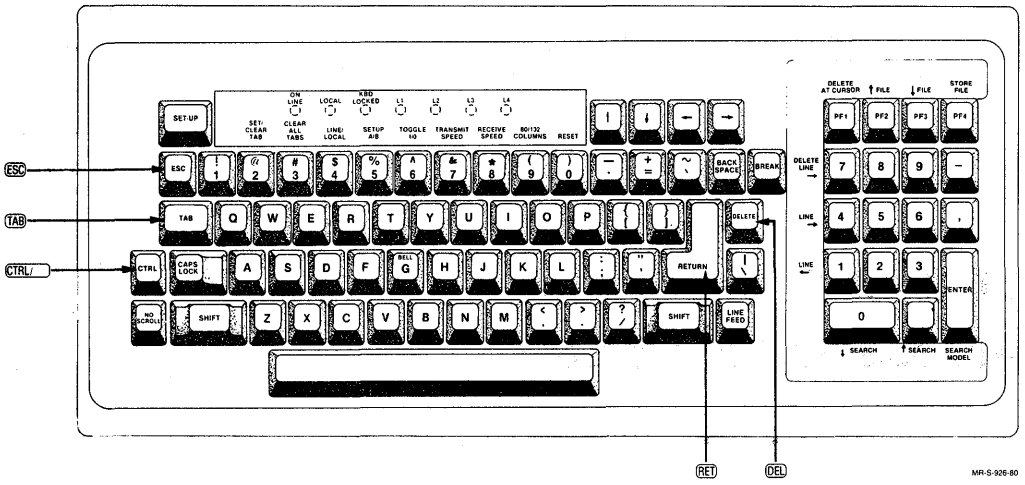
2.1.1 Function Keys

Most terminals have keys that allow you to send special signals to the computer. These are called *function keys.*

Figure 2–1 shows the keyboards of two different DIGITAL terminals: the VT100 and the VT52. Arrows point to the most commonly used function keys. The symbols used in these diagrams are used throughout this text as a shorthand notation to indicate that these keys should be pressed.

Note that function keys are not always in the same position on different types of terminals, so check the keyboard layout each time you use a terminal for the first time.

Figure 2–1: The VT100 and VT52 Keyboards



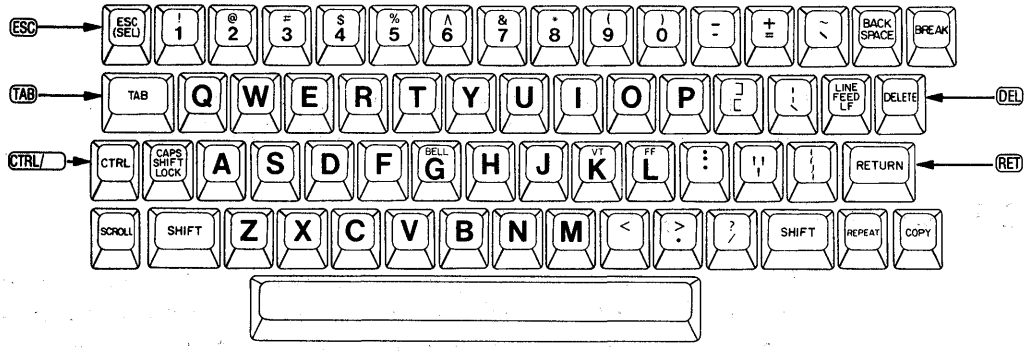


Table 2–1 describes the important function keys that appear on your terminal.

Table 2–1: Function Keys

Function Key	Description
CTRL/	Part of many 2–key combinations that perform a variety of functions, as described in Section 2.1.2. The control characters used most often are described in Table 2–2.
DEL	Deletes the last character entered on a video terminal and backspaces over it. On a hard-copy (printing) terminal, DEL instructs the system to ignore the last character typed.
ESC	Terminates lines typed at the terminal, if escape sequence processing is not enabled. See Section 6.5.4 for more information about escape sequence processing.
RET	Carriage return; transmits the current line to the system for processing and advances the cursor to the leftmost position of the next line.
TAB	Moves the printing position on the terminal to the next tab stop.

Terminal keyboards do not always represent function characters in the same way. The following are some common variations:

- The ESCAPE key may be labeled ALT MODE or ESC (SEL).
- The circumflex (^) may be represented as an up-arrow.
- The underscore (_) may be represented as a back-arrow.
- The DELETE key may be labeled RUBOUT or DEL.
- The RETURN key may be labeled CR for carriage return.

If you do not have a DIGITAL terminal, consult your terminal's user guide for information about the representation of function characters.

2.1.2 Control Characters

A *control character* is a character sequence entered by pressing a letter key and the CTRL key simultaneously. Control characters that you enter are sometimes displayed on the terminal by a circumflex (^) followed by the selected letter. Other control characters, such as CTRL/S, are not displayed.

Table 2–2 summarizes the functions of the most commonly used control characters.

Table 2–2: Control Characters

Control Character	Function
CTRL/C	Generally cancels processing of DSM commands (unless CTRL/C recognition is disabled). See Sections 4.3 and 6.5.3 for more information about CTRL/C recognition. Initiates the VAX/VMS log-in sequence.
CTRL/I	Duplicates the function of the TAB key.
CTRL/L	Form feed.
CTRL/O	Suppresses display of terminal output the first time it is pressed, and continues output to the terminal the second time it is pressed.
CTRL/Q	Restarts terminal output that was suspended by CTRL/S.
CTRL/R	Retypes the current line (including the prompt, if any) and leaves the cursor positioned at the end of the line.
CTRL/S	Suspends terminal output until CTRL/Q is pressed.
CTRL/U	Discards the current input line and retypes the prompt, if any.
CTRL/Y	Interrupts command or routine execution and returns control to the VAX/VMS command language interpreter (unless CTRL/Y recognition is disabled). See Sections 4.3 and 6.5.3 for more information about CTRL/Y recognition. Initiates the VAX/VMS log-in sequence.

2.2 Logging into VAX/VMS

The procedure used to access VAX/VMS from the terminal is called *log-in.* The log-in procedure identifies you to the operating system and distinguishes you from other system users.

You must have an account on VAX/VMS before you can log in. Accounts are set up by the system manager, or whoever authorizes the use of the system at your installation.

After you have an account, you log in by providing the operating system with two forms of identification:

- A User Name
- A Password

The following sections describe each form of identification and show an example of the VAX/VMS log-in procedure.

2.2.1 User Names

A user name is an alphabetic character string that identifies you to the VAX/VMS operating system. It is the name that the system manager gives to your account. A user name is often the user's first or last name; however, it can be any continuous character string.

2.2.2 Passwords

A password is an additional code that identifies you to the operating system. It can be any continuous character string, and is generally selected by the individual user.

VAX/VMS does not display your password when you enter it at the terminal. This is for your protection. If you maintain the secrecy of your password, other users cannot access the system with your user name.

For security reasons, DIGITAL recommends that your password should be at least six characters long.

2.2.3 Sample Login

You let the system know you want to log in by pressing one of the following keys:

- ESC
- RET
- CTRL/C
- CTRL/Y

After you press one of these keys, VAX/VMS responds by prompting for your user name. Type your user name followed by RET. VAX/VMS then prompts for your password. Enter your password and press RET.

If you enter your user name or password incorrectly, or wait too long to enter either form of identification, VAX/VMS displays an error message. If this occurs, you must repeat the log-in procedure.

The following example shows the complete log-in procedure:

```
RET
Username: BLOCKRET
Password:          RET

Welcome to VAX/VMS Version 3.0

$
```

When the dollar sign ($) appears at the far left of the terminal, it indicates that login was successful and that you can begin your terminal session. The dollar sign is the VAX/VMS command language interpreter (CLI) prompt.

2.2.4 Auto–Login

A terminal can be enabled for *auto-login* to ensure that only specified routines and data can be accessed from the terminal. A terminal on which auto-login is enabled is called a "tied" terminal.

Auto-log-in is a VAX/VMS log-in option that forces the automatic and protected execution of a VAX/VMS command procedure. Such a command procedure can be used to start up a VMS image, such as VAX–11 DSM. (VAX–11 DSM is the only VAX/VMS layered product that provides utilities for the use of auto-login.)

To log in to VAX/VMS from a terminal set up for auto-login, you simply type any of the keys listed in Section 2.2.3. See Sections 7.8 and 11.1.3.3 for more information about auto-login.

2.3 Running a DSM Image

To run the DSM image, follow these steps:

1. Log in to VAX/VMS.
2. Issue the DSM command at the CLI prompt ($).

For example:

— Log in to VAX/VMS —

```
$ DSMRET
                        VAX-11 DSM Version 2.0

>
```

After you enter the DSM command, VAX–11 DSM prints a sign-on message and displays the DSM interpreter prompt, the right angle-bracket (>). When this character appears at the far left of the terminal, it indicates that the interpreter is ready to accept commands to build, edit, or run DSM routines. You are now in Programmer Mode.

To get help in using the DSM language, type ?<RET> after the DSM prompt. DSM responds by running the ^%HELP utility, which provides interactive help.

Programmer Mode is one of two user modes available to VAX–11 DSM users. The other user mode, called Application Mode, is used for executing routines only. Chapter 4 describes the characteristics of each user mode in detail.

2.4 Switching between DSM and VAX/VMS

If you start up DSM in Programmer Mode, there are two ways to exit to VAX/VMS command level (so that the CLI prompt is displayed, rather than the DSM interpreter prompt). If you issue either of the following commands, control is transferred from the DSM interpreter to the CLI:

- CTRL/Y
- HALT

Issuing CTRL/Y interrupts the DSM image and transfers control to the CLI. Because this command does not destroy the current DSM context, you can issue the DIGITAL Command Language (DCL) CONTINUE command to resume DSM (if you do not invoke another VAX/VMS image). Table 2–3 lists the DCL commands that do not invoke an image; you can issue any of these and then return to your DSM image with CONTINUE.

Table 2–3: VAX/VMS Commands That Do Not Invoke an Image

=	EOD	SET VERIFY
ALLOCATE	EXAMINE	SHOW DAYTIME
ASSIGN	GOTO	SHOW DEFAULT
ATTACH	IF	SHOW QUOTA
CLOSE	INQUIRE	SHOW PROTECTION
CONTINUE	ON	SHOW STATUS
DEALLOCATE	OPEN	SHOW SYMBOL
DEASSIGN	READ	SHOW TIME
DEBUG	SET CONTROL	SHOW TRANSLATION
DECK	SET DEFAULT	SPAWN
DEFINE	SET ON	WAIT
DELETE/SYMBOL	SET PROTECTION/DEFAULT	WRITE
DEPOSIT	SET UIC	

HALT runs down the DSM image. During DSM image rundown, all resources allocated to VAX–11 DSM are returned to VAX/VMS. All open files are closed; any devices allocated from DSM are deallocated, and the DSM context is lost. Once you have issued this command, you must reissue the DSM command to restore the DSM image.

In Application Mode, both CTRL/Y and HALT transfer control from the DSM interpreter to the CLI, with the same side effects as their execution in Programmer Mode. A DSM application can also use the QUIT command to transfer control from DSM to the CLI, unless the current routine was called from another DSM routine.

See Section 4.3 for more information about the behavior of these commands in each user mode.

2.5 Logging Out of VAX/VMS

The log-out procedure deletes your process and ends a terminal session. You must be at VAX/VMS command level to log out.

To log out, issue the DCL LOGOUT command in response to the CLI prompt, as shown in the following example:

```
$ LOGOUT RET
```

After you issue the LOGOUT command, VAX/VMS prints the following message:

```
BLOCK logged out at 23-JULY-1980 12:43:10.38
```

If you issue the LOGOUT command with qualifiers, it causes VAX/VMS to print a summary of accounting statistics for your terminal session. (Section 3.1 describes using qualifiers with DCL commands.) See the *VAX/VMS Command Language User's Guide* for a description of specific LOGOUT command qualifiers.

The LOGOUT command causes any sub-processes you have created during this session to be terminated.

NOTE

If you shut your terminal off while you are logged in, VAX/VMS does not log you out. However, if you log in on a remote terminal (a terminal that communicates with the processor over telephone lines) or over a computer network, and you break the connection prior to issuing the LOGOUT command, VAX/VMS does log you out.

Chapter 3
Using VAX/VMS

This chapter describes the features of the VAX/VMS operating system you must know to use VAX–11 DSM.

3.1 Using the DIGITAL Command Language

To use VAX–11 DSM effectively, you need to use the DIGITAL Command Language (DCL). DCL commands help you manage and maintain your account. Of particular significance to VAX–11 DSM users are commands that help you:

- Manipulate files and directories
- Create logical names
- Set default characteristics of your process
- Mount and initialize magnetic tapes
- Determine system status
- Create command procedures (such as a log-in command file)

Throughout this manual, references are made to specific DCL commands and their effects. Thus, you must understand the general format of the DCL command string, described in the following paragraphs. For a detailed description of DCL and the commands discussed in subsequent parts of this document, see the *VAX/VMS Command Language User's Guide.*

The general format of a DCL command string is:

$[Label:]Command Name[/Qualifier(s)]Ⓢ[Parameter 1]...[Parameter n]

Elements in brackets are optional.

Dollar Sign ($) The dollar sign is the command language interpreter (CLI) prompt. This character must appear on the far left of the terminal before you can enter any DCL command. In a command procedure, you must place a dollar sign before every DCL command.

Label All DCL command strings can be labeled. A label precedes the command name and is separated from it by a colon (:). Generally, you use labels in command procedures when you want the ability to transfer control to other language statements by means of such DCL commands as GOTO. See the *VAX/VMS Command Language User's Guide* for a complete description of command procedures.

Command Name The command name is an English-language verb that describes the action the command performs. You can truncate all command names to four characters. Many command names can be truncated to fewer characters.

Qualifiers Qualifiers modify the action of a command or parameter. You often do not need to specify a qualifier, because its default value provides the most commonly used function of the command.

Qualifiers always begin with a slash (/).

Parameters Parameters describe the object of a command. In some cases, a parameter is a keyword. In others, it is the name of a routine to execute or the name of a file or device to manipulate

At least one space or tab must separate the first parameter from the command name.

If you need information about a DCL and its qualifiers and parameters, issue the DCL HELP command in response to the DCL prompt. The parameter of the HELP command is the name of the command about which you want information.

3.2 File Specifications

A *file* is a collection of logically related data stored on a medium such as a disk or magnetic tape. All information that VAX/VMS reads or writes on behalf of its users is defined in terms of files.

To access files that already exist, or to give names to files that you create, you must know how the operating system identifies files. VAX/VMS identifies files by a *file specification* in the following format:

node::device:[directory]file name.file type;version number

All punctuation marks (colons, brackets, period, semicolon) are required to separate the components of the file specification. The elements of a file specification are:

Node	Identifies a node on a computer network. A node name is a one- to six-character alphanumeric string. This part of the file specification only applies to systems that support VAX–11 DECnet.
Device	Identifies the device on which a file is stored or is to be written, as described in Section 3.2.2.
Directory	Identifies the name of the directory in which the file is catalogued. You can delimit directory names with either square brackets, as shown, or with angle brackets (< >). Directories are described in Section 3.3.
File name	Identifies a file; a file name can be up to nine alphanumeric characters long.
File type	Describes the kind of data in the file; the file type can be up to three alphanumeric characters long.
Version Number	A decimal number from 1 to 32767 that specifies the version of the file. Version numbers are incremented by 1 each time a new version of a file is created. You can use either a semicolon (;) or a period (.) to separate a file type from the version number.

3.2.1 Defaults in File Specifications

It is not necessary to explicitly state all elements of a file specification each time you wish to perform an operation on an input file, or whenever you create an output file. When a field of a file specification is omitted, VAX/VMS supplies a default value. Table 3–1 summarizes the default value for each field.

Table 3–1: File Specification Defaults

Field	Default
Node	Local network node
Device	Device established as your default by (1) the entry in your user authorization file, or (2) executing the DCL SET DEFAULT command.
Directory	Directory established as your default by (1) the entry in your user authorization file, or (2) executing the DCL SET DEFAULT or SET UIC command.
File Name	DSM routine directories appear in VAX/VMS directories as ROUTINES.DSM. See Section 5.6 for more information about DSM directories and directory specifications. Other languages have no default file names.
File Type	Depends on the context in which a file specification is used for input or created for output. Table 3–2 lists the default file types that are significant for VAX–11 DSM users.
Version Number	For input files: Highest existing version. For output files: Version number 1 for new files; highest existing version plus 1 for old files.

Table 3–2 lists the default file types that are significant for VAX–11 DSM users.

Table 3–2: Default File Types

File Type	File Contents
COM	VAX/VMS command procedure
DAT	Input or output data file
DIR	VAX/VMS directory file
DSM	DSM routine directory file
EXE	Executable image file
GBL	DSM global file
JRN	DSM journaling file
LOG	DSM journal log file or error log file; also a VAX/VMS batch job output file
OBJ	Object file (created by a language compiler)
VIR	DSM mapped routine file

3.2.2 Device Names

Each physical device known to the system is uniquely identified by a device name specification. VAX/VMS device names consist of a device mnemonic, controller designation, and unit number, in the following format:

devcu

dev A two- or three-character mnemonic for the device type, as listed in Table 3–3.

c A controller designation made up of one alphabetic character. For example, MTA designates magnetic tape controller A.

u The device unit number, a decimal number between 0 and 65535.

Thus, a system can have two tape units on different controllers, one designated MTA0 and the other designated MTB0. Each drive is unit 0 on its controller.

Table 3–3 lists the VAX/VMS device types and their mnemonics.

Table 3–3: VAX/VMS Device Types

Mnemonic	Device Type
CR	Card Reader
CS	Console Storage Device
DB	RP04, RP05, RP06 Disk
DD	TU58 Cassette Tape
DL	RL02 Disk
DM	RK06 and RK07 Disk
DQ	RB02 and RB80 Disk
DR	RM03, RM05, RM80, RP07 Disk
DX	RX01 Floppy Diskette
DY	RX02 Floppy Diskette
LP	Line Printer
MB	Mailbox
MF	Magnetic Tape
MS	TS11, TS04 Magnetic Tape
MT	TE16, TM03, TU45, TU77 Magnetic Tape
NET	Network Communications Device
NL	Null Device
OP	Operator's Console
RT	Remote Terminal
TT	Interactive Terminal
XJ	DUP11 Synchronous Communications Line
XM	DMC11 Synchronous Communications Line

3.3 VAX/VMS Directories and Subdirectories

A directory is a structure that organizes files on a disk device. The files belonging to one user on a VAX/VMS system all reside in directories belonging to that user. The user can sort the files into different directories as desired.

Directories are represented in file specifications by *directory names*.

3.3.1 Directory Names

Directory names can be represented in either of two ways:

[directory name]

[g,m]

A directory represented by [directory name] contains a one- to nine-character alphanumeric string within the brackets, as in the following example:

[LEWIS]

The directory representation [g,m] generally (but not necessarily) corresponds to the UIC of the owner of the directory. The following example shows a directory name in UIC format:

[122,26]

Section 3.4 provides more information about UICs.

In addition to your main directory, represented by [directory name], VAX/VMS allows you to organize your files into *subdirectories*, as explained in the next section. Subdirectory names have the following format:

[name.name.name...]

An example of a subdirectory name is [BWV431.JSB]. You can include up to eight levels (separated by seven periods) in a subdirectory name.

3.3.2 Creating Subdirectories

Each VAX/VMS user has at least one directory, a main directory, created by the system manager when the account is established. Usually that directory's name is [username].

Optionally, you can create, in your main directory, one or more subdirectories. The DCL CREATE command allows you to create subdirectories. For example, the following command creates a subdirectory to a main directory called [OWNER]:

```
$ CREATE/DIRECTORY [OWNER.VIDEO]
```

This command places an entry for the directory file VIDEO.DIR in the main directory [OWNER]. Subsequently, you can use the subdirectory [OWNER.VIDEO] in a file specification.

A subdirectory can contain a directory file for another subdirectory; that subdirectory can contain a directory file for another subdirectory, and so on. The maximum number of directory levels, including the main directory, is eight.

3.3.3 The Default Directory

Whenever you are using VAX/VMS, you are associated with a *default directory*. When you first log in, your default directory is generally your main directory.

You change the default directory for the current session by issuing the DCL SET DEFAULT command. The parameter of this command is the directory name that you wish to use as a default. Note that this command does not change what your default directory will be the next time you log in.

To see what your current default directory is, issue the DCL SHOW DEFAULT command.

3.3.4 Examining Directory Contents

You examine the contents of a directory by issuing the DCL DIRECTORY command, as shown in the following example:

```
$ DIRECTORY RET
```

This command shows you the contents of the current default directory.

To examine the contents of a directory that is not your default directory, issue the DIRECTORY command with the name of the desired directory (or subdirectory) as the parameter.

To examine the contents of a subdirectory of the current default directory, issue the DIRECTORY command followed by the subdirectory name, for example:

```
$ DIRECTORY [.VIDEO]
```

Note that in this case you need not explicitly state the default directory name to examine the contents of your subdirectories.

3.4 Maintaining File Protection

VAX/VMS bases its file protection mechanism on the User Identification Code (UIC). The system manager assigns each account in the system a UIC. The UIC consists of two identification numbers that specify a group and member number, as in the following example:

[226,60]

VAX/VMS defines four user categories based on the UIC, as follows:

SYSTEM	All users who have low group numbers, usually 1 through 10 (octal). However, the exact range of system group numbers is determined by the system manager at VAX/VMS sysgen time. These group numbers are generally for system managers, system programmers, and operators. Users who have SYSPRIV, LOG_IO, and PHY_IO privileges are also included in this category.
OWNER	The UIC of the person who created and therefore owns the file.
GROUP	All users who have the same group number in their UICs as the owner of the file, including the owner.
WORLD	All users who do not fall into any of the other three categories. This category also includes the owner of the file and all users in the owner's group.

Each user category can be permitted or denied the following types of file access:

READ	The right to examine, print, or copy a file.
WRITE	The right to modify a file, including deleting records from it.
EXECUTE	The right to execute a file that contains an executable image.
DELETE	The right to delete a file.

3.4.1 Setting File Protection

The DCL SET PROTECTION command establishes the protection that applies to a file or group of files. It also establishes the default protection for files created during the current terminal session. You will often need to use this command to control access to your files and directories by other system users.

3.4.2 Examining File Protection

The SHOW PROTECTION command displays the default protection for files created during the current terminal session. You do not use this command to examine the protection of a particular file. To look at the protection of one or more particular files, issue the DIRECTORY command with the /PROTECTION qualifier.

3.5 Using Logical Names

Using logical names, you can write routines that are independent of physical file specifications. Logical names also provide a shorthand way to specify files that you refer to frequently.

When VAX/VMS expects a file specification in a command or routine but encounters a logical name instead, it translates the logical name to its equivalence name.

You assign logical names with the DCL ASSIGN command. This command equates a logical name with an equivalence name, which can be a device, a complete file specification, or another logical name, for example:

```
$ ASSIGN DBC0:[JONES]DEMOG.DAT;1 DEMOG
```

This command creates the process logical name DEMOG and associates it with the file specification DBC0:[JONES]DEMOG.DAT;1. This file specification will be used whenever VAX/VMS encounters the logical name DEMOG in a command or routine during execution.

Since the ASSIGN command allows you to equate a logical name with another logical name, logical name translation is a recursive procedure. After VAX/VMS finds and translates a logical name, it uses the equivalence name as the argument for another logical name translation. VAX/VMS continues in this manner until it cannot translate the equivalence name.

When VAX/VMS cannot translate an equivalence name, it applies system defaults to fill in any part of a file specification that remains unspecified. For example, if an equivalence name does not specify a device, VAX/VMS adds your default device to the file specification. If an equivalence name does not specify a directory, VAX/VMS adds your current default directory to the file specification.

3.5.1 Concealed Devices

To avoid having to change references to device names when you run DSM routines on a new device, you can use *concealed device names* to refer to devices. A concealed device name is a logical name that represents a device, but that is not translated to the physical device name unless you explicitly ask for a translation with the SHOW LOGICAL command. Using concealed device names allows you and VAX/VMS to refer to devices consistently by a logical name.

You create a concealed device name by preceding the physical device name (or file specification) with a double underscore (_) in the ASSIGN or DEFINE command string for the logical name. The following command defines the logical name TAPE2 as the physical device name MTA2:

```
$ DEFINE TAPE2 _MTA2:
```

For more information on defining and using concealed device names, see the *VAX/VMS Command Language User's Guide.*

3.5.2 Logical Name Tables

When you assign a logical name to a file specification, the logical name and its translation are placed in one of three logical name tables. The table in which the logical name is placed depends on whether you want the logical name to be local to the current process, available to the group, or available to the entire system. VAX/VMS maintains a logical name table for each category, as follows:

- Process Logical Name Table

  Contains logical names that are local to the process. When an entry is placed in this table, it is available to all images that run in the process until it is deassigned, or until the process is deleted (for example, as a result of logout).

- Group Logical Name Table

  Contains logical names that are available to all processes that have the same group number in their UICs. Entries remain in this table until explicitly deleted. User privilege is required to place and delete entries in the group logical name table.

- System Logical Name Table

  Contains logical names that are available to all processes in the system. Entries remain in this table until they are explicitly deleted. User privilege is required to place and delete entries in the system logical name table.

To translate a logical name to its equivalence name, VAX/VMS searches the process, group, and system logical name tables, in that order, and uses the first match it finds. Thus, entries in the process logical name table take precedence over those in the group and system logical name tables, and entries in the group logical name table take precedence over entries in the system logical name table.

3.5.3 Processor Modes

When you use the ASSIGN (or DEFINE) command to create process logical names, you can use command qualifiers to assign them in either of the following processor modes:

- Supervisor mode
- User mode

Supervisor mode and user mode are two of the four VAX/VMS processor access modes. By default, the ASSIGN command enters logical names in the process logical name table in supervisor mode.

User mode process logical names take precedence over supervisor mode process logical names. However, user mode process logical names are deleted from the process logical name table after the image completes its execution.

3.5.4 Process–Permanent Logical Names

VAX/VMS defines a set of logical names for every process created during login. These logical names, called process-permanent logical names, remain defined for the life of a process. Table 3–4 lists some process-permanent logical names. You can check the process-permanent logical names for your process by entering the following DCL command:

```
SHOW LOGICAL/PROCESS
```

Table 3–4: Process–Permanent Logical Names

Logical Name	Equivalence Name
SYS$INPUT	Default input stream for the process. For an interactive user, SYS$INPUT equates with your terminal. For a batch job, SYS$INPUT equates with the batch input stream.
SYS$OUTPUT	Default output stream for the process. For an interactive user, SYS$OUTPUT equates with your terminal. For a batch job, SYS$OUTPUT equates with the batch job log file.
SYS$COMMAND	Original SYS$INPUT for a job. When a process executes an indirect command file, SYS$INPUT is assigned to that file, and SYS$COMMAND remains assigned to the original command stream. The name TT is equivalent to SYS$COMMAND.
SYS$ERROR	Default output stream to which the system writes messages. For an interactive user, SYS$ERROR equates with the terminal. For a batch job, SYS$ERROR equates with the batch job log file.
SYS$DISK	Default device established at login, or by the SET DEFAULT command.
SYS$LOGIN	Default device and directory specification at login.
SYS$SCRATCH	Default work disk and directory.
TT	Current terminal.

The equivalence names for SYS$INPUT, SYS$OUTPUT, SYS$COMMAND, and SYS$ERROR define files that remain open for the life of the process. These files can be read or written from routines.

3.5.5 System–Permanent Logical Names

VAX/VMS defines a number of logical names for the system logical name table. Entries in this table are available to all processes in the system. Among these logical names are:

SYS$HELP	Device and directory of the system help files
SYS$LIBRARY	Device and directory of the system libraries
SYS$MANAGER	Device and directory containing (among other files) SYSTARTUP.COM
SYS$MESSAGE	Device and directory of system error message files
SYS$SHARE	Device and directory of system shareable images
SYS$SYSTEM	Device and directory of operating system programs and procedures
SYS$UPDATE	Device and directory used in installation

For a complete list of system-permanent logical names, see the *VAX/VMS System Management and Operations Guide*, or type the following DCL command:

```
SHOW LOGICAL/SYSTEM SYS$*
```

Part 2
Using VAX–11 DSM

Chapter 4
VAX–11 DSM Command Line Options

This chapter describes the full syntax of the DSM command, including command qualifiers, and discusses VAX–11 DSM user modes.

4.1 Introduction

To gain access to the DSM language, it is sufficient to issue the DSM command in its simplest form, as described in Chapter 2. However, the DSM command also accepts a series of qualifiers and a command parameter that establish initial conditions for the execution of the DSM image.

DSM command qualifiers instruct VAX–11 DSM to perform a number of tasks. Specifically, they allow you to:

- Enable and disable routine interrupts by CTRL/C
- Enable and disable the setting of breakpoints for use in debugging
- Change your principal input and output device
- Change your default directory for DSM application and library routines, and for globals
- Change the default size of your source and precompiled routine buffers, your terminal's output buffer, the local symbol table, and the call stack
- Optimize global accesses
- Enable and disable the various aspects of DSM file sharing
- Map routines in virtual memory sections

The DSM command parameter allows you to load and execute a DSM routine (or application) at DSM image start-up. The presence or absence of the command parameter also determines your DSM user mode. VAX–11 DSM can be operated in either of two user modes:

- Programmer Mode
- Application Mode

Each user mode provides an operating environment with a different set of default characteristics, as described in Section 4.3.

4.2 DSM Command Syntax

The full syntax of the DSM command line is:

$ DSM[/Qualifier(s)[/Subqualifier]]ⓈⓅ[Parameter]

Where:

- Command elements in brackets are optional
- All punctuation (slash, space) is required to separate the components of the command line

Sections 4.2.1 and 4.2.2 describe the syntax of the DSM command line in detail. Section 4.4 describes specific DSM command qualifiers.

4.2.1 DSM Command Qualifier Syntax

DSM command qualifiers have five forms:

Form	Example
/ql	/SHARED
/ql/sql	/MAPPED/LIBRARY
/ql=n	/SOURCE_BUFFER_SIZE=10100
/ql[=func]	/CENABLE=BREAK
/ql[=fs][/sql]	/MAPPED=file specification/LIBRARY

Where:

- Elements in brackets are optional
- ql is a qualifier name
- sql is a subqualifier that further defines the action specified by the qualifier
- n is a numeric argument

- func is a functional argument
- fs is any part of a VAX/VMS file specification

Some qualifiers have a negative form that indicates the opposite effect of the action specified by the qualifier. Negative qualifiers have the form:

/NOql

You can truncate all DSM command qualifiers to the shortest unambiguous string. In all cases, this is four or fewer characters.

4.2.2 DSM Command Parameter Syntax

The DSM command parameter is the name of a DSM routine that you want to load from one of your directories and execute as soon as you terminate the DSM command line. The command parameter has the form:

[Label[Offset]]^Routine_Name

Parameter elements in brackets are optional. The elements are:

Label	A DSM routine line label that identifies a line in the specified routine.
+Offset	A positive integer offset from Label that indicates the routine line where you want to start execution of the specified routine; the plus sign [+] is required if you include an offset.
^Routine_Name	The name of the routine or application you want to execute; the circumflex (^) is required whether you specify a label or not.

At least one space must separate the command parameter from the DSM command.

The DSM command parameter can also be specified following DSM command qualifiers, as shown in the command line:

```
$ DSM/CENABLE LAB+2^DATA
```

This command line causes the routine DATA to execute starting from line LAB+2 with CTRL/C recognition enabled. See Sections 4.3.1 and 4.3.2 for more information about CTRL/C recognition.

4.3 VAX–11 DSM User Modes

Depending on the presence or absence of the DSM command parameter, you can operate VAX–11 DSM in either of two user modes, Programmer Mode or Application Mode.

The difference between Programmer Mode and Application Mode operation is the default behavior of VAX–11 DSM with respect to the following:

- CTRL/C
- Recognition of the BREAK command
- The HALT and QUIT commands
- File sharing
- Use of the DSM Job Controller
- Use of the DSM Journal Process
- Error processing
- The ZESCAPE command
- The mapping of virtual memory sections

The following sections describe the default characteristics of each user mode.

4.3.1 Programmer Mode

To operate VAX–11 DSM in Programmer Mode, issue the DSM command (at the CLI prompt) without a command parameter, as shown in Chapter 2.

When you operate VAX–11 DSM in Programmer Mode, you interact with the DSM interpreter directly. The interpreter prompt at the far left of the terminal indicates that the interpreter is ready to accept commands to create, modify, load, store, delete, and execute routines.

The following defaults apply to VAX–11 DSM when you operate it in Programmer Mode:

- CTRL/C and CTRL/Y

  CTRL/C recognition is enabled unless you append the /NOCENABLE qualifier to the DSM command at start-up (refer to Section 4.4 for details about this qualifier). Typing CTRL/C while a DSM routine is running interrupts execution and causes the interpreter to display the DSM interpreter prompt. The DSM context is preserved, and you can continue to enter DSM commands.

CTRL/Y recognition is generally enabled. (However, the VAX/VMS system manager can explicitly disable CTRL/Y recognition on a user-by-user basis. See Section 6.5.3 for more information about CTRL/Y recognition.) If CTRL/Y recognition is enabled and you type CTRL/Y, control returns to the CLI.

Invoking a VAX/VMS image after typing CTRL/Y causes DSM image rundown. If you do not invoke a VAX/VMS image, you can type the DCL CONTINUE command to return the DSM image. Refer to Table 2–3 for a complete list of the DCL commands that do not invoke a VAX/VMS image.

- HALT and QUIT

  Execution of the HALT command causes DSM image rundown and transfers control from the DSM interpreter to the CLI. The DSM HALT message is displayed.

  If a routine executes a QUIT command at the outermost nesting level, execution of the routine stops. However, the DSM context is preserved. The DSM interpreter displays the interpreter prompt, and you can continue to enter DSM commands.

- File sharing

  File sharing is a VAX–11 RMS option that allows many users to access the same files with read, write, and delete privileges. For VAX–11 DSM users, file sharing applies primarily to globals and routines. However, library globals and routines, like other globals accessed read-only, do not have to be explicitly shared. RMS allows many users to read files simultaneously. This is called implicit sharing.

  In Programmer Mode, file sharing is disabled unless you append the /SHARED qualifier to the DSM command at start-up. See Section 4.4 for a description of the /SHARED qualifier.

- The DSM Job Controller

  The primary function of the DSM Job Controller is to handle interlock requests by multiple DSM user processes, that is, requests by programmers or applications involving the DSM LOCK and ZALLOCATE commands. (Refer to the *VAX–11 DSM Language Reference Manual* for details about these commands.)

  In Programmer Mode, the DSM Job Controller does not process lock requests; if you issue LOCK or ZALLOCATE commands, VAX–11 DSM ignores them.

  See Chapter 12 for more information about the DSM Job Controller.

- Journaling

  VAX–11 DSM journaling is never enabled in Programmer Mode.

- Error processing

  If an error related to language syntax, routines, I/O, or the DSM data base occurs, the DSM context is preserved; control returns to the DSM interpreter.

  Generally, users running the DSM image cannot cause a fatal error (that is, an error that causes DSM image rundown) in Programmer Mode. However, DSM command line errors are fatal to the DSM image in Programmer Mode.

  See Section 5.7 for a general description of VAX–11 DSM error processing and Appendix A for a description of specific DSM errors.

- ZESCAPE

  When VAX–11 DSM executes ZESCAPE in Programmer Mode, it generates an error. Execution of the routine stops, and control returns to the DSM interpreter.

- BREAK

  When a BREAK command is encountered, a breakpoint is generated, unless the /NOBREAK qualifier was specified with the DSM command.

- Mapped virtual memory sections

  In programmer mode, mapped virtual memory sections are mapped as private sections rather than as global (shared) sections.

4.3.2 Application Mode

To operate VAX–11 DSM in Application Mode, issue the DSM command (at the CLI prompt) with a parameter, for example:

```
$ DSM "DEMOG<RET>
```

DEMOG is the name of a routine or application that you want to load from your directory and execute at start-up.

The following defaults apply to VAX–11 DSM when you operate it in Application Mode:

- <CTRL/C> and <CTRL/Y>

  <CTRL/C> recognition is disabled unless you append the /CENABLE qualifier to the DSM command at start-up.

  <CTRL/Y> recognition is generally enabled in Application Mode. (However, it can and in many cases should be disabled at VAX/VMS command level by the system manager.) Typing <CTRL/Y> transfers control from the DSM interpreter to the CLI. If you invoke a VAX/VMS image after typing <CTRL/Y>, you cause DSM image rundown. However, if you do not invoke a VAX/VMS image, you can type the CONTINUE command to return the DSM image. See the Table 2–3 for a list of DCL commands which do not invoke a VAX/VMS image.

- HALT and QUIT

  Executing the HALT command in Application Mode causes DSM image rundown and transfers control from the DSM interpreter to the CLI. No exit message is displayed, however.

  Executing the QUIT command at the outermost nesting level also causes DSM image rundown and transfers control from the DSM interpreter to the CLI.

- File sharing

  File sharing is enabled, unless you append the /NOSHARED qualifier to the DSM command at start-up.

- The DSM Job Controller

  The DSM Job Controller processes all interlock requests involving the LOCK and ZALLOCATE commands unless you append the /NOSHARED qualifier to the DSM command at start-up. The DSM Job Controller can also enable journaling for applications users, and permit or disable start-up in Application Mode.

- Error processing

  A DSM user process can cause a fatal error in Application Mode.

  If an error related to language syntax, routines, I/O, or the DSM data base occurs (and error trapping is not enabled), the DSM image is run down; control returns to the CLI. If CTRL/C recognition is enabled, typing CTRL/C also causes DSM image rundown and returns control to the CLI.

- ZESCAPE

  When VAX–11 DSM executes a ZESCAPE command in an application, it exits from the application and transfers control to the DSM interpreter in Programmer Mode. After the DSM interpreter prompt appears, you can enter any valid DSM command or examine any accessible routine.

- BREAK

  BREAK commands are ignored in Application Mode, since applications users should not be involved in debugging routines.

- Virtual memory sections

  Virtual memory sections are mapped as global (shared) sections for applications users.

4.4 Description of DSM Command Qualifiers

Table 4–1 summarizes the functions of the DSM command qualifiers. Table 4–2 shows the defaults for each qualifier. Sections 4.4.1 through the end of the chapter describe specific qualifiers in detail. These sections are arranged in alphabetical order by qualifier name.

Table 4–1: DSM Command Qualifiers

Qualifier	Function
/[NO]BREAK	Enables and disables BREAK command recognition.
/[NO]CENABLE[=BREAK]	Enables and disables CTRL/C recognition or instructs the interpreter to treat CTRL/C like the BREAK command.
/CLUSTER_SIZE=n	Sets the number of pages brought into memory as the result of a global section page fault.*
/DELETE	Deletes permanent global sections.*
/ERROR=file-spec	Specifies the output device for system messages.
/GLOBALS=file-spec[/LIBRARY]	Specifies a default global directory or library global directory for the duration of the DSM session.
/INPUT=file-spec	Specifies the principal input device.
/INSTALL	Installs a mapped routine file in the system as a permanent global section.*
/KEY_SIZE=n	Establishes the maximum size of a global key.
/[NO]MAPPED[=file-spec][/LIBRARY]	Establishes random access input from a global section, or allows you to create a virtual memory section.
/OPEN_GLOBALS=n	Establishes the number of globals that can be open simultaneously.
/[NO]OPTIMIZE_BUFFER_COUNT	Overrides VAX–11 DSM optimization of the RMS multi-buffer count for open globals.
/OUTPUT=file-spec	Establishes your principal output device.
/ROUTINES=file-spec[/LIBRARY]	Changes your default routine directory or library routine directory for the duration of the DSM image.
/SECTION_NAME=name	Specifies the name of the DSM mapped routine section.
/[NO]SEQUENTIAL_OPTIMIZATION	Optimizes the sequential access to global variables.
/[NO]SHARED	Determines whether file sharing is enabled or disabled, if your process communicates with the DSM Job Controller, and whether virtual memory sections are mapped privately or globally.
/SOURCE_BUFFER_SIZE=n	Establishes the size (in bytes) of the DSM source routine buffer and precompiled routine buffer.
/STACK_SIZE=n	Specifies the interpreter's stack size.
/SYMBOL_TABLE_SIZE=n	Establishes the size (in bytes) of the local symbol table.

Table 4–1 (Cont.): DSM Command Qualifiers

Qualifier	Function
/SYSTEM	Used with /MAPPED and /SECTION_NAME to indicate a system-wide application section.
/TERMINAL_BUFFER_SIZE=n	Establishes the size (in bytes) of your terminal's output buffer.
/TYPEAHEAD	Indicates that DSM READ commands with literals will result in a VMS "read with prompt", but the typeahead buffer will never be purged.
/[NO]UNWIND_STACK[=ALL]	Enables and disables unwinding of the DSM call stack when an error occurs.

* Sufficient user privileges are required to use this qualifier. See Part III of this manual for more information.

Table 4–2: DSM Command Qualifier Defaults

Qualifier	Default
/[NO]BREAK	/BREAK (Programmer Mode) /NOBREAK (Application Mode)
/[NO]CENABLE[=BREAK]	/CENABLE (Programmer Mode) /NOCENABLE (Application Mode)
/DELETE	Permanent global sections not deleted
/ERROR=file-spec	/ERROR=SYS$ERROR*
/GLOBALS=[file-spec][/LIBRARY]	/GLOBALS=default VAX/VMS device and directory: /GLOBALS=SYS$LIBRARY:
/INPUT=file-spec	/INPUT=SYS$INPUT*
/INSTALL	Global section not installed
/KEY_SIZE=n	/KEY_SIZE=64
/[NO]MAPPED[=file-spec][/LIBRARY]	/NOMAPPED (Programmer Mode) /MAPPED (Application Mode)
/OPEN_GLOBALS=n	/OPEN_GLOBALS=7
/[NO]OPTIMIZE_BUFFER_COUNT	/OPTIMIZE_BUFFER_COUNT
/OUTPUT=file-spec	/OUTPUT=SYS$OUTPUT*
/ROUTINES=[file-spec][/LIBRARY]	/ROUTINES=default VAX/VMS device and directory:ROUTINES.DSM** /ROUTINES=SYS$LIBRARY:ROUTINES.DSM**
/SECTION_NAME=name	/SECTION_NAME=DSM$ROUTINE_SEC
/[NO]SEQUENTIAL_OPTIMIZATION	/SEQUENTIAL_OPTIMIZATION
/[NO]SHARED	/NOSHARED (Programmer Mode) /SHARED (Application Mode)
/SOURCE_BUFFER_SIZE=n	/SOURCE_BUFFER_SIZE=10240
/STACK_SIZE=n	/STACK_SIZE=8192
/SYMBOL_TABLE_SIZE=n	/SYMBOL_TABLE_SIZE=102400
/SYSTEM	Not present by default
/TERMINAL_BUFFER_SIZE=n	/TERMINAL_BUFFER_SIZE=992
/TYPEAHEAD	No typeahead
/[NO]UNWIND	/UNWIND

* The translation of this process-permanent logical name.

** This is the original default. You can define a logical name to represent another default, as described in the text below.

4.4.1 /BREAK and /NOBREAK

/BREAK
/NOBREAK

Instructs the interpreter to recognize or ignore the DSM BREAK command, which sets breakpoints for use in debugging DSM routines. The default in Programmer Mode is /BREAK, so that the interpreter recognizes the BREAK command. The default in Application Mode is /NOBREAK, so that the interpreter ignores the BREAK command. (Thus, you need not edit out BREAKs when you run a routine in Application Mode.)

4.4.2 /CENABLE and /NOCENABLE

/CENABLE[=BREAK]
/NOCENABLE

Instructs the interpreter to:

1. Recognize or ignore CTRL/C
2. Treat CTRL/C like the DSM BREAK command if CTRL/C recognition is enabled

How the interpreter treats CTRL/C depends on the user mode. Section 4.3 describes the behavior of CTRL/C in each user mode.

The default in Programmer Mode is /CENABLE. In Programmer Mode, the interpreter recognizes CTRL/C. The default in Application Mode is /NOCENABLE. In Application Mode, the interpreter ignores CTRL/C.

VAX–11 DSM returns an error if you use the form /NOCENABLE=BREAK.

You can use the [NO]CENABLE option on the USE command to override the setting of the /[NO]CENABLE qualifier on the DSM command. See Chapter 6 for a description of USE command options.

4.4.3 /CLUSTER_SIZE

/CLUSTER_SIZE=n

Sets the number of pages brought into memory as the result of a global section page fault. The qualifier overrides the VAX/VMS default cluster size specified with the PFCLUST sysgen parameter.

4.4.4 /DELETE

/DELETE

Indicates that the application or library installed as a permanent global section is to be be deleted. See Section 11.2.2 for details on how to use /DELETE.

4.4.5 /ERROR

/ERROR = file-spec

Specifies the output device or output stream to which VAX–11 DSM writes messages. The default output device for messages is SYS$ERROR. SYS$ERROR is a process-permanent logical name that, for interactive users, equates with your terminal.

4.4.6 /GLOBALS

/GLOBALS = [file-spec][/LIBRARY]

Changes your default directory for globals created in DSM application routines or library routines until DSM image rundown. You apply the /LIBRARY qualifier to specify a library directory; otherwise, a routine directory is assumed.

The default global directory is the VAX/VMS default directory and device, unless you have defined the logical name DSM$GLOBAL_DIRECTORY to refer to another default directory and device. The default library directory is SYS$LIBRARY, unless you have defined the logical name DSM$GLOBAL_LIBRARY to refer to another default library directory. To access a global directory or library global directory other than the default, you append any part of a file specification except a file name to the qualifier name. The file name field is always the global name.

NOTE

To access a directory with the /GLOBALS qualifier, you must have created the directory previously with the DSM CREATE/DIRECTORY command.

See Section 5.2 for a complete description of VAX–11 DSM directories and their specifications.

4.4.7 /INPUT

/INPUT = file-spec

Specifies your principal input device or input stream. The default principal input device is SYS$INPUT. SYS$INPUT is a process-permanent logical name that, for interactive users, equates with your terminal.

4.4.8 /INSTALL

/INSTALL

Indicates that the DSM mapped routine file specified should be installed as a permanent global section. See Sections 5.8.2 and 11.2.2 for information on how to use the /INSTALL qualifier.

4.4.9 /KEY_SIZE

/KEY_SIZE=n

Changes the maximum permissible length of a global key, that is, the subscript field of a global variable.

This qualifier only applies to globals created during the current terminal session. The maximum key size of existing globals cannot be modified.

The value of n must be greater than or equal to 4 and less than or equal to 255. The default key size is 64 characters.

Refer to Section 9.7.3.1 for more information about this qualifier.

4.4.10 /MAPPED and /NOMAPPED

/[NO]MAPPED[=file-spec][/LIBRARY]

Serves either of the following functions:

1. Establishes access to a set of DSM routines that have been mapped in a global virtual memory section.
2. Allows you to create a private or global virtual memory section and map a set of DSM routines in it.

You must have sufficient user privileges to create a global virtual memory section.

The default in Programmer Mode is /NOMAPPED. The default in Application Mode is /MAPPED. Thus, in Application Mode, VAX–11 DSM searches any virtual memory section for a routine that you want to execute before it searches your default directory.

For more information about mapped directories and libraries, virtual memory sections, and the /MAPPED qualifier, see Section 5.8.2. Also see the description of the following qualifiers:

- /SHARED
- /SECTION_NAME
- /CLUSTER_SIZE
- /SYSTEM
- /INSTALL
- /DELETE

4.4.10.1 Mapping a Global Section — A virtual memory section is a copy of an image or data file that can be mapped in a process's virtual address space at run time. Virtual memory sections can be private or shared. When a virtual memory section is shared, it is called a global section.

To map an existing global section in your process's virtual address space, you use the /MAPPED qualifier along with the /SHARED qualifier. If you append the /LIBRARY subqualifier to /MAPPED, a library section is mapped. Otherwise, a routine directory is mapped.

4.4.10.2 Creating a Private Virtual Memory Section — To create a private virtual memory section and map a directory or library in it, you must append a file specification to /MAPPED. You must not specify /SHARED in this case. VAX–11 DSM then requests the Create and Map Section (CRMPSC) system service to create a virtual memory section in your process's virtual address space. The specified file is then mapped to it.

To map a directory privately, you append a file specification to the /MAPPED qualifier. To map a library, you must append the /LIBRARY subqualifier to the file specification, as in the following example:

/MAPPED = newfile/LIBRARY

4.4.11 /OPEN_GLOBALS

/OPEN_GLOBALS = n

Changes the maximum number of globals that can be open simultaneously. The default number of globals that DSM keeps open simultaneously is 7.

Refer to Section 9.7.3.2 for more information about this qualifier.

4.4.12 /OPTIMIZE_BUFFER_COUNT and /NOOPTIMIZE_BUFFER_COUNT

/[NO]OPTIMIZE_BUFFER_COUNT

Overrides VAX–11 DSM optimization of the RMS multi-buffer count for existing globals. The default is /OPTIMIZE_BUFFER_COUNT. If you specify /NOOPTIMIZE_BUFFER_COUNT, the system uses the RMS default buffer counts that apply to indexed sequential (ISAM) files for all global access.

The /OPTIMIZE_BUFFER_COUNT qualifier does not apply to newly created globals. For new globals, the RMS default multi-buffer counter is used, whether or not /OPTIMIZE_BUFFER_COUNT was specified on the DSM command.

Refer to Section 9.7.3.3 for more information about the RMS multi-buffer count and this qualifier.

4.4.13 /OUTPUT

/OUTPUT=file-spec

Specifies your principal output device or output stream.

The default principal output device is SYS$OUTPUT. SYS$OUTPUT is a process-permanent logical name that, for interactive users, equates with your terminal.

4.4.14 /ROUTINES

/ROUTINES=[file-spec][/LIBRARY]

Changes your default directory for DSM application routines or library routines until DSM image rundown.

To access a DSM application routine directory other than the default, append any part of a file specification to the qualifier name. To access a DSM library routine directory, you must append the /LIBRARY subqualifier to the file specification. Otherwise, a routine directory is assumed.

When you use this qualifier, the original default routine directory is the file ROUTINES.DSM in the default VAX/VMS device and directory. If you define the logical name DSM$ROUTINE_LIB, the translation of the logical name is used as the default instead.

The original default file specification for a library directory is SYS$LIBRARY:ROUTINES.DSM. If you define the logical name DSM$ROUTINE_DIR, the translation of the logical name is used as the default instead.

See Section 5.2 for further details about VAX–11 DSM directories.

4.4.15 /SECTION_NAME

/SECTION_NAME=name

Specifies the name of the DSM mapped routine section. This qualifier must be used in conjunction with the /MAPPED qualifier.

The /SYSTEM, /INSTALL, or /DELETE qualifiers can be used with /SECTION_NAME.

The argument name is the name of the DSM mapped routine section. It can be up to 15 characters long. The default name is the translation of the logical name DSM$ROUTINE_SEC.

The /SECTION_NAME qualifier can only be used for application sections. You cannot use it for mapped library sections.

4.4.16 /SEQUENTIAL_OPTIMIZATION and /NOSEQUENTIAL_OPTIMIZATION

/[NO]SEQUENTIAL_OPTIMIZATION

Optimizes the sequential access to global variables through $NEXT, $ORDER, $ZNEXT, and $ZORDER. It does this by caching the Record File Address (RFA) of selected operations.

4.4.17 /SHARED and /NOSHARED

/[NO]SHARED

Serves three functions:

1. Determines whether file sharing is enabled or disabled for this DSM user.

   /SHARED enables file sharing. /NOSHARED disables file sharing. This qualifier does not affect files accessed for implicit sharing. Thus, /[NO]SHARED does not apply to libraries or other files accessed read only.

2. Determines whether your process communicates with the DSM Job Controller.

   /SHARED establishes a communications link between your process and the DSM Job Controller. This link allows the DSM Job Controller to process your lock requests.

   /NOSHARED inhibits communication between your process and the DSM Job Controller; in this case, VAX–11 DSM ignores LOCK and ZALLOCATE commands.

3. Determines whether routines are mapped in private or global virtual memory sections.

   /SHARED causes virtual memory sections to be mapped globally. /NOSHARED causes virtual memory sections to be mapped privately.

   See the description of the /MAPPED qualifier and Section 5.8.2 for more information about virtual memory sections.

The default in Programmer Mode is /NOSHARED. The default in Application Mode is /SHARED.

4.4.18 /SOURCE_BUFFER_SIZE=n

/SOURCE_BUFFER_SIZE=n

Changes the size of your DSM source routine buffer and precompiled routine buffer. The DSM source routine buffer stores the lines of a routine written in the DSM language as ASCII character strings. The precompiled routine buffer stores DSM routines in precompiled format, as described in Chapter 1. The default size of these buffers is 10240 bytes.

4.4.19 STACK_SIZE

/STACK_SIZE=n

Changes the size of your DSM call stack. The size of the stack determines how many call frames an application can keep active at the same time — that is, how many nested DO statements, FORs, or XECUTE commands can be used. If the stack is too small, VAX–11 DSM issues the %DSM–E–STKOVF error message.

4.4.20 /SYMBOL_TABLE_SIZE=n

/SYMBOL_TABLE_SIZE=n

Changes the size of your local symbol table. The local symbol table contains an entry for all defined local variables, subscripted or unsubscripted. The default size of the local symbol table is 102400 bytes.

4.4.21 /SYSTEM

/SYSTEM

Used with /MAPPED and /SECTION_NAME to indicate a system-wide application section.

4.4.22 /TERMINAL_BUFFER_SIZE=n

/TERMINAL_BUFFER_SIZE=n

Changes the size of your terminal's output buffer, that is, the output buffer size for the principal device. The size of this buffer determines the maximum size of a single physical write operation to the terminal. The default size of the output buffer is 992 bytes. The maximum value of the parameter argument n depends on the value of the VAX/VMS sysgen parameter MAXBUF. Consult your system manager for more information about MAXBUF.

4.4.23 /TYPEAHEAD

/TYPEAHEAD

Overrides the DSM default by which the READ literal command, issued from a terminal, causes typeahead to be purged. If you specify /TYPEAHEAD, typeahead is never purged.

4.4.24 /[NO]UNWIND_STACK[=ALL]

/[NO]UNWIND_STACK[=ALL]

When an error occurs, VAX–11 DSM unwinds the DSM call stack automatically, back to the call frame of the last declared error handler. The [NO]UNWIND_STACK qualifier is provided for backward compatibility with VAX–11 DSM Version 1.0. If /NOUNWIND is specified, the stack is not unwound (the same behavior as in Version 1.0). If you specify /NOUNWIND, you can use the ZCLEAR command to unwind the stack explicitly.

If you specify /UNWIND=ALL, the DSM call stack is cleared whenever an error occurs.

4.5 Examples of the Extended Command Line

The following are examples of the DSM command with qualifiers:

1. `DSM/KEY_SIZE=83/SOURCE_BUFFER_SIZE=11000/SYMBOL_TABLE=14000`

   Invokes VAX–11 DSM in Programmer Mode and overrides the following defaults:

   - Global key size
   - Source and precompiled routine buffer size
   - Symbol table size

2. `DSM/CENABLE=BREAK/GLOBALS=WRKD$:[CYG.DSM]/LIBRARY`

   Invokes VAX–11 DSM in Programmer Mode and changes the default library global directory to a VAX/VMS subdirectory located on the device indicated by the logical name WRKD$. It also instructs the interpreter to treat CTRL/C like the DSM BREAK command.

3. `DSM/INPUT=DATA.DAT;2/OUTPUT=LPA0:DATA.OUT`

   Processes the input file DATA.DAT;2, and sends the output to the line printer. The line printer listing has DATA.OUT as a header. Note that this happens only if the line printer is a spooled device.

4. `DSM/BREAK/NOSHARED ^INFO`

   Invokes VAX–11 DSM in Application Mode and executes the routine INFO at start-up. BREAK recognition is enabled, and file sharing is disabled.

5. `DSM/OPEN_GLOBALS=10/ROUTINES=NEWDIR ^RECORDS`

   Invokes VAX–11 DSM in Application Mode and:

   - Executes the routine RECORDS
   - Overrides the default maximum number of open globals
   - Changes the default routine directory to a file called NEWDIR.DSM. The routine RECORDS resides in this directory.

6. `DSM/NOMAPPED ^INVENTORY`

   Invokes VAX–11 DSM in Application Mode; loads and executes the routine INVENTORY from the default routine directory.

7. `DSM/MAPPED=LIBRARY/LIBRARY`

   Invokes VAX–11 DSM in Programmer Mode and creates a private virtual memory section for the file LIBRARY.VIR.

Chapter 5
Developing and Maintaining Application Routines

This chapter describes how to develop and maintain VAX–11 DSM application routines. It describes:

- Creating routines
- Saving and loading routines
- Deleting and renaming routines
- Storing routines on sequential files
- Editing routines
- Debugging routines
- Using directories
- Error processing
- Mapping routines

To use this chapter effectively, you should be familiar with the syntax of the DSM language as described in the *VAX–11 DSM Language Reference Manual.*

5.1 Creating Routines

The VAX–11 DSM language can be used in two ways:

1. In *Direct Mode*, in which all commands entered at the terminal are executed (interpreted) immediately.
2. Through *routines* made up of multiple lines of DSM code. Routines can be stored and used in future DSM sessions.

Introduction to DSM, included in your documentation set, provides a tutorial in developing routines. The following sections summarize these subjects for the purpose of reference.

5.1.1 Direct Mode

When you enter DSM in Programmer Mode, you may want to use the language in Direct Mode. Direct Mode is useful for learning how the language works, and for getting quick answers to computational problems.

In Direct Mode, you enter commands after the DSM prompt (>). VAX–11 DSM executes the command immediately. The following example shows your input to DSM in Direct Mode and the interpreter's response:

```
> WRITE "Hello"
Hello
```

5.1.2 Entering Lines in the Routine Buffer

Direct Mode is limited because you must wait for DSM's response after each line of code. For more sophisticated programming tasks, you use routines instead of individual lines of code. You signal to DSM that a line of code belongs to a routine by starting the line with a TAB or a label followed by a TAB. The following are examples of routine lines:

```
>START<TAB>WRITE "Powers of 2"
><TAB>SET X=1
><TAB>FOR I=1:1:10 SET X=X*2 WRITE X," "
```

The routine lines shown above make up a routine that calculates and displays on the terminal the first ten powers of 2. You execute these lines by entering the DO command, using the first label in the routine as the point at which execution begins:

```
>DO START
```

DSM responds by displaying the output of the routine:

```
Powers of 2
2 4 8 16 32 64 128 256 512 1024
```

Because they start with a TAB or a label and a TAB, the lines stay in your routine buffer. You can execute them again by repeating the DO command.

5.1.3 Using the ZPRINT and ZREMOVE Commands

You can display the lines in your routine buffer by issuing the ZPRINT command. The ZPRINT command leaves the pointer (showing your location in the buffer) at the end of the routine.

If you specify a line label with ZPRINT, DSM displays the line specified and positions the pointer after that line.

The ZREMOVE command deletes the contents of the routine buffer. If you specify a line label with ZREMOVE, that line is deleted.

After you delete a line with ZREMOVE, you can replace the line by entering a new line starting with TAB or with a label and TAB. The following lines show how ZREMOVE is used in deleting and replacing a routine line:

```
>ZREMOVE START
>START<TAB>WRITE "Powers of two"
```

To insert a line without removing one, you first move the pointer to the position after a line by using ZPRINT. Then, you enter the new line, as shown in the following example:

```
>ZPRINT START
START   WRITE "Powers of two"
><TAB>WRITE !,"from 1 to 10"
```

To insert a line at the beginning of the routine buffer (the area in which your routine lines are kept), type ZPRINT +0 to position the pointer before the first line in the buffer. Then enter the new line.

5.2 Saving and Loading Routines

If you want to save a DSM routine for later use, you can store in in a *routine directory*. A routine directory is a VAX/VMS file with the default name ROUTINES.DSM. If you want your routine directory to have another name, include the /ROUTINES qualifier with the DSM command, as described in Section 4.4.14.

You must restore the routine to your routine buffer if you want to add more lines, remove lines, or edit the routine with the DSM editor (described in Section 5.5.1).

To display a list of the routines in your routine directory, invoke the ^%RD (Routine Directory) utility, as follows:

```
>DO ^%RD

   VAX-11 DSM Routine Directory of WRK$:[TEST]ROUTINES.DSM;

ABC     DEF     XXX

   3 routines

>
```

5.2.1 Saving a Routine in a Routine Directory

The ZSAVE command places the routine currently in your routine buffer into your DSM routine directory.

The argument of the ZSAVE command is the name under which you want to save the routine. For example, the following command stores the contents of the routine buffer in the routine directory under the name ABC:

```
>ZSAVE ABC
```

You use the same name when you load the routine back into your routine buffer, as described in the next section. You also use the routine name when you execute the routine from the routine directory. To execute a routine that is in your routine directory, you issue the DO command with the name of the routine, preceded by a circumflex (^), as shown in the following example:

```
>DO ^ABC
Powers of two
From 1 to 10
```

If you try to ZSAVE an unnamed routine (that is, if you enter ZSAVE without an argument while code is in your routine buffer), VAX–11 DSM returns a PNAME (Bad Routine Name) error.

A routine in your routine buffer may already have a name because it was already stored and then loaded back into the routine buffer. You determine the name (if any) of the current routine by issuing the following WRITE command:

```
>W $TEXT(+0)
```

The function $TEXT(+0) always contains the string DSM uses to name the current routine.

5.2.2 Loading a Routine from the Routine Directory

To restore a routine to the routine buffer, enter the ZLOAD command with the name of the routine as the argument:

```
>ZLOAD ABC
```

5.2.3 Forms in Which a Routine Is Stored

The ZSAVE command stores each routine in the directory in two forms:

1. In *source form*, as the ASCII character strings that make up each DSM language statement in the routine.
2. In *precompiled* form, as source code that has been partially processed to expedite subsequent interpretation. See Section 1.1.3 for more information about precompilation.

5.3 Deleting and Renaming Routines

As described in Section 5.1.3, the ZREMOVE command deletes lines from your routine buffer. In conjunction with other commands, the ZREMOVE command can be used to delete routines stored in your routine directory and rename routines. The following sections describe these uses of ZREMOVE.

5.3.1 Deleting a Stored Routine

To delete a routine stored in your routine directory, first enter the ZREMOVE command to clear your routine buffer. Then enter the ZSAVE command with the name of the routine that you want to delete as its argument. This sequence saves an empty routine buffer under the same name as the routine that you want to delete. After you delete a routine from the routine directory, DSM also removes that routine's name from the list displayed by ^%RD.

The following command sequence deletes from disk a routine called BOOKSTAT that you once saved:

```
>ZREMOVE
>ZSAVE BOOKSTAT
```

5.3.2 Renaming Routines

The procedure for renaming a routine stored on disk is similar to the procedure for deleting a routine stored on disk. First use the ZLOAD command to load the routine into your routine buffer. Then refile the routine using the ZSAVE command with the new name as its argument. This procedure produces two copies of the same routine in the routine directory.

To keep only a single copy of the routine under the new name, you must delete the routine under its original name. You do this by saving an empty routine buffer using the old routine name. The following example shows how to change the name of the routine BOOKSTAT to BKCHK and delete the original version of the routine:

```
>ZL BOOKSTAT
>ZS BKCHK
>ZR
>ZS BOOKSTAT
```

5.4 Using Sequential Files to Store Routines

In addition to storing routines in a routine directory for execution, you can write routines to a sequential file for later printing or backup, as described in the following sections. However, the DSM utilities ^%RS (SAVE) and ^%RR (RESTORE) move routines to and from a sequential storage medium, so you need not execute the steps in Sections 5.4.1 or 5.4.2 manually.

NOTE

Remember that you cannot execute (use the DO command on) a routine from a sequential file.

5.4.1 Writing a Routine Onto a Sequential File

To write a routine to a sequential file, follow these steps:

1. Issue the OPEN command to gain ownership of the device.
2. Issue the USE command to make the device current.
3. Issue the ZPRINT command followed by two spaces to write the routine.

The following command sequence writes the routine currently in your routine buffer to a sequential file called FILEM.DAT:

```
>O "FILEM" U "FILEM" ZP  W ! C "FILEM"
```

Note the two blanks after the ZP command in this command sequence. Two blanks are required because the ZP command has no argument.

NOTE

You must not use the file types .DSM, .GBL, or .VIR for routines stored in sequential files, since these file types have special meaning for DSM operations. Use a file type such as .DAT or .BAC.

You can also invoke the DSM routine ^%RS to save a routine in a sequential file. The ^%RS routine is useful if you intend to edit your routine with a VAX/VMS–supported editor, as described in Section 5.5.2.

5.4.2 Loading a Routine from a Sequential File

To load a routine from a sequential file manually, use the following procedure:

1. Issue the OPEN command to gain ownership of the device.
2. Issue the USE command to make the device current.
3. Enter an argumentless ZLOAD command to load the first routine in the file into your routine buffer. ZLOAD reads lines into your buffer until it encounters a null line or an end-of-file.

For example:

```
> O "FILEM" U "FILEM" ZL
```

If the routine you want to load is not the first record of the file, (that is, in the first physical position), use multiple ZLOAD commands separated by two spaces to load the desired routine. The number of ZLOAD commands you use should match the position of the routine in the file. For example, two ZLOADs loads the second routine in the file; three ZLOADs loads the third, and so forth.

The DSM utility ^%RR (RESTORE) loads a routine from a sequential file. The ^%RR utility is useful if you are using a VAX/VMS–supported editor, as described in Section 5.5.2.

5.5 Using Editors

If you want to edit a routine, you have several methods to choose from, in addition to the use of ZREMOVE described in Section 5.1.3. VAX–11 DSM provides a text editor that can be used interactively in Programmer Mode. This editor is described below in Section 5.5.1.

You can also invoke one of the text editors supported by VAX/VMS, as described in Section 5.5.2.

5.5.1 Using the DSM Editor

VAX–11 DSM has an interactive text editor that allows you to edit the code in your routine buffer. The DSM routine editor can change text on one line or in a range of lines.

The editor displays a sequence of question-like prompts. At each prompt, you enter the information required to do your editing task. If you enter a question mark (?) at an editor prompt, the editor displays a list of your options.

Before using the DSM editor, bring the routine that you want to edit into the routine buffer (if it is not already there). The following command invokes the routine editor:

```
> XECUTE ^%ED
```

or, in abbreviated form:

```
> X ^%ED
```

The editor's initial response to this command is:

```
LINE>
```

This response indicates that you have entered the DSM editor and that the editor is waiting for you to respond to its first query. To exit %ED, simply type a period (.).

The DSM editor displays seven prompts. Appendix B describes each prompt in detail. See *Introduction to DSM* for tutorial information on using the DSM editor.

5.5.2 Using VAX/VMS Editors

The VAX/VMS operating system provides a number of powerful text editors to help you develop and maintain your application software. Among these editors are EDT, TECO, SOS, EDIT, and VTECO. Some of these editors are documented in other DIGITAL publications, such as the *VAX–11 SOS Text Editing Reference Manual* and the *EDT Editor Manual.* Both of these books are included in your VAX/VMS documentation set. All of the editors are documented in the VAX/VMS HELP facility.

To use one of these editors to edit DSM routines, follow these steps:

1. Use the DSM library utility Routine Save (^%RS) to transfer a routine or group of routines to a VAX/VMS sequential file.
2. Through this utility, place the output file in your default VAX/VMS directory, after which you can edit it with any editor you choose.
3. After your editing session, use the DSM library utility Routine Restore (^%RR) to transfer the routine(s) to your DSM routine directory.

   Refer to Chapter 7 for a description of these utilities.

The following example shows how a routine is transferred to a file for editing and then restored to the DSM routine directory.

```
> D ^%RS

Routine Save

Output file?  MYROUT<RET>
routine (s)>ABC<RET>
routine (s)?<RET>

Saving routines on WRK$:[TEST]MYROUT.DAT;1

ABC

   1 routines saved

> HALT

%DSM-I-HALT, HALT command executed

$ EDIT MYROUT.DAT
      .
      .
      .
$ DSM
```

```
   VAX-11 DSM Version 2.0

> D %RR

Routine Restore

Input file?  MYROUT↵

Restoring routines from WRK:[TEST]MYROUT.DAT;1
Saved on 19-AUG-1982 17:05:09.55

ABC

>
```

5.6 Starting and Stopping Routines

In Programmer Mode, you can execute the routine that is currently in your routine buffer or a routine that is stored on disk, as described in Sections 5.1.2 and 5.2.1. In Application Mode, a routine is executed immediately after DSM image start-up. In either case, DSM executes the precompiled version of the routine.

Section 5.6.1 summarizes the rules for executing the routine that is currently in your routine buffer. Section 5.6.2 describes how you execute a routine stored in your routine directory.

Section 5.6.3 lists the conditions under which a routine stops executing.

5.6.1 Executing the Routine in Your Routine Buffer

To execute a routine that is currently in your routine buffer, you use the DO command in one of the following forms:

1. The DO command *without an argument* causes VAX–11 DSM to execute the routine lines in the buffer, starting from the first executable line.
2. The DO command *with an argument* causes VAX–11 DSM to begin executing at the line specified in the argument. For example, enter:

   ```
   > DO PARA+3
   ```

   Where PARA + 3 is a routine line specified by label plus offset. VAX–11 DSM executes the routine starting with line PARA + 3.

5.6.2 Executing a Routine from the Routine Directory

To load and execute a routine stored in your routine directory, you issue the DO with an argument. The argument is the name of the routine in your routine directory that you wish to execute. It is always preceded by a circumflex (^).

For example, the following command line loads the routine PRIME into your routine buffer and executes it:

```
>DO ^PRIME
```

You can also specify a routine line in this syntax, for example:

```
>DO GG+1^PRIME
```

This command loads the routine PRIME into your routine buffer and executes it starting from line GG+1.

5.6.3 Conditions for Execution to Stop

VAX–11 DSM continues to execute a routine until one of the following takes place:

- DSM executes a HALT command (which causes rundown of the DSM image and transfer of control to the CLI).
- DSM executes a QUIT command at the outermost nesting level. Control returns to the DSM interpreter if execution was begun in Programmer Mode; control returns to the CLI if execution was begun in Application Mode.
- DSM executes a BREAK command. See Section 5.7 for more information about the BREAK command and debugging.
- You type a CTRL/C from the Principal I/O Device (if CTRL/C recognition is enabled). The Principal I/O Device is generally the terminal where you logged in. Transfer of control is the same as for the QUIT command.
- You type a CTRL/Y on the Principal I/O Device (if CTRL/Y recognition is enabled). Control returns to the command language interpreter.
- An error occurs.

5.7 Using the VAX–11 DSM Debugger

The VAX–11 DSM Debugger is a facility for monitoring the execution of routines. It allows you to stop the execution of your routine at any point you desire and enter *break mode*, a variety of direct mode in which you can enter any DSM command except ZLOAD, ZREMOVE, and ZINSERT. Then you can start execution of your routine again. You can interrupt execution as often as you like.

The points where you stop your routine are called *breakpoints*. You can set as many breakpoints as you like, of which nine are valid at any given time. You can also execute your routine in *single-step mode*, in which execution is interrupted after every command in the routine.

The debugger is implemented through a set of VAX–11 DSM commands, a set of qualifiers on the DSM command line, and a system variable. The commands, described in full in the *VAX–11 DSM Language Reference Manual*, are:

- BREAK[:postconditional] *argument*
- ZDEBUG[:postconditional] *argument*
- ZSTEP[:postconditional] *argument*
- ZGO[:postconditional]

Using these commands is described later in this chapter.

The qualifiers on the DSM command are /[NO]CENABLE=BREAK and /[NO]BREAK. As described in Section 4.4.2, /NOBREAK inhibits the recognition of breakpoints altogether. The /CENABLE=BREAK qualifier causes DSM to interrupt execution of your routine when a CTRL/C is entered from the terminal. The negative form, /NOCENABLE, causes DSM to ignore CTRL/C.

The special variable $ZBREAK, described in detail in the *VAX–11 DSM Language Reference Manual*, is used to contain a reference to the location where you set a breakpoint. Section 5.7.1.1 below describes how you use $ZBREAK.

5.7.1 Breakpoints

VAX–11 DSM allows you to set nine breakpoints for normal execution and one breakpoint for single-step execution. (These ten breakpoints do not include the breakpoints set with the BREAK command, described in Section 5.7.1.2.)

You refer to the nine breakpoints for normal execution as $ZBREAK(1) through $ZBREAK (9). You refer to the single-step breakpoint as $ZBREAK(0).

You set these breakpoints, examine their contents, and clear them as described in the following sections.

When VAX–11 DSM encounters a breakpoint in a routine, and the ZDEBUG flag (described in Section 5.7.4) is ON, DSM displays the following messages:

```
%DSM-I-BREAK BREAK command executed
-DSM-I-ATLABEL entry-ref routine-line
BREAK b:n>
```

The arguments entry-ref and routine-line identify the location in the routine where the breakpoint was encountered.

The argument b identifies the breakpoint. For normal execution, b is a number from 1 to 9. For single-step execution, b is 0. When CTRL/C is enabled to cause a BREAK, the characters "^C" are displayed instead. (If you reach a breakpoint through the BREAK command, described in Section 5.7.1.2, b:n is the value of the argument of the BREAK command.)

The argument n is the command number (on the routine line) before which the breakpoint took place.

5.7.1.1 Setting Breakpoints with $ZBREAK — You set a breakpoint at a particular location in your program by issuing the DSM SET command with the breakpoint variable $ZBREAK as its argument. (You can set, modify, or kill elements in $ZBREAK regardless of the value of the DEBUG flag.) With the $ZBREAK variable, you specify the entry reference and command number of the desired location, in the following format:

SET $ZBREAK="Entry Reference:Command Number"

Entry Reference — Must be a complete routine reference in the form:

label + offset ^routine name

The label must be the closest label to the line you are specifying. If the line you are specifying has its own label, you must specify that label, not an offset from an earlier label. (See Section 4.2.2 for information about labels and offsets.)

Command Number — Must be included; there is no default command number. VAX–11 DSM does not recognize breakpoints on lines with no commands (comment lines, for example).

VAX–11 DSM assigns this location to the first unused breakpoint, beginning with breakpoint 1. If all nine breakpoints are in use, VAX–11 DSM clears breakpoint 9 and assigns the value you specify to breakpoint 9.

You set a particular breakpoint by specifying a breakpoint number with the $ZBREAK variable, as follows:

SET $ZBREAK(b)="Entry Reference:Command Number"

You can specify any number from 1 to 9 as b for normal execution. For single-step execution, specify 0 as b.

5.7.1.2 Setting Breakpoints with the BREAK Command — You can also set a breakpoint in a routine by including the BREAK command in the routine, as follows:

BREAK *argument*

The argument (a number, expression, or string) is evaluated and displayed in the prompt issued by DSM when execution is suspended and you enter break mode through the BREAK command. For example, if you include the command BREAK 4 in a routine, DSM displays the following prompt after its informational messages:

```
BREAK 4>
```

The BREAK command also takes a postconditional expression. For example, the following command string suspends execution only if the value of M is less than 30:

```
BREAK:M<30
```

Recognition of the BREAK command depends on the presence (implicit or explicit) of the /BREAK qualifier with the DSM command. In Programmer Mode, /BREAK is the default. In Application Mode /NOBREAK is the default. However, execution of the BREAK command is independent of the status of the DEBUG flag, set with the ZDEBUG command.

See the *VAX–11 DSM Language Reference Manual* for more information on the BREAK command.

5.7.1.3 Breakpoint Actions — A breakpoint action string is a command or set of commands that is executed whenever a breakpoint is encountered. You can specify an action string as part of the $ZBREAK value for the breakpoint you specify, in the following format:

SET $ZBREAK(b)="Entry Reference:Command Number>Action"

The breakpoint action string can contain any commands that are legal in break mode, that is, any commands but ZL, ZR, and ZI. The commands are executed when the breakpoint is reached, before the BREAK message is printed.

You should use an action string to perform simple tasks that you would otherwise have to type every time a breakpoint is reached. For example, you could use an action string to examine the contents of a variable. Breakpoint action strings are also useful for more sophisticated tasks, such as the %TRACE utility described in Section 5.7.5.

You cannot specify a breakpoint action string with the BREAK command.

You can specify a breakpoint action string for use in single-step execution. To do this, specify breakpoint 0, as follows:

SET $ZBREAK(0)=">Action"

This causes the breakpoint action to be performed after each step in your program. Note that you do not specify an entry reference or command number in this command.

5.7.1.4 Examining Breakpoints — To examine the contents of all existing breakpoints, enter the following command:

ZWRITE $ZBREAK

To examine the contents of any particular breakpoint, enter the WRITE command, specifying which breakpoint you want to look at:

WRITE $ZBREAK(b)

In response, VAX–11 DSM displays the contents of the breakpoint as in the following example:

```
$ZB(4)="START+1 TEST:3>W A"
```

5.7.1.5 Clearing (Killing) Breakpoints — To clear all existing breakpoints, enter the following command:

KILL $ZBREAK

To clear a particular breakpoint, specify the breakpoint number in the KILL command:

KILL $ZBREAK(b)

You can also clear a breakpoint by setting the breakpoint to the null string, as follows:

SET $ZBREAK(b)=""

5.7.2 Continuing Execution after a Breakpoint

There are two VAX–11 DSM commands for continuing execution after a breakpoint.

You use the ZGO command (abbreviated as ZG) to continue execution after a breakpoint reached through the BREAK command or through a breakpoint set with SET $ZBREAK. Execution is resumed and continues until another breakpoint is reached.

You use the various forms of the ZSTEP command to continue execution in single-step mode. To start using single-step mode, you first stop your routine at a breakpoint. Then you enter one of the forms of the STEP command, as follows:

ZSTEP OVER or ZSTEP

This form of ZSTEP treats any DO command and the entire subroutine associated with the DO command as one unit. Execution is not halted at any of the lines in the subroutine. Execution is halted when another line is reached in the "top" routine.

ZSTEP INTO

This form of ZSTEP steps to the next command if the current command is a DO. Thus, it steps "into" the subroutine instead of stepping "over" it. Execution is halted at each line in the subroutine.

ZSTEP OUTOF

This form of ZSTEP steps to the first command following the QUIT from the current routine. You usually use ZSTEP OUTOF when you have stepped into a routine with ZSTEP INTO, executed a number of steps, and then decide that you do not need to monitor the remainder of the subroutine.

See Section 5.9.2.2 for more information about nested DO statements and Section 5.9.2.3 for more information about the QUIT command.

5.7.3 Interrupting Execution with CTRL/C

If you equate CTRL/C with the BREAK command, routine execution stops whenever CTRL/C is entered at the terminal. VAX–11 DSM displays the following message:

```
%DSM-I-BREAK BREAK command executed
-DSM-I-ATLABEL entry-ref routine-line
BREAK ^C>
```

You equate CTRL/C with a break by including the /CENABLE=BREAK qualifier on the DSM command.

5.7.4 Enabling and Disabling Debugging

The ZDEBUG command (abbreviated ZDEB, not ZD or ZDE) sets an internal flag that turns the debugger on or off. When this flag is ON, VAX–11 DSM recognizes breakpoints set by means of the $ZBREAK special variable. (However, DSM does not recognize breakpoints if /NOBREAK is in effect, even if ZDEBUG is ON.)

The flag is set to OFF until you set it to ON by entering the following command:

ZDEBUG [ON]

The argument ON is optional. This command is abbreviated ZDEB.

To disable recognition of breakpoints set with $ZBREAK, you set the flag to OFF with the following command:

ZDEBUG OFF

When the debugger is off, VAX–11 DSM still recognizes the BREAK and SET $ZBREAK commands. You disable recognition of the BREAK command and breakpoints set through $ZB by including the /NOBREAK qualifier on the DSM command, as described in Chapter 4.

Both ZDEBUG ON and ZDEBUG OFF can be specified with a postconditional argument. Using a postconditional argument allows you to turn debugging off or on depending on conditions that exist at run time.

5.7.5 Debugging Utilities Provided with VAX–11 DSM

The VAX–11 DSM software includes two utilities, %TRACE and %STACK, that use the debugging facility to provide information about the execution of routines. You access these utilities by issuing the D ^%TRACE or D ^%STACK commands.

The %TRACE utility allows you to trace the execution path of a program or to monitor changes to variables. It uses the debugging commands and breakpoint action strings. Section 7.4.4.2 shows a sample of %TRACE's output.

The %STACK utility displays the DSM call stack at a breakpoint. Section 7.4.4.1 shows a sample of %STACK's output.

5.8 Using VAX–11 DSM Directories

VAX–11 DSM provides *routine directories* for all DSM routines and *global directories* for globals. Section 5.2 introduced the functions of the routine directory. The following sections describe these directories in greater detail.

5.8.1 VAX–11 DSM Routine Directories

VAX–11 DSM provides two routine directories for each user. These directories catalog two kinds of routines:

- Application routines
- Library routines (routines whose names begin with the percent (%) character)

VAX–11 DSM routine directories are VAX–11 RMS indexed (ISAM) files. (See Section 6.6.1 for a description of indexed file organization). VAX–11 DSM stores every application and library routine as two records of a directory file: one record for the source version of the routine and one record for the precompiled version of the routine.

5.8.1.1 File Specifications of Routine and Library Directories — The file specification that corresponds to your DSM routine and library directories is determined by a set of defaults. The system creates a full VAX/VMS file specification for your directory files according to the following procedure:

1. If the logical names DSM$ROUTINE_DIR and DSM$ROUTINE_LIB are defined, they translate to a VAX/VMS device, directory (or subdirectory), file name, and file type. DSM$ROUTINE_DIR and DSM$ROUTINE_LIB can translate to the same string or to different strings.

2. If the logical names DSM$ROUTINE_DIR and DSM$ROUTINE_LIB are not defined, the system uses your default VAX/VMS device and directory for your application routines, and the translation of the system logical name SYS$LIBRARY for your library routines.
3. The default file name is ROUTINES.
4. The default file type is .DSM.

Thus, depending on how you define the logical names, the default file specification for DSM application routines can be either of the following:

- The translation of DSM$ROUTINE_DIR
- Default Device:[Default Directory]ROUTINES.DSM

The default file specification for DSM library routines can be either:

- The translation of DSM$ROUTINE_LIB
- SYS$LIBRARY:ROUTINES.DSM

You can override any part of the default file specification for DSM application and library routine directories by either of the following methods:

- Issue the DCL ASSIGN command before invoking DSM, as in the following example:

```
$ ASSIGN [MYDIR.DSM] DSM$ROUTINE_DIR
```

  As a result of this assignment, DSM considers your routine directory for this VAX/VMS session to be [MYDIR.DSM]ROUTINES.DSM. To change the routine directory specification for the duration of the DSM image, append the /USER qualifier to the ASSIGN command.

- Use the /ROUTINES qualifier with the DSM command, described in Section 4.4.14. This changes the directory's file specification for the duration of the DSM image.

To examine the contents of your application and library routine directories, run the routine utilities ^%RD and LIB^%RD, described in Section 7.4.2.

5.8.1.2 Size of Routines in Directories — The default maximum size of a DSM routine in your routine buffer is 10240. You can change this maximum size (up to 16290 bytes) with the SOURCE_BUFFER_SIZE qualifier on the DSM command, described in Chapter 4. This maximum size is determined by the maximum size of a VAX–11 RMS record (16300 bytes).

However, the /SOURCE_BUFFER_SIZE qualifier does not increase the maximum size of a routine that you can store in a routine directory. The maximum size of a routine that you can store in a routine directory is determined by its maximum record size, set when the routine directory is first created. This maximum record size is the size of the routine buffer at the time, plus ten.

To increase the maximum size of the DSM routines that can be stored in a routine directory, follow this procedure:

1. Save all your routines in a sequential file, using ^%RS.
2. Using the DCL RENAME command, give the routine directory a temporary name, for example:

   $ RENAME ROUTINES.DSM ROUTINES.OLD
3. Invoke DSM, specifying a larger value with /SOURCE_BUFFER_SIZE.
4. Restore the routines from the sequential file, using ^%RR.
5. Delete the renamed directory (ROUTINES.OLD).

Remember to invoke DSM with the new buffer size in the future. You may want to define the DSM command as DSM/SOURCE=new value in your VAX/VMS log-in command file.

5.8.1.3 Routine Directory Protection and Access Modes — VAX–11 DSM routine files are subject to the same file protection mechanism as any VAX/VMS file. This mechanism is described in Section 3.4.

When you first access a DSM routine directory file (by loading, saving, or executing a routine), VAX–11 DSM opens the file. If the DSM routine name does not start with the percent (%) character, the routine is by definition an application routine; DSM opens the routine directory file with read/write/delete access.

If the DSM routine name starts with the percent (%) character, the routine is by definition a VAX–11 DSM library routine, and the system opens the routine directory file with read-only access.

If the routine directory file is protected against write access, but DSM was invoked in shared mode, DSM opens the file with read-only access.

If, however, the fully expanded file specifications of your application and library routine directories are identical (called the resultant file specification), VAX–11 DSM opens your library routine directory with read/write/delete access.

5.8.1.4 Creating or Modifying a DSM Library Directory — The system library directory is specified by the translation of the logical name SYS$LIBRARY. SYS$LIBRARY contains the library routines supplied with the VAX–11 DSM system. (It also contains the globals used by these routines.) SYS$LIBRARY:ROUTINES.DSM is your default routine library directory unless DSM$ROUTINE_LIB is defined.

If the resultant file specifications of your application and library routine directories are identical, VAX–11 DSM treats %Routine_Name and Routine_Name as synonyms. In this case, you can save routines that start with "%" without restriction or special preparation.

If, however, the resultant file specifications of your DSM application and library routine directories are not identical, you must use the following procedure to place routines in a directory to be accessed as a library directory:

1. Save a series of routines in an application directory (established with the /ROUTINES=file-spec qualifier of the DSM command, for example) with the names of each routine stripped of the leading "%".
2. Run down the DSM image
3. Reinvoke VAX–11 DSM, declaring the previous application directory to be your library routine directory, by appending /ROUTINES=file-spec/LIBRARY to the DSM command. Using this command qualifier also causes VAX–11 DSM to define DSM$ROUTINE_LIB. (You can also redefine the library name using the DCL ASSIGN command.)
4. Execute the same routines by including the leading "%" in the argument of the DO command.

The following example illustrates this procedure:

```
$ DSM/ROUTINES=MYLIB
> ZSAVE X
> HALT
$ DSM/ROUTINES=MYLIB/LIBRARY

   VAX-11 DSM Version 2.0

> DO ^%X
```

You must have the same UIC as the system library directory (UIC [1,4]) to save routines in this directory.

5.8.1.5 VAX–11 DSM Routine Directories and VAX–11 RMS — All VAX–11 RMS defaults relating to indexed files apply to DSM routine directories. Thus, if you know how to use the RMS utilities, you can create, analyze, convert, and reclaim any DSM routine directory.

The following RMS attributes apply to DSM routine directories:

- Indexed Organization
- Primary Key Position = 0
- Primary Key Size = 10
- Primary Key Type = string
- Maximum Record Size = 10250 or size set with /SOURCE_BUFFER_SIZE, plus 10
- Record Format = variable

The system and process RMS default parameters for indexed files, such as the Multi–Buffer Count, also apply to DSM routine directories. Refer to Chapter 9 for more information about VAX–11 RMS.

5.8.2 VAX–11 DSM Global Directories

VAX–11 DSM provides two global directories for each user. These directories catalog your:

1. Application globals
2. Library globals (global names that start with "%")

Globals comprise the VAX–11 DSM data base. The system represents each global by one VAX–11 RMS ISAM file.

VAX–11 DSM places globals in your application global directory whenever you execute a routine or command line that defines a global variable with the SET command. You can delete an application global (or a sub-tree) by issuing the KILL command. For a complete description of the global SET and KILL procedure, refer to the *VAX–11 DSM Language Reference Manual*. Refer to Chapter 9 for a detailed description of global variables.

At installation time, the system places a set of library globals in SYS$LIBRARY. You can read the contents of these globals (all of which start with %), but you cannot SET or KILL any nodes in them.

The correspondence between a global, the ISAM file that represents it, its VAX–11 DSM directory, and its VAX/VMS file specification is determined by a set of defaults. The system creates a full file specification for your application and library globals according to the following procedure:

1. If defined, the logical names DSM$GLOBAL_DIR and DSM$GLOBAL_LIB translate to a VAX/VMS node, device, directory (or subdirectory), and file type. DSM$GLOBAL_DIR and DSM$GLOBAL_LIB can translate to the same string or to different strings.
2. If the previous logical names are not defined, the system uses the local node and your default VAX/VMS device and directory for application globals, and the translation of the system logical name SYS$LIBRARY for library globals.
3. If the global variable name does not begin with "%", the file name of the ISAM file that represents the global equals the global variable name.

   If the global variable name begins with "%", the file name equals the global variable name stripped of the leading "%" sign.

NOTE

If a global variable name begins with "%", the global is by definition a library global. Generally, library globals can only be accessed for reading. However, if the resultant file specifications of your application and library global directories are identical, you can access library globals read/write.

4. GBL is the default file type for application and library globals. VAX–11 DSM applies this default to the specification after it applies:

   a. The global variable's user field, which specifies a directory or a node in a computer network, as described in Section 9.2.

   b. DSM$GLOBAL_DIR or DSM$GLOBAL_LIB (since these logical names can include a file type in their definition).

5. The default version number for globals is 1. Global file version numbers are not updated each time they are accessed.

Depending on how you define your logical names, the default file specification for any global used in an application routine can be either of the following:

- DSM$GLOBAL_DIR:globalname.GBL
- Default Device:[Default Directory]globalname.GBL;1

The default file specification for any global used in a library routine can be either:

- DSM$GLOBAL_LIB:globalname.GBL
- SYS$LIBRARY:globalname.GBL;1

You can override any part of the default file specification for DSM application and library globals (except the file name) with the DCL ASSIGN command or with the /GLOBALS qualifier of the DSM command. The following examples show these two methods:

```
$ ASSIGN [MYDIR.DSM] DSM$GLOBAL_DIR
$ DSM/GLOBALS=[MYDIR.TEST]
```

To examine the contents of your application and library global directories, run the GLOBAL DIRECTORY (^%GD) and LIBRARY DIRECTORY (LIB^%GD) utilities from the Global Utilities menu, as described in Sections 7.4.1.4 and 7.4.1.6.

Chapter 9 describes how you create and modify your own library globals.

5.9 Error Processing

VAX–11 DSM's error-processing facilities enable the system to behave predictably when an error occurs. They also allow you to control the handling of errors that occur while your routines are running.

5.9.1 Error Severity Levels

VAX–11 DSM treats all errors according to their severity. Errors can have one of five severity values, as follows:

0 = Warning 3 = Information

1 = Success 4 = Severe Error

2 = Error

Standard VAX–11 DSM error processing considers levels 2 and 4 as the basis for stopping a job, and levels 0, 1, and 3 as the basis for continuing a job.

In Programmer Mode, VAX–11 DSM considers all of the following to be level 2 errors:

- Language syntax (interpreter) errors
- Global access errors
- Routine interrupt errors
- I/O errors

Such errors interrupt the execution of the current routine. In Programmer Mode, the system normally responds to errors by:

1. Transferring control from the current device to the Principal I/O Device; this sets the $IO special variable to the principal device.
2. Interrupting the routine.
3. Printing a series of linked messages on the Principal I/O Device that indicate:
   - The line and position in the line of the DSM routine that caused the error, and the type of DSM error that occurred
   - The VAX–11 RMS error (if one occurred)
   - The VAX/VMS system error (if one occurred)

   Refer to Appendix A of this manual for a complete description of DSM error messages and the VAX/VMS documentation for description of VAX/VMS and VAX–11 RMS error messages.
4. Storing device-specific error information in the $ZA and $ZB special variables (see Chapter 6 for further details).

5. Returning to the DSM prompt (>).

In Application Mode, VAX–11 DSM turns all errors into severe errors (severity level 4). Severe errors cause DSM image rundown and transfer control to the CLI.

VAX–11 DSM always considers as severe errors such start-up errors as the following:

- DSM command-line errors
- Failure to open the principal device
- Faulty interaction with the DSM Job Controller or the DSM Journal Process

5.9.2 Error–Processing Routines

You use error-processing routines in a DSM application to prevent errors from being turned into severe errors. The DSM language provides a set of commands and special variables for error processing. Some of these are specifically designed for use in error processing. Others perform special actions when executed in error-processing routines.

The mode of error handling used for your DSM routine depends on your use of the /[NO]UNWIND qualifier on the DSM command.

If you do not specify a qualifier, or do specify /UNWIND, which is the default, DSM treats errors as described in Sections 5.9.2.1 through 5.9.2.4. If you specify /NOUNWIND, DSM handles errors as described in Section 5.9.2.5. The error-handling mode used when you specify /NOUNWIND is compatible with earlier versions of VAX–11 DSM.

NOTE

The ZCLEAR command has no effect unless /NOUNWIND is specified.

5.9.2.1 Default Error–Handling Mechanism — You use the $ZTRAP special variable to establish an error-handling routine for your DSM routine. The value of $ZT, as described in the *VAX–11 DSM Language Reference Manual*, is a reference to the line and/or routine to which control should pass when an error occurs in a routine called from the routine that sets $ZT. ($ZT can be set through the SET command.)

The $ZTRAP special variable should not be confused with the ZTRAP command. When this command is issued, an error occurs, and the value specified as the argument of the command is passed to the error handler.

If you set the special variable $ZT to the null string, and an error occurs, VAX–11 DSM performs standard error processing.

If you set $ZT to "LABEL^ROU", and an error occurs, DSM:

1. Sets the $ZERROR special variable to a string containing the DSM error code, the error message text, and the label and routine name that indicate where the error occurred. This information is followed by VAX/VMS or VAX–11 RMS error messages, if any.
2. Transfers control to the location referenced in $ZTRAP: routine called ROU at entry point LABEL.
3. Unwinds the DSM call stack to the frame in which $ZT was set.

You can examine $ZE to determine which errors occurred and where. At the end of the error handler, you must set $ZE to the null string (" "). This indicates to DSM that the error handler has successfully treated the error condition, so that DSM continues execution of the application (at the point determined by the use of the QUIT or GOTO, as described in Section 5.9.2.3). If $ZE is not null at the end of the error handler, DSM behaves as if the error condition was never corrected. In addition, when the next error occurs, a non-null value of $ZE causes VAX–11 DSM to perform no error recovery at all.

5.9.2.2 The ZQUIT Command and Nested DO Statements — You use the ZQUIT command in your error-processing routines if your application calls its subroutines with nested DO statements. If executed within one of an application's declared error handlers (a routine referenced in $ZTRAP), the ZQUIT command transfers control to the previous error handler, that is, to the error handler of the last subroutine to set $ZTRAP. The ZQUIT command unwinds the call stack down to the frame in which this handler was declared.

When used together, $ZTRAP and ZQUIT provide a mechanism for linking all the error handlers in an application into a hierarchy of handlers, systematically passing an error from one handler to another until the error condition is resolved.

To clarify the context in which error handlers are used, the following paragraphs explain how a call stack is constructed.

Each time a routine calls another routine with a DO statement, VAX–11 DSM builds a data structure called a *DO frame*. The DO frame is built on the call stack (one of the private data structures in your process's virtual address space). VAX–11 DSM stores the context of each routine that issues a DO command in a DO frame. The context includes the current status of any FOR loops, the value of the $ZTRAP special variable, and the position of the return from the subroutine.

Thus, when routine A issues a DO statement to call routine B, VAX–11 DSM builds a DO frame on the call stack to preserve the context of A. When routine B issues a DO to call routine C, VAX–11 DSM adds a DO frame to the stack to preserve the context of B, and so forth.

Each routine in an application can establish its own error-handling routine by setting $ZTRAP to the name of an error handler to which VAX–11 DSM passes control in the event of an error.

For example, consider the following sequence of $ZTRAP settings:

- Routine C (which contains the DO command to execute routine D) sets $ZTRAP to "^CERR"
- Routine B sets $ZTRAP to "^BERR" (and contains the DO command to execute routine C)
- Routine A sets $ZTRAP to "^AERR" (and contains the DO command to execute routine B)

If an error occurs in routine D (the frame of which contains no handler), VAX–11 DSM transfers control to CERR to process the error condition, and unwinds the DO frame for routine D from the call stack. If an error occurs in routine C, VAX–11 DSM transfers control to ^CERR to process the error condition, but does not unwind any frame, since the error occurred in the same frame that $ZT was set to ^CERR in.

5.9.2.3 Exiting from an Error Handler — If an error handler can correct an error condition, it can exit using either of the following commands:

- QUIT
- GOTO

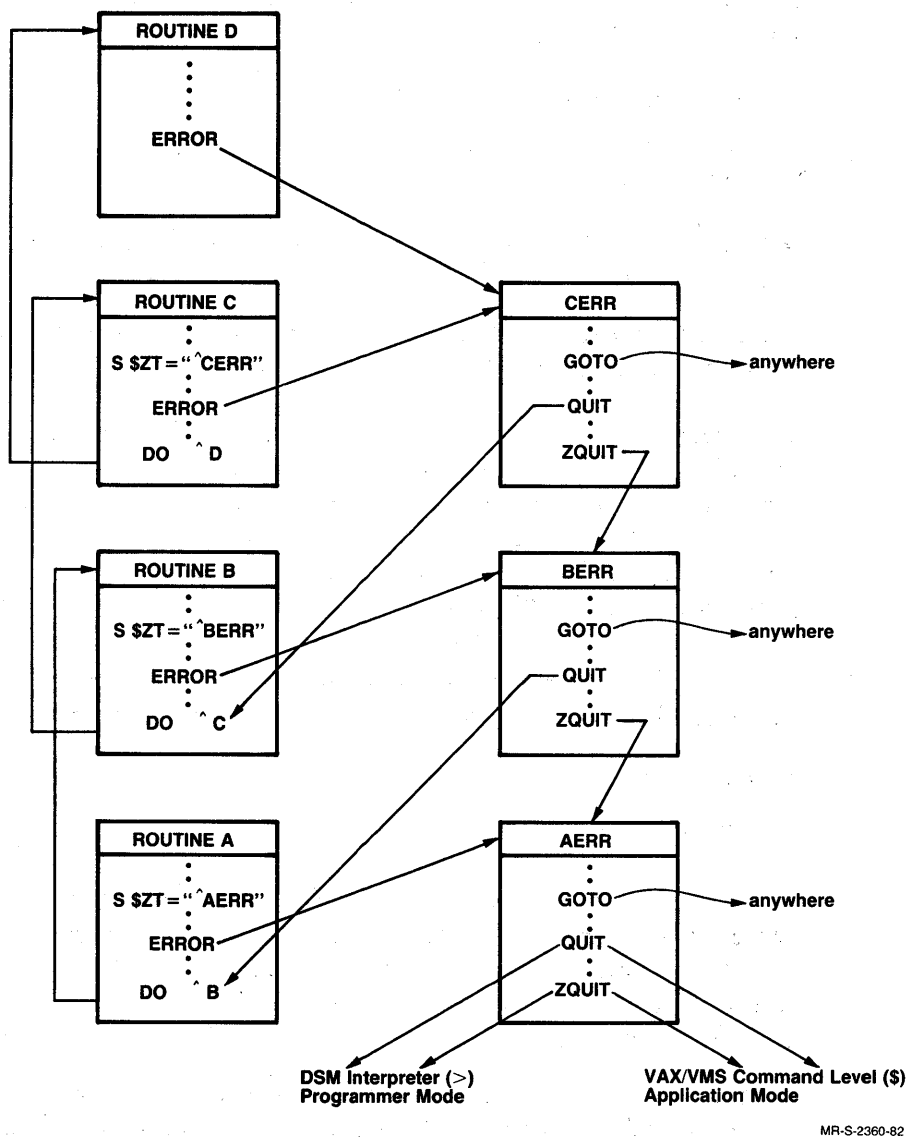
From a declared error handler, QUIT transfers control to the routine in which the error handler was declared. Thus, if routine D, called from routine C, contains an error that transfers control to ^CERR, the QUIT command in ^CERR transfers control to the statement in routine B that follows the DO ^C command.

Usually, you terminate an error-processing routine that can correct an error condition with GOTO. In this way the application can proceed to the next routine (or to any appropriate destination). You can use GOTO to exit an error handler (after setting $ZE to "") because DSM does not add a frame to the call stack when it transfers control to an error handler.

If an error handler cannot correct an error condition, you can terminate the error processing code with ZQUIT to transfer control to the error handler of the last subroutine to set $ZTRAP. For example, if an error occurs in routine D (in the application described above) and the error handler ^CERR cannot correct the error condition, ZQUIT transfers control to ^BERR and unwinds routine C's DO frame from the call stack. If ^BERR cannot correct the error condition, ZQUIT transfers control to ^AERR and unwinds B from the stack, and so forth. If the last declared error handler issues a ZQUIT, the DSM error is handled as if no handler were present.

Figure 5–1 shows the flow of error processing when the error handlers terminate with ZQUIT.

Figure 5–1: Flow of Error Processing with ZQUIT



As shown in Figure 5–1, if you started the application in Programmer Mode, control passes to the DSM interpreter. If this occurs, VAX–11 DSM processes the error using standard error processing. If you started the application in Application Mode, control passes to VAX/VMS and the $STATUS symbol is set to the binary equivalent of the error. In this case, you may want the last error handler in the application to exit with the HALT command.

When DSM executes a HALT command in an error-processing routine, it automatically passes control to VAX/VMS. VAX–11 DSM also passes the status of the last DSM error to the VAX/VMS system variable $STATUS. This variable always contains the status of the last image to execute. In this case, $STATUS contains the binary equivalent of the current DSM error message. (When you run down DSM normally, $STATUS contains the binary equivalent of the message %DSM–I–HALT).

You can test $STATUS in a command procedure and respond to it with the DCL ON command (refer to the *VAX/VMS Command Language User's Guide* for details). Thus, you can continue to process the error condition at VAX/VMS command level if desired.

5.9.2.4 Error Processing if /NOUNWIND Is Specified — If you specify /NOUNWIND on the DSM command, the transfer of control to an error handler does not automatically cause the stack to be unwound down to the frame of the routine that declared the handler. If /NOUNWIND is in effect and you want the stack to be unwound, you must issue the ZCLEAR command.

Similarly, if /NOUNWIND is in effect and a ZQUIT command is issued, control is transferred to the next error handler "down" the stack, but no frames are unwound.

The QUIT and GOTO commands can also behave differently with /NOUNWIND in effect than with /UNWIND. When /NOUNWIND is in effect, you QUIT from the routine in which the error occurred, not from the routine that declared the handler. GOTO transfers control to the location specified, but a subsequent QUIT would transfer control to a different location with /NOUNWIND, because the frame in which the handler was declared is not unwound.

5.10 Using the VAX–11 DSM Mapped Routine Facility

The VAX–11 DSM mapped routine facility allows you to map a precompiled DSM routine or set of routines in a VAX/VMS virtual memory section. After you map the section, you can access its contents, read only, from the DSM image.

Virtual memory sections can be private or shared. A shared virtual memory section is called a global section. Global sections can be accessed by more than one process at a time. Thus, they provide a way for many users to access frequently used code or data in a way that produces the least overhead.

Normally, only one copy of a global section actually resides in memory. When a group of processes require data in the same section, the system maps the contents of the section into the virtual memory space of each requesting process.

If you install a VAX–11 DSM application (or library) in a permanent global section, the following improvements in performance occur:

1. Virtual memory paging replaces reading a record from an indexed file, which improves response time because paging I/O is the fastest I/O in VAX/VMS.

2. The most frequently used DSM routines tend to remain in physical memory, so page faults occur from the free page list rather than the disk page file. (The free page list is a list of physical memory pages that are available for use; it also acts as a cache for pages that were recently discarded by a process as the result of a page fault. Refer to the *VAX/VMS Summary Description and Glossary* and the *VAX–11 Software Handbook* for more information about paging.)

5.10.1 Types of Virtual Memory Sections

A global virtual memory section can be any of the following types:

- Group temporary
- Group permanent
- System temporary
- System permanent

A group global section can be shared by all processes that have the same group number in their UICs.

A system global section can be shared by all processes in the system.

If a group or system global section is temporary, VAX–11 DSM deletes it when no processes are mapped to it. If a group or system global section is permanent, the section remains in existence even when no processes are mapped to it.

You must install permanent global sections in the system with the /INSTALL and /SHARED qualifiers of the DSM command and delete them with the /DELETE and /SHARED qualifiers. These qualifiers are described in Section 11.2. The /CLUSTERSIZE qualifier can be used in conjunction with /INSTALL to override the default VMS page fault cluster size for the section installed. You can also specify /SECTION_NAME and /SYSTEM.

VAX–11 DSM also allows you to map a file in your own process virtual address space. This type of mapping is called private mapping. Privately mapped sections are not shared by other processes in the system. VAX–11 DSM deletes private virtual memory sections during DSM image rundown. If you are in Programmer Mode, or specified /NOSHARED, DSM maps sections privately.

5.10.1.1 Global Section Names — VAX–11 DSM applies two default names to global sections:

DSM$ROUTINE_SEC

The global sections that contain DSM application routines have this logical name. Each group can have only one DSM application installed as a global section. You can include the /SYSTEM qualifier to make this section a system global section instead of a group global section. When you create a

group global section, the system qualifies DSM$ROUTINE_SEC with the group number of your UIC. Thereafter, only processes with the same group number can access that DSM$ROUTINE_SEC. If you include the qualifier /SECTION_NAME=n, the specification named as n supersedes the logical name DSM$ROUTINE_SEC.

DSM$LIBRARY_SEC

The global section that contains DSM library routines has this logical name. There is only one section named DSM$LIBRARY_SEC because only one library can be installed as a global section. This section is always mapped on a system-wide basis. This logical name cannot be overridden.

5.10.1.2 Privileges Required to Create Virtual Memory Sections — You must have special user privileges to create a group or system virtual memory section and make it permanent. Table 5–1 shows the privileges needed to create each type of section.

Table 5–1: Privileges Needed to Create Virtual Memory Sections

Section Type	Privileges
Group Permanent	PRMGBL
System Temporary	SYSGBL
System Permanent	SYSGBL PRMGBL

See the *VAX/VMS System Management and Operations Guide* for information about these user privileges.

No special privileges are required to:

- Create a private virtual memory section
- Map to an existing group or system global section

Thus, all users can access files that are mapped in a global section.

5.10.2 Creating and Mapping Virtual Memory Sections

To map a DSM routine or set of routines, either as a global section or a private section, you must build a mapped routine file with a VAX–11 DSM library utility called ^%RBUILD. (You access this utility by choosing the BUILD option from the MAPPED menu, found under ROUTINES in the library utilities package.) This utility builds a file that contains selected precompiled DSM routines in a format that can be mapped into a virtual memory section. The ^%RBUILD utility is described in Section 7.4.2.12.

To map the file generated by this utility as a permanent global section, either for the group (if an application) or for the system (if a library), you have to install it in the system with the /INSTALL qualifier of the DSM command. You must have the privileges described in Section 5.10.1.2 to use this qualifier. Section 11.2.1 describes the /INSTALL qualifier and the procedure for installing permanent global sections.

To map the file generated by ^%RBUILD as a temporary global section (either group or system) you do not have to install it in the system, but you must have the privileges described in Section 5.10.1.2. If you have these privileges, you can create a temporary global section when you invoke the DSM image by issuing the DSM command in either of the following forms:

DSM/SHARED/MAPPED = file name/INSTALL

Creates DSM$ROUTINE_SEC and qualifies it with the group number of your UIC, then maps the specified file in it. The /SHARED qualifier instructs the system to map the file in DSM$ROUTINE_SEC. You should in general use the /INSTALL qualifier unless the section is private. You can also include /SYSTEM and /SECTION_NAME in this command line, as in the following command:

```
DSM/SHARED/MAPPED=NEWP/INSTALL/SYSTEM/SECTION_NAME=RADIO$SEC
```

DSM/SHARED/MAPPED = file name/LIBRARY/INSTALL

Creates DSM$LIBRARY_SEC and maps the specified file in it. DSM$LIBRARY_SEC is always created or mapped on a system-wide basis. The /SHARE qualifier (with the /LIBRARY qualifier) instructs the system to create this section for the file. The /INSTALL qualifier specifies that the section should be created as a permanent section.

NOTE

If you do not specify a file type in the file specification, VAX–11 DSM uses .VIR by default.

If you try to create a global section when either DSM$ROUTINE_SEC or DSM$LIBRARY_SEC already exists, VAX–11 DSM ignores the file specification, maps the existing section in your process's virtual address space, and reports a warning message to inform you that the file was not mapped.

To map explicitly to an existing global section, issue one of the following forms of the DSM command when you invoke the DSM image:

DSM/SHARED/MAPPED

Maps DSM$ROUTINE_SEC in your process's virtual address space. In this case, the /SHARED qualifier instructs the system to look for a group global section. VAX–11 DSM returns an error message if DSM$ROUTINE_SEC does not exist. You can also include /SYSTEM and /SECTION_NAME in this command line.

If you know the name of the starting routine in a mapped application, you can invoke VAX–11 DSM in Application Mode to map (and execute) the section; this also allows you to omit the /SHARED qualifier from the command line because the default in Application Mode is /SHARED.

DSM/SHARED/MAPPED/LIBRARY

Maps DSM$LIBRARY_SEC in your process's virtual address space. The /SHARED qualifier instructs the system to look for this system global section. If you want to execute a specific mapped library routine, you can invoke VAX–11 DSM in Application Mode. This allows you to omit /SHARED from the command line.

To create a private virtual memory section, use the /MAPPED command qualifier with the DSM command when you invoke the DSM image, and add a qualifier to the file name. In either of the forms shown below, this command causes VAX–11 DSM to call the Create and Map Section system service ($CRMPSC). This system service creates a virtual memory section in your process's virtual address space and maps the specified file to it.

DSM/MAPPED = file name

Creates a private section for a DSM application.

DSM/MAPPED = file name/LIBRARY

Creates a private section for a DSM library.

The system deletes all private virtual memory sections at DSM image rundown.

5.10.3 Running Mapped Routines

The procedure for executing an application or library routine from a virtual memory section is the same as the procedure for executing a routine that has not been mapped. Thus, to execute a mapped application or library routine, you simply specify the name of the routine you want to execute in the argument of the DO command, as follows:

>DO ^Routine Name

This command causes VAX–11 DSM to search DSM$ROUTINE_SEC or your private section for the specified routine name. If it is found, the interpreter executes the routine directly from virtual memory. If the routine name is not found, the system searches your default routine directory for the routine, and, if found, loads and executes it.

To execute a library routine from a virtual memory section, precede the routine name with the percent character (%), as follows:

>DO ^%Routine Name

This causes VAX–11 DSM to search DSM$LIBRARY_SEC (or the section specified with /SECTION_NAME) or your private section for the specified routine name stripped of the leading "%".

You can execute mapped and non-mapped routines during the same terminal session without restriction. VAX–11 DSM simply searches a virtual memory section for the presence of a routine before searching your default directory. However, you can map to only two DSM mapped routine sections simultaneously:

1. One mapped application, either private or shared (group or system)
2. One mapped library, either private or shared (system-wide)

To list mapped routines, issue the command D MAP^%RD for mapped sections or the command D ^%MAPCON to list the contents of a mapped file.

5.10.4 Optimization Considerations

The following DSM operations always require the presence of DSM source code:

- Executing the $TEXT function (unless the line starts with a double semicolon)
- Generating a DSM error message that indicates the line where the error occurred

Because mapped routine files consist of precompiled code only, VAX–11 DSM must be able to locate the source version of the routines in which these operations occur; that is, DSM must be able to locate the directory in which the routines are stored in source form.

When either of the preceding operations have to be performed, VAX–11 DSM locates and interprets the source code, and then continues to execute code from the mapped section. This process is totally transparent. However, it does affect the performance of a mapped application if these operations are performed frequently.

You can optimize the processing of $TEXT for an application by including a double semicolon (;;) at the beginning of lines that reference $TEXT. The double semicolon causes DSM to include the line in the precompiled code, and not to access the source code at run time.

Chapter 6
Input/Output Processing

This chapter describes the input and output (I/O) options for VAX–11 DSM. It provides information about:

- DSM terminal I/O
- DSM file-handling procedures
- Interprocess communication through mailboxes
- Networking

6.1 Overview of I/O Processing

The VAX–11 DSM I/O subsystem is the link between the DSM language and a subset of the I/O capabilities of VAX/VMS.

VAX–11 DSM performs device I/O for the following types of devices:

- Terminals
- Mailboxes
- Relative files
- Indexed files
- Sequential files
- Magnetic tapes

I/O to these classes of devices is discussed in Sections 6.5 through 6.11.

VAX–11 DSM uses the VAX/VMS system service Queue I/O (QIO) to handle terminals and mailboxes.

For file-structured devices (disks and magnetic tape), VAX–11 DSM uses the file-handling capabilities of VAX–11 RMS, the file and record access subsystem of the VAX/VMS operating system. Through DSM, you can manipulate all file types supported by VAX/VMS, including relative files, indexed files, sequential files, and sequential files on magnetic tape.

If your system is one of the nodes in a DECnet–VAX network, the DSM I/O subsystem also allows you to access and manipulate files on remote computers (resource sharing) and exchange data with routines on remote computers (task-to-task communication).

Although VAX–11 DSM defines the general device types listed above, DSM I/O operations are largely independent of physical devices. By using VAX/VMS logical names, you can associate any device or file specification with an arbitrary string. Thus, you can write a routine or application that does not explicitly refer to particular VAX/VMS devices or files; at run time, you can simply associate the logical names in your application with the file specification appropriate to your particular needs. If VAX–11 DSM encounters I/O directives that do not apply to the device being used, the system typically ignores the directives, instead of causing an error.

6.2 Assigning I/O Devices or Gaining Access to Files

To gain access to a file in a VAX/VMS directory, or to allocate and deallocate any device in the system other than your Principal I/O Device (defined below in Section 6.2.4), you use the DSM *assignment commands*:

OPEN — The OPEN command establishes ownership of the device indicated in its argument, or enables access to the file indicated. The OPEN command can also reserve a number of devices or files for a routine or application, if you include several device specifiers in its argument list. When you issue the OPEN command, it places a device in a process's pool of devices.

USE — The USE command makes one of a process's devices the current device,or establishes access to a file. DSM directs all I/O requests to the device specified in the argument of the USE command (until you issue another USE command). Thus, a routine cannot communicate with more than one device at a time.

DSM stores the device specifier of the current device or file in the $IO special variable, described in detail in Section 6.4. All I/O operations (such as READ or WRITE) are directed to the device whose device specifier is in $IO. You can reference $IO in any expression, but you can only change its value through the USE command. The full VAX/VMS device and file specification for the device or file is stored in $ZIO.

CLOSE The CLOSE command releases a device or file to the system and allowsit to be used by other system users. When a routine HALTs, or when you rundown the DSM image, all currently owned devices and files are automatically closed (unless output is pending, in which case the I/O operation completes before the system closes the device or file).

6.2.1 Assignment Command Syntax

The DSM assignment commands have the following general syntax:

OPEN device-specifier[:(parameter 1:...parameter n)][:timeout]

USE device-specifier[:(parameter 1:...parameter n)]

CLOSE device-specifier[:(parameter 1:...parameter n)]

Note that:

- Command elements in brackets are optional.
- All spaces and colons are required to separate the components of the command.
- Parentheses are required to delimit parameters if more than one parameter is specified; quotation marks are required to delimit the device specifier unless the device specifier is a variable or an expression, as described in Section 6.2.2.
- Parameters are device-specific. Refer to Sections 6.5 through 6.11 for lists of parameters for different kinds of devices.
- The timeout argument can only be used with the OPEN command. The timeout is an integer value that specifies the number of seconds VAX–11 DSM suspends execution until the requested device or file is available. If VAX–11 DSM cannot open the device or file in the specified period of time, it sets the $TEST special variable to 0 and resumes execution. If VAX–11 DSM can open the device or file during the specified period, it sets $TEST to 1 and resumes execution.

6.2.2 I/O Device Specifiers

To refer to a device in any DSM assignment command, you use a *device specifier*. A device specifier is any part of a VAX/VMS file specification, enclosed in quotes, or an expression evaluating to such a specification.

A DSM device specifier need not include all components of a VAX/VMS file specification. VAX–11 DSM defaults are applied to missing components, followed by VAX/VMS defaults. Logical names can replace any field or fields of a DSM device specifier.

A DSM device specifier can be any valid DSM variable or expression. If you use a variable or expression, you need not enclose the device specifier in quotes.

The following are the defaults applied to device specifiers:

Node = local node

Device = default VAX/VMS device

Directory = default VAX/VMS directory

File name = user defined

File type = .DAT

Examples:

The following examples assume 3 is a logical name and ABC is a variable whose value is the following string: [OWNER]INFO.DAT;1.

Device Specifier	Full File Specification
O "MTA0:TEST"	MTA0:[Default Directory]TEST.DAT;1
U "[CYGNUS.DSM]LOCAT"	Default Device:[CYGNUS.DSM]LOCAT .DAT;1
C "3"	Translation of the logical name 3
O "LZ002"	Default Device:[Default Directory] LZ002.DAT;1
U "DISK$USER:"	The device represented by logical name DISK$USER. File name null, type .DAT.
O ABC	Default Device:[OWNER]INFO.DAT;1

6.2.3 Device Recognition

When you allocate a device with the OPEN command, VAX–11 DSM recognizes the device type either by its device specifier or by an identifying parameter on the OPEN command itself.

VAX–11 DSM recognizes terminals and magnetic tape drives by their device characteristics.

VAX–11 DSM recognizes relative files and indexed files by the presence of the RELATIVE or INDEXED parameters on the OPEN command.

VAX–11 DSM recognizes a mailbox by the presence of the MAILBOX parameter on the OPEN command when you first create the mailbox. At any later point when you use the mailbox, VAX–11 DSM recognizes it by its device characteristics alone.

If a device cannot be classified in any of the previous categories, VAX–11 DSM recognizes the device as a sequential file. Thereafter, the device can *only* be accessed sequentially.

6.2.4 The Principal I/O Device

By convention, the terminal at which you log in is considered to be your *principal I/O device.* VAX–11 DSM interprets the following device specifiers to mean the principal device:

- 0
- The null string
- The value of the $IO special variable when at DSM command level

VAX–11 DSM equates your principal I/O device with the VAX/VMS process permanent logical names SYS$INPUT and SYS$OUTPUT. For interactive users, these logical names refer to your terminal.

You can establish a new default principal input and output device by using the /INPUT and /OUTPUT qualifiers with the DSM command. You can separate the input and output functions by specifying one device with /INPUT and another with /OUTPUT.

When you run DSM from a command procedure, the input stream is the file (file type .COM) that contains the procedure. In batch mode, the input stream is the batch command file; output goes to the log file.

6.3 I/O Commands

The following DSM commands allow you to perform all I/O operations:

ZPRINT	Writes the contents of the source routine buffer to the current device.
READ	Accepts data from the current input device and stores it in local variables.
READ *	Performs an unformatted read of one 8–bit binary character.
ZLOAD	Without arguments, loads a routine from the current I/O device to the source routine buffer.
WRITE	Writes variables and string literals to the current device or file.
WRITE *	Performs a physical write of one 8–bit binary character.
ZWRITE	Writes the contents of all or part of the local symbol table to the current file or device.

You can use the DSM I/O commands freely with any applicable device. You cannot, however, use them to perform operations that are inapplicable to a device, such as trying to READ from a line printer.

Section 6.5 through the end of this chapter provide device-specific information about the behavior of these commands.

6.4 I/O Special Variables

The following VAX–11 DSM special variables report device-specific information about the current I/O operation:

$IO
: Contains the DSM specification of the current I/O device for all devices except the principal device. For the principal device, $IO returns the fully parsed VAX/VMS device and file specification of SYS$INPUT.

$X
: Contains an integer value that indicates the horizontal cursor position on the current line. The value of $X can be between 0 and 65535 (16 bits); VAX–11 DSM resets $X to 0 when its value exceeds 65535 or when a new line (!) is written. If more than one DSM device specifier maps to the same device, all accesses to that device modify a common $X.

  READ * and WRITE * do not update $X.

$Y
: Contains an integer value that indicates the current line on the current page of the I/O device. The value of $Y, indicating the number of line feeds executed since the last form feed, can be between 0 and 65535. VAX–11 DSM resets $Y to 0 when its value exceeds 65535 or when you enter a form feed (^L). If more than one DSM device specifier maps to the same device, all accesses to that device modify a common $Y.

  READ * and WRITE * do not update $Y.

$ZA
: Contains device-dependent status or error information about the last I/O operation.

$ZB
: Contains device-dependent status or error information about the last I/O operation.

$ZC
: Contains a truth value that indicates whether a CTRL/C was typed on a terminal on which CTRL/C recognition is disabled. $ZC contains 1 if CTRL/C was typed; otherwise it contains 0. Testing $ZC resets it to 0; $ZC is also reset to 0 before each READ or READ * command, and whenever the DSM interpreter prompt appears. $ZC always contains 0 for terminals on which CTRL/C recognition is enabled, and for non-terminal devices.

$ZIO
: Contains the fully parsed VAX/VMS device and file specification of the current device. This specification is obtained from the DSM device specifier and the application of defaults. For the principal device, $ZIO contains the same string as $IO.

For the principal device only, the special variables $X and $Y can be modified through the cursor control functions, specified with the USE command. These functions are specified in the following forms:

U 0:X=n
U 0:Y=n
U 0:UPSCROLL
U 0:DOWNSCROLL

See Section 6.5.2.2 for more information on these functions.

6.5 Using Terminals

I/O to terminals uses the VAX/VMS QIO interface. QIO is a VAX/VMS system service, described in detail in the *VAX/VMS System Services Reference Manual.*

6.5.1 Setting Terminal Characteristics

Default terminal characteristics for a VAX/VMS system are defined at system generation time based on the most common type of terminal in use. You can examine these characteristics by issuing the DCL command SHOW TERMINAL. To change these characteristics for the duration of your process, you can issue the DCL command SET TERMINAL, in the following format:

SET TERMINAL[/Qualifier(s)]⏎>[device name]

The command can take the following qualifiers to indicate which terminal characteristics you want to set or clear:

/[NO]ECHO	/[NO]TAB	/VT52	/[NO]WRAP
/[NO]ESCAPE	/WIDTH=n	/VT55	
/[NO]FORM	/VT05	/VT100	

The argument of the SET TERMINAL command is the device name of the terminal whose characteristics you want to change. If you do not specify a device name, the current terminal device is assumed.

From VAX–11 DSM, you can change terminal characteristics by issuing the USE command with parameters. Most of these parameters change terminal characteristics for all subsequent reads and writes until your DSM image is run down. Some affect only the next operation. Some affect the terminal after rundown as well. Section 6.5.2.2 discusses the parameters in detail.

6.5.2 Terminal Commands

The VAX–11 DSM commands used for terminal I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE	ZLOAD	ZPRINT
		ZWRITE

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands. The command descriptions are followed by discussions of issues related to terminal I/O.

6.5.2.1 The OPEN Command — You rarely need to use the OPEN command for terminals, since nearly all terminal I/O is to the principal device. You generally need to use terminal OPEN for slave terminals and line printers.

For terminal I/O, the OPEN command accepts one parameter:

BLOCKSIZE=n — This parameter sets the size (in bytes) of the output buffer for the terminal indicated in the argument of the OPEN command. The output buffer size is the logical block size, which determines the maximum size of a single logical write operation. The maximum value of n depends on the value of the VAX/VMS sysgen parameter MAXBUF. If the parameter argument exceeds the value of MAXBUF, VAX–11 DSM processes your I/O requests in segments that are less than MAXBUF. Consult your system manager for the value of MAXBUF at your installation.

VAX–11 DSM ignores any attempt to open a device that is already open. Thus, the BLOCKSIZE parameter cannot change the buffer size of the principal device because VAX–11 DSM opens this device when you invoke the DSM image. To change the buffer size of the principal device, you must use the /TERMINAL_BUFFER_SIZE qualifier of the DSM command.

6.5.2.2 The USE Command — Parameters for the USE command allow you to set and clear various terminal characteristics. Once you set a particular characteristic with this command, it can be cleared only by issuing the USE command with the opposite form of the parameter appended to it.

The USE command parameters for terminals are:

Parameter	Function
CANCTLO	Disables CTRL/O recognition for the next WRITE operation only. VAX–11 DSM automatically reenables CTRL/O recognition for subsequent WRITEs.
[NO]CENABLE	Enables or disables CTRL/C recognition on the principal device. See Section 6.5.3 for more information about CTRL/C.
CLEARSCREEN	For the principal device only, clears the screen from the current cursor position to the end of the screen. See the description of the Y=n parameter below for an example of the use of CLEARSCREEN.
[NO]CONVERT	CONVERT changes lowercase characters to uppercase after a READ. NOCONVERT, the default, prevents conversion. Note that the CONVERT condition lasts only for the duration of the DSM image. At DSM rundown, the default condition is reestablished.
CTRAP=string	Establishes a set of trap characters for the principal device. The string specified can include any control characters from 0 to 31. If an application user enters any of the characters specified as trap characters, DSM generates the following error message: %DSM–E–CTRAP, character trap $C(n) received where n is the character entered. The trap occurs asynchronously, independent of pending READs on the principal device. You can enable trapping on control characters regardless of the enabling or disabling of CTRL/C. However, if you specify CTRAP=$C(3), and CTRL/C recognition is also enabled through /CENABLE, the character trap takes precedence. You can use CTRAP to enable CTRL/C trapping while /NOCENABLE is in effect. You clear character trapping by setting CTRAP to the null string (" ") or by setting it to another value.

Parameter	Function
CTRAP=string (Cont.)	The following example shows how CTRAP is used:

```
Q     U 0:CTRAP=$C(26,7)
      S $ZT="ERR^THISROUT"
^Z <<user input>>
ERR   ;
      I $ZE'[["CTRAP" ZQ
      U 0:CTRAP=" "
      W !,"Operation aborted",!
```

The first line of this example declares ^Z (ASCII 26) and ^G (the bell, ASCII 7) as trap characters. The second line sets $ZT. Execution continues until the user types a CTRL/Z. Then control transfers to ERR.

At ERR, the routine checks to see if this is a CTRAP error; if not, it resignals the error. For CTRAP errors, the routine disables character trapping and (in the last line) sends a message to inform the user that execution was interrupted.

DOWNSCROLL	For the principal device only, moves the cursor up and scrolls if necessary. Subtracts 1 from $Y unless $Y=0.
[NO]ECHO	Enables or disables the display of terminal input on the terminal. ECHO is the default. The NOECHO parameter inhibits terminal input from being displayed on interactive READ and READ * commands, that is, READ commands in the form:

READ "String Literal ",X

or

WRITE "String Literal " R X

The NOECHO condition lasts only for the duration of the DSM image; if you rundown DSM and then reinvoke the image, VAX–11 DSM reestablishes the default condition.

ERASELINE	For the principal device only, clears the current line from the cursor position to the end of the line.

Parameter	Function
[NO]ESCAPE	Enables or disables escape sequence processing for the duration of the image. NOESCAPE is the default. See Section 6.5.4 for more information about escape sequence processing.
FIELD=n	Sets the size (in bytes) of the terminal's input buffer. The default size of the input buffer is 255 bytes, the maximum length of a single DSM string. Specifying n=0 resets the default buffer size. Note that the READ X#n form of the READ command overrides the value specified in FIELD for the duration of a single READ.
TERMINATOR=string	Declares a set of terminators for READ commands. A terminator can be any control character from 0 to 31. The default terminators, RET (13) and ESC (27), are always recognized. If a null string is specified with TERMINATOR, all terminators except for the defaults are disallowed for the terminal. For example, the following command declares that CTRL/Z (26), form feed (12), and line feed (10) are terminators: `U 0:TERM=$C(12,10,26)` Once this command is in effect for a terminal, the string ABCLF is read by VAX–11 DSM as ABC. You cancel the effects of the USE command shown above by entering the following command: U 0:TERM="" Once this USE command takes effect, only <CR> and ESC are recognized as terminators.
UPSCROLL	For the principal device only, moves the cursor down and scrolls as necessary. Adds 1 to $Y.
WIDTH=n	Sets the right margin for the terminal to position n; the terminal driver inserts a carriage return/line feed after the nth character. Note that margins set with this parameter remain in effect for the duration of your process; DSM image rundown does not reset a margin to the default. Specifying n=0 sets the terminal characteristics to NOWRAP.

Parameter	Function
X=n	Sets the cursor on the screen and updates $X, for the principal device only. See the description of the Y=n parameter for an example of the use of X=n.

Y=n	Sets the cursor on the screen and updates $Y, for the principal device only.
	
	The following examples demonstrate the use of the Y=n, X=n, and CLEARSCREEN parameters. The first command moves the cursor to coordinate 40,10, clears the remainder of the screen, and sets $X and $Y:
	
	`U 0:(X=40:Y=10:CLEAR)`
	
	The next command moves the cursor to the "home" position (0,0), sets $X and $Y, but does not clear the screen:
	
	`U 0:(X=0:Y=0)`

6.5.2.3 The CLOSE Command — The CLOSE command releases a terminal to the system (after processing any pending output). You cannot CLOSE the principal I/O device, however. VAX–11 DSM automatically closes this device at DSM image rundown and ignores attempts to close the principal device through commands such as CLOSE $I or CLOSE 0.

6.5.2.4 The READ Command — The READ command causes VAX–11 DSM to read a string from the terminal. Keep the following points in mind when using the READ command for terminal I/O:

- A READ command in the following form purges typeahead and uses the VAX/VMS "read with prompt" feature:

  READ "String Literal",X

  READ without a literal never purges typeahead. If you specify /TYPEAHEAD with the DSM command, typeahead is never purged.

- A READ command can only be terminated by RET, ESC, or other terminators specified with the TERMINATOR parameter for the USE command. All other characters, including control characters not specified with TERMINATOR, are returned in the string read.

- If a READ command timeout expires during data entry, the variable returns the characters entered prior to the occurrence of the timeout.

- The smallest permissible timeout is 0.

- Each READ clears $ZC.

- Each read returns the terminator in the low byte of $ZB and the escape character in the high byte of $ZB, as described in Section 6.5.5.
- Each read affects the special variables $X and $Y, described in Section 6.4.

6.5.2.5 The READ * Command — The READ * command reads one binary character. During a READ * operation, VAX–11 DSM performs a limited interpretation of the character. It reads the character in 8–bit binary format and requests the Read QIO function IO$_READVBLK qualified by the NOFILTER function modifier (as described in the *VAX/VMS I/O User's Guide*). As a result, the terminal driver does not intercept CTRL/U, CTRL/R, and DELETE; they are passed to the terminal. (However, CTRL/Y, CTRL/C, and CTRL/O are intercepted.)

If a READ * command timeout expires during data entry, the character read is assumed to be -1.

The READ * command does not affect the special variables $X and $Y.

6.5.2.6 The WRITE Command — The WRITE command causes VAX–11 DSM to perform an asynchronous write of data to a terminal.

6.5.2.7 The Formatted WRITE Command — A formatted WRITE is a WRITE command with one of the form control characters as its argument. For terminal I/O, the formatted WRITE performs the following functions:

Command	Function
WRITE !	Writes a carriage return and line feed. Clears $X and adds 1 to $Y.
WRITE #	Clears the terminal screen of a recognized type of video terminal (that is, if one of the VT terminal characteristics is set). Otherwise, WRITE # writes a formfeed (binary 12). WRITE # clears $X and $Y.
WRITE ?n	Performs tabulation relative to $X. If a terminal has mechanical tabs (that is, if TAB is a terminal characteristic), DSM writes the appropriate number of tabs (8 spaces each) instead of spaces, to increase terminal output speed.

6.5.2.8 The WRITE * Command — The WRITE * command performs a physical write of one or more characters by putting the terminal in NOFORMAT mode for the duration of the operation. The WRITE * command accepts an integer argument and writes the ASCII character whose code is equivalent to the integer expression you use.

If the character is a control character, VAX–11 DSM performs the operation specified by the control character on the current device. For example, W *7 rings the terminal bell.

You can use WRITE *27 to introduce an escape sequence, but $C(27) is preferred because it generates fewer QIOs.

WRITE * does not affect the values of the special variables $X and $Y.

6.5.2.9 Optimizing Terminal Output — The WRITE command causes VAX–11 DSM to perform an asynchronous write of as much data as possible to a terminal. The maximum size of a single write operation depends on the default size of the sysgen parameter MAXBUF and the value specified in the argument of the /TERMINAL_BUFFER_SIZE qualifier of the DSM command. The default maximum size is 992.

The WRITE command for terminal I/O has different effects depending on the form of the command used. A WRITE command in the following form causes a single asynchronous write operation (if the total size is less than the buffer size):

```
WRITE A,!,B,!,C,!
```

A WRITE command in the following form causes three separate QIOs.

```
WRITE A,! WRITE B,! WRITE C,!
```

Switching from WRITE to WRITE * also causes a separate QIO. For example, the following command sequence causes three QIOs:

```
WRITE *7,"ABC",*27
```

Section 6.5.2.8 above describes the WRITE * command in detail.

6.5.3 CTRL/C and CTRL/Y Recognition

By default, VAX–11 DSM enables CTRL/C recognition in Programmer Mode and disables CTRL/C recognition in Application Mode and on slave terminals (allocated with the OPEN command).

However, you can explicitly enable or disable CTRL/C recognition with the /CENABLE qualifier of the DSM command, described in Section 4.4. You can also specify the [NO]CENABLE parameter on the DSM USE command (described above) to enable or disable CTRL/C recognition on the principal device. The DSM application can still test the value of $ZC if CTRL/C is disabled.

The VAX/VMS system manager can enable and disable CTRL/Y recognition on a user-by-user basis by means of the User Authorization File, described in the *VAX/VMS System Management and Operations Guide.*

Inhibiting CTRL/Y recognition is useful in an application environment where you do not want users to interrupt the execution of the application. If the system intercepts a CTRL/Y interrupt when CTRL/Y recognition is disabled, control does not pass to the command language interpreter.

VAX/VMS also allows you to enable and disable CTRL/Y recognition with the DCL SET NOCONTROL_Y command, described in the *VAX/VMS Command Language User's Guide.*

6.5.4 Escape Sequence Processing

Escape sequences are strings of two or more characters, beginning with the escape character (decimal 27), which indicate that control information follows. Escape sequences take two forms:

ESC character (any character except question mark)

ESC ?[??...] character

Terminals read and write escape sequences to enable special character sets, to set the position of a cursor, or to use function keys (for a READ). Refer to your terminal's user guide for terminal-specific information about escape sequence processing.

Keep the following points in mind when doing escape sequence processing from DSM:

- To enable escape sequence processing on a terminal, you must issue the USE command with the ESCAPE parameter for reading only. The ESCAPE parameter is not necessary for writing.
- Escape sequence processing uses the $ZB special variable. If an application user enters a valid escape sequence while escape sequence processing is enabled, the escape character is returned in the $ZB special variable. If a READ * is issued while escape sequence processing is enabled, the variable that you read contains 0; $ZB contains the escape character. In both of these cases, the format of the escape character in $ZB is:

  The terminator (27) in the low byte

  [(Character Code + 16 (decimal)) MODULO 64] in the high byte

  This format is compatible with other implementations of the DSM language.
- The escape character itself is always the last character in the terminal's input buffer, that is, the last character of the escape sequence. For example, the escape character of the escape sequence ESC P is "P". The escape character of the sequence ESC ? u is "u".

6.5.5 Terminal Status and Error Conditions

Terminal I/O status and error conditions are reported in the $ZA and $ZB special variables as shown in Table 6–1.

Table 6–1: $ZA and $ZB Assignments for Terminal I/O

Special Variable	I/O Operation Successful	I/O Operation Failed
$ZA	Physical length (in bytes) of last read or write.	Error message string
$ZB	Escape sequence processing enabled — terminator plus escape character entered on last read. Escape sequence processing disabled — terminator entered on last READ: 13 if RET, 27 if ESC, or other, if TERMINATOR is specified. If no terminator (last READ exceeds FIELD length, or READ x#n issued), value is 0.	0

6.6 Using Files

VAX–11 DSM provides a subset of the file-handling capabilities of VAX–11 RMS. You can create and manipulate all file types supported by RMS: sequential, relative, and indexed.

However, VAX–11 DSM imposes several restrictions on the record access methods that you can use for each file type. It also imposes some restrictions on the attributes of indexed files.

The following section describes the general organization of each file type. For a detailed description of file types and file access methods, refer to the *Introduction to VAX–11 Record Management Services.*

6.6.1 File Organization

The term *file organization* refers to the way records are logically arranged on a storage device. You specify a file's organization when you create the file; once you choose a file organization, you cannot change it later.

VAX–11 DSM supports three file organizations:

1. Sequential files

   Sequential files consist of records arranged in the sequence in which they are written to the file. A file with sequential organization can contain records of either fixed or variable length. Sequential file organization is permitted on all file-structured devices supported by VAX–11 DSM, including disks and magnetic tapes.

2. Indexed files

   Indexed files, permitted on disk devices only, consist of fixed- or variable-length records organized in a pyramid-like configuration of linked buckets that store pointer information and data records. Each bucket consists of a number of contiguous 512–byte physical blocks (see Chapter 9 for more information about the structure of indexed files). The indexed organization permits random insertion or retrieval of records based on a fixed-length key; a key is simply a contiguous string of unique characters. The location of a given record in an indexed file depends upon the sort order defined for the keys in the file.

3. Relative files

   Relative files, permitted on disk devices only, consist of numbered positions called cells. Cells are fixed in size, and are numbered consecutively from 1 to n, where 1 is the first cell and n is the last cell in the file. This arrangement lets you place records in the file according to cell number. Cell numbers are referred to by a relative record number, which specifies the location of a given record relative to the beginning of the file. There is no requirement, however, that every cell contain a record. Empty cells can be interspersed among cells that contain records. The relative file organization supports records that are either fixed or variable length.

6.6.2 File Access Methods

You can manipulate files using either of the following methods:

- Record I/O, most frequently used
- Block I/O, less frequently used

6.6.2.1 Record I/O — Record I/O allows you to access the logical records in a file by record access methods in these two general categories:

- Sequential access, most commonly used
- Random access

Sequential access to a file means that you can read a particular record only after reading all records preceding it. Thus, you can only write new records at the end of a file open for sequential access. All file types support sequential record access.

Random access to a file means that you can specify the order in which records are read or written. VAX–11 DSM supports three random record access methods:

Random by RFA (for sequential and indexed files only)	*Record File Address* (RFA) refers to an address that uniquely identifies the location of a record in a file. In file types that support random access by RFA, the RFA remains constant while the record remains in the file. VAX–11 DSM returns the RFA to you in the $ZB special variable in the form "n,m" where n is the number of the block that contains the record, and m is the byte offset into that block, indicating where the record actually begins.
Random by Primary or Alternate Key	In random access by key, you qualify your read requests with the key value of the record. VAX–11 DSM supports random access by primary or alternate key value. When you access a record by key, VAX–11 RMS searches for the corresponding key in the index and reads the record that the key points to.
Random by Relative Record Number (for relative files only)	The Relative Record Number is an identification number that indicates the position of a record relative to the beginning of the file. Random access by Relative Record Number is only supported on files with relative organization. VAX–11 DSM returns the Relative Record Number to you in the $ZB special variable as a positive integer.

Each file organization supports several record access methods. Table 6–2 shows the record access methods that VAX–11 DSM supports for each file organization.

Table 6–2: VAX–11 DSM Record Access Methods

File Organization	Access Methods
Sequential	Sequential Random by RFA
Indexed	Sequential Random by Primary or Alternate Key Random by RFA
Relative	Sequential Random by Relative Record Number

6.6.2.2 Block I/O — Block I/O allows you to access and manipulate the blocks (physical records) that make up a file, rather than its logical records. All file types support block I/O. For disk-resident files, you can specify the size of the block you want to read in an OPEN command parameter (BLOCKSIZE = n); this block size must be a multiple of 512. For sequential files on magnetic tape, you can specify any block size.

Performing block I/O on any disk-resident file overrides the disk's logical organization. As a result, I/O must proceed as if the file were a disk-resident sequential file.

6.6.3 Creating and Opening Files

For sequential, indexed, and relative files, the DSM OPEN command takes two parameters that determine whether a new file should be created: NEWVERSION and READONLY.

If you issue the OPEN command (followed by a device specifier) with no parameters, VAX–11 DSM opens a file on your default disk device. If the file does not exist, VAX–11 DSM creates it. (This is not true for files on magnetic tape, for which READONLY is the default.)

For example, the following command line creates a sequential file named REDLINE.DAT:

```
>O "REDLINE"
```

If REDLINE.DAT already exists, this command opens the file. You can manipulate its contents with any I/O command.

If you include the NEWVERSION parameter, as in the next example, the OPEN command always creates a new file, even if there is an existing file with the same name:

```
>O "SCRATCH":NEWVERSION
```

If you include the READONLY parameter (as in the following example), a new file is never created. An error occurs if the file named does not already exist.

```
>O "LIST":READONLY
```

If the device specifier includes a VAX/VMS device other than your default device, VAX–11 DSM creates or opens the file on the device specified.

6.6.4 Positioning Files

After you open a file, or after you issue the USE command with the DISCONNECT parameter, the position of the file is undefined. The repositioning of the file upon the next I/O operation follows these rules:

- If the next I/O operation after the OPEN or USE x:DISCONNECT is a READ, the file is positioned at the beginning.
- If the next I/O operation after the OPEN or USE x:DISCONNECT is a WRITE, the file is positioned at the end.

6.6.5 Reading and Writing Records

The WRITE command, described in detail under each type of file, packs a buffer. Followed by one exclamation point, WRITE writes the current record to a file. Followed by two exclamation points (!!), it writes a null record.

The following example shows how WRITE ! is used:

```
> O "FILE":NEWVERSION
> U "FILE" W "ABC"
> U "FILE" W "DEF ,! W !,"XXXXX",!
> U "FILE":DISCONNECT R X,Y,Z
> W X,!,Y,!,Z,!
ABCDEF
(null)
XXXXX
```

6.6.6 File Sharing

If the /NOSHARED qualifier is in effect implicitly or explicitly, the default behavior of DSM is that no files are shared. However, the SHARED option on the OPEN command can be used to specify that a particular file should be shared.

If /SHARED is in effect implicitly or explicitly, indexed and relative files are shared by default, but sequential files are never shared. To specify that a sequential file should be shared, you must use the SHARED parameter with the OPEN command. However, sharing sequential files has some drawbacks, as described in the *VAX–11 Record Management Services Reference Guide.*

6.7 Sequential Files on Disk

The following sections describe I/O to sequential files on disk devices. Sequential files on magnetic tape are described in Section 6.8.

6.7.1 Sequential File Commands

The commands used for sequential file I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE	ZLOAD	ZPRINT
		ZWRITE

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands. The command descriptions are followed by an explanation of the status and error conditions returned for sequential file I/O.

6.7.1.1 The OPEN Command — The OPEN command reserves a sequential file on disk for use by your process, or creates a sequential file if the device specifier indicated in its argument does not exist.

Parameters for the OPEN command allow you to set various characteristics of the file you create, such as its protection mask. They also allow you to establish certain conditions for the current I/O operation, such as making it READ only.

For sequential file I/O, you use the OPEN command timeout parameter to set a limit on how long VAX–11 DSM tries to open a file if the file has already been opened by another user with sharing options that conflict with yours. Ordinarily, VAX–11 DSM tries to open the file at one-second intervals until the timeout expires. If you try to open the file without a timeout on the OPEN command, VAX–11 DSM continues trying to open the file.

Table 6–3 summarizes the OPEN command parameters for sequential file I/O. All parameters refer to VAX–11 RMS options and are described in detail in the *VAX–11 Record Management Services Reference Manual.*

Table 6–3: OPEN Command Parameters for Sequential File I/O

Parameter	Function
ALLOCATION=n	Specifies the amount of space, in blocks, to allocate to the file when you create it.
BLOCKSIZE=n	Requests block I/O and specifies the buffer size (in bytes); n must be positive.
CONTIGUOUSBESTRY	Allocates the file as contiguously as possible on the disk.
DELETE*	Deletes the current version of the file when you close it or at DSM image rundown.
EXTENSION=n	Specifies the number of blocks to add to the file when the current space is exhausted.
NEWVERSION	Explicitly creates a file.
NOSEQUENTIAL	Enables random access to the records in the file by RFA.
PROTECTION=opts*	Sets the protection that applies to the file.
READONLY	Opens an existing file for reading only.
RECORDSIZE=n	Specifies the maximum record size used when a file is created.
SHARED	Enables file sharing.
SPOOL*	Sends the file to the spool file of the default print queue (SYS$PRINT) when you close it or at DSM image rundown.
SUBMIT*	Submits the file to the default batch queue (SYS$BATCH) when you close it or at DSM image rundown.
UIC="[n,m]"	Establishes the owner UIC for the file when you create it.

* Indicates the parameter is also available on CLOSE.

The ALLOCATION=n parameter accepts a numeric argument that represents a number of disk blocks, so it can only be used for disk files. The number n is the amount of disk space allotted to the file when it is created; n must be in the range of 0 to 4294967295. A value of 0 indicates no allocation. This parameter is equivalent to the RMS ALQ option.

The BLOCKSIZE=n parameter enables block I/O operations on the specified file by creating fixed-length blocks. This parameter accepts a numeric argument (greater than 0) that specifies the size of the buffer (in bytes). The buffer size determines the maximum size of a single write or read. For disk devices, the buffer size must be a multiple of 512. For magnetic tape, you can select any buffer size. See Section 6.8 for more information about block I/O on sequential files. If you create the file, n also determines the size of the file's fixed-length blocks.

The CONTIGUOUSBESTRY parameter instructs VAX–11 DSM to allocate the file contiguously on a "best effort" basis. This parameter is equivalent to the RMS CBT option. It is valid for disk files only.

The EXTENSION=n parameter accepts a numeric argument that represents a number of disk blocks. It is valid for disk files only. The value of n determines the number of blocks added to a file when its initial allocation of blocks is exhausted. This number must be in the range of 0 to 65535. If you specify an extension of 0, the file is extended to the VAX–11 RMS default extension value. This parameter is equivalent to the RMS DEQ option.

NOTE

Automatic extension occurs whenever you perform a write operation on a file that has used all of its initial allocation of disk space.

The NEWVERSION parameter creates a new file. A previous version of the file may or may not exist. NEWVERSION is required for writing to magnetic tape, but you can use it for disks as well.

The NOSEQUENTIAL parameter enables random access to a sequential file by Record File Address (RFA). You must include this parameter in the OPEN command if you want to access records in a file by RFA. Specifying this parameter clears the SQO bit in the RMS File Access Block (FAB). The SQO bit is set in the RMS FOP parameter.

The PROTECTION=opts parameter sets the protection that applies to the file. The PROTECTION parameter keywords are identical to the keywords used with the DCL SET PROTECTION command. User categories are: GROUP, WORLD, SYSTEM, OWNER. You can specify READ (R), WRITE (W), EXECUTE (E), or DELETE (D) protection for each user category. Unlike the SET PROTECTION command syntax, however, each user category must be separated from the protection codes by an equal sign (=). You must use the following syntax to specify no protection:

```
>O "File name":PROTECTION=(W=)
```

In all other respects, the PROTECTION parameter syntax is the same as the SET PROTECTION command syntax.

```
>O "File name":PROTECTION=(GROUP=RWED,WORLD=R)

>O "File name":PROTECTION=(SYSTEM=RWED,GROUP=R,WORLD=RE)
```

The READONLY parameter overrides the default VAX–11 DSM access mode for sequential files. By default, VAX–11 DSM opens a sequential file with GET and PUT privileges, equivalent to READ/WRITE access to the file. The READONLY parameter opens a file with GET privilege only, equivalent to READ access; all you can do to the file is read it. READONLY is the default behavior for magnetic tape files.

If READONLY is present (either explicitly or implicitly), an error occurs if you try to open a file that does not exist. A new file is never created in response to OPEN if READONLY is present.

The RECORDSIZE parameter specifies the maximum size of a record. This size, set only when the file is created, corresponds to the RMS option MRS. If you do not specify RECORDSIZE, the default is 255, the size of a DSM string.

The SHARED parameter enables shared access to a sequential file. The SHARED parameter is equivalent to the RMS SHR options. RMS support for sharing sequential files is limited (and in general not recommended). See the *VAX–11 Record Management Services Record Manual* for a description of these limitations.

The NOSHARED parameter disables the RMS shared file option. This parameter opens a file with the RMS access privilege NIL only, so that all explicit file sharing is disabled.

You use READONLY and BLOCKSIZE in conjunction with SHARED or NOSHARED to specify a file-sharing mask for a file. See Section 6.10.1.1 for details on this operation, which is more appropriate for indexed and relative files than for sequential files.

The UIC="[n,m]" parameter establishes the owner UIC of the file when you create it. You must have sufficient VAX/VMS privileges to set the UIC to one other than your own.

6.7.1.2 The USE Command — Parameters for the USE command allow you to perform record and block I/O operations on the file specified in its device specifier field. Table 6–4 summarizes the USE command parameters for sequential file I/O. The paragraphs following this table describe each parameter in detail.

Table 6–4: USE Command Parameters for Sequential File I/O

Parameter	Function
[NO]CONVERT	Converts lowercase characters to uppercase on READ, or disables lowercase conversion (default).
DISCONNECT*	Positions the file at the beginning or end depending on the next operation.
RFA="n,m"	Positions the file to the specified Record File Address.
SPACE=n*	Positions the file n blocks forward or backward for block I/O operations.

* Indicates a VAX–11 RMS option.

The CONVERT parameter instructs VAX–11 DSM to convert lowercase characters to uppercase characters after a READ. NOCONVERT (the default) instructs the system not to convert lowercase characters to uppercase.

The DISCONNECT parameter positions the file at the beginning if the next operation is READ. It positions the file at the end if the next operation is WRITE. The DISCONNECT parameter is equivalent to the RMS $DISCONNECT macro. You need not specify DISCONNECT immediately after OPEN.

The RFA parameter allows you to read or write records in the file by their Record File Address (provided you have included the NOSEQUENTIAL parameter on the OPEN command). However, you can only write a record to a Record File Address if it is *exactly* the same length as the record that currently occupies that address. The parameter argument "n,m" represents the block and offset: n specifies the block number that contains the desired record, and m specifies the byte offset into that block that indicates where the record actually begins. A typical RFA specification is "2,478".

To get the RFA for each record in the file, you must initially read or write the entire contents of the file sequentially and save the value of the $ZB special variable after each operation; for sequential file I/O, $ZB returns the RFA after each successful read or write, as described in Section 6.7.2. Once you save the RFA for a record, you can access it either by specifying the RFA as a block and offset, as shown above, or by specifying the variable in which you saved the RFA. Figure 6–1 shows a routine that creates a sequential file consisting of 100 records. This routine simultaneously saves the value of $ZB for each record in a local array.

Figure 6–1: Routine to Retrieve Records by RFA

```
    Q ;Demonstrate RFA on sequential files
    O "A":(NOSEQ:NEWVERSION)
    F I=1:1:100 U "A" W "RECORD"_I,! S A(I)=$ZA,B(I)=$ZB
    U 0 W !,"Now, retrieve records",!
ACC U "A":RFA=B(29) R X U 0 W !,X,!,"should be RECORD29",!
    U "A":RFA=B(60) W $E("XXXXXXXXXXXXXXXXXXXXXXXXXXXX",1,A(60)),!
    U "A":RFA=B(50) R X U 0 W X,!,"should be RECORD50",!
    U "A":RFA=B(60) R X U 0 W X,!,"should be all X's !!",!
    U 0 W !!,"End...",!
```

In Figure 6–1, routine lines ACC to ACC+3 perform the following tasks:

- ACC reads the record in the RFA specified by the variable B(29) and writes it to the principal device.
- ACC+1 writes a record consisting of Xs to the RFA specified by the variable B(60). The $EXTRACT statement makes sure that the length of the record matches the length of the record that currently occupies this RFA.
- ACC+2 reads the record in the RFA specified by the variable B(50) and writes it to the principal device.
- ACC+3 reads the record that replaced record 60 and writes it to the principal device; this record consists of eight Xs.

The SPACE=n parameter, equivalent to the RMS $SPACE, positions a file n blocks forward or backward for block I/O operations. The argument n can be a positive or negative integer, where negative specifies backwards positioning. You can only use this parameter if you have included the BLOCKSIZE=n parameter on the OPEN command, unless your first operation on the file is block I/O. Specifying the SPACE parameter automatically sets the access mode to block I/O.

6.7.1.3 The CLOSE Command — Parameters for the CLOSE command allow you to perform various exit options, such as deleting the file or sending it to the line printer. Several CLOSE parameters can also be used on the OPEN command. If you specify one of these parameters when you open the file, the parameter takes effect automatically when you close the file. You do not need need to respecify the parameter when you close the file.

Table 6–5 summarizes the CLOSE command parameters for sequential file I/O. The paragraphs that follow describe the parameters in greater detail.

Table 6–5: CLOSE Command Parameters for Sequential File I/O

Parameter	Function
DELETE*	Deletes the current version of the file
PROTECTION=opts*	Sets the protection that applies to the file
QUEUE=q	Sends the file to a queue other than SYS$PRINT or SYS$BATCH
RENAME=name	Renames an existing file to the name specified in the parameter argument
SPOOL*	Sends the file to the spool file of the default print queue (SYS$PRINT)
SUBMIT*	Submits a file to the default batch queue (SYS$BATCH) for batch processing
UIC="[n,m]"	Establishes the owner UIC for an existing file

* Indicates the parameter is available on the OPEN command.

The DELETE parameter explicitly deletes the current version of the file. If you create a file but do not write anything to it, VAX–11 DSM deletes the file automatically when you close it. If you use this parameter along with the SPOOL or SUBMIT parameters, VAX–11 DSM deletes the file after it has been processed.

The PROTECTION=opts parameter sets the protection that applies to the file, as discussed in the description of this parameter for the OPEN command.

The QUEUE=q parameter sends a file to a queue other than the default print or batch queues. The parameter argument q specifies the name of the queue to which you want to direct the file. You use this parameter with the SPOOL and SUBMIT parameters, as shown in the following examples:

C "File name":(SPOOL:QUEUE=queue name)

C "File name":(SUBMIT:QUEUE=queue name)

The RENAME=name parameter changes the name of an existing file to the name specified in the parameter argument. The parameter argument name can include any part of a file specification. If you omit fields from the file specification, VAX–11 DSM constructs a full specification by using all fields included in the original file specification and applying VAX/VMS defaults.

The SPOOL parameter sends the file to the spool file of the default print queue. The default print queue is the translation of the logical name SYS$PRINT. If you want to direct the file to a queue other than SYS$PRINT, you must include the QUEUE=q parameter in the CLOSE command in addition to SPOOL.

The SUBMIT parameter sends the file to the default batch queue for batch processing. The default batch queue is the translation of the logical name SYS$BATCH. If you want to direct the file to a queue other than SYS$BATCH, you must include the QUEUE=q parameter in the CLOSE command in addition to SUBMIT.

The UIC="[n,m]" parameter changes the owner UIC of an existing file. You must have sufficient VAX/VMS privileges to change the file owner UIC with this parameter.

6.7.1.4 The READ Command — For record I/O operations, the READ command reads a record from the file into a variable as a DSM string. The maximum record size that VAX–11 DSM can read is 255 bytes; this is the default input buffer size and the maximum length of a single DSM string. If you try to read a record that is larger than 255 characters, VAX–11 DSM returns the record to you in pieces in subsequent READs, for example:

```
>U "File" R X S A(1)=$ZA ;$ZA returns length of record
>R Y S A(2)=$ZA
```

In this example, if the record read is larger than 255 characters, READ X only returns the first 255 characters. If A(1)=256 (that is, the value of $ZA=256), then the length of Y is 1 and the value of A(2) is 0. If A(2) is 0, READ Y did not return a physical record; instead it returned the remaining portion of record X.

For block I/O, the READ command returns a 255–byte binary string. If you try to read a record larger than 255 characters, VAX–11 DSM returns the record to you in pieces in subsequent READs.

You can use a timeout parameter on the READ command, but it is ignored for sequential I/O.

6.7.1.5 The READ * Command — For record I/O, the READ * command reads one character from a record in 8–bit binary format. To retrieve an entire record, you must issue as many READ * commands as there are characters in the record.

For block I/O, READ * retrieves one binary character from a physical record (block).

VAX–11 DSM ignores the timeout parameter on the READ * command.

6.7.1.6 The WRITE Command — The WRITE command writes a DSM string to the file's output buffer. When the output buffer becomes full, or if you execute a formatted WRITE (as described in Section 6.7.1.8), VAX–11 DSM writes the contents of the buffer to the file as a record. When you write consecutive strings to the buffer, they are concatenated into a single record until the length exceeds 255 bytes; then the system automatically creates a new record. (Thus, a single READ actually returns a concatenated string.)

VAX–11 DSM uses the RMS RAT=CR option for carriage control. When a sequential file is written to a carriage control device, such as a terminal or line printer, this option instructs the system to perform a line feed/carriage return after writing each record to the device. This information is not, however, stored as data in the file itself. The RMS RAT=CR option corresponds to the FORTRAN "List" format.

6.7.1.7 The WRITE * Command — The WRITE * command writes a single character to the file's output buffer in 8–bit binary format.

6.7.1.8 The Formatted WRITE Command — For sequential file I/O, the formatted WRITE performs the following functions:

Command	Function
WRITE !	Writes the current record to the file. Two consecutive carriage return/line feeds (!!) write a null record to the file.
WRITE #	Writes the current record to the file plus a record that contains a form feed.
WRITE ?n	Writes the specified number of spaces (relative to $X) to the current record buffer.

6.7.1.9 The ZPRINT Command — The ZPRINT command transfers the contents of the routine buffer to the file. ZPRINT causes the entire contents of the buffer to be written as one record per source line of code. To separate multiple routines in the file from one another, you must explicitly write a carriage return/line feed (!) after each routine. This action writes a null record (blank line) to the file.

6.7.1.10 The ZLOAD Command — The ZLOAD command transfers a routine from a sequential file to the routine buffer. ZLOAD loads records until it encounters either of the following:

- A blank line
- The end-of-file

If ZLOAD encounters an end-of-file, it causes an error and returns an empty buffer.

6.7.2 Sequential File Status and Error Conditions

Sequential file I/O status and error conditions are reported in the $ZA and $ZB special variables as shown in Table 6–6. Note that, for READ and WRITE, $ZA and $ZB values apply to the current device and are stored in the access context of the file.

Table 6–6: $ZA and $ZB Assignments for Sequential File I/O

Special Variable	I/O Operation Successful	I/O Operation Failed
$ZA for READ or WRITE	Length (in bytes) of last record read or written	Error message string corresponding to the primary RMS error status (RMS STS value)
$ZB for READ or WRITE	RFA of the last record read or written	Error message string corresponding to the secondary RMS error status (RMS STV value) If no STV value, returns the null string.
$ZA after OPEN	Maximum record size (RMS MRS status)	
$ZB after OPEN	0	

6.8 Sequential Files on Magnetic Tape

With certain operational restrictions, VAX–11 DSM supports most magnetic tape options provided by VAX/VMS. VAX/VMS uses the magnetic tape structure defined by the American National Standards Institute standard ANSI X3.27–1978. Magnetic tapes must be ANSI labeled and coded in ASCII format.

VAX–11 DSM can process nonstandard tapes (such as DOS). However, you must mount such tapes at VAX/VMS command level with the /FOREIGN qualifier (or through $ZCALL), and you must not access the information on the tape in block I/O mode. VAX/VMS uses 9–track tape drives only. Refer to the *VAX–11 Magnetic Tape User's Guide* for information about labeling and file-structuring format.

VAX–11 DSM supports the following kinds of magnetic tape operations:

- Record I/O on file-structured tapes (in nearly all cases)
- Block I/O on file-structured tapes
- Block I/O on non-file-structured tapes

6.8.1 Magnetic Tape Operations

VAX–11 DSM imposes the following restrictions on magnetic tape operations:

1. You must initialize magnetic tapes at VAX/VMS command level with the DCL INITIALIZE command.
2. You can mount magnetic tapes either at VAX/VMS command level, with the DCL MOUNT command, prior to invoking the DSM image, or with the $ZCALL (%MOUNT) and (%DISMOUNT) facilities from DSM. The only time that you do not need to explicitly mount the tape is when you want to request the next volume of a volume set. A parameter on the USE command allows you to set automatic mounting of the next volume.

See the *VAX/VMS Command Language User's Guide* for a complete description of the INITIALIZE and MOUNT commands and the mount and initialize procedures. See the description of $ZCALL for instructions on using %MOUNT and %DISMOUNT.

You can position magnetic tapes directly from the DSM language, however. Parameters on the assignment commands, described later in this chapter, allow you to rewind a tape file-by-file or rewind the entire tape. They also allow you to position the tape forward or backward by block-for-block I/O operations.

6.8.2 Magnetic Tape Access Modes

Magnetic tapes support sequential access only. Only one user can have access to a given volume set at any one time, and only one file in the volume set can be open for processing at a time.

You can access either file-structured or non-file-structured tapes from the DSM language, as described in the following sections.

6.8.2.1 Accessing File–Structured Tapes — To access a file-structured tape, you must first mount the tape at VAX/VMS command level (or through $ZCALL). Thereafter, I/O is very similar to I/O for sequential files on disk.

If you issue the OPEN command with a device specifier that includes a VAX/VMS magnetic tape device and a file name, VAX–11 DSM opens the specified file for reading only (READONLY is the default); then you can manipulate the file's contents with any I/O command, for example:

```
>O "TAPENAME:File name.DAT"
> "TAPENAME:File name.DAT" R RECORD
```

To create a new sequential file on tape, you append the NEWVERSION parameter to the OPEN command, as described in Section 6.8.4.1. The following example shows how a file is created and a record is written to it:

```
$ MOUNT MTO:  MYTAPE
$ DSM
     VAX-11 DSM Version 2.0
>"MTO:MYFILE.DAT":NEWVERSION
>U "MTO:MYFILE.DAT" W "RECORD",!
```

File-structured tapes support both record and block I/O.

6.8.2.2 Accessing Non–File–Structured Tapes — To access a non-file-structured tape, you must first mount the tape at VAX/VMS command level (or through $ZCALL) with the /FOREIGN qualifier appended to the MOUNT command. This qualifier indicates that the tape is not in the standard ANSI format used by the VAX/VMS operating system. Remember that if you mount a tape with the /FOREIGN qualifier, your routines must be able to process the labels on the volume.

After you mount the tape and invoke the DSM image, you issue the OPEN command with a device specifier that includes a VAX/VMS tape device name only. Since non-file-structured tapes support block I/O only, you must also include the BLOCKSIZE=n parameter, where n is larger than the largest block that you want to read on that tape. For example:

```
>O "TAPENAME":BLOCKSIZE=4096
```

6.8.3 Magnetic Tape Status and Error Conditions

Status and error conditions for magnetic tape are returned in $ZA and $ZB in the same forms as status and error conditions for sequential files on disk, described in Section 6.7.2.

6.8.4 Magnetic Tape Commands

The commands used for magnetic tape I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE	ZLOAD	ZPRINT
		ZWRITE

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands.

6.8.4.1 The OPEN Command — For file-structured tapes, the OPEN command's behavior is similar to its behavior for disk-resident sequential files. That is, the OPEN command either opens or creates a sequential file on the device specified in the device specifier field.

For non-file-structured tapes, the OPEN command establishes a sequential input or output stream for block I/O operations only.

Table 6–7 summarizes the OPEN command parameters for magnetic tape I/O. The paragraphs following this table describe each parameter in detail.

Table 6–7: OPEN Command Parameters for Magnetic Tape I/O

Parameter	Function
BLOCKSIZE = n	Requests block I/O and specifies the block size (in bytes); for existing files, n must be greater than or equal to the largest block size on the tape. The VAX/VMS default is 2048 bytes; the DSM–11 default is 1024.
NEWVERSION	Creates a new file.
READONLY	Opens a file for reading only (default).
RECORDSIZE = n	Specifies the maximum record size (default is 255).
REWIND	Rewinds the tape; positioning depends on whether access is to a file-structured or non-file-structured tape.

The BLOCKSIZE = n parameter enables block I/O operations on the specified file-structured or non-file-structured tape. When you create a file, the parameter argument n specifies the size of one tape block in bytes. When you read an existing file, the parameter argument should be larger than the largest block on the tape.

The NEWVERSION parameter creates a new sequential file on the specified tape drive. For example:

```
>O "MTA0:File name":NEWVERSION
```

This command line creates a new sequential file on the tape mounted on drive MTA0. The name of the file is file name.DAT;1.

The READONLY parameter opens an existing file for reading only. Unlike disk-resident sequential files, magnetic tape sequential files are opened read-only by default. Thus, this parameter never has to be explicitly specified.

The RECORDSIZE = n parameter specifies the maximum record size. The default is 255 bytes. This parameter is equivalent to the RMS option MRS. The REWIND parameter positions a file-structured tape at the beginning of the current file. It positions a non-file-structured tape at the beginning of the tape (that is, it rewinds the entire tape).

6.8.4.2 The USE Command — Parameters for the USE command allow you to perform record or block I/O operations on either file-structured or non-file-structured tapes. The parameters for this command specify RMS options. Table 6–8 summarizes the USE command parameters for magnetic tape I/O. The following paragraphs describe them in detail. Refer to the *VAX–11 Record Management Services Reference Manual* for additional information about these parameters.

Table 6–8: USE Command Parameters for Magnetic Tape I/O

Parameter	Function
DISCONNECT	Positions the tape depending on the next operation and whether access is to a file-structured or non-file-structured tape.
NEXT	Requests the next volume in a volume set.
SPACE=n	Positions the tape forward or backward n blocks for block I/O operations.

For file-structured tapes, the DISCONNECT parameter positions the tape at the beginning of the file if the next operation is READ. If the next operation is WRITE, the tape is positioned at the end of the file.

For non-file-structured tapes, the DISCONNECT parameter positions the tape at the beginning.

The NEXT parameter requests the next volume in a volume set to be mounted. For input, this parameter causes the following to occur:

- If the current volume is the last volume in the set, the system reports end-of-file.
- If another file section exists (a part of a file that physically spans tapes), the next volume is mounted. When necessary, the current volume is rewound, and a request to mount the next volume is sent to the operator.

For output, this parameter causes the following to occur:

- The file section on the current volume is closed with the appropriate end-of-volume labels, and the volume is rewound.
- The next volume is mounted.
- A file with the same file name and the next higher file section number is opened for output, and processing continues.

This parameter is equivalent to the RMS $NXTVOL macro.

The SPACE=n parameter positions the tape n blocks forward or backward for block I/O operations. The parameter argument n can be a positive or negative integer; a negative integer specifies backwards positioning. For input, you can use this parameter only if you have included the BLOCKSIZE=n parameter on the OPEN command. This parameter is equivalent to the RMS $SPACE macro.

6.8.4.3 The CLOSE Command — For magnetic tape I/O, the CLOSE command accepts one parameter, REWIND. This parameter positions a file-structured tape at the beginning of the current file. It positions a non-file-structured tape at the beginning of the tape.

6.8.4.4 The READ and WRITE Commands — Except for random read by RFA and file sharing, the READ and WRITE commands behave for magnetic tape exactly as they behave for sequential files on disk. The READ *, WRITE *, and Formatted WRITE commands behave exactly as they behave for sequential files on disk.

See Sections 6.7.1.4 through 6.7.1.8 for information on how these commands work for sequential files on disks.

6.9 Indexed Files

VAX–11 DSM provides a subset of the options supported by VAX–11 RMS for indexed sequential file (ISAM) organization. These options include:

- Random record access by primary or alternate key
- Random record access by RFA
- Sequential access to the file
- File sharing
- Automatic record interlocking

To create an indexed file, issue the OPEN command followed by a device specifier and the INDEXED and NEWVERSION parameters. To open a file, used INDEXED. If the file does not exist, VAX–11 DSM creates it, as described in Section 6.6.3. If the file does exist, VAX–11 DSM opens it. For example:

```
>O "GREENLINE":INDEXED
```

This command line opens or creates the indexed file GREENLINE.DAT on your default disk device.

When you create an indexed file from the DSM language, VAX–11 DSM allows you to specify the following file parameters: KEYSIZE (see Section 6.10.1.1 for details), ALLOCATION, EXTENSION, CONTIGUOUSBESTRY, and RECORDSIZE. All other file parameters are fixed by DSM and cannot be changed, except with the RMS utility CREATE/FDL, described in Section 9.7.2, or the $ZCALL %FDLCREATE.

The following are the default parameters for indexed files created from the DSM language:

- Key position = 0
- Key size = 64

- Definition of the primary key only (no alternate keys)
- No duplicate keys allowed
- ASCII collating sequence
- Maximum record size = maximum size of a DSM string (255) + key size
- Bucket size = 2
- One area

VAX–11 DSM allows you to open an existing indexed file without the INDEXED parameter. However, if you do so, the file can only be processed sequentially. That is, you must process the file as if it were a disk-resident sequential file. Opening an indexed file without the INDEXED parameter disables all random record access and record locking.

6.9.1 Indexed File Commands

The commands used for indexed file I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE		

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands. The command descriptions are followed by sections explaining record locking and error conditions.

6.9.1.1 The OPEN Command — When specified with the INDEXED parameter, the OPEN command reserves an indexed file for use by your process or creates an indexed file. If specified without the INDEXED parameter, the OPEN command establishes a sequential input/output stream from an indexed file.

Many of the OPEN command parameters specify VAX–11 RMS options. The following list shows the OPEN parameters for indexed file I/O; an asterisk (*) indicates that the parameter specifies an RMS option.

ALLOCATION = n*
CONTIGUOUSBESTRY*
DELETE
EXTENSION = n*
KEYSIZE = n
PROTECTION = opts
READONLY
RECORDSIZE = n*
[NO]SHARED
NEWVERSION
UIC = "[n,m]"

The READONLY parameter overrides the default VAX–11 DSM access mode for sequential files. By default, VAX–11 DSM opens an indexed file with GET, PUT, UPD, DEL, and MSE privileges, equivalent to READ/WRITE access to the file. The READONLY parameter opens a file with GET privilege only, equivalent to READ access; all you can do to the file is read it.

If READONLY is present (either explicitly or implicitly), an error occurs if you try to open a file that does not exist. A new file is never created in response to OPEN if READONLY is present.

The SHARED parameter enables shared access to the file. The SHARED parameter is equivalent to the RMS SHR options. If the /SHARED qualifier, described in Section 4.4.19, is specified, the default is SHARED for indexed files.

The NOSHARED parameter disables the RMS shared file option. This parameter opens a file with the RMS access privilege NIL only, so that all explicit file sharing is disabled. If the /NOSHARED qualifier is in effect, NOSHARED is the default for indexed files.

The following examples show various file-sharing options and the equivalent RMS access privileges.

OPEN Option	RMS File–Sharing Mask
O "File":(NOSHARED:INDEXED:READONLY)	NIL
O "File":(INDEXED:SHARED)	GET,PUT,UPD,DEL,MSE
O "File":(INDEXED:SHARED:BLOCKSIZE=512)	GET,PUT,UPD,DEL,MSE,UPI
O "File":(INDEXED:SHARED:READONLY)	GET,MSE
O "File":(INDEXED:SHARED:READONLY:BLOCKSIZE=512)	GET,UPI
O "File":(INDEXED:NOSHARED)	NIL

Refer to the *VAX–11 RMS Reference Manual* for a complete description of the RMS keywords listed above.

Except for the KEYSIZE parameter, the remaining OPEN parameters for indexed file I/O (including the timeout) perform the same functions as their counterparts for sequential files, as described in Section 6.7.1.1.

The KEYSIZE=n parameter allows you to select the size of the primary key for the records written to an indexed file. The value of the parameter argument n can be 1 to 255. By default, the primary key size is 64; the system uses 64 as the value of n if you do not include this parameter when you open the file.

6.9.1.2 The USE Command — Parameters for the USE command allow you to perform random or sequential record I/O. Table 6–9 summarizes the USE command parameters for indexed file I/O. The paragraphs following this table describe each parameter in detail.

Table 6–9: USE Command Parameters for Indexed File I/O

Parameter	Function
CONVERT	Converts lowercase characters to uppercase on READ.
NOCONVERT	Disables lowercase character conversion on READ (default).
DELETE	Deletes the current record.
DISCONNECT*	Positions the file at the beginning or end depending on the next operation.
REFERENCE=n	Specifies an alternate key of reference.
KEY="key"	Positions the file to the specified key for read operations. If reference is not equal to 0, the alternate key specified in the REFERENCE parameter is used.
RFA="n,m"	Positions the file to the specified Record File Address for read operations.

* Indicates a VAX–11 RMS option.

The CONVERT, NOCONVERT, and DISCONNECT parameters perform the same functions as their sequential file I/O counterparts, described in Section 6.7.1.2.

The DELETE parameter deletes the current record. This parameter can be used to delete records either randomly (by key or RFA) or sequentially.

The REFERENCE=n parameter specifies a key other than the primary key. (The primary key, which is the default, is REFERENCE=0.) The value specified must be in the range 0 to 255. If the KEY parameter is absent, the REFERENCE parameter is ignored. The KEY="key" parameter allows you to read records randomly by primary or alternate key. It uses any alternate key number specified as the REFERENCE parameter.

KEY positions the file to the record associated with the parameter argument key, which is either the primary or the alternate key. The form of the parameter argument is a character string of 1 to 255 characters.

When you read a record, you can specify a key that is longer than, shorter than, or the same size as the key that was established when the file was created. (The key size is established with the OPEN command parameter KEYSIZE=n for the primary key, and by RMS utilities or $ZCALL (%FDLCREATE) for alternate keys.)

If you specify a longer key, VAX–11 DSM positions the file to the record that is strictly greater in key value (RMS KGT option) than the desired key. For example, if the key for a record is KEYREC and you specify KEYREC0 in the KEY parameter, VAX–11 DSM positions the file beyond the record with which KEYREC is associated. In addition, the next READ returns the null string. The following read returns the next record.

If you specify a key that is shorter than the established key size, VAX–11 DSM does an approximate key match. For example, if the key for a record is KEYREC, and it is unique to four characters, VAX–11 DSM considers any of the following keys a match:

- KEYR
- KEYRE
- KEYREC

The RFA="n,m" parameter allows you to read records in the file randomly by Record File Address. For indexed file I/O, VAX–11 DSM returns the RFA for a record in the $ZB special variable after each successful read or write operation. Refer to the description of this parameter for sequential file I/O (in Section 6.7.1.2) for more information about RFAs.

If you specify a key length that exactly matches the key size established when the file was created, DSM does an approximate key match. The first read returns a null string. If the next record exists, it is returned in the next read. If the next record does not exist, another null string is returned. The file is always positioned at a valid record boundary for successive reads.

6.9.1.3 The CLOSE Command — For indexed file I/O, the CLOSE command accepts the following parameters:

- DELETE
- PROTECTION=opts
- UIC="[n,m]"
- RENAME

These parameters perform the same functions as their counterparts for sequential file I/O. All CLOSE parameters are available on the OPEN command as well. If you specify one of these parameters when you open the file, you need not respecify it when you close the file.

6.9.1.4 The READ Command — The READ command retrieves records from the file either randomly or sequentially.

You can read records randomly either by key (primary or alternate) or by RFA. The following paragraphs describe each procedure.

1. Random by Primary Key

   OPEN the file with the INDEXED parameter, and issue the USE command with the KEY="key" parameter. This positions the file to the record associated with the specified key. A READ retrieves this record. If the record does not exist, a READ returns the null string. To retrieve another record, issue the USE device-specifier:KEY="key" command

again to reposition the file. READ the record; reissue the USE command, and so on. If you try to read a record that collates past the last key in the file, VAX–11 DSM returns the null string.

2. Random by Alternate Key

   OPEN the file with the INDEXED parameter, and issue the USE command with the KEY="key" and REFERENCE parameters. This positions the file to the record associated with the specified key. A READ retrieves this record. If the record does not exist, a READ returns the null string. To retrieve another record, issue the USE device-specifier:KEY="key":REFERENCE command again to reposition the file. READ the record; reissue the USE command, and so on. If you try to read a record that collates past the last key in the file, VAX–11 DSM returns the null string.

3. Random by RFA

   OPEN the file with the INDEXED parameter, and issue the USE command with the RFA="n,m" parameter. This positions the file to the record associated with the specified Record File Address. Once positioned to this RFA, the READ procedure is the same as reading a record by primary key.

To read an indexed file sequentially or to mix random and sequential access to the file, use one of the following procedures:

1. Open the file or issue the USE command with the DISCONNECT parameter. The first READ positions the file at the beginning. Subsequent READs sequentially retrieve each record in the file in the collating sequence of their keys, described below. If you try to read a record when you reach end-of-file, VAX–11 DSM returns an ENDOFILE error.

2. To mix random and sequential access to the file, OPEN it (with the INDEXED parameter) and issue the USE command with either KEY="key" or RFA="n,m". This positions the file to the record associated with the specified key or Record File Address. The first READ returns this record; subsequent READs sequentially retrieve the remaining records in the file by the collating sequence of their keys (ASCII, if created from the DSM language).

   For example, if a file consists of three records called ARECORD, BRECORD, and CRECORD, and each record has a primary key size of 4, the first READ returns ARECORD because its key (AREC) comes first in the ASCII collating sequence. The next READ returns BRECORD because its key (BREC) follows AREC and precedes CREC in the ASCII collating sequence. The next READ returns CRECORD because its key (CREC) follows BREC in the ASCII collating sequence.

For indexed file I/O, the READ command timeout is used to read locked records. If you try to read a record that is locked, VAX–11 DSM will try to read it at one second intervals until the timeout expires.

The READ * command performs an 8–bit binary read of one character. Thus, to retrieve an entire record, you must issue the number of READ * commands equal to the length of the record. (For relative file I/O, the $ZA special variable returns the length of the record (in bytes) after each physical read or write operation.)

6.9.1.5 The WRITE Command — The WRITE command writes a string to the file's output buffer. VAX–11 DSM writes the contents of the buffer to the file when:

- The buffer becomes full
- You issue a formatted WRITE (W !)
- You issue a USE:KEY="key" command

When you write consecutive strings to the buffer, VAX–11 DSM concatenates them into a single record until its length exceeds 255 bytes plus the primary key size; then VAX–11 DSM automatically creates a new record.

VAX–11 DSM places a record in the file according to the collating sequence of its primary key. When writing a record to an indexed file, you cannot explicitly specify a key, since the key value is always embedded in the record itself. For indexed files created from the DSM language, the key starts at character position 0 of the record and terminates at the point specified in the OPEN parameter KEYSIZE="key".

NOTE

Trying to write a null record (or a record shorter than the primary key) to an indexed file generates an error, because a record always has to be large enough to contain the primary key.

The WRITE * command performs an 8–bit binary write of one character to the file's output buffer. To write this character to the file, you must subsequently issue a formatted WRITE command.

The formatted WRITE command behaves for indexed files as for sequential files on disk, as described in Section 6.7.1.8.

6.9.2 Record Locking

The term *record locking* refers to a condition that prevents all users except the current user from accessing a record in any way. This section describes the actions that lock and unlock records in an indexed file.

A record becomes locked when:

1. You read a record sequentially. Subsequent READs lock each subsequent record until the operation is complete.
2. You read a record randomly by primary or alternate key.

3. You position the file randomly by RFA.

A record becomes unlocked when:

1. You update the record with a WRITE.
2. You sequentially read or write another record.
3. You position the file to another record by primary or alternate key (with the USE command).

You can use the READ command with timeout to find out whether a record is locked.

6.9.3 Indexed File Status and Error Conditions

Indexed file I/O error and status conditions are reported in the $ZA and $ZB special variables as shown in Table 6–10.

Table 6–10: $ZA and $ZB Assignments for Indexed File I/O

Special Variable	I/O Operation Successful	I/O Operation Failed
$ZA for READ or WRITE	Length (in bytes) of last record read or written	Error message string corresponding to the primary RMS error status (RMS STS value)
$ZB for READ or WRITE	RFA of the last record read or written	Error message string corresponding to the secondary RMS error status (RMS STV value) If no STV value, returns the null string.
$ZA on OPEN	Maximum record size	
$ZB on OPEN	Primary key size	

6.10 Relative Files

VAX–11 DSM provides a subset of the options supported by VAX–11 RMS for the relative file organization. These options include:

- Random access to the file by relative record number
- Sequential access to the file
- File sharing
- Automatic record interlocking and locking of nonexistent records

To create a relative file, issue the OPEN command followed by a device specifier and the RELATIVE parameter. If the file does not exist, VAX–11 DSM creates it. If the file does exist, VAX–11 opens it. For example:

```
>O "BLUELINE":RELATIVE
```

This command line opens or creates the relative file BLUELINE.DAT on your default disk device.

You can open an existing relative file without specifying the RELATIVE parameter. However, if you do so, the file can only be accessed sequentially; I/O must proceed as if the file were a disk-resident sequential file. Opening a relative file this way disables record access by relative record number and record locking.

6.10.1 Relative File Commands

The commands used for relative file I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE		

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands. The command descriptions are followed by a section on record locking and a description of the status and error conditions returned for relative files.

6.10.1.1 The OPEN Command — The OPEN command, specified with the RELATIVE parameter, reserves a relative file for use by your process or creates a relative file. When specified without the RELATIVE parameter, the OPEN command establishes a sequential access input/output stream from a relative file.

The OPEN command parameters for relative file I/O (including the timeout) perform the same functions as their sequential or indexed file I/O counterparts. The following list shows the OPEN parameters for relative file I/O; starred items (*) indicate that the parameter specifies an RMS option. Refer to Sections 6.7.1.1 and 6.9.1.1 for a complete description of these parameters.

ALLOCATION = n*	PROTECTION = opts
CONTIGUOUSBESTRY*	READONLY
DELETE	RECORDSIZE = n*
EXTENSION = n*	[NO]SHARED
NEWVERSION	UIC = "[n,m]"

6.10.1.2 The USE Command — Parameters for the USE command allow you to perform random or sequential record I/O. Table 6–11 summarizes the USE command parameters for relative file I/O. The paragraphs following this table describe each parameter in detail.

Table 6–11: USE Command Parameters for Relative File I/O

Parameter	Function
CONVERT	Converts lowercase characters to uppercase on READ
NOCONVERT	Disables lowercase character conversion on READ (default)
DELETE	Deletes the current record
DISCONNECT*	Positions the file at the beginning or end depending on the next operation
KEY=record number	Positions the file by relative record number

* Indicates a VAX–11 RMS option.

The CONVERT, NOCONVERT, and DISCONNECT parameters perform the same functions as their counterparts for sequential file I/O. Refer to Section 6.7.1.2 for details about these parameters.

The DELETE parameter deletes the current record. This parameter can only be used to delete records randomly, that is, after you position the file by key (with the KEY parameter).

The KEY=record number parameter accepts a positive integer argument that specifies the relative record number of a record. When you include this parameter in the USE command, VAX–11 DSM positions the file at the specified record. See the description of the READ and WRITE commands for more information.

6.10.1.3 The CLOSE Command — For relative file I/O, the CLOSE command accepts the following parameters:

- DELETE
- PROTECTION=opts
- UIC="[n,m]"
- RENAME

These parameters perform the same functions as their counterparts for sequential file I/O, described in Section 6.7.1.3.

The CLOSE parameters are all available on the OPEN command as well. If you specify one of these parameters when you open the file, you need not respecify it when you close the file.

6.10.1.4 The READ Command — The READ command reads a record from the file into a variable as a DSM string. You can read records either randomly by relative record number or sequentially.

To read a relative file randomly, issue the USE command with the KEY=record number parameter to position the file at the record. Then READ the record. (For relative file I/O, the $ZB special variable returns the relative record number after each READ or WRITE.)

To retrieve another record, you must position the file again by issuing the USE command with KEY=record number. If you try to read an empty cell on a random access to the file, VAX–11 DSM returns a null string.

To read a relative file sequentially or to mix random and sequential access to the file, use one of the following procedures:

1. Open the file and USE it. The first READ positions the file at the beginning. All subsequent READs sequentially retrieve the contents of each cell (until you explicitly override sequential access with the KEY parameter of the USE command). VAX–11 DSM skips empty cells on sequential read operations.

2. To mix random and sequential access, OPEN the file (with the RELATIVE parameter) and issue the USE command with the KEY=record number parameter. This positions the file at the record associated with the specified relative record number. The first READ returns this record, and subsequent READs sequentially return the remaining records in the file. After VAX–11 DSM reads the last record, it returns a %DSM–E–ENDOFILE message.

For relative file I/O, the READ command timeout is used to read locked records. See Section 6.9.2 for more information about record locking. The timeout parameter behaves as follows. If you try to read a record that is locked, VAX–11 DSM tries to read it at one second intervals until the timeout expires.

The READ * command behaves exactly like its counterpart for indexed files, described in Section 6.9.1.4.

6.10.1.5 The WRITE Command — The WRITE command writes a string to the file's output buffer. When the buffer becomes full, or if you execute a formatted WRITE, VAX–11 DSM writes the contents of the buffer to the file. When you write consecutive strings, VAX–11 DSM concatenates them into a single record until the length exceeds 255 bytes; then VAX–11 DSM automatically creates a new record.

Records can be written either randomly or sequentially. To write records randomly, issue the USE command with the KEY=record number parameter followed by the WRITE command. This sequence writes the record to the specified relative cell. If the cell contains a record, it is updated. If the cell is empty, the record is inserted into the cell. If the cell does not exist, and the file has exhausted its initial allocation of space, the file is automatically extended to accommodate the new record.

To WRITE records sequentially, OPEN the file and issue the USE command. The first WRITE positions the file at the end and writes the record to the first cell after the last record in the file. All subsequent WRITEs are sequential (until you explicitly override sequential access with the KEY parameter of the USE command).

NOTE

VAX–11 DSM performs an implicit WRITE ! prior to every USE command to ensure that records are written to the appropriate cell. Thus, the following command sequence writes the string "ABC" to the last cell referenced and the string "CDE" to cell 100:

```
W "ABC" U "X":KEY=100 W "CDE",!
```

The WRITE * behaves like its counterpart for indexed file I/O, described in Section 6.9.1.5.

For relative file I/O, the formatted WRITE behaves exactly as it behaves when used with sequential files on disk. Refer to Section 6.7.1.8 for details.

6.10.2 Record Locking

Record locking is the same for relative files as for indexed files, except that you access a relative file by relative record number, not by key. See Section 6.9.2 for information on record locking of indexed files.

6.10.3 Relative File Status and Error Conditions

Relative file I/O error and status conditions are reported in the $ZA and $ZB special variables as shown in Table 6–12.

Table 6–12: $ZA and $ZB Assignments for Relative File I/O

Special Variable	I/O Operation Successful	I/O Operation Failed
$ZA for READ or WRITE	Length (in bytes) of last record read or written	Error message string corresponding to the primary RMS error status (RMS STS value)
$ZB for READ or WRITE	Relative record number	Error message string corresponding to the secondary RMS error status (RMS STV value) If no STV value, returns the null string.
$ZA for OPEN	Maximum record size	
$ZB for OPEN	0	

6.11 Using Mailboxes

A mailbox is a VAX/VMS pseudodevice that can be shared by a number of processes in the system. Though a mailbox is a software data structure in virtual memory, it is treated exactly as if it were a record-oriented I/O device.

Mailboxes are used for interprocess communication. One process can create a mailbox and write data to it, and other processes can subsequently read data from the mailbox or write data back to it. Thus, you can use a mailbox to pass status information, return codes, messages, or any other data. Mailboxes can also be used to control other processes.

To create a mailbox from the DSM language, issue the OPEN command followed by a device specifier and the MAILBOX and NEWVERSION parameters, for example:

```
>O "T":(MAILBOX:NEWVERSION)
```

When you create a mailbox, you can make it either temporary or permanent. You can make it available to a single group or to the entire system. A temporary mailbox remains in existence until the number of channels assigned to it reaches zero, that is, when all processes close it. A permanent mailbox remains in existence until it is explicitly deleted, as described in Section 6.11.2.3.

VAX/VMS allows temporary mailboxes for groups only. You can specify that DSM create a system permanent mailbox, but you cannot create a group permanent mailbox or a system temporary mailbox.

6.11.1 Privileges Required to Create a Mailbox

To create a group temporary mailbox, you must have the TMPMBX privilege. To create a system permanent mailbox, you must have the PRMMBX privilege. (You also need this privilege to delete a system permanent mailbox).

No special privileges are required to access an existing mailbox.

6.11.2 Mailbox Commands

The commands used for mailbox I/O are:

Assignment	Input	Output
OPEN	READ	WRITE
USE	READ *	WRITE *
CLOSE		

These commands are described in the following sections. Assignment commands are described first, then input commands, and finally output commands. The command descriptions are followed by a description of status and error conditions for mailbox I/O.

6.11.2.1 The OPEN Command — If specified with the parameters MAILBOX and NEWVERSION, the OPEN command creates a mailbox. If specified without parameters, the command can reserve a mailbox that already exists. For example, if a mailbox called LZ004 exists, the following command opens it for reading or writing:

```
>O "LZ004"
```

However, if no mailbox of this name exists but there is a sequential file called LZ004, this command opens the file. If there is no device at all called LZ004, this command creates a new sequential file, not a mailbox.

If a mailbox and a sequential file have the same name, you can use the following OPEN command syntax to ensure you access the right device:

`>O "LZ004"` Opens mailbox LZ004

`>O "_LZ004"` Opens file LZ004

In the first command line, VAX–11 DSM opens the mailbox because LZ004, like all mailbox names, is a logical name, and device specifiers always undergo logical name translation. In the second command line, the underscore character preceding the device specifier inhibits logical name translation; thus, VAX–11 DSM opens the file. Table 6–13 summarizes the OPEN parameters for mailbox I/O.

Table 6–13: OPEN Command Parameters for Mailbox I/O

Parameter	Function
BLOCKSIZE=n	Specifies the maximum size of a single message when you create the mailbox (default is 255, the maximum size of a DSM string).
DELETE	Marks a system mailbox for deletion on CLOSE, as described in Section 6.11.2.3.
MAILBOX	Specifies that a mailbox should be created or reserved.
NEWVERSION	Creates a group temporary or system permanent mailbox depending on the parameters specified with it.
PROTECTION=opts	Specifies the protection that applies to the mailbox (READ protect and WRITE protect only).
RECORDSIZE=n	For mailboxes, the record size is equal to the block size.
SYSTEM	Creates a system permanent mailbox when specified with the NEWVERSION parameter

The BLOCKSIZE=n parameter accepts a positive integer argument that specifies the maximum size of a message that can be written to or read from a mailbox. The maximum value of n depends on the value of the VAX/VMS sysgen parameters associated with mailboxes. Consult your system manager for the value of these parameters at your installation. By default, VAX–11 DSM uses the maximum length of a DSM string (255 characters)

for the maximum message size if you do not include the BLOCKSIZE parameter on the OPEN command when you create the mailbox. As noted above, the record size for mailboxes is equal to the block size.

The NEWVERSION parameter creates a new mailbox. By default, VAX–11 DSM creates a group temporary mailbox. If you specify the NEWVERSION parameter when a mailbox exists, VAX–11 DSM ignores it.

The PROTECTION=opts parameter specifies the protection that applies to the mailbox. Mailbox protection is limited to READ access, WRITE access, and READ/WRITE access. By default, mailboxes have no protection. All users can access the mailbox READ/WRITE. To change the protection, follow the procedure outlined in the description of this parameter for sequential file I/O (in Section 6.7.1.1).

The SYSTEM parameter creates a system permanent mailbox. You must have the privileges described in Section 6.11.1 to use this parameter. If you do not have these privileges and you try to use this parameter, an error is generated.

For mailbox I/O, the OPEN command timeout parameter allows a process to wait for a mailbox to be created (after which it can perform any appropriate operation). To do this, issue the OPEN command with only the MAILBOX parameter and a timeout, for example:

```
> O "ABC":MAILBOX:15
```

In this example, VAX–11 DSM waits 15 seconds for mailbox ABC to be created. If you issue the OPEN command in this form without the timeout, VAX–11 DSM continues trying to access the mailbox.

6.11.2.2 The USE Command — For mailbox I/O, the USE command accepts the following parameter:

WAIT This parameter instructs VAX–11 DSM to wait for a written message to be read by the receiving process before returning control to the DSM interpreter. By default, control returns to the DSM interpreter as soon as the message is written to the mailbox.

6.11.2.3 The CLOSE Command — For mailbox I/O, the CLOSE command accepts one parameter, DELETE. This parameter marks a system permanent mailbox for deletion. VAX/VMS only deletes a system permanent mailbox when the number of channels assigned to it reaches 0, that is, when no processes are accessing it.

6.11.2.4 The READ Command — The READ command retrieves one message from the mailbox. The length of the message read depends on the length of the record when it was written to the mailbox. If you try to read a message that is longer than 255 characters, VAX–11 DSM returns the message to you in pieces in subsequent READs. Refer to Section 6.7.1.4 for a general description of this process.

For mailbox I/O, a READ with timeout causes VAX–11 DSM to try to read a message at one-second intervals until the timeout expires. A timeout of 0 causes DSM to try to read a message once and then quit. If no message is read, the null string is returned. For timed READ, the value of $T indicates whether an end of file was read (if $T = 1) or no message was read ($T = 0).

If you try to read a mailbox without a timeout, and the mailbox does not contain a message, the read operation hangs until someone else writes to the mailbox.

The READ * command retrieves one 8–bit binary character from the message string.

6.11.2.5 The WRITE Command — The WRITE command transfers a string to the mailbox's message buffer. Only a formatted WRITE actually sends the contents of the buffer to the mailbox.

If the buffer becomes full before you write its contents, VAX–11 DSM reports an error.

If you issue the USE command with the WAIT parameter, a write operation hangs until the receiver reads the message.

NOTE

The maximum number of messages that can be written to a mailbox depends on the value of VAX/VMS sysgen parameters. Consult your system manager for the value of this parameter at your installation.

The WRITE * command writes one 8–bit binary character to the mailbox message buffer.

For mailbox I/O, the formatted WRITE performs the following functions:

Command	Function
WRITE !	Writes the contents of the mailbox buffer to the mailbox. Two consecutive carriage return/line feeds (!!) write an end-of-file. (However, VAX–11 DSM reads end-of-file as a null string. Thus, mailbox I/O never causes a %DSM–E–ENDOFILE error.)
WRITE #	Writes the contents of the mailbox buffer to the mailbox followed by a message that contains a form feed.
WRITE ?n	Writes the specified number of spaces (relative to $X) to the mailbox message buffer.

6.11.3 Mailbox Status and Error Conditions

Mailbox I/O status and error conditions are reported in the $ZA and $ZB special variables as shown in Table 6–14.

Table 6–14: $ZA and $ZB Assignments for Mailbox I/O

Special Variable	I/O Operation Successful	I/O Operation Failed
$ZA on READ or WRITE	Length (in bytes) of last message read or written	Error message string corresponding to the VAX/VMS I/O error
$ZB on READ or WRITE	After READ — Process ID (PID) of sender After WRITE with WAIT — PID of receiver After WRITE without WAIT — 0	0
$ZA after OPEN	Maximum message size of mailbox	
$ZB after OPEN	0	

6.12 Communicating with Remote Computers — Networks

If your computer is one of the nodes in a DECnet network, you can use VAX–11 DSM I/O procedures to communicate with other nodes in the network. These procedures allow you to exchange data with routines on a remote system (task-to-task communication) and to access files on a remote system (resource sharing).

Both task-to-task communication and resource sharing between systems are transparent. That is, these intersystem exchanges do not appear to be different from local interprocess and file access exchanges.

The following sections describe the special considerations that you should keep in mind when communicating across the network.

6.12.1 Limitations on Operations Across the Network

The I/O options listed earlier in this chapter for the OPEN, USE, and CLOSE commands can be used for cross-network access only if the corresponding options are supported by both DECnet/VAX and the system running on the remote node.

You cannot use any device-dependent options in I/O to terminals or mailboxes across the network. Timed read is not supported. Writing to mailboxes is synchronous.

6.12.2 Reading and Writing Files Across the Network

To read and write files across the network (resource sharing), you specify a node name as the first element of a VAX–11 DSM device specifier, for example:

```
>O "BOSTON::TEST$DISK:[OWNER]TESTDATA.DAT;2"
```

This command line opens the file TESTDATA.DAT;2 on the device represented by TEST$DISK: on the BOSTON node of the network. This syntax allows you to access remote files of any organization (sequential, relative, indexed).

See the *DECnet/VAX User's Guide* for information about the use of sequential, relative, and indexed files across the network.

6.12.3 Task–to–Task Communication Across the Network

Task–to–task communication requires a special field in the device specifier. You must use the notation TASK= in place of the device field, and supply a task name, for example:

```
> O "BOSTON::" "TASK=NETJOB" " "
```

This command line establishes a link with a task on the remote node. There must be a VMS command file (in this case, called NETJOB.COM) in the default directory of the account being accessed on the remote node. The command file invokes the task that you want to run. Subsequently, your application can receive data from this task and process it, or your application can pass data across the network to be processed.

The task invoked by the command file must contain the following statement:

OPEN "SYS$NET"

or the equivalent command in a language other than DSM.

6.12.4 Ending Communication Across the Network

To break the link established during task-to-task communication or resource sharing, simply close the device on the remote node with the CLOSE command, for example:

```
> C "BOSTON::TEST$DISK:[OWNER]TESTDATA.DAT;2"
```

If you use logical names in your applications, you can equate the logical names with either local or remote files as required.

6.12.5 Accessing DSM Globals Across the Network

To access DSM globals on a remote node, you specify an alternate global directory in the user field of the global specification (described in Section 9.2).

The directory name can be specified in either of the following ways:

- As a full VAX/VMS specification (node name, device name, and directory name)

- As a node name followed by access information (in quotes) and a device name

In the second case, you can use the dummy device name SYS$DISK, which is translated on the remote node. An example of the user field produced by the second method is shown in the following command line:

```
WRITE *["FARNODE" "ACCESSCODE" "::SYS$DISK:"]B
```

6.12.6 Using Mailboxes Across the Network

You open a mailbox across the network by issuing the DSM OPEN command with the parameter NOSEQUENTIAL specified after the node name and the mailbox name, as in the following example:

```
OPEN "FARNOD::MBXNAME:":NOSEQUENTIAL
```

You must include a colon after the logical name of the mailbox to force logical name translation on the remote node rather than on your local node.

Note that you cannot use the MAILBOX parameter on the OPEN command.

Remember that I/O to network mailboxes is synchronous.

Chapter 7
VAX–11 DSM Utilities

This chapter describes the VAX–11 DSM utility routines. It also describes how to create an interactive menu interface for utilities written at your installation.

7.1 Overview of VAX–11 DSM Utilities

The DSM utilities are a collection of routines written in the VAX–11 DSM language that perform commonly needed tasks or provide commonly needed information. The utilities fall into two categories:

- Library Utilities
- System Utilities

In general, library utilities serve the needs of application programmers, and system utilities serve the needs of system managers and system operators.

VAX–11 DSM provides access to all utilities through an interactive menu-driven package. When you invoke the package, DSM displays options on your terminal. You select the options required to do a given task.

The utility package provides on-line information about the utilities that make up the package. If you are ever unsure of the appropriate response to a utility prompt, you can always enter a question mark (?) for help. DSM responds by listing your options or by asking you to identify which option you need clarified.

A library utility called ^%HELP also provides on-line assistance; it provides information about all DSM commands, variables, and functions. You can run ^%HELP with the DO command or invoke it (in Programmer Mode) by typing a question mark (?).

Figure 7–1: The VAX–11 DSM Utilities

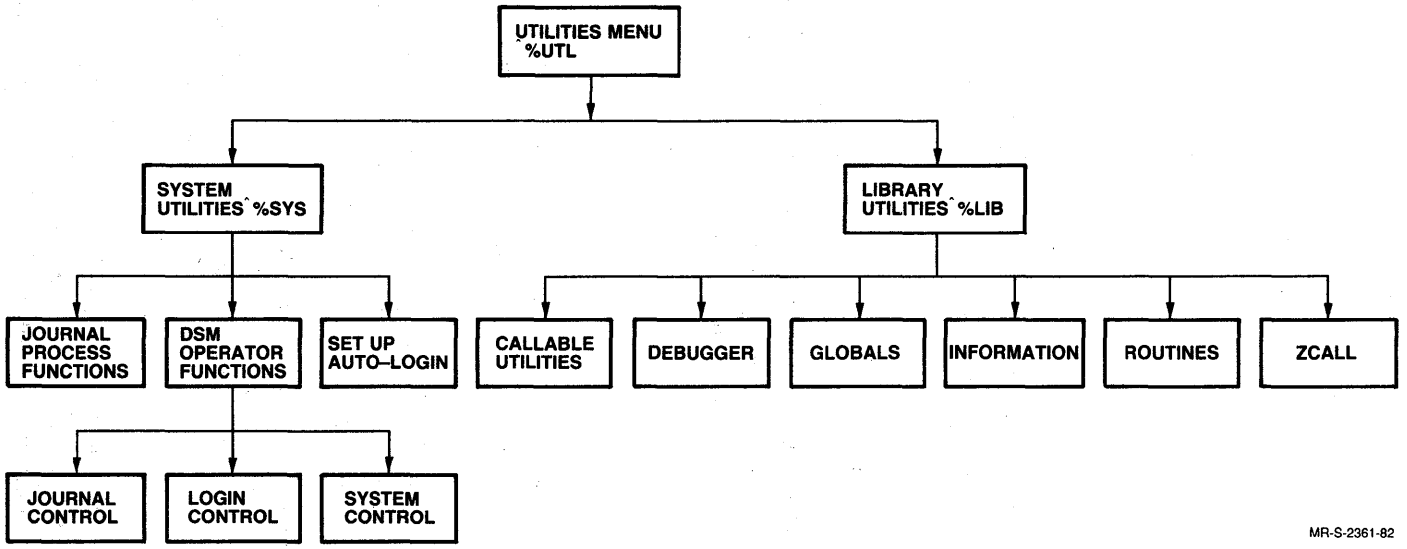


Figure 7–1 shows the categories of the VAX–11 DSM utilities and the tree-like structure of the utility menu.

7.2 Running the DSM Utilities

You can access the VAX–11 DSM utilities in three ways:

- Through the utility package menu, by typing:

  >D ^%UTL

  Or, if you know the utility's category but not its name, you can access the menu at a specific entry point by typing:

  >D ^%LIB for all library utilities

  >D ^%SYS for all system utilities

  >D ^%JOURNAL for all journaling utilities (included under ^%SYS)

  >D ^%OPER for all operator functions (included under ^%SYS)

- By typing the utility's VAX–11 DSM routine name directly in Programmer Mode, in one of these formats:

  >D ^%Utilityname or >D Label^%Utilityname

- By invoking VAX–11 DSM in Application Mode using a utility's routine name as the DSM command parameter, in one of these formats:

  $ DSM ^%Utilityname or $ DSM Label^%Utilityname

7.3 Utility Conventions

All utilities use the following conventions, unless otherwise specified in the description of the utility:

- To get information about any utility or utility prompt, enter a question mark (?). In response, the system prints help text.
- Each menu displays a list of numbers followed by options. Options can point to utility categories or to actual utilities, for example, CALLABLE FUNCTIONS or VERIFY DSM USERS. To select an option, enter either its number or its name (enough characters to distinguish it from other names). The menu also displays the VAX–11 DSM routine name for many utility categories and all utility routines, for example, ^%LIB or ^%RD. You cannot, however, enter such names in response to an option prompt.
- If you do not respond to a prompt within 60 seconds, the utility returns to the previous question; it eventually returns to the DSM interpreter prompt if you do not type anything. The system manager can alter the default timeout period by changing the library global ^%TIMEOUT. (Section 9.3.1 describes how to modify library globals.)

- The correct way to exit a menu is to back up, by typing either a RET or a circumflex (^) at each prompt. To exit any utility, use the same procedure. If the utility interprets a carriage return as a default value, use the circumflex instead of the carriage return to back up.
- After a utility has begun to operate, do not stop the utility before execution is completed. You run the risk of leaving files open, corrupting the data base, or leaving variables undefined. Let the utility run to completion, then rerun it correctly.
- When a utility requests an input or output file, it accepts any valid VAX/VMS device and file specification. You can refer to your own terminal as 0, but you must specify all other devices by device name or logical name. If your VAX/VMS installation spools the line printer, you can specify LP in the device field of the output file specification. When you include the line printer name in the file specification (for example, LP0:MYLIST.DAT), the file name appears on the header page of your output.
- For some utilities, such as those that manipulate routines or globals, you may want to specify more than one name in response to a utility prompt. List multiple names separated by commas, or use the asterisk (*) as a wild card character. The asterisk by itself indicates ALL. When the asterisk is the last character in an alphanumeric string, the utility interprets that string as meaning all names with that string as their root. For example, to specify ALP, ALPHA, ALP123, type ALP*.

  Another valid response when prompted for a routine or global name is circumflex–D (^D); this response displays the contents of your global or routine directory (to help you make your selection). You can also enter circumflex–L (^L); this response produces a list of the routines or globals you have already selected.
- Access to some system utilities depends upon whether your UIC matches the UIC of the DSM Journal Process (for ^%OPER) or the DSM Job Controller (for ^%JOURNAL). If you try to run utilities in this category and your UIC does not match, VAX–11 DSM reports an error message indicating that it could not interact with the DSM Journal Process or the DSM Job Controller.
- VAX–11 DSM always returns numbers in decimal, and always reads numbers as being decimal. You can, however, do numeric conversions to other bases, as described in Section 7.4.3.1.

7.4 The Library Utilities

The library utilities fall into five categories:

1. Callable functions which provide date and time and numerical conversion
2. Routines for use in debugging DSM routines

3. Routines to manipulate globals
4. Routines to provide information about your VAX–11 DSM process
5. Routines to manipulate DSM routines

The following sections describe each of these categories and the utilities within them. Refer to the on-line help text for detailed information about particular utilities and the prompts they employ.

7.4.1 Global Utilities

When you select GLOBALS from the ^%LIB menu, DSM displays a menu of utilities that examine and manipulate globals. The global utilities, described in the following sections, are:

Utility	Routine Name
COMPATIBILITY RESTORE	^%GR11
COPY	^%GC
CREATE	^%GLCRE
DIRECTORY	^%GD
EDIT	^%GEDIT
LIBRARY DIRECTORY	LIB^%GD
LIST	^%GL
RESTORE	^%GR
SAVE	^%GS
SET ATTRIBUTES	SET^%GBLATR
SHOW ATTRIBUTES	SHOW^%GBLATR
SIZE	^%GBLSIZ

Because globals are VAX–11 RMS ISAM files, you can manipulate or inspect them with the RMS utilities as well as the VAX–11 DSM utilities. Section 9.7.2 provides an overview of the RMS utilities. Refer to the *VAX–11 RMS Utilities Manual* for a detailed description.

7.4.1.1 Compatibility Restore (^%GR11) — Restores globals from DSM–11–generated ANSI–formatted magnetic tapes to VAX–11 DSM. The Compatability Restore utility can restore globals from tapes generated by both DSM–11 Version 1 and DSM–11 Version 2. In addition, it can restore a global directly from tape or from a copy of it on disk.

To make transfers from DSM–11 to VAX–11 DSM with this utility, you must set the DSM–11 magnetic tape default to AVL (ANSI variable-length labelled) and use the DSM–11 Global Save utility (^%GS). Do not use ^%GTO. When you bring the tape to your VAX–11 DSM system, use the ^%GR11 utility to restore tapes generated by DSM–11 Version 1 or the V2^%GR11 entry point to restore tapes generated by DSM–11 Version 2.

When you restore a global from magnetic tape, the ^%GR11 utility prompts for an input file specification. This specification must be:

MTn:MUMPS.SRC

where n is the unit number.

7.4.1.2 Global Copy (^%GC) — Copies globals or parts of globals from one global to another.

7.4.1.3 Global Create (^%GLCRE) — Allows you to create a global and define its keysize, initial allocations, and default file extension.

If the current device is the principal device, the ^%GLCRE utility is interactive. If the current device is not the principal device, this utility automatically reads from the current device. You can use the latter method to read the file generated by the Global Size (^%GBLSIZ) utility to recreate globals of the same name with more suitable parameter values.

7.4.1.4 Global Directory (^%GD) — Prints the names of the globals in your current global directory.

7.4.1.5 Global Edit (^%GEDIT) — Allows you to perform node-by-node edits on one or more globals or parts of globals. Global Edit only allows you to alter the data associated with a global node, not the node reference itself.

7.4.1.6 Library Directory (LIB^%GD) — Prints the names of the globals in your current library global directory.

7.4.1.7 Global List (^%GL) — Prints the full references and values for a specified set of globals or global nodes on your terminal.

7.4.1.8 Global Restore (^%GR) — Restores to your current default directory the globals that were saved (using ^%GS) on a disk or magnetic tape file. You can restore all, some, or none of the globals in the specified file; you can also rename each global as you restore it.

The Global Restore utility optionally merges a global being restored with a global of the same name. It also allows you to KILL a global of the same name before you restore it.

7.4.1.9 Global Save (^%GS) — Writes the references and values of a set of specified globals or global nodes from your current global directory to a sequential medium, such as a disk file, printer, terminal, or magnetic tape.

7.4.1.10 Set Attributes (SET^%GBLATR) — Sets the attributes of all or selected globals.

7.4.1.11 Show Attributes (SHOW^%GBLATR) — Displays the attributes of all or selected globals.

7.4.1.12 Global Size (^%GBLSIZ) — Scans a set of specified globals and determines the maximum keysize used by each. Optionally, the Global Size utility can write the specified global names and the maximum keysize associated with each one to an output file. This file can subsequently be passed to the Global Create (^%GLCRE) utility to redefine the globals with more suitable parameter values.

7.4.2 Routine Utilities

When you select the ROUTINES option from the ^%LIB menu, DSM displays a menu of utilities useful for examining and manipulating routines. The routine utilities, described in the following sections, are:

Utility	Routine Name
COMPARE	^%RCMP
COMPATIBILITY RESTORE	^%RR11
CONTENTS	^%RCON
COPY	^%RCOPY
ROUTINE DIRECTORY	^%RD
FIRST LINE LIST	^%FL
LIBRARY DIRECTORY	LIB^%RD
RESTORE	^%RR
SAVE	^%RS
SEARCH	^%RSE
SIZE	^%RSIZE

Among the options listed in the ROUTINE menu is MAPPED. If you select the MAPPED option, DSM displays a menu from which you can choose any of the following utilities:

Utility	Routine Name
BUILD	^%RBUILD
CONTENTS	^%MAPCON
DIRECTORY	MAP^%RD

The following sections describe the utilities accessed through the ROUTINES menu and the MAPPED menu.

7.4.2.1 Routine Compare (^%RCMP) — Compares two VAX–11 DSM routines and prints out all lines that are different.

7.4.2.2 Compatibility Restore (^%RR11) — Restores routines from DSM–11–generated ANSI–formatted magnetic tapes to VAX–11 DSM. The Compatability Restore utility can restore tapes generated by both DSM–11 Version 1 and DSM–11 Version 2. In addition, it can restore a routine directly from tape or from a disk copy.

To make transfers from DSM–11 to VAX–11 DSM with %RR11, you must set the DSM–11 magnetic tape default to AVL (ANSI variable-length labelled) and use the DSM–11 Global Save utility. When you bring the tape to your VAX–11 DSM system, use the ^%RR11 utility to restore tapes generated by DSM–11 Version 1 or the V2^%RR11 entry point to restore tapes generated by DSM–11 Version 2.

When you restore a routine from magnetic tape, ^%RR11 prompts for an input file specification. This specification must be:

MTn:MUMPS.SRC

where n is the unit number.

7.4.2.3 Routine Contents (^%RCON and FULL^%RCON) — Lists the names of the routines in the file you specify; this file must have been created by the Routine Save (^%RS) utility. The FULL^%RCON format lists both the names and the first lines of the routines in the file. To execute Routine Contents from the FULL^%RCON entry point, you must issue the following command in Programmer Mode:

> D FULL^%RCON

7.4.2.4 Routine Copy (^%RCOPY) — Copies segments of DSM source code from one routine to another. Routine Copy can also append several routines into one routine. To copy one line of code only, enter the same line reference (line label or line label plus offset) for both the *From Line* and *Through Line* prompts.

7.4.2.5 Routine Directory (^%RD) — Produces an alphabetical list of the names of the routines in your current routine directory.

7.4.2.6 First Line List (^%FL) — Lists the first lines of the routines you specify. (By convention, these lines should contain the routine's name and a brief description of the routine.)

7.4.2.7 Library Directory (LIB^%RD) — Lists the names of the routines in your current library routine directory.

7.4.2.8 Routine Restore (^%RR) — Restores routines saved on a disk or magnetic tape file to your current routine directory. You can restore all or some of the routines in the specified file; you can also rename each routine as you restore it.

7.4.2.9 Routine Save (^%RS) — Writes specified routines from your current routine directory to any sequential medium. The Routine Save utility also allows you to:

- Get a listing of any routine(s) you want
- Edit a routine with a VAX/VMS editor, and then restore it with the Routine Restore utility
- Transport routines from a VAX–11 DSM system to a DSM–11 system

If you specify the line printer as your output device, the routines are printed with a form feed between each routine.

7.4.2.10 Routine Search (^%RSE) — Searches a routine or set of routines for the occurrence of a specified string. The Routine Search utility prints any line that contains the string, and a reference to the routine in which the line appears.

7.4.2.11 Routine Size (^%RSIZE) — Scans a set of specified routines and determines their size (in bytes) in source and precompiled format. The Routine Size utility also flags the largest source routine specified and the largest routine in precompiled format.

7.4.2.12 Build Mapped Routine File (^%RBUILD) — Builds a file from selected precompiled DSM routines that can be mapped in a virtual memory section. Only files generated by ^%RBUILD can be specified in the argument of the /MAPPED=file-spec qualifier of the DSM command.

7.4.2.13 Contents of Mapped Routine File (^%MAPCON) — Lists the names of all routines in the specified mapped routine file. The specified file name must have been built with the ^%RBUILD utility.

7.4.2.14 Mapped Directory (MAP^%RD) — Lists the names of all DSM application routines and DSM library routines that are currently mapped in virtual memory.

7.4.3 Callable Functions

Descriptions and examples of nine callable functions can be accessed through the CALLABLE FUNCTIONS option of the %LIB menu. (To use these functions, invoke them from your application routine.)

DATE (^%D)	Writes current date in the following format: day-month-year For example, 5–JUN–82
HOROLOG (^%H)	Converts DSM date and time, as described below in Section 7.4.3.2

DECIMAL TO HEX (^%DH)	Converts decimal number to hexadecimal
DECIMAL TO OCTAL (^%DO)	Converts decimal number to octal
HEX TO DECIMAL (^%HD)	Converts hexadecimal number to decimal
HEX TO OCTAL (^%HO)	Converts hexadecimal number to octal
OCTAL TO DECIMAL (^%OD)	Converts octal number to decimal
OCTAL TO HEX (^%OH)	Converts octal number to hexadecimal
TIME (^%T)	Writes current time in the following format: hours:minutes AM or PM For example, 10:24 AM

7.4.3.1 Using the Numerical Conversion Functions — When you select one of the conversion functions from the CALLABLE FUNCTIONS menu, the utility prompts for input. You usually, however, call these functions directly from your DSM application routines, non-interactively.

These functions need not be interactive, however. If you call them from an application routine, you can pass input to them in variables and receive output from them in variables.

To create a routine to call one of these functions, set a local variable with the same name as the function equal to the number you want to convert. Then issue the following command:

>D ^%Function Name

This sequence creates a variable with that function name whose value is equal to the results of the conversion. For example, to convert your DSM job number (which is the VAX/VMS Process I.D.) from decimal (DSM format) to hexadecimal (VAX/VMS format), type:

```
>SET %DH=$J DO ^%DH WRITE %DH
710019
```

where $J is the DSM Job Number.

If an input error occurs during the execution of any numeric conversion routine, (for example, trying to convert 9 from octal to decimal), the variable receives the value *.

7.4.3.2 Using the ^%H Function — The ^%H function is a special case. This function converts date and time from $HOROLOG format to the current date and time, and vice versa. $HOROLOG expresses date and time in the format:

n,m

where n is the number of days since December 31, 1840, and m is the number of seconds since midnight.

You can perform four conversions using ^%H:

1. To convert the date in $H format (n) to the current date, type:

   >SET %DT=$P($H,",",1) DO %CDS^%H WRITE %DAT

   This command sequence returns the date in the format month/day/year, for example 6/5/1983.

2. To convert the current date to $H format, type:

   >SET %DT="mm/dd/yyyy" DO %CDN^%H WRITE %DAT

   This command sequence returns a 5–digit number that represents the first value (n) of the $H format.

3. To convert $H time (m) to the current time, type:

   >SET %TM=$P($H,",",2) DO %CTS^%H WRITE %TIM

   This command sequence returns the current time in the format hours:minutes:seconds.

4. To convert current time to $H format, type:

   >SET %TM="hh:mm:ss" DO %CTN^%H WRITE %TIM

   This command sequence returns the number of seconds since midnight, that is, the $H m value.

7.4.4 Debugger Utilities

When you select the DEBUGGER option from the ^%LIB menu, DSM displays the DEBUGGER menu. This menu provides access to the following three utilities useful in debugging DSM routines:

Utility	Routine Name
STACK	^%STACK
TRACE	^%TRACE
VERIFY ROUTINE	^%ERRCHK

These utilities are described in the following sections.

7.4.4.1 The ^%STACK Utility — The ^%STACK utility prints out the current state of the call stack, including the program name, line number, and command number of all currently active DOs in reverse order. You can use this utility at a breakpoint to determine exactly what path was taken to reach a particular place in a program. You can also call the utility as a breakpoint action, as follows:

SET $ZBREAK(4)="TRACK^PREFIT>D ^%STACK"

The following example shows the output of ^%STACK:

```
> SET $ZBREAK="DUMMY+1TEST2:1"
> ZDEB
> D TEST
 This is a test
 4
 %DSM-I-BREAK, BREAK command executed
 -DSM-I-ATLABEL, DUMMY+1TEST2 S X=""
 BREAK 1:1>DO %STACK
 Program Name =    TEST2    Line Number =    2    Command Number =    1
 Program Name =    TEST1    Line Number =    3    Command Number =    1
 Program Name =    TEST     Line Number =    3    Command Number =    3
 Program Name =             Line Number =    0    Command Number =    0

 BREAK 1:1>
```

7.4.4.2 The ^%TRACE Utility — The ^%TRACE utility uses the single-step breakpoint, $ZBREAK(0), and breakpoint action strings to produce an execution trail of the routine(s) being debugged. ^%TRACE allows you to monitor the execution of a program or a set of programs, or to monitor the state of a variable or set of variables.

You can choose any of the following reporting modes when you use TRACE:

- Command — Reports any change in the command number.
- Line — Reports any change in the entry reference.
- Tag — Reports any change in the tag part of the entry reference.
- Routine — Reports any change in the routine name.
- Variable — Reports any change in the value of a variable or set of variables. Reports old and new values, along with the location (entry reference:command number) where the change occurred. This location is called a *watchpoint.*

The following example shows a session with ^%TRACE.

```
> DO ^%TRACE

Trace utility

Enter routine reference (? for help) : START+1^TEST<RET>

Output file ? <RET>

Trace options are : C(OMMAND), L(INE), T(AG), R(OUTINE) or W(ATCH)
Enter trace option (or ? for help) : R <RET>

START+1^TEST
This is a test

PRINT+1^TEST1
4
DUMMY+1^TEST2

PRINT+3^TEST1

START+2^TEST
```

```
PRINT+1^TEST1
8
....

> DO ^%TRACE

Trace utility

Enter routine reference (? for help) : START+1^TEST<RET>

Output file ? <RET>

Trace options are : C(OMMAND), L(INE), T(AG), R(OUTINE) or W(ATCH)
Enter trace option (or ? for help) : W<RET>
Variable ? A<RET>
Variable ? <RET>

This is a test

A Changed at START+1^TEST:2
Old Value =
New Value = 2

A Changed at START+2^TEST:2
Old Value = 2
New Value = 4
4
A Changed at START+2^TEST:2
Old Value = 4
New Value = 8
8


 ....
```

7.4.4.3 The ^%ERRCHK Utility — This utility verifies the syntax of a VAX–11 DSM routine. The following example shows a session with the ^%ERRCHK utility:

```
>DO ^%ERRCHK
Routine name ? CHTEST<RET>
Output file ? <RET>

VAX-11/DSM Error Report for CHTEST       26-AUG-1982
11:19:29.52        Page 1


%DSM-E-COMAND, bad command detected
-DSM-I-ATLABEL, START+1        W "This is a test",!  S A=2

%DSM-E-COMAND, bad syntax at end of command or Xecute string
-DSM-I-ATLABEL, START+2        F I=2:1:10 S A=A*2 D PRINT^TEST1

%DSM-E-COMAND, bad command detected
-DSM-I-ATLABEL, PRINT+3        XXXX

Routine name ? <RET>
>
```

7.4.5 The Information Utilities

When you select the INFORMATION option from the ^%LIB menu, DSM displays the INFORMATION menu. This menu contains two options:

- STATISTICS, which generates the STATISTICS menu
- ZCALL, which provides access to the utility ^%ZD

7.4.5.1 Utilities on the Statistics Menu — When you select the STATISTICS option from the INFORMATION menu, DSM displays the STATISTICS menu, which contains six options. These options are entry points into the ^%STAT utility, as follows:

Utility	Entry Point
1. ALL	ALL^%STAT
2. DEVICES	IO^%STAT
3. DIRECTORIES	DIR^%STAT
4. GENERAL	GENERAL^%STAT
5. GLOBALS	GBLSTAT^%STAT
6. OPEN GLOBALS	GBLGLO^%STAT

7.4.5.2 The ^%ZD Utility — When you select the ZCALL option from the INFORMATION menu, DSM transfers control to entry point ^%ZD. This utility lists the currently installed external functions. You can use any of these functions with the $ZCALL special function, described in Chapter 8.

The ^%HELP utility, invoked by typing ? at the DSM prompt, describes under the keyword EXTERNAL FUNCTIONS the functions (beginning with a %) supplied as $ZCALL entries.

7.5 Overview of the System Utilities

You access the DSM system utilities through the ^%SYS menu. This menu contains three options that provide access to system utilities:

- The ^%OPER menu, whose options allow the DSM operator to control the use of the DSM Job Controller. To use the options on the ^%OPER menu, you must have the same UIC as the DSM Job Controller.
- The ^%JOURNAL menu, whose options allow the DSM operator to control the use of the DSM Journal Process. To use the options on the ^%JOURNAL menu, you must have the same UIC as the DSM Journal Process.
- The ^%ALF utility, which sets up auto-login. You can use this utility with any UIC.

The following sections describe the classes of utilities accessed through these options on the ^%SYS menu. Section 7.6 and its subsections describe the utilities accessed through the ^%OPER menu. Section 7.7 and its subsections describe the utilities accessed through the ^%JOURNAL menu. Section 7.8 describes the ^%ALF utility.

7.6 Utilities Accessed through ^%OPER

The ^%OPER menu contains three options which are themselves menus. These options are:

1. The menu of journal control utilities
2. The menu of log-in control utilities
3. The menu of system control utilities

The following sections describe these utilities.

7.6.1 Journal Control Utilities

These utilities enable and disable journaling by DSM users (whether or not a DSM Journal Process is running). You must have the same UIC as the DSM Job Controller to run these utilities.

7.6.1.1 Add Group to Journal List (ADDJRN^%MJCJRN) — Adds a VAX/VMS group (where group is determined by the group number of a process' UIC) to the list of groups allowed to journal data base transactions. Running ADDJRN^%MJCJRN expands the list of user groups initially established in the DSMMJCPAR.OPT file. Additions made with this utility are only effective if the current Journal–Enable mode is SELECT_GROUPS.

Note that you must subsequently run the Start Journaling For DSM Users utility for these groups to be able to run the DSM image in shared mode (Application Mode).

7.6.1.2 Change Journal–Enable Mode (JOPT^%MJCJRN) — Allows you to change the Journal–Enable mode from ALL to SELECT_GROUPS or from SELECT_GROUPS to ALL.

7.6.1.3 Delete Group from Journal List (DELJRN^%MJCJRN) — Deletes a VAX/VMS group (as determined by the group number of a process' UIC) from the list of groups allowed to journal data base transactions. Deletions made with DELJRN^%MJCJRN are only effective if the current Journal–Enable mode is SELECT_GROUPS.

7.6.1.4 Show Groups Journal List (SHOJRN^%MJCJRN) — Displays the list of all groups that are allowed to journal data base transactions.

7.6.1.5 Start Journaling for DSM Users (JRNON^%MJCJRN) — Prompts the DSM Job Controller Process to send a journaling-enabled indicator to the DSM users who are allowed to perform journaling. This indicator is sent to all designated DSM images as they start up.

However, if a DSM image for which journaling is allowed is already running, it does not receive the journaling-enabled indicator. Thus, this image will not be able to perform journaling.

7.6.1.6 Stop Journaling for DSM Users (JRNOFF^%MJCJRN) — Prompts the DSM Job Controller Process to stop sending the journaling-enabled indicator to the DSM users who are allowed to perform journaling. This indicator is sent to all designated DSM images as they start up.

If a DSM image is already running, and currently journaling data base transactions, it continues to perform journaling.

7.6.2 Log–in Control Functions

The utilities in the following subsections control DSM operations and access to VAX–11 DSM from VAX/VMS. You must have the same UIC as the DSM Job Controller to run these utilities.

7.6.2.1 Add a Group to Log–in List (ADDLOG^%MJC) — Adds a VAX/VMS group (where group is determined by the group number of a process's UIC) to the list of groups allowed to run the DSM image. Running ADDLOG^%MJC expands the list of user groups initially established in the DSMMJCPAR.OPT file. Additions made with this utility are only effective if the current Login–Enable mode is SELECT_GROUPS.

Note that you must subsequently run the Allow Future DSM Logins utility (described in the following section) for these groups to be able to run the DSM image in shared mode (Application Mode).

7.6.2.2 Allow Future DSM Logins (LOG^%MJC) — Allows all designated DSM users to run the DSM image in a shared mode, that is, Application Mode.

7.6.2.3 Change DSM Login–Enable Mode (LOPT^%MJC) — Allows you to change the DSM Login–Enable mode from ALL to SELECT_GROUPS or from SELECT_GROUPS to ALL.

7.6.2.4 Delete a Group from Login List (DELLOG^%MJC) — Deletes a VAX/VMS group (as determined by the group number of a process's UIC) from the list of groups allowed to run the DSM image. Deletions made with DELLOG^%MJC are only effective if the current Login–Enable mode is SELECT_GROUPS.

7.6.2.5 Show Groups (DISLOG^%MJC) — Lists the groups allowed to run the DSM image. Groups are designated by their group numbers.

7.6.2.6 Prevent Future DSM Logins (NOLOG^%MJC) — Prevents all system users from running the DSM image in shared mode.

7.6.3 DSM System Control Functions

When you select the SYSTEM CONTROL option from the ^%OPER menu, DSM displays the SYSTEM CONTROL menu. This menu provides access to six utilities that deal with the DSM Job Controller, as described in the following sections.

Note that you must have the same UIC as the DSM Job Controller to run these utilities.

7.6.3.1 Job Table Display (^%JOBTAB) — Displays the contents of the DSM Job Table. The Job Table is an internal data structure maintained by the DSM Job Controller that contains the following information about each active DSM user:

- Process ID (PID)
- Process name
- UIC
- User name
- Principal Input Device

7.6.3.2 Kill a DSM User (KILLUSE^%MJC) — Instructs the DSM Job Controller to force DSM image rundown of a specified DSM image. In addition, KILLUSE^%MJC clears the DSM Job Table and Lock Table of all entries made by that image. Note that issuing the DCL command STOP/ID=PID does not clear these tables, because it has no effect on the DSM Job Controller.

7.6.3.3 Lock Table Display (^%LCKTAB) — Displays the contents of the Lock Table. The Lock Table is an internal data structure maintained by VAX–11 DSM that contains the following information about each active DSM user:

- Process ID (PID)
- Locked local variables
- Locked global variables
- Pending lock requests

7.6.3.4 Shutdown DSM (SHUTUP^%MJC) — Causes the DSM Job Controller to exit the system automatically, after the last DSM user known to it has exited. SHUTUP^%MJC provides an easy way to shut down the Job Controller if there are active DSM users and you want to avoid having to check until the last one has logged off. After you run this utility, you can restart the Job Controller only by performing the start-up procedures outlined in Section 12.3. This utility does not force the exit of any DSM users. (Use KILLUSE^%MJC to force exit.)

7.6.3.5 Status of DSM Job Controller (STATUS^%MJC) — Displays the status of the DSM Job Controller, including:

- What groups are allowed to run the DSM image
- Whether DSM login is enabled or disabled

- What groups are allowed to journal
- Whether journalling is enabled or disabled for selected groups or for all users

7.6.3.6 Verify DSM Users (VERIFY^%MJC) — Verifies that each DSM user known to the Job Controller is an active VAX/VMS user. If any user is found not to be an active VAX/VMS user, VERIFY^%MJC clears the DSM Job Table entry for that user, and any entries in the Lock Table made by that user. You may want to run this utility before shutting down DSM.

7.7 Utilities Accessed through ^%JOURNAL

The ^%JOURNAL menu contains four options:

1. The DEJOURNAL utility
2. The menu of journal file utilities
3. The menu of journal globals utilities
4. The menu of journal process operator utilities

The utilities accessed through options 2 and 4 interact with the DSM Journal Process. You must have the same UIC as the DSM Journal Process to run the utilities accessed through these two options.

7.7.1 The DEJOURNAL Utility

The DEJOURNAL utility, ^%DEJRNL, extracts information from journal files. Dejournaling is described below in Chapter 13.

7.7.2 Journal File Utilities

The journal file utilities affect the behavior of the DSM Journal Process by changing parameters initially established in the Journal Process start-up option file, described in Section 13.5.2.

The utilities described in the following sections add, delete, purge, and display file names in the list of file names maintained by the Journal Process. The Journal Process acquires names for output files from this list.

You must have the same UIC as the DSM Journal Process to use the journal file utilities.

7.7.2.1 Add Journal File Name (ADDFIL^%JRNL) — Adds a file name to the list of output files maintained by the Journal Process. Although ADDFIL^%JRNL adds the file name to the list, the Journal Process does not write to the file until its status is current, open, and ready. If no files are open, you must open the added file with the Open Current Journal File utility. If any other journal file is open, the Journal Process automatically opens the added file when needed.

7.7.2.2 Close Current Journal File (CLOSEFI^%JRNL) — Changes the status of the current journal file from current to closed. After you run CLOSEFI^%JRNL, records of data base transactions cannot be written to this file.

The CLOSEFI^%JRNL utility does not automatically open a file in the Journal Process's list of files, however. If you want journaling to continue, you must subsequently open a file with the Open Current Journal File utility.

NOTE

If a DSM user writes information to the Journal Process while a file is closed, the information is not lost. It is buffered in the Journal Process's input mailbox.

7.7.2.3 New Log File (NEWLOG^%JRNL) — Creates a new Journal Log file (.LOG). A Journal Log file contains the file specifications of the journal file or files that have been opened. (The Journal Log file itself is used in dejournaling. See Section 13.1.4 for details about dejournaling.)

7.7.2.4 Delete Journal File Name (DELFIL^%JRNL) — Deletes the journal file name you specify from the list of journal file names maintained by the Journal Process. Using DELFIL^%JRNL does not affect the file itself, however. If you specify the file name of a closed file, the contents of the file remain intact and its name is not deleted from the Journal Log file.

7.7.2.5 Show Journal File Names (SHOW^%JRNL) — Displays the Journal Process's list of file names. The display includes information on the status (current, open, ready, closed) of each file, and displays any message associated with the file name when you added it to the list. (Such messages are sent to the VAX/VMS operator.)

7.7.2.6 Open Current Journal File (OPENFIL^%JRNL) — Changes the status of the current journal file so it becomes the open journal file; that is, OPENFIL^%JRNL causes the file to be the file written to by the Journal Process.

7.7.2.7 Purge Journal File Names (PURGE^%JRNL) — Purges the names of all closed journal files from the Journal Process's list. Names of open files remain in the Journal Log file, and the closed files themselves remain intact.

7.7.3 Journal Process Globals Utilities

When you choose the GLOBALS option from the ^%JOURNAL menu, DSM displays a menu containing three options. These options are utilities that clear, set, and test the status of globals, as described in the following sections.

You do not need to have the same UIC as the DSM Journal Process to run these utilities.

7.7.3.1 Clear Journaling for Global (CLRGLO^%JRNL) — Removes the marker causing journaling from the globals you specify. This utility invokes SET^%GBLATR (set global attribute).

7.7.3.2 Set Journaling for Global (SETGLO^%JRNL) — Allows you to mark specified globals to be journaled. This utility calls SET^%GBLATR. After you run SETGLO^%JRNL, subsequent modifications of these globals are journaled.

If you KILL a global marked for journaling, the journal indicator associated with that global disappears. Thus, if you recreate that global, you must remark it for journaling with SETGLO^%JRNL or SET^%GBLATR.

7.7.3.3 Test Journaling for a Global (TSTGLO^%JRNL) — Indicates whether a specified global is marked for journaling. This utility calls SET^%GBLATR.

7.7.4 Journal Process Operator Utilities

When you select the OPERATOR option from the ^%JOURNAL menu, DSM displays a menu of six operator utilities for the DSM Journal Process. These utilities enable and disable the DSM Journal Process, KILL the DSM Journal Process, pause and resume operation of the DSM Journal Process, and display the status of the process, as described in the following sections.

You must have the same UIC as the DSM Journal Process to run these utilities.

7.7.4.1 Disable Journal Process (DISABLE^%JRNL) — Causes the Journal Process to suspend operations. While the Journal Process is suspended, information written to the Journal Process is not written to the current output file.

NOTE

The DISABLE^%JRNL utility should be used with discretion, because it can cause loss of information.

7.7.4.2 Enable Journal Process (ENABLE^%JRNL) — Reenables the Journal Process after it has been disabled with the Disable Journal Process utility. After you run ENABLE^%JRNL, the Journal Process resumes writing to the output file. Any information written to the Journal Process while it was disabled is lost.

7.7.4.3 Kill Journal Process (KILL^%JRNL) — Deletes the Journal Process; that is, causes the Journal Process to exit the system. Any information not written to an output file is lost.

If you run KILL^%JRNL, you must perform the entire Journal Process start-up procedure (as described in Section 13.5) to restore the system's journaling capability. Thus, KILL^%JRNL should only be used at system shutdown.

7.7.4.4 Pause Journal Process (PAUSE^%JRNL) — Temporarily stops the Journal Process from writing to the output file. Information can still be written to the Journal Process, but it is buffered in the Journal Process's input mailbox until you resume journaling by running the Resume Journal Process utility, described below.

Running this utility does not cause information to be lost. However, if the Journal Process's input mailbox fills up, all users running the DSM image hang.

7.7.4.5 Resume Journal Process (RESUME^%JRNL) — Causes the Journal Process to continue writing to the output file after it has been stopped by the Pause Journal Process utility.

7.7.4.6 Status of Journal Process (STATUS^%JRNL) — Generates a report that indicates:

- Whether there is any pending input from the Journal Process's input mailbox
- Whether the Journal Process is in a Wait state
- Whether the Journal Process is enabled to write to an output file (file type .JRN)

7.8 The Set Up Auto–Login Facility (^%ALF)

The ^%ALF utility allows you to "tie" terminals to a VAX/VMS user name. Through ^%ALF, you manipulate a copy of the VAX/VMS Auto–Login file, SYS$SYSTEM:SYSALF.DAT.

The utility first asks you for a name for the copy of this file. The default file name is SYSALF.DAT. After you specify a file name, the utility asks you to identify the terminals you want tied and the user name with which you want those terminals associated. (You set up the VAX/VMS account with that user name according to the guidelines for special options listed in Chapter 11.)

Once you enter this information, you must copy your file to the system directory SYS$SYSTEM under the name SYSALF.DAT (that is, its full file specification must be SYS$SYSTEM:SYSALF.DAT). After you copy your file to the system directory, the information in it is implemented.

NOTE

Do not try to edit SYS$SYSTEM:SYSALF.DAT itself. If you do, the file becomes locked, and thus interferes with the execution of the VAX/VMS LOGINOUT utility.

7.9 Creating Your Own Menu

VAX–11 DSM supplies a utility called ^%MENU that allows you to link utilities or functions written at your installation to form an interactive, menu-driven package, similar in format to the VAX–11 DSM menus described earlier in this chapter. The ^%MENU utility can also invoke on-line information about your utilities.

The utility ^%MENU interprets tree-structured data. Thus, routines that call ^%MENU must refer to a global whose nodes contain data that comprise the utility prompts and options you want displayed. Then, when you execute ^%MENU, the list of options for the calling routine(s) is printed and the options a user selects are executed.

To link your utilities through ^%MENU, follow these steps:

1. Create a global that has the menu prompts and list of options for the utilities you want displayed. See the header comment of the source code of ^%MENU for the format to use. To examine the ^%MENU source code, invoke the ^%RS utility, select %MENU, and specify a file name for the output of this command sequence: LP0:MENU.LIS, for example.

2. Write a one-line routine that calls ^%MENU for the global you created, for example:

   ```
   >MYMENU<TAB>SET %MENU="^globalname(""node"")" DO ^%MENU
   >ZSAVE MYMENU
   ```

   Note that ^%MENU cannot process a global without nodes (^A, for example). Thus, "node" is mandatory, but its name is unimportant.

After you complete these steps, type DO ^MYMENU, and the list of options you put in that global is displayed; when you select the option you want, that option is executed.

If you want your utility to be called by ^%MENU, precede the above code with the following:

```
DO:$D(%MENU) PUSH^%MENU
```

7.10 Error Messages

Error messages reported by the VAX–11 DSM utilities are described in Appendix A.

Chapter 8
Procedure Calling

This chapter describes how to call procedures written in languages other than DSM, and provides information on calling VAX/VMS system services and VAX–11 Run–Time Library procedures.

8.1 Overview of Procedure Calling

VAX–11 DSM allows you to access the following kinds of procedures outside of DSM:

- Routines written in other languages
- VAX/VMS system services
- Routines in the VAX–11 Run–Time Library

Section 8.2 defines the term *procedure* in more detail.

The ability to call procedures is especially useful when a feature is not available in VAX–11 DSM. For example, the DSM language does not include a square root function. Through the procedure-calling mechanism, however, you can write a program to calculate square roots (in a language such as FORTRAN), pass arguments to this program from your DSM routine, and have the results passed back to DSM.

VAX–11 DSM implements procedure calling through the $ZCALL function. The following sections describe this function and the procedure-calling mechanism in detail. The material presented here assumes some knowledge of:

- VAX/VMS procedure calling mechanisms
- VAX/VMS argument passing mechanisms

- VAX–11 MACRO
- VAX–11 FORTRAN

You should be familiar with these subjects before you try to use the information presented in this chapter.

8.2 The VAX Procedure Calling Standard

External routines called by the VAX–11 DSM interpreter must conform to the standard defined for VAX procedure calls. This standard defines a procedure as a routine that is entered by a CALLG or CALLS instruction. The calling standard prescribes how arguments are passed to such routines, how function values are returned, and how procedures receive and return control.

VAX–11 DSM uses the CALLG instruction to call procedures.

The following sections describe how arguments are listed and passed in accordance with the VAX Procedure Calling Standard. See the *VAX Architecture Handbook* for further details about this standard.

8.2.1 Argument Lists

Whenever you call a procedure, VAX–11 DSM builds an argument list. This list is the primary means of passing information to and receiving results from the procedure.

As defined in the VAX Procedure Calling Standard, the argument list is a sequence of longword (4–byte) entries. The first byte of the first entry is the argument count, indicating how many arguments follow in the list. The remaining entries are the arguments passed to the procedure for processing.

8.2.2 Argument–Passing Mechanisms

The VAX Procedure Calling Standard defines three mechanisms by which arguments are passed to procedures:

1. By Value

   When you pass an argument by value, the argument list entry contains the actual, uninterpreted 32–bit value of the argument. Usually, arguments passed by value are integer constants that fit in a longword. For example, to pass 100 by value, the caller puts 100 directly into the argument list.

2. By Reference

   When you pass an argument by reference, the argument list entry contains the address of the location that contains the value of the argument. That is, the argument list entry points to the value's location. For example, if the variable X is allocated to location 1000, which currently contains the value 100, the argument list entry contains 1000.

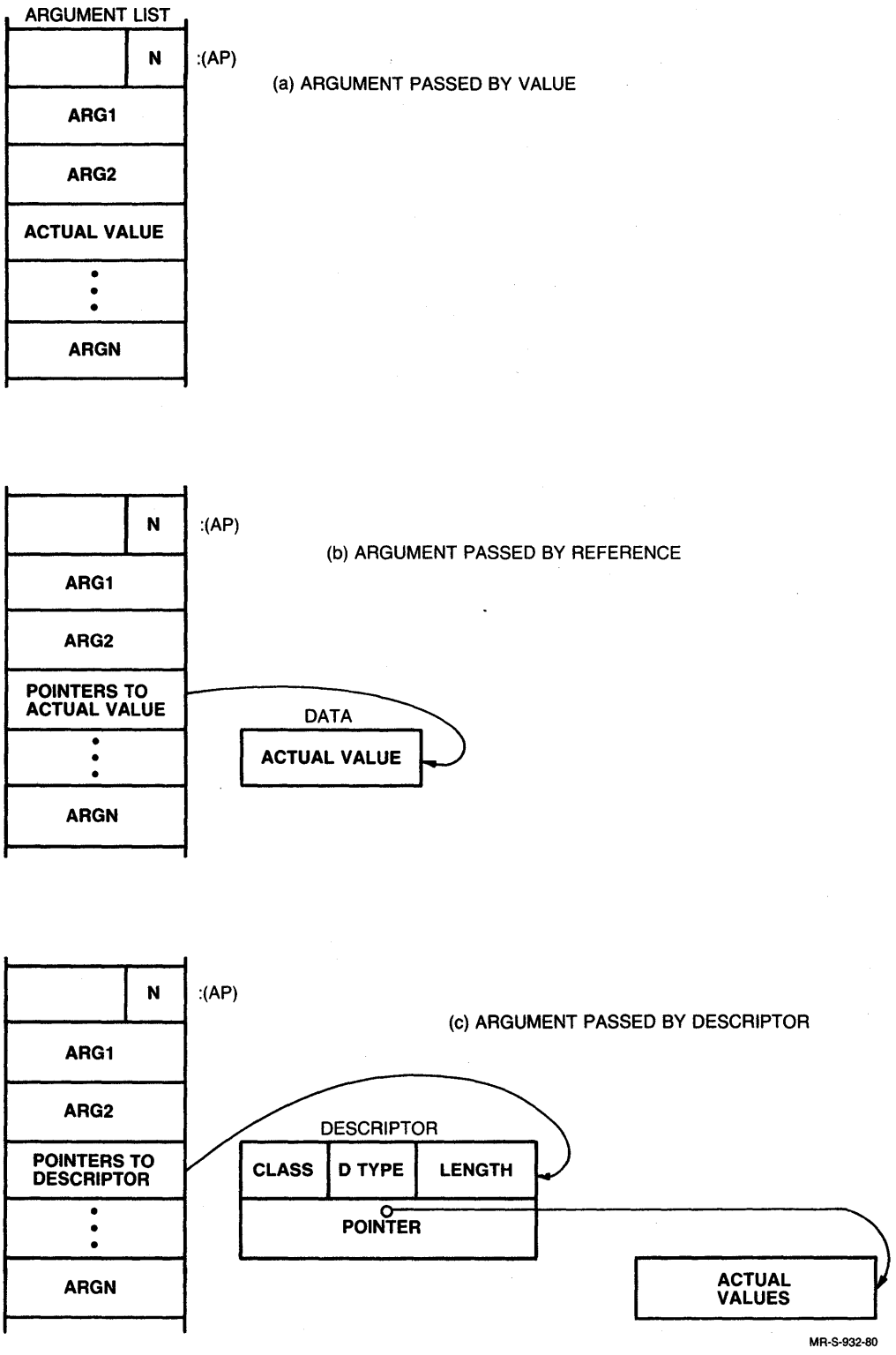
3. By Descriptor

   When you pass an argument by descriptor, the argument list entry contains the address of a VAX–11 descriptor of the argument. This passing mechanism is used to pass string data, which is more complicated than other data types. Descriptors include the following fields to describe data:

DSC$W_LENGTH	Data length in bytes
DSC$B_DTYPE	Data type
DSC$B_CLASS	Data class
DSC$A_POINTER	Address of the first byte of data

Figure 8–1 illustrates the argument-passing mechanisms.

Figure 8–1: Argument–Passing Mechanisms for Procedures



8.2.3 Data Types for Argument Passing

Each hardware data type can only be passed to and from a procedure with certain argument-passing mechanisms, as follows:

- String data can be passed *only* by descriptor.
- The other data types — longword, quadword, floating-point, double floating-point, G_floating, and H_floating — can be passed either by value or by reference, but *not* by descriptor.

Procedures that pass quadword output data return a quadword in $ZCALL as a pair of longwords. The data is arranged in the form n,m where n is the high longword and m is the low longword.

8.3 Calling Procedures from the DSM Language

In VAX–11 DSM, procedure calls are handled by a subsystem called the ZCALL Module. This module makes many details of procedure calling transparent. However, you must take certain steps to establish the proper links between the VAX procedure-handling mechanism and VAX–11 DSM. The following sections describe these steps.

8.3.1 The $ZCALL Function

The interface between a procedure, the ZCALL Module, and the DSM language is a DIGITAL–implemented extension to Standard MUMPS called *$ZCALL*. $ZCALL is a function; like other functions in the DSM language, it accepts one or more arguments and returns a value. The form of the function arguments and the value returned depend on the type of operation performed by the procedure called by $ZCALL.

The general format of $ZCALL is:

Value = $ZCALL(Callname,arg 1,...arg n)

You can abbreviate $ZCALL to $ZC.

Value	The value of the function, that is, the output produced by the procedure. Typically, you assign Value to a DSM variable. If the procedure produces multiple outputs, Value consists of concatenated strings separated by commas.
Callname	The name of the procedure you want to call. This name is established in the ZCALL Table entry for the procedure (see Section 8.3.2). Procedure names beginning with the percent sign (%) are reserved for DIGITAL–supplied $ZCALL functions, such as the VMS system services and Run–Time Library routines listed later in this chapter.
arg 1,...arg n	Inputs passed to the procedure for processing.

8.3.2 ZCALL Tables

VAX–11 DSM uses internal data structures called *ZCALL tables* to recognize procedure names and build the argument list for procedures called from the DSM language. ZCALL table source files (called "table source files" in the following discussion) are processed by the VAX–11 MACRO assembler and linked with the DSM image by the command procedure SYS$LIBRARY:DSMBUILD. While DSM is executing, the ZCALL Table and external procedures reside in memory.

Whenever you write a procedure (in another language) that you want to call through $ZCALL, you must make an entry for it in a ZCALL Table. You can make these entries by editing the table source file provided (ZCALLT.MAR, in the VAX/VMS directory SYS$LIBRARY) or by creating a new table source file. (See Section 8.3.3 for more information about creating your own table source files.) Then you compile the source file and relink the object module to the DSM interpreter as described in Section 8.5.

To make an entry for a procedure in ZCALLT.MAR, follow these steps:

1. Copy ZCALLT.MAR from SYS$LIBRARY into your default directory (with the DCL COPY command).
2. Open the file for editing with any VAX/VMS editor.
3. Type in the required data in the format shown in Figure 8–2 and described in the following sections.

Note that, whatever changes and additions you make to a table source file, every file must begin with the following line:

ZCALLINI

Every table source file must end with the following lines:

ZCALLFIN
.END

The .END command is the required final line for any VAX–11 MACRO source file.

Figure 8–2: Procedure Entry from a ZCALL Table Source File

```
;Procedure Description (any brief descriptive text)
ROUTINE         CALLNAME=name,LINKNAME=name,INPUTS=n,OUTPUTS=m
RETURN          {STATUS, VALUE, or IGNORED},TYPE=data type
INPUT           TYPE=data type,MECHANISM=argument passing mechanism,POSITION=I1
.
.
.
INPUT           TYPE=data type,MECHANISM=argument passing mechanism,POSITION=In
OUTPUT          TYPE=data type,MECHANISM=argument passing mechanism,POSITION=O1
.
.
.
OUTPUT          TYPE=data type,MECHANISM=argument passing mechanism,POSITION=Om
```

As shown in Figure 8–2, there are four types of lines in a procedure entry:

- A routine line
- A return line
- Input lines
- Output lines

These lines specify information about the procedure, as described in the following sections.

8.3.2.1 The Procedure Entry's Routine Line — In each procedure entry in a table source file, there is one ROUTINE line describing the procedure. As shown in Figure 8–2, the ROUTINE line contains four keyword entries:

- CALLNAME = name
- INPUTS = n
- OUTPUTS = m
- LINKNAME = name

These entries specify general information about the procedure, as follows:

CALLNAME	Specifies the name used in the callname field of the $ZCALL function. CALLNAME should be eight or fewer characters in length. (Names longer than eight characters are accepted, but only the first eight characters are used by VAX–11 DSM to recognize the callname.) If you use callnames longer than eight characters, make sure that the first eight characters are unique. VAX–11 DSM considers upper- and lower-case callnames to refer to different routines.
INPUTS	Specifies how many input arguments the procedure expects. The argument n must be a positive integer or zero. You must include as many input lines in the procedure entry as you specify here with the INPUTS keyword.
OUTPUTS	Specifies how many values the procedure produces as output. The argument m must be a positive integer or 0. You must include as many output lines in the procedure entry as you specify here with the OUTPUTS keyword.
LINKNAME	The name by which the VAX/VMS linker identifies a procedure when you link it to the interpreter. LINKNAME must be 30 or fewer characters in length. LINKNAME and CALLNAME can be the same, or they can be different.

8.3.2.2 The Procedure Entry's Return Line — The RETURN line indicates how the value of R0 and R1 returned by the called routine should be interpreted by $ZCALL. The RETURN line must precede all INPUT and OUTPUT lines. The four possible keywords used in return lines, of which no more than two are used in any line, are:

- TYPE = data type
- STATUS
- VALUE
- IGNORED

TYPE — Specifies the data type of the value returned in R0 and R1: LONG, QUAD, FLOATING, DOUBLE, or G_FLOATING. (Note that you cannot use the STRING or H_FLOATING data types.) The other keyword used in the return line can influence the data type that you must specify with TYPE. You include this keyword only if VALUE is also specified.

STATUS, VALUE, and IGNORED are mutually exclusive keywords. You can include only one of these in a return line, as follows:

STATUS — The value returned in R0 is interpreted as a status. STATUS implies a longword data type. You need not include the TYPE keyword if you specify STATUS. If you enter any data type other than LONG with STATUS, you cause an error.

You should use this keyword when calling VMS system services and some of the VMS Common Run–Time Library functions.

VALUE — R0 and R1 are interpreted as a function value. They are returned as the first (and typically the only) output of the $ZCALL function. The value of TYPE determines whether only R0 or both R0 and R1 will be considered, and what data type will be returned. (R0 and R1 are longwords, like all registers.) Use this keyword when you call FORTRAN functions and mathematical procedures from the VAX–11 Run–Time Library.

IGNORED — Indicates that the value returned in R0 will be ignored; that is, the status returned by $ZCALL is always success. Errors that occur in the called routine should be signalled. Use this keyword when you call FORTRAN subroutines.

If you omit the the RETURN line from the procedure entry, STATUS is assumed.

8.3.2.3 The Procedure Entry's Input Lines — Each INPUT line describes an argument passed to the procedure for processing. INPUT lines must precede all OUTPUT lines. You must include as many INPUT lines as the number specified with the INPUT keyword in the routine line.

The INPUT line contains a maximum of five keywords from the following list, as described below:

- TYPE = data type
- MECHANISM = argument-passing mechanism
- POSITION = argument position
- CONSTANT
- OPTIONAL
- DEFAULT
- VALUE = value
- REQUIRED

TYPE

Specifies the data type of the input arguments. VAX–11 DSM supports a subset of the data types defined in the VAX Procedure Calling Standard. The data types supported by VAX–11 DSM are listed below, along with the abbreviations that you use for them in a procedure entry:

• String data	STRING
• Longword	LONG
• Quadword	QUAD
• Floating-point	FLOATING
• Double-precision floating point	DOUBLE
• G_floating	G_FLOATING
• H_floating	H_FLOATING

If a procedure requires a byte or word data type, you can generally substitute a longword for it without affecting the performance of the procedure.

The data type you choose for an input argument depends on what the procedure does. For example, if you write a program to calculate the square root of a real number, the input data type should be floating-point or double floating. If you write a program that manipulates strings, you must use the string data type for input.

MECHANISM — Specifies the mechanism by which VAX–11 DSM passes the input argument to the procedure. This mechanism can be any of the three argument-passing mechanisms described in Section 8.2.2 that is appropriate for the data type. In a procedure entry, you specify these mechanisms as VALUE, REFERENCE, or DESCRIPTOR.

POSITION — Specifies the position of the input argument in the argument list of the called procedure.

The keywords CONSTANT, OPTIONAL, DEFAULT, and REQUIRED are mutually exclusive. If you include CONSTANT or DEFAULT, you must also specify VALUE. If you specify none of these three keywords, DSM assumes that the argument is required. You can specify the keyword REQUIRED, if you like, to make your ZCALL Table more readable.

CONSTANT — Indicates that a constant value, specified with the VALUE keyword, will be passed to the called routine at the specified position in the argument list. You use CONSTANT when you know that the routine being called must always receive a particular value to behave in the way that you want, and you know that you will never need to change that value.

OPTIONAL — Indicates that if this input is present in the $ZCALL call, the corresponding parameter will be passed to the routine being called; however, if this input is omitted from the $ZCALL function reference, no parameter will be passed to the routine.

Note that optional input parameters can only be omitted if their position is last in the argument list of the CALLG, as well as last in the list of INPUT lines for this procedure entry.

DEFAULT — Indicates that the called procedure uses the default value for the argument, specified with the VALUE keyword, at this position unless a value is explicitly specified in the DSM routine's $ZCALL function reference. Note that default parameters must appear last in the list of INPUT lines in this procedure entry.

VALUE — Specifies the value to be used as a constant or as a default for the argument, depending on whether the CONSTANT keyword or the DEFAULT keyword is specified. If the value is a special character in VAX–11 MACRO, it must be enclosed in angle brackets (<>). Special characters, which include blanks, commas, and semicolons, are listed in the *VAX–11 MACRO Language Reference Manual.* You can specify the null string as a value by entering an empty pair of angle brackets (<>).

REQUIRED — Indicates that a value must be passed to the procedure in the $ZCALL call. This is the default state for INPUT lines unless CONSTANT, OPTIONAL, or DEFAULT is specified. You need not explicitly include the keyword REQUIRED.

Depending on the procedure, there can be zero or more INPUT lines. You use more than one INPUT line when the procedure takes more than one input argument. The INPUT lines can be placed in the procedure entry in any order; the value assigned with the POSITION keyword, rather than the location of the INPUT line in the procedure entry, determines the order in which the procedure accepts input arguments. However, arguments must be ordered in the $ZCALL call in the sequence used for their INPUT lines in the procedure entry. The first INPUT line corresponds to arg 1, the second to arg 2, and so on.

The only exceptions to this ordering rule are caused by INPUT CONSTANT lines, which do not correspond to an argument in the $ZCALL function reference. Therefore, if INPUT CONSTANT is the fourth INPUT line in a procedure entry, arg3 in the $ZCALL function reference corresponds to the third INPUT line, but arg4 corresponds to the fifth INPUT line.

8.3.2.4 The Procedure Entry's Output Lines — Each OUTPUT line describes a value produced by the procedure and the way the value is passed to VAX–11 DSM. OUTPUT lines cannot precede INPUT lines in the procedure entry.

The OUTPUT line contains five keywords:

- TYPE = data type
- MECHANISM = argument-passing mechanism
- POSITION = argument position
- DUMMY
- REQUIRED

TYPE — Specifies the data type of the output arguments: STRING, LONG, QUAD, FLOATING, DOUBLE, G_FLOATING, or H_FLOATING.

MECHANISM — Specifies the mechanism by which the output value is passed: either REFERENCE or DESCRIPTOR. (Note that VALUE is not allowed.)

POSITION — Specifies the position of the output argument in the argument list of the called procedure.

DUMMY — Indicates that, although an output parameter is passed to the routine and a value is received, that value is not returned as part of the value of the $ZCALL function. You use this keyword if you know that there are some values returned by the called routine that you will never want to examine or use. DUMMY also helps you keep the length of the returned value within the 255 character limit for the returned value of a $ZCALL function.

REQUIRED — Indicates that a value is returned to DSM through this parameter. This is the default state for OUTPUT lines unless DUMMY is specified, so you need not explicitly include REQUIRED.

There can be zero or more OUTPUT lines in a procedure entry. You use multiple OUTPUT lines when the procedure returns more than one value. Procedures that produce more that one value return them to you in $ZCALL as a concatenated string separated by commas. The order of the OUTPUT lines determines the position of the values returned in this string. This string must not be longer than 255 characters.

8.3.3 Multiple ZCALL Tables

You can link more than one file containing ZCALL Table procedure entries with the VAX–11 DSM interpreter. Using multiple ZCALL Tables allows you to select the procedures to which access is possible from a DSM application.

You assign file names to the table source files that you create; since these are VAX–11 MACRO source files, the default file type is .MAR. When you run DSMBUILD.COM, as described in Section 8.5, you specify the names of the table source files. If you do not specify a name, ZCALLT.MAR is used.

To help you set up interface routines for the procedures that you call, DSM provides the ZCEXAMPLE.MAR file, which contains sample $ZCALL interface routines. Look at this file to see how to construct $ZCALL interfaces, and at ZCALLT.MAR to see how to construct procedure entries. Both files reside in the VAX/VMS directory SYS$LIBRARY.

The VAX–11 DSM system software also includes (in SYS$LIBRARY) a compiled version of ZCEXAMPLE.MAR called ZCEXAMPLE.OLB.

To call the procedures in ZCEXAMPLE.MAR, follow these steps:

1. Copy SYS$LIBRARY:ZCALLT.MAR to your default directory.
2. Assign LNK$LIBRARY to SYS$LIBRARY:ZCEXAMPLE.OLB.
3. Execute DSMBUILD.COM.

NOTE

When you install VAX–11 DSM, the binary ZCALL Table linked to the DSM image contains only DIGITAL–supplied procedure entries.

8.4 Calling User–Defined Functions

A user-defined function is a program written in a language that produces native-mode code that can be called as an external procedure through $ZCALL. The following sections provide some guidelines for writing user-defined functions in VAX–11 MACRO and VAX–11 FORTRAN.

To see the list of user-defined functions that can be called through $ZCALL on your system, use the DSM utility ^%ZD.

8.4.1 Writing User–Defined Functions in VAX–11 MACRO

Keep the following points in mind when writing a user-defined function in VAX–11 MACRO:

1. VAX–11 MACRO can pass arguments by any of the three methods defined in the VAX Procedure Calling Standard.
2. When the CALLG instruction is executed by the DSM ZCALL module, general register 12, the AP (argument pointer) points to a valid argument list. For an argumentless procedure, the argument list consists of a single longword containing the value 0.
3. You can pass strings to and from a MACRO program using either of two techniques, as shown in Section 8.4.2.

8.4.2 Passing String Arguments from User–Defined Functions

You can pass output strings from user-defined functions in two ways:

1. Pass the output string by descriptor as an empty dynamic descriptor.

   When you use this method, the program you want to call must provide for the virtual memory space required to store the output string. Thus, you must use the Run–Time Library procedure LIB$SGET1_DD in your program to allocate the necessary virtual memory for this string.

   Alternatively, you can use the Run–Time Library procedure LIB$SCOPY_DXDX to copy the resultant string to the empty dynamic output string.

2. Pass the input string by descriptor and have the procedure operate on the input string.

   When you use this method, the program you want to call does not have to allocate space for the output string.

To understand the difference between the two ways to pass string arguments, consider the following two VAX–11 MACRO programs and their respective ZCALL Table entries:

Program 1

```
;+
; CONVERT UPPER CASE TO LOWER CASE
;-

            .ENTRY CONV2,^M<R2,R3,R4>
            MOVL 8.(AP),R5   ; GET ADDRESS OF OUTPUT DESCRIPTOR
            PUSHL R5    ; PUSH ADDRESS OF OUTPUT DESCRIPTOR
            MOVL 4(AP),R2   ; ADDRESS OF DESCRIPTOR
            MOVZWL (R2),R3    ; GET LENGTH OF STRING
            PUSHL R2    ; PUSH ADDRESS OF NEEDED LENGTH
            CALLS #2,G^LIB$SGET1_DD ; GET SPACE FOR OUTPUT
            MOVL 4(R2),R4   ; POINT TO INPUT STRING
            MOVL 4(R5),R5   ; POINT TO OUTPUT STRING
100$:       CMPB (R4),#^A/A/   ; IS IT UPPERCASE ?
            BLSS 150$    ; NO
            CMPB (R4),#^A/Z/
            BGTR 150$
            ADDB3 #<^A/a/-^A/A/>,(R4)+,(R5)+ ; CONVERT
            BRB 200$
150$:       MOVB (R4)+,(R5)+   ; COPY BYTE UNTOUCHED
200$:       SOBGTR R3,100$    ; LOOP
            MOVL #1,R0    ; SET SUCCESS STATUS
            RET     ; DONE
            .END
```

The ZCALL table procedure entry for Program 1 is:

```
; CONVERT UPPER CASE TO LOWER CASE

ROUTINE     CALLNAME=CONV2,LINKNAME=CONV2,INPUTS=1,OUTPUTS=1
RETURN      STATUS
INPUT       TYPE=STRING,MECHANISM=DESCRIPTOR,POSITION=1
OUTPUT      TYPE=STRING,MECHANISM=DESCRIPTOR,POSITION=2
```

Program 2

```
;+
; CONVERT UPPER CASE TO LOWER CASE IN PLACE
;-

            .ENTRY CONV,^M<R2,R3,R4>
            MOVL 4(AP),R2   ; ADDRESS OF DESCRIPTOR
            MOVZWL (R2),R3    ; GET LENGTH OF STRING
            MOVL 4(R2),R4   ; POINT TO STRING
100$:       CMPB (R4)+,#^A/A/   ; IS IT UPPERCASE ?
            BLSS 200$    ; NO
            CMPB -1(R4),#^A/Z/
            BGTR 200$
            ADDB #<^A/a/-^A/A/>,-1(R4) ; CONVERT
200$:       SOBGTR R3,100$    ; LOOP
            MOVL #1,R0    ; SET SUCCESS STATUS
            RET     ; DONE
            .END
```

The ZCALL table procedure entry for Program 2 is:

```
; CONVERT UPPER CASE TO LOWER CASE IN PLACE

ROUTINE     CALLNAME=CONV,LINKNAME=CONV,INPUTS=1,OUTPUTS=0
RETURN      STATUS
INPUT       TYPE=STRING,MECHANISM=DESCRIPTOR,POSITION=1
```

Programs 1 and 2 perform the same task. When called through $ZCALL, each accepts a string input. If the string consists of uppercase characters, they convert it to lowercase. However, VAX–11 DSM builds a different argument list for each procedure, and each procedure produces different side effects.

Program 1 passes the output string by descriptor, and thus uses LIB$SGET1_DD to allocate the virtual memory necessary to store it. When you call this program through $ZCALL, the value of the function equals the string stored in the output descriptor. For example:

```
>S X="ABCD"
>S Y=$ZC(CONV2,X)
>ZW
X=ABCD
Y=abcd
```

Program 2 does not pass an output string, so its procedure entry specifies 0 outputs. Instead, the program operates on the input string and passes the modified input back in $ZCALL, as shown in the following example:

```
>S X="ABCD"
>S Y=$ZC(CONV,X)
>ZW
X=abcd
Y= ;Value of Y is null
```

8.4.3 Writing a User–Defined Function in VAX–11 FORTRAN

Keep the following points in mind when writing a user-defined function in VAX–11 FORTRAN:

1. If the procedure is a FORTRAN subroutine, you must specify RETURN IGNORED in the procedure entry, as shown in the first FORTRAN example below.
2. If the procedure is a FORTRAN numeric function, you must specify RETURN VALUE in the procedure entry, as in the second FORTRAN example.
3. The program must pass and receive arguments by reference or by descriptor only. By default, VAX–11 FORTRAN passes arguments by reference. For more information about passing string arguments, refer to the *VAX–11 FORTRAN User's Guide.*

The following example shows a FORTRAN program in the format necessary to call it from the DSM language. This program calculates the roots of a quadratic equation. Following the program are its procedure entry and several examples of its use with $ZCALL.

```
C       This program calculates the roots of a quadratic equation
C       A,B, and C are input quadratic coefficients.
C       FLAG = 1 if roots are real, 0 if imaginary.

        SUBROUTINE QUADX(A,B,C,ROOT1,ROOT2,FLAG)
        INTEGER FLAG
        REAL*8 A,B,C,ROOT1,ROOT2
        D = (B**2 - 4*A*C)
        IF (D .LE. 0.0) GOTO 100
        ROOT1 = (-B + SQRT(D))/(2*A)
        ROOT2 = (-B - SQRT(D))/(2*A)
        FLAG = 1      ! ROOTS ARE REAL
        RETURN
100     FLAG = 0      ! ROOTS ARE IMAGINARY
        RETURN
        END
```

The ZCALL table procedure entry for this program is:

```
; QUADRATIC FORMULA

ROUTINE     CALLNAME=QUAD,LINKNAME=QUADX,INPUTS=3,OUTPUTS=3
RETURN      IGNORED
INPUT       TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=1
INPUT       TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=2
INPUT       TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=3
OUTPUT      TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=4
OUTPUT      TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=5
OUTPUT      TYPE=LONG,MECHANISM=REFERENCE,POSITION=6
```

Note that output 3 (in position 6) is a longword, because output 3 can be only 1 (for real roots) or 0 (for imaginary roots).

The following examples show $ZCALL function references for this FORTRAN program:

```
>S A=1,B=0,C=-1/1
>W $ZC(QUAD,A,B,C)/1
1,-1,1

>S L=43
>W $ZC(QUAD,4,L,9)
-,21354437,-10,536456,1

>W $ZC(QUAD,1,-3,8)
0,0,0
```

The following is a FORTRAN function with its procedure entry:

```
C       This function returns the absolute value of its input

        REAL*8 FUNCTION ABSX(ORIG)
        REAL*8 ORIG
        ABSX = ABS(ORIG)
        RETURN
        END
```

The ZCALL table procedure entry for this procedure is:

```
; ZCALL TO RETURN ABSOLUTE VALUE

ROUTINE CALLNAME=ABSX,LINKNAME=ABSX,INPUTS=1,OUTPUTS=0
RETURN VALUE,TYPE=DOUBLE
INPUT TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=1
```

The following examples show $ZCALL function references for this procedure:

```
> S A=67.56,X=-987.54
> S B=$ZC(ABSX,A),Y=$ZC(ABSX,X)
> ZW
A=67.56
B=67.56
X=-987.54
Y=987.54
```

8.5 Linking Procedures to the Interpreter

To call procedures from the DSM language, you must link them to the VAX–11 DSM interpreter. Follow these steps:

1. Copy ZCALLT.MAR from SYS$LIBRARY to your default directory.
2. Edit ZCALLT.MAR to make the necessary procedure entries (as described in the preceding sections).
3. Create any additional table source files that you want.
4. Assign the logical names LNK$LIBRARY_1 through LNK$LIBRARY_n to the object libraries needed, unless the procedure entries refer to procedures in the System Service Library or the Run–Time Library.
5. Run the DSMBUILD.COM command file, supplied with VAX–11 DSM. DSMBUILD.COM resides in SYS$LIBRARY and can be run by all VAX–11 DSM users without special privileges.

Running DSMBUILD.COM builds a new version of the DSM image (in your default VMS directory or any directory you specify) that includes the entry points for all procedures to be called through $ZCALL. The output produced by this command procedure is a file called DSM.EXE (or DSMD.EXE if the VAX/VMS symbolic debugger is included).

DSMBUILD.COM does the following:

1. Locates the directory in which your table source files reside.
2. Assembles all ZCALL Tables from the source files.
3. Links all ZCALL Tables (the object modules produced by the assembler) to the DSM image.

8.5.1 Linking User–Defined Functions

To link user-defined functions (MACRO and FORTRAN procedures), you must first create a VAX/VMS object library (file type .OLB) to catalog their object modules (file type OBJ). To create a VMS object library, use the DCL LIBRARY command qualified by the /CREATE command qualifier. This command and the process for creating object module libraries is described in detail in the *VAX/VMS Command Language User's Guide.*

After you catalog your object modules in this library, you *must* define the logical name LNK$LIBRARY and equate it with the full file specification of your object module library. All components of the file specification must be included when you make the logical name assignment.

If LNK$LIBRARY is already in use, define LNK$LIBRARY_1 through LNK$LIBRARY_999, as needed.

After you complete these tasks (and make the necessary table source files), execute DSMBUILD.COM. DSMBUILD.COM locates the directory in which your object module library resides (by translating LNK$LIBRARY), assembles the table source files, and links them to the DSM image.

8.5.2 Logical Names Used in Linking

For convenience, you can define any of the following logical names to tailor the linking process to your particular needs:

DSM$LIBRARY	Directory and device of VAX–11 DSM object module libraries, macro libraries, and command files. If undefined, DSMBUILD.COM uses SYS$LIBRARY.
DSM$TARGET	Directory and device in which you want the new DSM image to be built. If undefined, DSMBUILD.COM uses your default directory and device.
DSM$ZCALL	Directory and device in which your table source files reside. If undefined, DSMBUILD.COM uses your default directory and device.
LNK$LIBRARY or LNK$LIBRARY_n (n=1 – 999)	The complete file specification of an object library of user-written programs to be called through @ZCALL. This logical name must be defined to link any user-written programs to the DSM image.

The following examples show how you run DSMBUILD. The first example shows the command line for linking the default table source file ZCALLT.MAR. The second example shows the command line for linking DSM with several named table source files.

```
@SYS$LIBRARY:DSMBUILD

@SYS$LIBRARY:DSMBUILD ZCALLTBL1, ZCALLTBL2, ZCALLTBL3
```

8.5.3 Debugging

Your new version of the DSM image may contain bugs. To help you debug the code that you are linking, DSMBUILD.COM can be invoked with the VAX/VMS Symbolic Debugger activated. This debugger allows you to monitor the performance of software and isolate problems quickly.

The following examples show how you run DSMBUILD.COM with the debugger activated. To link DSM with the default ZCALL table source file, ZCALLT.MAR, plus the VAX/VMS symbolic debugger, enter the following:

```
@SYS$LIBRARY:DSMBUILD DEBUG
```

To link DSM, the debugger, and other table source files, enter the following:

```
@SYS$:LIBRARY:DSMBUILD DEBUG ZCALLTBL4,ZCALLTBL5,ZCALLTBL6
```

It is not recommended that you call a table source file DEBUG.MAR. However, if you do run DSMBUILD to link a table source file named DEBUG.MAR, you must include a null argument ("") before the file name in the command line, or include the word DEBUG twice, as shown in the following examples:

```
@SYS$LIBRARY:DSMBUILD "" DEBUG
@SYS$LIBRARY:DSMBUILD DEBUG DEBUG
```

When you link in the debugger, DSMBUILD.COM produces an executable image called DSMD.EXE that includes the debugger. To run DSMD.EXE subsequently, you must define the DSM command to invoke DSMD.EXE, as shown in the following example:

```
$DSM:==$Your device:[Your directory]DSMD.EXE
```

You may want to include this line in your VAX/VMS log-in command file, if you plan to run DSM with the debugger enabled.

Once this definition is made, executing the DSM command invokes DSMD.EXE in debug mode.

See the *VAX–11 Symbolic Debugger Reference Manual* for instructions in using the symbolic debugger, or the *VAX–11 MACRO Reference Manual* for a condensed description.

8.5.4 Reinstalling the DSM Image

After you link your procedures with DSMBUILD.COM and are satisfied with the operation of the DSM image, you must copy DSM.EXE to SYS$SYSTEM and reinstall it as a VAX/VMS known image to make the new version of the interpreter available to the system. Thereafter, anyone running the DSM image can call these procedures directly through $ZCALL. Section 10.4 describes in detail the procedure for installing known images.

8.5.5 Recompiling Stored Routines

If you add new procedure entries for general use on your system, and also modify the placement of previously-installed procedure entries, you must recompile all DSM routines that refer to the relocated entry points. The ^%REPLACE utility allows you to accomplish this task quickly. To use ^%REPLACE, invoke DSM and execute the utility, as shown in the following example. Enter the names of routines that need recompilation where prompted. Then enter RET to exit the utility.

```
> D ^%REPLACE

Replace routines to force recompilation

routine(s) ? > routine-name
routine(s) ? > RET
```

The ^%REPLACE utility scans your current DSM routine directory and performs a ZLOAD followed by a ZSAVE for each routine specified, forcing recompilation of the routine by the interpreter.

You must also rebuild any files containing DSM routines that are to be mapped as virtual memory sections. Use the ^%RBUILD utility, described in Section 7.4.2.12.

8.6 Supplied VAX/VMS Services and Routines

VAX–11 DSM supplies procedure entries for a number of VAX/VMS system services and Run–Time Library routines that can be called from the DSM language. VAX/VMS uses these services and routines to control resources available to processes, to provide for communication among processes, and to perform basic operating system functions, such as coordinating input/output operations.

To see a list of $ZCALL procedure entries supplied with VAX–11 DSM, use the DSM utility LIB^%ZD.

Figure 8–3 shows how you enter a $ZCALL function reference for VAX/VMS services.

Figure 8–3: Supplied VAX/VMS Services

%DAYS

SET X=$ZC(%DAYS)

Returns the number of days since a system zero date.
Calls the LIB$DAY library routine.

%CRELOG

SET dummy=$ZC(%CRELOG,arg1,arg2{,arg3})

Creates a user-mode logical name (default) or a supervisor-mode logical name.

arg1 = the logical name to create
arg2 = the equivalence string for the logical name
arg3 = "SUPERVISOR" or "PROCESS" means create logical name in supervisor mode. Otherwise create in user mode.

Calls (through an interface routine) the LIB$SET_LOGICAL library routine or the SYS$CRELOG system service.

%DELLOG

SET dummy=$ZC(%DELLOG,arg1{,arg2})

Deletes a logical name. Default is user mode. To delete a supervisor-mode name, specify "SUPERVISOR". Note that this deletes a supervisor-mode name only if a user-mode logical name by that name does not exist.
Always returns the null string, even if the logical name is not defined.

arg1 = logical name to delete

arg2 = "SUPERVISOR" or "PROCESS". Use to delete logical names created with the corresponding values in %CRELOG.

Calls (through an interface routine) the LIB$DELETE_LOGICAL library routine or the SYS$DELLOG system service.

%DELSYM

SET dummy=$ZC(%DELSYM,arg1{,arg2})

Deletes a local (default) or a global CLI symbol.

arg1 = the symbol to delete
arg2 = flags
if omitted or equal to 1,local symbol
if equal to 2, global symbol

Calls library routine LIB$DELETE_SYMBOL.

%GETMSG

SET X=$ZC(%GETMSG,arg1)

Returns the message text associated with a VMS status code.

arg1 = the VMS status code (in decimal)

Calls the LIB$SYS_GETMSG library routine.

%GETSYI

SET X=$ZC(%GETSYI)

Returns the following system information: VMS version identification, CPU type, system identification register value.

Calls (through an interface routine) the SYS$GETSYI system service.

%GETSYM

SET X=$ZC(%GETSYM,arg1,arg2)

Returns the value of a CLI symbol.

arg1 = the symbol to translate
arg2 = flags
if omitted or equal to 1,local symbol
if equal to 2, global symbol

Calls the LIB$GET_SYMBOL library routine.

%SETSYM

SET dummy=$ZC(%SETSYM,arg1,arg2{,arg3})

Sets a local (default) or global CLI symbol to a value.

arg1 = the symbol to set
arg2 = the symbol's value
arg3 = flags
if omitted or equal to 1,local symbol
if equal to 2, global symbol

Calls the LIB$SET_SYMBOL library routine.

%SPAWN

SET dummy=$ZC(%SPAWN{arg1,arg2,arg3,arg4,arg5})

Spawns a VMS subprocess and, optionally, executes a DCL command.

arg1 = DCL command. If a specified command fails, the $ZCALL fails and can be handled by setting $ZTRAP. If no command specified, $ZCALL always succeeds.

arg2 = an equivalence name to associate with SYS$INPUT in the subprocess

arg3 = an equivalence name to associate with SYS$OUTPUT and SYS$ERROR in the subprocess

arg4 = flags (sum for multiple flags)

1 main process and subprocess run at same time
2 no DCL symbols passed to subprocess
4 no logical names passed to subprocess

arg5 = subprocess name

Calls (through an interface routine) the LIB$SPAWN library routine.

%TRNLOG

SET X=$ZC(%TRNLOG,arg1)

Returns the translation of a logical name. If the logical name is not defined, returns the null string.

arg1 = logical name to translate

Calls (through an interface routine) the LIB$SYS_TRNLOG library routine.

%MOUNT

SET X=$ZC(%MOUNT,arg1,arg2{,arg3,arg4})

Mounts a magnetic tape. Mount failures are indicated by $ZCALL failures.

arg1 = the magnetic tape device name(s)
arg2 = the magnetic device label(s) (may be the null string)
arg3 = mount options, specified as in the *DCL Command Language User's Guide*. Each option must start with a slash (/), and no embedded spaces are allowed except within quotes on the /COMMENT qualifier.
arg4 = the logical name to equate to the device

Calls (through an interface routine) the SYS$MOUNT system service.

%DISMOUNT

SET X=$ZC(%DISMOUNT,arg1{,arg2})

Dismounts a magnetic tape device. Dismount failures are indicated by $ZCALL failures.

arg1 = the device name to dismount; only one name allowed
arg2 = qualifiers as specified in the *VAX/VMS Command Language User's Guide*

Calls (through an interface routine) the SYS$DISMOU system service.

%ENABLCTRL

SET X=$ZC(%ENABLCTRL,arg1)

Sets recognition of control characters by the VMS terminal handler. Returns the previous state of that control character as a DSM truth value (0 means off, 1 means on)

arg1 = the character to enable (for example, $CHAR(25) means CTRL/Y)
In VMS V3.0, only two control characters may be set or disabled: CTRL/T and CTRL/Y, $CHAR(20) and $CHAR(25) respectively.

Calls (through an interface routine) the LIB$ENABLE_CTRL library routine.

%DSABLCTRL

SET X=$ZC(%DSABLCTRL,arg1)

Stops recognition of control characters by the VMS terminal handler.
Returns the previous state of that control character as a DSM truth value (0 means off, 1 means on)

arg1 = the character to disable (for example, $CHAR(25) means CTRL/Y)
In VMS V3.0 only two control characters may be set or disabled: CTRL/T and CTRL/Y, $CHAR(20) and $CHAR(25), respectively.

Calls (through an interface routine) the LIB$DISABLE_CTRL library routine.

%GETDVI

SET X=$ZC(%GETDVI,arg1,arg2)

Returns device characteristics.

arg1 = the device specifier
arg2 = the mnemonic for the device characteristic (as specified for the lexical function F$GETDVI). Only one device characteristic may be specified.

Calls (through an interface routine) the SYS$GETDVI system service.

%GETJPI

SET X=$ZC(%GETJPI,arg1,arg2)

Returns job and process information.

arg1 = the process identification (pid) in decimal
arg2 = the mnemonic for the device characteristic (as specified for the lexical function F$GETJPI). Only one device characteristic may be specified.

Calls (through an interface routine) the SYS$GETJPI system service.

8.7 Calling Mathematical, Text–Related, and Other Functions

VAX–11 DSM provides procedure entries for a variety of mathematical functions and text-manipulation functions from the VAX–11 Run–Time Library, as well as some functions for converting time formats and creating files. The following sections describe how you call these procedures using $ZCALL function references.

8.7.1 Mathematical Functions

Figure 8–4 lists the mathematical functions from the VAX–11 Run–Time Library for which ZCALL Table entries are provided by VAX–11 DSM. Note that all angles are interpreted in degrees, not in radians.

Figure 8–4: Mathematical Functions

%SIN

SET X=$ZC(%SIN,arg1)

Returns the SIN of the argument.

arg1 = the angle expressed in degrees

Calls Run–Time Library routine MTH$DSIND.

%COS

SET X=$ZC(%COS,arg1)

Returns the COS of the argument.

arg1 = the angle expressed in degrees

Calls Run–Time Library routine MTH$DCOSD.

%TAN

SET X=$ZC(%TAN,arg1)

Returns the TAN of the argument.

arg1 = the angle expressed in degrees

Calls Run–Time Library routine MTH$DTAND.

%ARCSIN

SET X=$ZC(%ARCSIN,arg1)

Returns the ARCSIN of the argument in degrees.

arg1 = the angle's sine

Calls Run–Time Library routine MTH$DASIND.

%ARCCOS

SET X=$ZC(%ARCCOS,arg1)

Returns the ARCCOS of the argument in degrees.

arg1 = the angle's cosine

Calls Run–Time Library routine MTH$DACOSD.

%ARCTAN

SET X=$ZC(%ARCTAN,arg1)

Returns the ARCTAN of the argument in degrees.

arg1 = the angle's tangent

Calls Run–Time Library routine MTH$DATAND.

%EXP

SET X=$ZC(%EXP,arg1)

Returns the exponential of the argument (the number "e" raised to the power of the argument).

Calls Run–Time Library routine MTH$DEXP.

%POWER

SET X=$ZC(%POWER,arg1,arg2)

Raises the first argument to the power of the second.

Calls Run–Time Library routine OTS$POWDD.

%LOG

SET X=$ZC(%LOG,arg1)

Returns the natural (base "e") logarithm of the argument.

Calls Run–Time Library routine MTH$DLOG.

%LOG10

SET X=$ZC(%LOG10,arg1)

Returns the common (base 10) logarithm of of the argument.

Calls Run–Time Library routine MTH$DLOG10.

%MAX

SET X=$ZC(%MAX,arg1,arg2{,arg3,...})

Returns the largest number in the list of arguments.
A minimum of two and a maximum of 100 arguments may be specified.

Calls Run–Time Library routine MTH$DMAX1.

%MIN

SET X=$ZC(%MIN,arg1,arg2{,arg3,...})

Returns the smallest number in the list of arguments.
A minimum of two and a maximum of 100 arguments may be specified.

Calls Run–Time Library routine MTH$DMIN1.

%SQRT

SET X=$ZC(%SQRT,arg1)

Returns the square root of the argument.

Calls Run–Time Library routine MTH$DSQRT.

8.7.2 Text–Manipulation Functions

VAX–11 DSM provides procedure entries for various functions from the VAX–11 Run–Time Library that translate and examine character strings and pages of text. Figure 8–5 lists the $ZCALL function references for these functions.

Figure 8–5: Text–Manipulation Functions

%ASCEBC

SET X=$ZC(%ASCEBC,arg1)

Translates an ASCII string to EBCDIC.

arg1 = the string to translate

Calls the LIB$TRA_ASC_EBC library routine.

%EBCASC

SET X=$ZC(%EBCASC,arg1)

Translates an EBCDIC string to ASCII.

arg1 = the string to translate

Calls the LIB$TRA_ASC_EBC library routine.

%TRANSLATE

SET X=$ZC(%TRANSLATE,arg1,arg2,arg3)

Translates matched characters in a string.

arg1 = source string to translate
arg2 = match table
arg3 = translation table

Calls the STR$TRANSLATE library routine.

%UPCASE

SET X=$ZC(%UPCASE,arg1)

Translates a string to upper case.

arg1 = the string to translate

Calls the STR$UPCASE library routine.

%UPCASEQ

SET X=$ZC(%UPCASEQ,arg1)

Translates a string to upper case, except inside quoted strings.

arg1 = the string to translate

%BASEDIT

SETX=$ZC(%BASEDIT,arg1,arg2)

Edits a string using the BASIC (language) EDIT$ function.

arg1 = the string to edit
arg2 = an integers or a sum of integers, as described in the *VAX–11 Run–Time Library Language Support Reference Manual*

Calls the BAS$EDIT library routine.

%CURRENCY

SET X=$ZC(%CURRENCY)

Returns the local currency symbol.

Calls the LIB$CURRENCY library routine.

%DIGISEP

SET X=$ZC(%DIGISEP)

Returns the local symbol used to separate digit groups.

Calls the LIB$DIGIT_SEP library routine.

%LPLINES

SET X=$ZC(%LPLINES)

Returns the number of lines per printer page.

Calls the LIB$LP_LINES library routine.

%RADIXPT

SET X=$ZC(%RADIXPT)

Returns the radix point symbol.

Calls the LIB$RADIX_POINT library routine.

8.7.3 Calling File–Related Functions

VAX–11 DSM supplies procedure entries for two functions dealing with files and file information, as listed below in Figure 8–6:

Figure 8–6: File Functions

%GETFILE

SET X=$ZC(%GETFILE,arg1,arg2)

Returns file information.

arg1 = file specification
arg2 = any of the options supported by the lexical function F$FILE_ATTRIBUTES

%FDLCREATE

SET dummy=$ZC(%FDLCREATE,arg1{,arg2,arg3})

Creates a file using a File Definition Language (FDL) file.

arg1 = FDL file name
arg2 = file name (optional)
arg3 = default file name (optional)

8.7.4 Calling DSM–Specific Functions

In addition to the services listed above, VAX–11 DSM provides procedure entries for four time services that are particularly useful for DSM programmers. Figure 8–7 lists these functions.

Figure 8–7: Time Services

%CDATASC

SET X=$ZC(%CDATASC,arg1,arg2)

Converts and returns the first part of the $HOROLOG string to an ASCII date.

arg1 = the date in $HOROLOG format (the number of days since the DSM starting date)

arg2 = convert switch
- 1 convert to mm/dd/yy, where mm, dd, and yy are all numeric
- 2 convert to dd-mmm-yy, where dd and yy are numeric and mmm is the three-character abbreviation for the month
- 3 convert to the VMS date format, dd-mmm-yyyy, where dd and yyyy are numeric and mmm is the three-character abbreviation for the month. Note that the year is always four digits long, and the date is two characters, starting with a space for dates before the tenth of the month.

$CDATNUM

SET X=$ZC($CDATNUM,arg1)

Converts and returns date to the number of days since the DSM starting date (in $HOROLOG format).

arg1 = the date in any of the formats returned by %CDATASC (see above)

%CTIMASC

SET X=$ZC(%CTIMASC,arg1,arg2)

arg1 = the time in $HOROLOG format (the number of seconds since the previous midnight)

arg2 = convert switch
- 1 convert to hh:mm:ss AM or hh:mm:ss PM
- 2 convert to hh:mm:ss in 24–hour format
- 3 convert to the VMS time format, hh:mm:ss.cc

%CTIMNUM

SET X=$ZC(%CTIMNUM,arg1)

Converts and returns the time as the number of seconds since the previous midnight (in $HOROLOG format)

arg1 = the time in any of the formats returned by %CTIMASC (see above)

8.8 ZCALL Error Processing

If a procedure called through $ZCALL returns an error, and the RETURN line for the procedure specifies STATUS (see Section 8.3.2.2), DSM displays one of the following errors, followed by the status returned by the called procedure:

%DSM-E-ZCFATAL, Fatal error during $ZCALL
%DSM-E-ZCERROR, Error during $ZCALL
%DSM-E-ZCWARNING, Warning during $ZCALL

If an error generates a signal, DSM displays the following message:

%DSM-E-ZSIGNAL, condition signalled by the $ZCALL routine

DSM also places this message in $ZE for signals.

If a ZCFATAL error occurs in Programmer Mode, control returns to the DSM interpreter. If a ZCFATAL error occurs in Application Mode, it causes DSM image rundown and transfers control to the CLI.

If a procedure generates a fatal error and error trapping is enabled, that is, you set $ZT, VAX-11 DSM traps the error and puts the %DSM-E-ZCFATAL error message string in $ZE.

Refer to the *VAX Architecture Handbook* for details about procedure call condition handling.

8.9 Extended Example of Using $ZCALL

Figure 8-8 is an extended example showing how you create a procedure in a language other than DSM and call it through a $ZCALL function reference.

The example shows:

1. The creation and assembly of a VAX-11 FORTRAN procedure called ABSX.FOR.
2. The creation and assembly of a VAX-11 MACRO table source file called ZCALLTABS.MAR.
3. The invoking of DSMBUILD to link the table source file with the DSM image.
4. The redefinition of the DSM command to invoke a new version of DSM.
5. A DSM session in which the FORTRAN procedure is called from the DSM language through a $ZCALL function reference.

Figure 8–8: Extended Example of Using $ZCALL

```
$ CREATE ABSX.FOR
C         This function returns the absolute value of its input

          REAL*8 FUNCTION ABSX(ORIG)
          REAL*8 ORIG
          ABSX = ABS(ORIG)
          RETURN
          END
$ FORTRAN ABSX
$ LIBR/CREATE ABSX ABSX
$ CREATE ZCALLTABS.MAR
          .TITLE ZCALLT$ABS
          ZCALLINI  ; INITIALIZE THE ZCALL TABLE

          ; ZCALL TO RETURN ABSOLUTE VALUE
          ROUTINE CALLNAME=ABSX,LINKNAME=ABSX,INPUTS=1,OUTPUTS=0
          RETURN VALUE,TYPE=DOUBLE
          INPUT TYPE=DOUBLE,MECHANISM=REFERENCE,POSITION=1

          ZCALLFIN  ; TERMINATE ZCALL TABLE
          .END
$ ASSIGN DBA1:[VAXDSMV20]ABSX.OLB LNK__LIBRARY
$ @SYS$LIBRARY:DSMBUILD ZCALLTABS
Previous logical name assignment replaced

Building DSM image with user defined ZCALL routines

DSM target device and directory  DBA1:[VAXDSMV20]
DSM library device and directory SYS$LIBRARY:
DSM ZCALL source device and directoryDBA1:[VAXDSMV20]

Assembling ZCALLTABS

Linking image DBA1:[VAXDSMV20]DSM.EXE

Including user object libraries:
 LNK$LIBRARY DBA1:[VAXDSMV20]ABSX.OLB
```

At this point, the new image DBA1:[VAXDSMV20]DSM.EXE has been created. To invoke this image, define the symbol so that the command DSM is translated to the file specification $DBA1:[VAXDSMV20]DSM. Then, you can invoke this image by issuing the DSM command, as shown below.

```
$ DSM :== $DBA1:[VAXDSMV20]DSM
$ DSM
          VAX-11 DSM Version 2.0 Field Test 1
>S A=67.56,X=-987.54
>S B=$ZC(ABSX,A),Y=$ZC(ABSX,X)
>ZW
A=67.56
B=67.56
X=-987.54
Y=987.54
>H
%DSM-I-HALT, HALT command executed
```

Chapter 9
The VAX–11 DSM Data Base

This chapter describes the implementation of global variables under VAX–11 DSM. It provides an overview of global variable concepts, describes the logical structure of a global, and outlines the relationship between globals and RMS ISAM files. This chapter also provides some guidelines for optimizing the performance of the VAX–11 DSM data base.

9.1 Global Concepts

VAX–11 DSM implements its data base through hierarchical structures called global arrays, or simply *globals*. A global is a tree-structured system of nodes stored on disk, the data of which can be shared by a number of processes in the system.

VAX–11 DSM creates a global on the first write access to a global variable. A global variable is a simple or subscripted variable whose name begins with a circumflex (^); in all other respects, the naming conventions for global variables are the same as those for local variables (those accessible only to the process that creates them).

Global arrays are sparse arrays. In a sparse array, the system dynamically adds nodes to the array as you define them and deletes nodes as you delete them. Thus, you never have to preallocate space for globals through dimensioning.

VAX–11 DSM bases its implementation of globals on VAX–11 RMS indexed sequential (ISAM) files. VAX–11 DSM represents each global by one indexed file. The mapping of globals into the logical and physical structure of the ISAM files that represent them is handled entirely by VAX–11 DSM and is transparent to the user. Thus, there is no concept of "opening" and "closing" a global.

In general, global arrays are treated syntactically in the DSM language the same way as local arrays:

- To create a global, you issue the SET command.
- To access and manipulate a global's contents, you use any of a number of commands and functions in the DSM language set ($FIND, WRITE, $DATA, and so forth).
- To delete a global node, you issue the KILL command.
- To delete the entire global array, you kill its root node.

Because global operations are handled by VAX–11 DSM and are transparent to the user, applications programmers need not be concerned with the physical structure of files when designing a data base application; they need only be concerned with the data relationships that comprise the logical design of a data base. However, a programmer who understands the requirements of an application can optimize the system's access to globals by laying out the files carefully, as described in Section 9.7.

In addition, the system or application manager must be concerned with the optimal *physical* layout of global files. The manager needs little or no knowledge of the DSM applications to be run on the system, but needs to understand VAX–11 RMS parameters.

9.2 Global Variables

The rules that govern the formation of global variables are for the most part identical to the rules that govern local variables. The *VAX–11 DSM Language Reference Manual* provides a detailed description of these rules.

Unlike local variables, however, global variables support an extra syntactic form that includes a *user field*. The full syntax of a VAX–11 DSM global variable is:

^[<user>]Globalname(Subscript 1,Subscript 2,...Subscript n)

This syntax consists of three parts, as follows:

1. The User Field

   The user field refers to a particular VAX/VMS directory and/or a particular node on a computer network. You can use standard VAX/VMS syntax to specify the node, directory, and device, or use a logical name that translates to a node, directory, and device. This field should not include a file name; if you specify one, it is ignored. The contents of the user field override the corresponding elements of the default file specification for the ISAM file that corresponds to the global.

   All text within the user field must either be enclosed in quotation marks, or be a DSM expression that evaluates to a string. If the user field contains a logical name, you must include a colon (:) as the last character immediately before the right quotation mark. The colon forces logical translation of the string.

2. Globalname

   VAX–11 RMS uses the global name to construct the file name for the file that corresponds to the global. The file name cannot be superseded by the file specification in the user field.

3. Subscripts

   These are the subscripts for the variable named as globalname. VAX–11 DSM constructs a node key corresponding to the subscripts. VAX–11 RMS uses this key as the primary key for storing the record for that node in the indexed file that corresponds to the global.

The following section describes how the user field and the globalname relate to the file specification of the ISAM file that represents a global. Section 9.4.1 describes how the subscript field is related to the key used in the indexed file for the global.

9.2.1 Translating Global Variables into File Specifications

The VAX/VMS file specification of the ISAM file that corresponds to a global variable is determined by various conventions and defaults. The following paragraphs describe how the file specification is constructed. Section 9.2.2 shows examples of file specifications.

File name

For application globals, the file name equals the global variable name. For library globals (globals whose names begin with '%'), the file name equals the global variable name stripped of the leading '%'.

Node, Device, and Directory

If the global includes a user field, VAX–11 DSM uses the information in the user field for the file's node, directory, and device. Then DSM applies the translation of the logical name DSM$GLOBAL_DIR to application globals or DSM$GLOBAL_LIB to library globals. If portions of the file specification are still missing after VAX–11 DSM examines these sources, VAX/VMS defaults are applied.

If neither of the logical names is defined and the user field is absent, VAX–11 DSM uses the local node and your default device and directory for application globals, or the translation of SYS$LIBRARY for library globals.

File type

VAX–11 DSM uses the default file type .GBL only after it examines the user field (if present) and DSM$GLOBAL_DIR. A file type contained in one of these supersedes the default.

Version Number

The version number for globals created by DSM is always 1.

NOTE

You can use VAX/VMS utilities such as COPY to copy DSM globals. If you copy a global into a VAX/VMS directory that already contains a global by the same name, VAX/VMS will assign the "new" ISAM file a version number other than 1. The user is responsible for ensuring that only one version of a global is retained. If multiple versions are retained, a KILL of the global removes only the most recent version of the file corresponding to that global name.

See the *VAX–11 RMS Reference Manual* for a complete description of how file specifications are parsed.

9.2.2 Translation of Global Variables

The following examples show how global variables are translated into VAX/VMS file specifications. The examples assume that:

- DSM$GLOBAL_DIR: translates to the node, device, and directory HUDSON::WRKD$:[TOPDIR].
- LOGICAL is a logical name representing [CYG].LOG (the directory and file type elements of a file specification).

Example 1

Global Variable: ^ABC

Full File Specification:

HUDSON::WRKD$:[TOPDIR]:ABC.GBL;1

Example 2

Global Variable: ^[".TMP"]DEF

Full File Specification:

HUDSON::WRKD$:[TOPDIR]:DEF.TMP;1

Example 3

Global Variable: ^["BOSTON::[SMITH]"]GHI

Full File Specification:

BOSTON::WRKD$:[SMITH]GHI.GBL;1

This translation assumes that WRKD$: is the local default device on the node BOSTON. You should not, however, make assumptions about logical name translations on other systems.

Example 4

Global Variable: ^["LOGICAL:"]JKL

Full File Specification:

Local Node::WRKD$:[CYG]JKL.LOG;1

9.3 Global Protection and Access Privileges

Since VAX–11 DSM represents a global by a VAX/VMS file, the protection mechanism that applies to VAX/VMS files also applies to globals. Thus, access privileges to globals are governed by:

- The file's protection mask
- The file's owner UIC

You can set the protection mask for the file that corresponds to a global with the DCL SET PROTECTION command, described in Section 3.4.1. This command allows you to prohibit users in each user category from performing various operations on your globals. There are three types of access relevant to DSM globals: READ, WRITE, and DELETE (abbreviated R,W, and D). READ enables GET access, described below. WRITE enables PUT and UPD access, as well as DEL access for a global node. DELETE access allows deletion of the entire file; that is, DELETE enables KILLing of the global's root node. Thus, you can allow the nodes of a global to be deleted, while protecting the root node, by giving the file for the global RW access only.

The system manager establishes the UIC for each user in the system. VAX/VMS considers your UIC an attribute of your user name. You cannot change a file's owner UIC unless you have sufficient VAX/VMS user privileges.

By default, DSM opens the file that represents any global you create or access with access privileges represented by the following RMS codes:

- GET — Enables reading of nodes in a global array
- PUT — Enables writing of nodes in a global array
- UPD — Enables updating of nodes in a global array
- DEL — Enables deletion of nodes in a global array

When you access a library global, DSM opens it with GET privilege only, unless the resultant file specifications of DSM$GLOBAL_DIR and DSM$GLOBAL_LIB are identical. If they are identical, VAX–11 DSM opens library globals with all four access privileges listed above.

If you attempt to perform an operation on a global that is protected against that operation, VAX–11 DSM generates an error.

9.3.1 Creating and Modifying Your Own Library Globals

To create or modify your own library globals, follow these steps:

1. Invoke DSM with a global directory equal to the future library directory, as in the following example:

```
$ DSM/GLOBALS=[MYDIR.DSMLIB]
```

2. In response to the DSM prompt, set ^X to the null string:

```
> SET ^X=""
```

3. Create or modify the global X.GBL, and exit from DSM:

```
> HALT
```

4. Re–invoke DSM with [MAINDIR.DSMLIB] as your library global directory, using either of these methods:

```
$ ASSIGN [MAINDIR.DSMLIB] DSM$GLOBAL_LIB
$ DSM
```

or:

```
$ DSM/GLOBALS=[MAINDIR.DSMLIB]/LIBRARY
```

When you access the global ^%X, it is a library global.

9.3.2 Shared Access

The user mode in which you invoke VAX–11 DSM determines whether the system enables or disables explicit sharing of globals. By default, VAX–11 DSM disables explicit sharing of globals in Programmer Mode. In Application Mode, VAX–11 DSM enables explicit sharing.

Except for library globals, DSM attempts to open all globals accessed in Application Mode with the default file-sharing mask described above (GET, PUT, UPD, and DEL). This mask allows multiple users to simultaneously read, write, update, and delete nodes in a global array. If shared globals are protected against writing, DSM opens them with GET only. DSM also opens globals with GET only if the disk is write-locked, as during backup. (If the globals are also read-protected, DSM issues a privilege error.)

Unless the resultant file specifications of DSM$GLOBAL_DIR and DSM$GLOBAL_LIB are identical, library globals support implicit sharing (multiple simultaneous read operations) only.

You can override the file sharing default in either Programmer Mode or Application Mode by using the /[NO]SHARE qualifier of the DSM command. Refer to Section 4.4 for details about this qualifier.

9.3.3 Record Interlocking

The DSM LOCK and ZALLOCATE commands provide a record-interlocking convention for shared globals. Although VAX–11 RMS provides automatic record interlocking for ISAM files, VAX–11 DSM does not use this feature, except when deleting records using the KILL command. You must handle all record interlocking of globals open for sharing at the DSM application level. Refer to the *VAX–11 DSM Language Reference Manual* for details about the LOCK and ZALLOCATE commands.

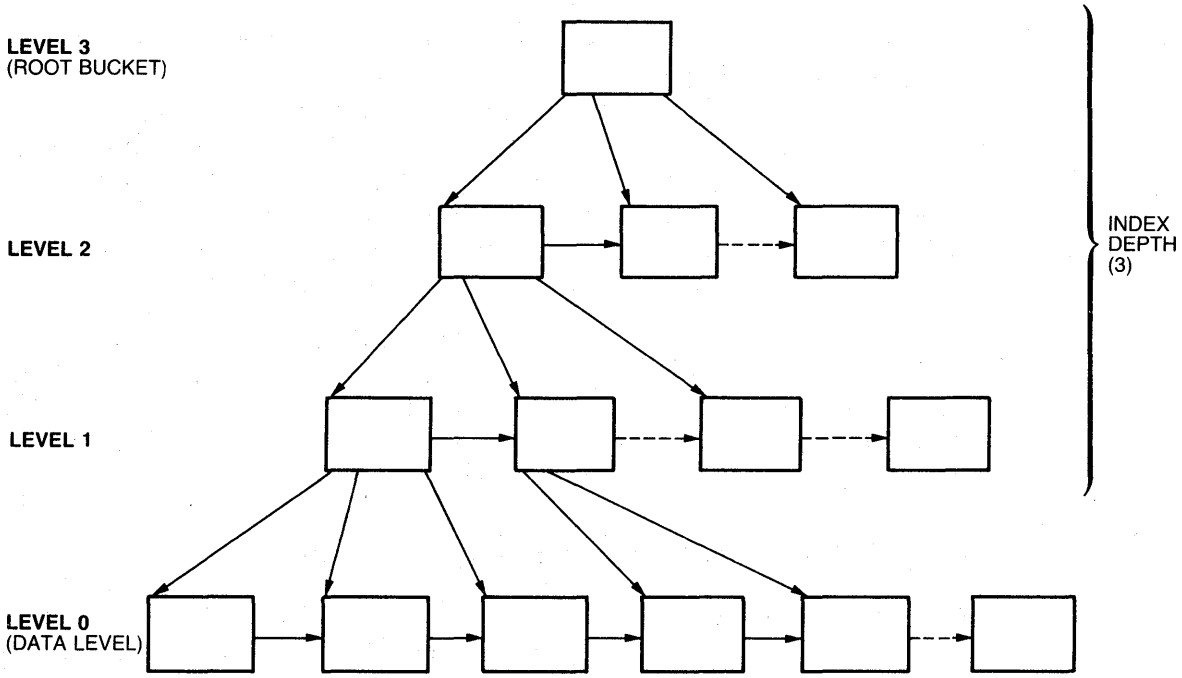
9.4 Structural Overview of the VAX–11 DSM Data Base

On the first write access (SET) to a global variable, VAX–11 RMS constructs one indexed file. The RMS file is subsequently maintained until the variable is killed (by killing the root node).

An indexed file may be visualized as a pyramid with levels that become smaller from bottom to top (see Figure 9–1). This structure is called an *index.*

The bottom level of an index (which has the largest number of blocks) is called the *data level,* or *level 0.* Levels above the data level are numbered in ascending order up to the top level. The highest level of an index is called the *index root.* The number of levels that make up an index is referred to as the *depth* of the index.

Figure 9–1: Three–Level Primary Key Index



Each level of an index consists of a linked and ordered chain of buckets. A *bucket* is the basic retrievable element of an indexed file. It consists of an integral number of contiguous 512–byte physical blocks. This number is called the *bucket size.* DSM uses a default bucket size of two blocks, or 1024 bytes.

Buckets at each level of the index store different types of information. Buckets at the bottom level (level 0) store the data records associated with a global. Buckets above level 0 up to the index root (which consists of a single bucket called the root bucket), store location pointers to buckets at levels below them.

Every indexed file consists of at least one index called the *primary index.* Although indexed files can have more than one index (alternate indices), VAX–11 DSM represents a global by the primary index only.

You access the records in the primary index through a record identifier called the *primary key.* A key is a prologue (prefix) to a data record that distinguishes it from other records in the file. For any global variable reference, VAX–11 DSM constructs the primary key from its subscripts. If a global variable is unsubscripted, VAX–11 DSM constructs a primary key that consists of a series of binary zeros.

When you set a global variable node equal to a data value, VAX–11 DSM constructs the primary key by arranging the characters in the subscript field in the DSM numeric collating sequence. After it constructs the key, VAX–11 DSM passes it to RMS. RMS then inserts the associated record in the file according to the ASCII collating sequence of the primary key. Section 9.4.2 describes this process in detail.

Whenever DSM requests a write or a read to an ISAM file, VAX–11 RMS compares the requested key value against the entries in the root bucket. When RMS finds a key whose value equals or exceeds the value of the requested key, it uses the bucket pointer associated with that value to locate the target bucket on the next lower level of the index. (This search is always successful because the root bucket always contains the key with the highest value in the collating sequence of the file.) This process repeats for each level until the target bucket at the data level is reached.

If there is insufficient room in a data level bucket to store a newly written record, RMS inserts a new bucket in the data level chain. It simultaneously adjusts the records in the surrounding buckets to preserve the pointer information and collating sequence. This is known as a bucket split. Successive bucket splitting adds more levels to the index, increasing its depth.

9.4.1 The Subscript Field and the Primary Key

VAX–11 DSM constructs the primary key for a record from the subscript field of a global variable. By default, the maximum number of characters that can be used in a key is 64. However, you can override this default in two ways:

- By using the /KEY_SIZE qualifier of the DSM command, setting the key size to any number from 4 through 255. This qualifier only applies to globals created during the current session. The keysize of existing globals cannot be changed.

  Refer to Section 9.7.3.1 for more information about the /KEY_SIZE qualifier.

- By constructing the global file through other means, such as the RMS utility CREATE/FDL and the DSM utility Global Create (^%GLCRE).

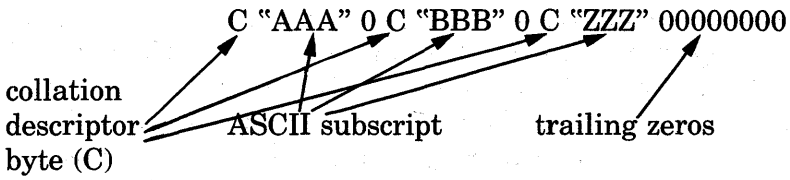
VAX–11 DSM constructs the primary key for a record by concatenating the subscript field from left to right and inserting a byte containing binary 0 between each subscript. Following the last character of the last subscript, VAX–11 DSM inserts zeros to fill out the key to the correct key size. (For Prologue 3 files, such as those newly constructed with VAX–11 DSM V2.0, RMS compresses the key to eliminate trailing zeros. RMS also compresses the beginning of the key if it is identical to the beginning of the preceding key in that bucket).

The first byte of each subscript in the key is a *collation descriptor byte*, indicating whether the subscript is a positive number, canonic zero, a negative number, or a non-canonic (that is, ASCII) string. See Appendix C for information on the value of this byte for different types and lengths of subscripts.

Figure 9–2 shows the structure of the primary key for the following node:

X("AAA","BBB","ZZZ")

Figure 9–2: The Primary Key



You determine the length of a key by calculating the sum of:

- The characters in the subscript (without the quotes, if any)
- Twice the number of subscripts (for each subscript's delimiter and collation descriptor byte)
- 1 (for the trailing zero)

9.4.2 Collating Sequence

The location of records in an indexed file depends on the order in which the keys associated with them are sorted. The order used is called the *collating sequence* for the file.

The default collating sequence for ISAM files is the ASCII collating sequence. This collating sequence dictates that all keys collate in the ascending order of the ASCII value of their characters.

VAX–11 DSM, however, uses a numeric collating sequence to arrange the records associated with a global in the disk structures that store them. The numeric collating sequence dictates that a global's subscripts sort in the following order:

1. Canonic numbers

   A canonic number is a number reduced to its simplest form. Such numbers contain only valid numeric characters and, optionally, a single decimal point and/or a minus sign. Canonic numbers have no leading or trailing zeros.

   Subscripts consisting of canonic numbers collate first in the ascending numeric order of the subscript characters. Negative canonic subscripts (those preceded by a minus sign) are placed first, followed by a subscript of zero and then by positive canonic subscripts.

2. Strings and non-canonic numbers

   VAX–11 DSM then sorts all nodes with subscipts consisting of alphanumeric strings (such as "B9") or non-canonic numbers (such as -0.5) in the ascending order of the total ASCII value of their subscript characters.

Refer to the *VAX–11 DSM Language Reference Manual* for more information about the numeric collating sequence.

Because VAX–11 RMS uses the ASCII collating sequence to arrange records in an indexed file, VAX–11 DSM implements the numeric collating sequence through the ASCII collating sequence. To do this, VAX–11 DSM represents alphanumeric subscripts in the key it constructs as the subscripts appear in a DSM string, that is, as pure ASCII characters preceded by the collation descriptor byte. This collation descriptor byte forces the arrangement of the records.

After VAX–11 DSM constructs the primary key in this fashion, it passes the primary key to RMS for processing.

9.4.3 RMS Defaults

When VAX–11 DSM creates an ISAM file to represent a global, it gives the ISAM file the following attributes by default:

1. Primary Key

   - Key position = 0
   - Key size = 64

- No duplicate keys allowed
- No alternate keys defined

2. Maximum Record Size = key size + maximum DSM string size(255) (default total = 319)
3. Bucket Size = 2 (1024 bytes)
4. Record Format = variable
5. Record Attribute (RMS RAT) = carriage return (CR)
6. Prologue 3 structure: index compression and data key compression, as described in Section 9.7.1.8 (but no data record compression).

9.5 Global Access from Other Languages and VAX/VMS Utilities

With the information provided in Sections 9.4 through 9.4.3, you can write a program in any language that supports RMS ISAM files to access and manipulate VAX–11 DSM globals.

Exercise extreme care when accessing the VAX–11 DSM data base from other languages, particularly if you intend to lock records. Because the DSM language provides its own locking mechanism for applications (LOCK, ZALLOCATE), VAX–11 DSM does not use the RMS autolocking feature for ISAM files. Thus, VAX–11 DSM (generally) expects to find records unlocked. Since VAX–11 RMS uses autolocking by default, a program could inadvertently lock a record or records. If VAX–11 DSM finds a record locked when it expects the record to be unlocked, VAX–11 DSM generates an error.

Global files can be manipulated by all VAX/VMS utilities that operate upon ISAM files (to copy, restructure, or back up a file). As most VAX/VMS utilities cannot access a file in shared mode, no DSM user should be accessing globals that are currently being handled by VAX/VMS utilities.

9.6 Global Access and DSM I/O

Globals can be created indirectly through the DSM I/O interface. For example, the following command line creates an indexed file called JPB, and predefines its key size to be 15 characters:

```
>OPEN "DSM$GLOBAL_DIR:JPB.GBL":(INDEXED:NEW:KEYSIZE=15)
```

After you create the indexed file this way, write one record to it to prevent VAX–11 DSM from deleting the file when you close it. The following example shows this procedure:

```
BAL   USE "device" F I=1:1:15 W $C(0)
      USE "device" W!
      CLOSE "device"
```

This procedure sets the root node to the null string (""). Thereafter, you can access the file directly through standard global variable syntax. That is, read and write requests to the global variable ^JPB are processed as if the global had been created by issuing a write request in the following form:

```
> SET ^JPB = " "
```

NOTE

If you create a global through the I/O interface using the KEYSIZE parameter, you cannot change the key size later because this action is equivalent to issuing the DSM command with the /KEY_SIZE qualifier appended to it. In addition, you cannot specify a key size less than 4 because this is the smallest permissible key size for global variables.

9.7 Global Optimization

You can manipulate a number of system and process-specific parameters to improve throughput on global variable accesses. You set up these parameters when you design (lay out) or restructure the file, using the VAX–11 RMS utilities or a set of DCL commands.

The parameters set during file layout are:

- Initial file allocation
- Division of a file into areas
- Contiguity of space within a file or within areas (if you use multiple areas)
- File extension size
- Bucket size
- Fill factor for initial loading of buckets
- Use of global buffers

You can change the file extension size and the global buffer count at a later time using the DCL SET FILE command. You cannot alter any other file layout parameters without copying the file. VAX–11 DSM sets the following parameters for you at access time:

- Multi–buffer count
- Window size
- Deferred write

The following sections describe the parameters that affect the performance of the data base. All parameters described in these sections can potentially affect the performance of the entire system. Moreover, altering one parameter often reverses the effect of altering another. Thus, setting up your system to yield optimum performance is a recursive process.

Refer to Section 11.1.7 for a description of the VAX/VMS parameters that affect not only the the data base, but VAX/VMS as a whole.

9.7.1 File Layout Parameters

The following sections describe the attributes of indexed files that you can define when you design the file. You use the VAX–11 RMS utilities, described in Table 9–1, to set up and examine these attributes.

9.7.1.1 The Initial File Allocation — You set up the initial file allocation with the FDL attribute FILE ALLOCATION. Careful use of this attribute (as well as of the file extension size) can minimize the overhead of extending a file. By reducing the number of retrieval pointers needed to map the file, a large initial file allocation provides more efficient random access (particularly in large files), reducing window-turning.

9.7.1.2 Use of Areas — You divide a file into areas by using Edit/FDL. Using areas allows you to reduce disk head movement when accessing records in a file. You should set up two areas: one for index buckets and one for data.

9.7.1.3 Contiguity — The contiguous use of space within a file (or within areas in a file) provides the most efficient use of disk space and the fastest access to data. To allocate space contiguously, use the CONTIGUOUSBESTRY keyword for both the file and the area.

9.7.1.4 Size of File Extensions — You can use the Edit/FDL utility to calculate the amount of space that is added to a file or an area when it reaches the limits of its allocated space. If you do not specify a file extension, RMS uses the disk cluster size as the default extension length, which may not be optimal for your application.

The disk cluster size is set using INITIALIZE (which erases the data on the disk).

You must specify file extensions at the area level, even if you use only one area. RMS does not recognize extensions specified at the file level.

9.7.1.5 Size of Buckets — You can use the Edit/FDL utility to calculate the best bucket size for your application. Increasing the bucket size decreases the number of index levels and therefore reduces disk access times. This is particularly important for very large files.

9.7.1.6 Fill Factor for Buckets — If your application is intended to insert records with randomly distributed keys into a file, you should specify that the initial loading of records into buckets does not completely fill the data or index buckets. Leaving some space in the buckets provides room for growth and reduces the number of bucket splits needed as the file is used. The fill factor is not important if your application only reads files, or if records are inserted with consecutive keys.

9.7.1.7 Use of Global Buffers — When multiple processes frequently reference the same bucket, you may want to allow that bucket to remain in memory, instead of being read into memory each time it is referenced. This is important for DSM applications, in which many DSM users access the same global variables concurrently. The root bucket, in particular, is accessed every time a record is inserted or retrieved.

You can specify the number of global buffers provided by using the FDL attribute FILE GLOBAL_BUFFER_COUNT when you create the file. You can change the number of global buffers by issuing the DCL command SET FILE with the qualifier /GLOBAL_BUFFERS.

Using global buffers allows deferred writing while sharing memory. Refer to Chapter 11 for information on setting up sysgen parameters for extensive use of global buffers.

9.7.1.8 Compression of Prologue 3 Files — Globals created with VMS version 3.0 or later are stored in Prologue 3 files. The prologue, an attribute of an ISAM file that defines its structure level, is set up so that space-saving compression operations can be performed on the file.

Because these files use the Prologue 3 format, you can specify that leading or trailing characters that are repeated should be compressed in the index buckets and in the keys.

You can also specify that repeating characters anywhere within a data record should be compressed. However, this compression is not recommended for most DSM globals.

You can use the VAX–11 RMS Convert/Reclaim utility, described in Table 9–1, to make empty buckets usable again.

9.7.2 Using the RMS Utilities

The VAX–11 RMS utilities let you optimize your files in all of the ways described above. There are five utilities:

- Edit/FDL, also known as the File Definition Language Editor
- Create/FDL
- Convert
- Convert/Reclaim
- Analyze/RMS_File

Table 9–1 summarizes the functions of the RMS utilities. For more information on these utilities, read the *VAX–11 Record Management Services Tuning Guide* and the *VAX–11 Record Management Services Utilities Reference Manual.*

Table 9–1: The RMS Utilities

Utility	Function
Edit/FDL	Allows you to set up values for the file attribute keywords that govern such file characteristics as initial file allocation, file extension size, bucket size, use of areas, and so on. To optimize an existing file, you must first run Analyze/RMS_File on the file, and then use the output of Analyze/RMS_File as input to Edit/FDL.
Create/FDL	Uses the keyword values set up with Edit/FDL to create an empty file.
Convert[/FDL]	Copies records from one or more files to an output file, either new or existing. You can use this utility to convert one indexed file to another, or to load data from a sequential file into an indexed file created with Create/FDL. You append /FDL to Convert to create and populate an indexed file in one command.
Convert/Reclaim	Makes empty buckets available so that new records can be written in them.
Analyze/RMS_File	Checks for structure errors in a file, and generates a report on the file's data structure. Can also set up the attributes of a file to be used as input for Edit/FDL.

When you install VAX–11 DSM, a sample file definition file for globals is inserted in SYS$LIBRARY with the file name DSMGLOBAL.FDL. You can use this file as a model or edit it to suit your situation.

9.7.3 Optimizing DSM Parameters

The following qualifiers on the DSM command perform data-base-related functions:

- /KEY_SIZE=n (for Prologue 2 files only)
- /OPEN_GLOBALS=n
- /[NO]OPTIMIZE_BUFFER_COUNT
- /[NO]SEQUENTIAL_OPTIMIZATION

These qualifiers tailor space and run-time requirements for global accesses. Section 4.4 provides a general description of these qualifiers. The following sections describe the impact of their use on the data base.

9.7.3.1 The /KEY_SIZE Qualifier — The /KEY_SIZE qualifier only affects "scratch" globals created during a session. It overrides, for the current terminal session, the default key size (64 characters) that VAX–11 DSM uses for globals.

You can use this qualifier to allocate a small key size if you know that you will never create a key larger than, for example, ten characters. Using a smaller key size allows faster data access time and better use of disk space for Prologue 2 files.

If your application needs to use large globals (up to 255 bytes), you can use /KEY_SIZE to provide a large key size.

As described in Section 9.7.1.8, Prologue 3 files allow automatic compression of leading or trailing characters in keys. In many cases, key compression provides good data access time and efficient use of bucket space, so that you can leave the key size at the default value.

9.7.3.2 The /OPEN_GLOBALS Qualifier — This qualifier overrides the default number of globals that can be open simultaneously (7).

VAX–11 DSM maintains a list called the Open Globals List that specifies the globals that are currently open. VAX–11 DSM updates this list on a least recently used basis. Every time a global is accessed that is not in the list, when the list has already reached its maximum size, VAX–11 DSM closes the least recently accessed global, and enters the needed global in its place.

Depending on the situation, the number of globals in the Open Globals List can improve or degrade the performance of your data base. For optimum performance, the list should be large enough to avoid frequent opening and closing. However, it should never be unnecessarily large, as this can lead to extra virtual memory paging.

As a rule, the size of the list should reflect the average number of globals in your data base that are accessed simultaneously. (If this number is fewer than the default number of globals maintained by the list, change the default to that number.)

To monitor the system-wide rate of file open operations, use the VAX/VMS MONITOR utility. To see statistics on each process's use of RMS operations (including OPENs and CLOSEs), use the VAX–11 DSM utility GBLSTAT^%STAT.

Each open file (including globals) uses up part of the process's BYTLM and FILLM quotas. Each open global is also noted in the process's Open Globals List. The count of the Open Globals List must be less than the value of FILLM. If you have many open globals, you may exceed your process's FILLM quota. If you exceed this quota, VAX–11 DSM sends a warning message to the terminal and automatically decreases the size of the Open Globals List. The difference between FILLM and the number of open globals plus 2 (for your two routine directories) is the number of files you can have open for I/O (OPEN, CLOSE, USE).

9.7.3.3 The /[NO]OPTIMIZE_BUFFER_COUNT Qualifier — The multi-buffer count is the number of I/O buffers that VAX–11 RMS allocates when you access a file (global or otherwise). RMS uses these buffers asynchronously whenever possible to overlap I/O time with compute time.

Unless you specify /NOOPTIMIZE_BUFFER_COUNT on the DSM command at image start-up (in either Programmer Mode or Application Mode), VAX–11 DSM automatically adapts the RMS multi-buffer count for existing globals to the number of levels in the global's index plus one. This value generally produces the optimum environment for global accesses.

VAX–11 DSM always uses the RMS default multi-buffer count for globals created during the current terminal session.

When you specify /NOOPTIMIZE_BUFFER_COUNT, RMS defaults are used in all cases, whether you create a new global or access an old one.

You can set the default multi-buffer count at the process level or at the system level. At the process level, you set the multi-buffer count with the following DCL command:

SET RMS_DEFAULT/BUFFER_COUNT=n/INDEXED

where n represents the number of buffers to be allocated; the value of n must be less than or equal to 127.

If the multi-buffer count for indexed files is undefined at the process level, VAX–11 RMS uses the system default established at VAX/VMS sysgen time (through the sysgen parameter RMS_DFMBIDX) or by the SET RMS_DEFAULT command qualified by the /SYSTEM qualifier. If the system default is also undefined, RMS allocates one buffer for the file.

The multi-buffer count does not affect the use of global buffers, explained in Section 9.7.1.7.

9.7.3.4 The /[NO]SEQUENTIAL_OPTIMIZATION Qualifier — The /SEQUENTIAL_OPTIMIZATION qualifier, which is the default, causes VAX–11 DSM to optimize (when possible) sequential access for a sequence of the following operations:

- Get the contents of a node, as with the command SET X=^A("xxx")
- $DATA
- $ORDER
- $ZORDER
- $NEXT
- $ZNEXT

VAX–11 DSM performs this sequential access by caching the RFA (record's file address) corresponding to the node just accessed. If the next operation on the same global refers to either the same node or the node that immediately follows, DSM performs an access by RFA instead of by key, thus avoiding the search through the index buckets for the key.

Operations on another global do not affect the data cached for a global.

Part 3
Operating VAX–11 DSM

Chapter 10
Installing VAX–11 DSM

This chapter describes the VAX–11 DSM distribution kit and the VAX–11 DSM software installation procedure. It describes how to install the DSM image as a known image, how to start the DSM Job Controller, and how to install the VAX–11 RMS shared-file option. For supplemental information about the material discussed in this chapter, refer to the *VAX/VMS System Management and Operations Guide.*

10.1 The VAX–11 DSM Distribution Kit

The VAX–11 DSM distribution kit is made up of software and documentation. DIGITAL distributes the VAX–11 DSM software on either a multivolume set of floppy disks or DECtape II magnetic tape cartridges. The VAX–11 DSM software consists of the following:

- Components needed to install DSM as a layered product
- DSM library utility routines and globals
- Components needed to rebuild DSM with user-defined functions

The documentation consists of:

- The *VAX–11 DSM User's Guide*
- The *VAX–11 DSM Language Reference Manual*
- The *VAX–11 DSM Summary*
- *Introduction to DSM*

10.2 Installing VAX–11 DSM

The general procedure for installing the VAX–11 DSM system software is:

1. Log in under the privileged system manager's account.
2. Set defaults as follows:

```
$ SET UIC [1,4]
$ SET DEFAULT SYS$UPDATE
```

3. At the console terminal, type:

```
$ @VMSUPDATE
```

   VAX/VMS then prints the following message on the terminal:

```
                    VMS Update Procedure

This command procedure performs VAX/VMS software updates and
optional software installations for VAX/VMS Version 3. During
this sequence, the standard console medium is not present in the
console drive. Thus, the system may be vulnerable to power fail-
ures or other fatal crashes. If a system crash occurs during
this time, the update sequence can be restarted at the beginning
of the first incomplete update.
```

4. Dismount the current medium (floppy disk or magnetic tape cartridge) from the console drive.

   As you remove the volume, note the direction in which it is facing: floppy disks or magnetic tape cartridges from the distribution kit must be inserted in the console drive so they face the same direction. (The label is on the front side of the volume.)

   Note that you receive a device-not-mounted message if no volume is currently mounted. Ignore this message.

5. Place the first floppy disk or magnetic tape cartridge from the distribution kit in the console drive. (The floppies or cartridges that comprise your distribution kit are numbered sequentially. You must install the volumes in ascending numerical order.)

   VAX/VMS then prints the following query:

```
ARE YOU READY TO CONTINUE?
```

6. Type Y for the installation procedure to proceed. When you respond to the query with Y, the command procedure on the volume assumes control, and the files on the medium are copied to the system disk.

   If you respond to the query by typing N, VAX/VMS repeats the request to put the volume in the console drive; then it repeats the READY TO CONTINUE? query.

7. After the files are copied from the first volume, repeat steps 5, 6, and 7 for each remaining floppy disk or magnetic tape cartridge in your distribution kit.

8. After you dismount the last volume, VAX/VMS begins to install the VAX–11 DSM system software. A series of milestone messages is printed, marking the installation of each component of VAX–11 DSM. Then the system initiates a verification procedure. The verification procedure consists of the following:

   - Invoking the DSM image.
   - Restoring utilities from the kit utilities file.
   - Restoring globals from the kit globals file. If the verification procedure is successful, you receive the following message:

```
VAX-11 DSM V2.0 test PASSED
```

NOTE

Stop the installation procedure and contact your Software Services representative if you receive the following message:

```
VAX-11 DSM V2.0 test FAILED
```

9. During installation, VAX–11 DSM asks a series of questions. By answering these questions, you specify whether previous versions of VAX–11 DSM related files should be purged:

   - `Do you wish to purge the previous versions of all VAX-11 DSM files?`

     If you answer YES, VAX–11 DSM asks no additional questions; the installation procedure purges all previous versions of VAX–11 DSM–related files in system directories.

     If you answer NO, the following additional questions are asked:

   - `VAX-11 DSM images?`

     Answering YES to this question causes all DSM executable images to be purged. Note that if you answer NO to this question, and DSM is installed as a shared image, you will have to run the INSTALL utility to install the new version of DSM. You need not run INSTALL if purging is selected.

   - `VAX-11 DSM object libraries?`

     This question refers to all DSM object library files kept in SYS$LIBRARY for linking the DSM interpreter with user-defined $ZCALL entry points.

   - `VAX-11 DSM option files and command procedures?`

If you answer YES, the installation procedure purges the command procedures and option files for starting up the DSM Job Controller and the DSM Journal Processes.

- VAX-11 DSM Library routines and globals?

  If you answer NO, the installation procedure preserves the previous version of the DSM library routine directory, SYS$LIBRARY:ROUTINES.DSM, and previous versions of DSM library globals.

- VAX-11 DSM $ZCALL and message libraries?

  Answering YES purges previous versions of files containing $ZCALL tables and examples in SYS$LIBRARY.

10. When the installation and verification procedures are complete, you receive a message informing you that VAX–11 DSM has been successfully installed. Control then returns to the system command procedure VMSUPDATE. This procedure prints the following query:

    ```
    ARE THERE MORE KITS TO PROCESS?:
    ```

    Type Y if you have additional software kits to install. Otherwise, type N to terminate the VMSUPDATE program.

11. Remount any previously mounted floppy disk or magnetic tape cartridge.

The installation procedure places all executable images that you need to operate DSM in the VAX/VMS directory SYS$SYSTEM. These include the DSM Job Controller (DSMMJC.EXE), the DSM interpreter (DSM.EXE), and the DSM journaling image (DSMJRN.EXE).

In addition, installation places the following components in the VAX/VMS directory SYS$LIBRARY:

- The DSM library routines and globals
- The command files and example option files for starting the DSM Job Controller and Journal Process
- The components needed to rebuild the DSM image with user-written functions, including the $ZCALL example file (ZCEXAMPLE.MAR)

At this point, the basic VAX–11 DSM software installation procedure is complete. To run the DSM image, type the DSM command in response to the DCL prompt:

```
$ DSM(RET)
```

The DSM command takes a number of qualifiers, as described in Chapter 4.

Note however, that your system is not yet configured to yield optimum performance as layered software under VAX/VMS. At this point, your system does not:

- Provide shared access to the DSM image
- Handle simultaneous accesses to the DSM data base and other disk files
- Handle DSM locks

The following sections address these issues.

10.3 Starting the DSM Job Controller

For any VAX–11 DSM system running applications in shared mode (that is, running in Application Mode or in Programmer Mode with the /SHARE qualifier specified), you must start up the DSM Job Controller. The DSM Job Controller is a process that manages lock requests for DSM applications and controls the use of journaling.

To start up the DSM Job Controller, you invoke a command procedure called DSMMJCSTA.COM, located in the directory SYS$LIBRARY. This command procedure creates a detached process and sets up a process name, privileges, and input, output, and error files. To invoke the command procedure, enter:

```
@SYS$LIBRARY:DSMMJCSTA " " [1,4]
```

You should also insert this command in the VAX/VMS start-up file SYS$MANAGER:SYSTARTUP.COM, so that the DSM Job Controller is started up automatically when VMS starts up. Refer to Chapter 12 for details about the DSM Job Controller and its start-up command procedure.

10.4 Installing the DSM Image as a Known Image

VAX/VMS allows you to install certain executable image files, such as the DSM image file, as known images. Known images are executable image files that can be quickly accessed and shared. They are usually installed when the system is booted.

Known images can have one or more of the following attributes:

- Permanently open
- Shared
- Privileged
- Protected
- Writeable
- Headers permanently resident in memory

To install the DSM image as a known image, you use the VAX/VMS INSTALL utility program. The following command sequence installs the DSM image as a known image with the attributes listed above:

```
$ SET DEFAULT SYS$SYSTEM
$ RUN SYS$SYSTEM: INSTALL
* DSM/OPEN/SHARED/HEADER_RESIDENT
```

You can also use INSTALL to delete or replace files in the system's list of known images. You will often have to replace one version of the DSM image with another to make user-written $ZCALL functions available to all VAX–11 DSM users. The following example shows how to use INSTALL to do this:

```
$ RUN SYS$SYSTEM: INSTALL
* DSM/REPLACE
```

NOTE

If you perform a system installation (as outlined in Section 10.2), and have previously installed a version of DSM as a known image, you do not have to use INSTALL. In this case, VAX/VMS automatically installs the new version of DSM as a known image. VAX/VMS deletes the previous version only if you answer YES to the purge question.

You should install DSM, however, each time you boot your VAX/VMS system, by inserting the commands shown above in SYS$MANAGER:SYSTARTUP.COM.

10.5 Installing the VAX–11 RMS Shared–File Option

Through the shared-file option, VAX–11 RMS allows several processes to access files simultaneously. VAX–11 DSM uses the RMS shared-file option in application environments to allow simultaneous accesses to the DSM data base and files created through the DSM I/O interface or by other languages.

Before you can use the RMS shared-file option, you must run the RMSSHARE utility program. You should run this utility each time you boot the system. Because of this, it is a good idea to include a command for running RMSSHARE in SYS$MANAGER:SYSTARTUP.COM.

The RMSSHARE utility tells you whether file sharing has been enabled. If file sharing is enabled, RMSSHARE displays current and maximum page counts (see Section 11.1.8). You can then increase or decrease the maximum page count to which the data base can grow. Because pages are not returned to the system pool until you reboot the system, you should not set the maximum lower than the current count.

To run the RMSSHARE utility program, you need the CMEXEC privilege. If you have this privilege, issue the following command to run the RMSSHARE utility program:

```
$ RUN SYS$SYSTEM:RMSSHARE
```

You use the EXIT command to terminate the execution of the RMSSHARE utility program. Type the EXIT command (in uppercase characters) in response to the program prompt.

Chapter 11
Managing VAX–11 DSM

This chapter describes the tasks involved in managing a VAX–11 DSM system. Topics discussed include:

- Establishing and deleting accounts for VAX–11 DSM programmers and applications users
- Setting up the user interface for VAX–11 DSM application users
- Installing DSM applications as permanent global sections
- Deleting and listing permanent global sections
- Performing VAX–11 DSM system backup
- Tuning DSM and VMS for optimal performance

For supplemental information on the topics described in this chapter, refer to the *VAX/VMS System Management and Operations Guide.*

11.1 Establishing Accounts for VAX–11 DSM Users

Each VAX–11 DSM user in Programmer Mode and each VAX–11 DSM application run in Application Mode has an *account* on the VAX/VMS system. Information to identify each account is kept in the system's user authorization file, SYS$SYSTEM:SYSUAF.DAT. This information includes (but is not limited to) the following:

- A user name

  The user name, as described in Section 2.2.1, is an alphabetic character string. It is VAX/VMS's principal way of identifying an account. You must ensure that each VAX–11 DSM account has a unique user name.

The user name for a programmer is often the programmer's name. The user name for an application is often the subject of the application, such as INVENTORY or RADIOLOGY.

- A password

  The password for a programmer's account should be known only to the user, as described below. The password for an application account is known by all users of the application. Note that the password of an application account set up with auto-login must be null, as described in Section 11.1.3.3.

- The name of the default device

  The default device is the storage device to which the user has access by default upon logging in. This is normally the device on which the volume that contains the account's most frequently used files is mounted.

- The name of the default directory

  The default directory is the directory (file containing a catalog of file names) to which the user is connected by default upon logging in. It is generally the user's main directory, as described in Section 3.3.

  Accounts can share the same default directory.

- The name of the log-in command file

  The log-in command file associated with an account is a command procedure (file type .COM) executed automatically when the user logs in.

- Privileges

  The privileges associated with a VAX/VMS account determine to what extent the user can affect important system facilities and thus influence how the system performs for other users.

- A User Identification Code (UIC)

  The VAX/VMS data protection scheme is based on the User Identification Code (UIC). You must assign each account a UIC.

  The UIC regulates each user's access to the data structures protected by UICs, such as files and interprocess communication facilities (mailboxes and shared areas of memory).

  A UIC is made up of two 16–bit numbers. The first number represents a user's group, and the second number represents a user's member number within the group.

To assign UICs optimally, be careful when assigning users to groups. VAX/VMS defines a group as a collection of users who normally have access to each other's files and mailboxes, to global virtual memory sections, and to the group logical name table. Group members share data and can control one another's processes.

Therefore, when establishing accounts for VAX–11 DSM users, observe the following guidelines:

- Assign different UICs and default directories to DSM programmers and applications accounts, so that application users cannot alter programs under development.

  (In some cases, however, you may want to assign the same directory and UIC to a programmer account and an application account to allow a programmer to have privileged access to the application's data.)

- Assign the same group number in the UIC and default directory to users who must share access to the same files.

11.1.1 How to Set Up User Accounts

To set up an account for a VAX/VMS user, you must have SYSPRV or SETPRV privilege.

The first step in setting up an account is to determine what the account's user name, password, default directory, and other identifying information will be. Then, you create the account's main directory and (if appropriate) a login command file. Finally, you insert information about the account into the system's user authorization file (UAF), using the AUTHORIZE utility. You specify the user name, password, default device, and so on to this utility.

The following example shows how part of your session with AUTHORIZE might look:

```
$ RUN SYS$SYSTEM:AUTHORIZE

UAF>ADD JONES/PASSWORD=TROMBONE/UIC=[013,007]-
/DEVICE=DISK$USER/DIRECTORY=[JONES]-
/LGICMD=DISK$USER:[RADIOLOGY]GRPLOGIN
```

The *VAX/VMS System Management and Operations Guide* contains a template of a command procedure for adding a new user to the UAF. This command procedure asks for all categories of information about the user, invokes AUTHORIZE, and cleans up after itself. You may find copying and using this command procedure more convenient than performing all the steps yourself.

11.1.2 Establishing Accounts for VAX–11 DSM Programmers

When you establish an account for a VAX–11 DSM programmer, follow the guidelines for establishing an account for VAX/VMS interactive users described in the *VAX/VMS System Management and Operations Guide*. To VAX/VMS, a VAX–11 DSM programmer is simply a time-sharing VAX/VMS user who happens to be running the DSM image.

You can recommend to the VAX–11 DSM programmers on your system that they create subdirectories with the logical names DSM$GLOBAL_DIR and DSM$ROUTINE_DIR. This allows programmers to separate their data files automatically from their routines and globals.

The password for a Programmer Mode user should be selected by the user. When you first set up the user's information in the UAF, you can assign a dummy password (USER, for example) and instruct the user to assign his or her own password with the DCL command SET PASSWORD. To enhance system security, encourage Programmer Mode users to select obscure passwords (not, for example, their first names) of at least six letters and to change passwords occasionally.

11.1.3 Establishing Accounts for DSM Applications Users

The three basic reasons for establishing separate DSM application accounts are:

- To enable protected and automatic start-up of a given application.
- To provide an easy-to-use interface for the user of the application.
- To prevent unintentional or unauthorized tampering with the data being manipulated by a particular application.

You can establish either of the following methods for application users to start a VAX–11 DSM application:

1. The user can enter a user name and password that correspond to a particular DSM application account. As described below, a log-in command procedure automatically starts the application. When the application exits, the user is automatically logged out. If you take the precautions listed below, all interruptions, such as CTRL/Y, are disabled while the application executes.
2. You can tie an application account to a terminal (or set of terminals), as described in Section 11.1.3.3. When a terminal is tied, typing a control character (such as RET, BREAK, or CTRL/C) forces the automatic start-up of the application.

11.1.3.1 Protecting Application Accounts — When you establish a VAX–11 DSM application account with the VAX/VMS AUTHORIZE utility, take the following precautions:

1. Specify the /FLAGS=DISCTLY qualifier to AUTHORIZE to disable CTRL/Y in the user authorization file record.
2. Specify the /FLAGS=CAPTIVE qualifier to AUTHORIZE to prevent the application user from specifying another command interpreter (or another start-up command file) when logging in.
3. Specify the/FLAG=LOCKPWD qualifier to AUTHORIZE to prevent the application user from changing the password of the application account.

In addition, you may want to specify /DISWELCOME and /DISNEWMAIL to keep non-application text from being displayed, and /DISNETWORK and /DISDIALUP to prevent remote access to the application.

To prevent interactive access to an application account, specify the /DISUSER qualifier to AUTHORIZE. This does not prevent automatic logins through null passwords on tied terminals, described in Section 11.1.3.3.

For additional suggestions on managing application accounts, see the descriptions of "turnkey" accounts in the *VAX/VMS System Management and Operations Guide.*

11.1.3.2 Starting an Application through a Command Procedure — You may find it useful to start the DSM application for the user, through a log-in command procedure. You specify the procedure's file name with the /LGICMD qualifier to AUTHORIZE. A log-in command procedure for start-up of the application APPL would look like the following:

```
$ Miscellaneous DCL Commands (for example, ASSIGN commands)
        .
        .
        .
$ SET NOON
$ DSM/INPUT=TT:   ^APPL
$ LOGOUT
```

11.1.3.3 Starting an Application through Automatic Login — The VAX/VMS Automatic Facility (Auto–Login) is a VAX/VMS start-up feature that allows you to "tie" a user name to a terminal to enable the automatic and transparent start-up of an application.

Auto–Login is handled by the VAX/VMS LOGINOUT utility program. VAX–11 DSM provides an interface to LOGINOUT through a system utility routine called %ALF. This utility sets up the data structures required by LOGINOUT to tie terminals to an application.

%ALF creates, lists, and modifies the contents of the automatic login file SYS$SYSTEM:SYSALF.DAT, which contains the records that associate a terminal with a particular user name. The primary key of each record is interpreted as a terminal name, and the data is interpreted as a user name.

NOTE

> Always use a temporary copy of SYSALF.DAT to perform modifications. After you finish, you can then copy the updated version of SYSALF.DAT to the system directory (SYS$SYSTEM), automatically replacing the old version.

For an application to be tied to a terminal, the application account must have a null password ("").

When unsolicited input (such as RET, BREAK, or CTRL/Y) occurs at a terminal, the LOGINOUT utility looks for SYSALF.DAT. If the utility finds SYSALF.DAT, it checks to see whether the current terminal name corresponds to the key of a record in SYSALF.DAT. If the LOGINOUT utility cannot locate a corresponding record, login proceeds as usual, with VAX/VMS prompting you for a user name and a password.

If LOGINOUT does locate a record that corresponds to the current terminal, and the password specified is null, LOGINOUT attempts to perform automatic login. You are not requested to enter a user name or a password.

NOTE

If you tie a non-existent user name to a terminal, you will receive a log-in error each time unsolicited input is entered at that terminal.

11.1.3.4 Suppressing VAX/VMS Messages — When you set up an application account, you can disable the VAX/VMS welcome and mail messages through the qualifiers described in Section 11.1.3.1. However, these qualifiers do not suppress messages from DCL commands (such as ASSIGN commands when you replace a logical name's equivalence string) or the VAX/VMS logout message.

To suppress these messages, you can send them to a null device (device type NL:). List the commands whose output you want to suppress in a command procedure. Then, invoke that command procedure (from your start-up file, for example) with the /OUTPUT=NL: qualifier.

To suppress the VAX/VMS logout message, include the following command at the end of your start-up file:

```
$ STOP 'F$PROCESS()
```

11.1.4 Deleting User Accounts

When a user leaves your VAX/VMS system, or when a VAX–11 DSM application is no longer needed, you should delete the account for that user or application. You delete an account using AUTHORIZE. Follow these steps:

1. Copy any files of possible future value to another account.
2. Delete the account's files and directories from the bottom up (that is, starting with the files in the most nested subdirectories).
3. Invoke AUTHORIZE and delete the account, as described in the *VAX–11 Utilities Reference Manual.*

11.1.5 Assigning Privileges to VAX–11 DSM Users

Because many VAX/VMS system functions perform extremely sensitive tasks and affect all users of the system, they are protected by privileges. When you assign privileges to VAX–11 DSM users, carefully consider whether each user has the skill and experience to use the privileges assigned, as well as a valid need for the privilege.

However, VAX–11 DSM programmers must have certain privileges to perform some VAX–11 DSM operations:

DSM Command	Description	Privilege
ZUSE	Broadcast to a terminal	OPER
ZJOB	Start a process	GRPNAM TMPMBX
ZJOB x:DETACH	Start a detached process	DETACH
ZJOB x:PRIORITY=n	Start process with higher priority	ALTPRI
OPEN x:(MAILBOX:NEWVERSION)	Create a temporary mailbox	TMPMBX
OPEN x:(MAILBOX:NEWVERSION:SYSTEM)	Create a permanent mailbox	PRMMBX

DSM Qualifier	Description	Privilege
/INSTALL	Create a permanent section	PRMGBL
/DELETE	Delete a permanent section	PRMGBL
/SYSTEM	Create or delete a system wide section	SYSGBL

You must record all privileges that you assign to a user in the user's account record in the User Authorization File (UAF), using AUTHORIZE.

In most instances, the privileges that you assign to a user can only be used to execute an image that has corresponding privileges. However, when a user executes a known image, the user's privileges can be temporarily amplified so they correspond with the privileges of the known image. Thus, a user having few privileges can execute an image that requires many privileges.

NOTE

The DSM image is normally installed without additional privileges.

For more information about privileges, consult the *VAX/VMS System Management and Operations Guide.*

11.1.6 Assigning Limits for VAX–11 DSM Users

Each VAX/VMS user must have limits set on certain system resources that affect system performance. You set each user's assigned limits in that user's account record in the UAF.

The following are the VAX/VMS limits that directly affect VAX–11 DSM system operation:

1. Default AST Queue Limit (ASTLM)

   Restricts the number of terminals (ASTLIM–1) that a DSM application can have open concurrently. You should specify the VAX/VMS default value for this limit.

2. Buffered I/O Count Limit (BIOLM)

   Restricts the number of outstanding buffered I/O operations allowed to a user's process. You should specify the VAX/VMS default value for this limit.

3. Buffered I/O Byte Count Limit (BYTLM)

   Restricts the amount of buffer space that a process can use for various operations, such as writes to terminals or mailboxes. Accessing files, particularly large files, also uses up the BYTLM quota. The default byte count limit is not adequate for DSM operation because of DSM's heavy use of buffer space for writes. You should significantly increase the value of this limit.

   Note that subprocesses started with ZJOB share the buffer space allocated to a process with BYTLM.

4. Open File Limit (FILLM)

   Restricts the number of files that a user's process can have open simultaneously. The Open File Limit is significant for VAX–11 DSM, and may have to be made larger than the VAX/VMS default value. Refer to Section 9.7.3.2 for additional information about this limit.

5. Direct I/O Count Limit (DIOLM)

   Restricts the number of outstanding direct I/O operations permitted to a user's process. The Direct I/O Count Limit can affect VAX–11 DSM disk I/O performance. You should specify the VAX/VMS default value for this limit.

6. Enqueue Quota (ENQLM)

   Establishes the maximum number of locks that a process can own. You should set this limit in the 50–100 range, higher than the limit (20) suggested for typical VMS operation.

7. Paging File Limit (PGFLQUOTA)

   Establishes the number of pages that a process can use in the system paging file. The paging file provides temporary disk storage for pages forced out of memory by a memory management operation. Thus, PGFLQUOTA limits the writable portion of a user's address space.

   The first user who gains shared access to a file with a non-zero global buffer count is charged by VAX/VMS for page file use resulting from the creation of a global page-file section to hold these global buffers.

8. Subprocess Creation Limit (PRCLM)

   Restricts the number of subprocesses that a process can start concurrently with the ZJOB command or the %SPAWN $ZCALL function. Note that each subprocess, if started, deducts from the creating process' deductible limits. You should tailor this limit to accommodate your application.

9. Shared File Limit (SHRFILLM)

   Restricts the use of system resources required for a process to perform file sharing.

10. Working Set Size Limit (WSQUOTA)

    Restricts the size to which the working set of a user's process can be expanded. You should tailor this limit to accommodate your application. DSM applications require a minimum working set size of 300.

11. Default Working Set Size (WSDEFAULT)

    Sets the initial working set size for a user's process. You should tailor this limit to accommodate your application.

12. Working Set Extent Limit (WSEXTENT)

    Specifies the maximum size to which a user's physical memory usage can grow, independent of the system load. WSEXTENT should always be greater than or equal to WSQUOTA. The minimum value is set with the system parameter WSMAX, described in the following section.

For more information about limits, consult the *VAX/VMS System Management and Operations Guide.*

11.1.7 Optimizing SYSGEN Parameters

After you install VAX–11 DSM, you should modify a few VAX/VMS SYSGEN parameters to optimize the performance of VAX–11 DSM. Only those parameters that specifically affect the DSM user environment (either programmer or application) are listed in this section. For a complete description of each VAX/VMS SYSGEN parameter, refer to the *VAX/VMS System Management and Operations Guide.*

You should record site-specific parameter values in a file with the name SYS$SYSTEM:PARAMS.DAT. This file is consulted by the VAX/VMS AUTOGEN procedure to establish system parameters, as described in the *VAX/VMS System Management and Operations Guide.*

Parameter	Applicability to VAX–11 DSM
SYSMWCNT	Because DSM processes perform frequent, shared accesses to VAX–11 RMS ISAM files, increase this parameter to enhance VAX–11 DSM performance. You determine the optimal value by running the PAGE option of the VAX/VMS MONITOR utility and observing the system page faults. If the system page fault count is greater than 2, increase SYSMWCNT.
WSMAX	This parameter should be high enough to achieve maximum performance in a DSM application environment (for example, to minimize swapping while keeping paging to a reasonable rate). The VAX/VMS default is usually adequate. If the working set limit and extent quota values assigned to individual users in the user authorization file are lower than WSMAX, they are used as maximums. Moreover, VAX/VMS provides an automatic working set adjustment mechanism within the limit set by WSMAX.
PAGEDYN	This parameter must be large enough to accommodate the RMS shared pool space. The RMS shared pool space is usually allocated at VAX/VMS startup by means of the RMSSHARE utility. See the next section for information about the RMS shared page count, which is related to this SYSGEN parameter.
ACP PARAMETERS	These parameters affect the performance of DSM because they affect the performance of all disk handling. See the *VAX/VMS System Management and Operations Guide* for recommended settings.
RMS PARAMETERS	These parameters affect the performance of DSM in its access to routines and globals and when performing sequential I/O. In particular, the multibuffer count for indexed files affects routine access and newly created globals. You should set the value of RMS_DFMBFIDX to 4 or 8.
LOCKIDTBL	This parameter must be increased for VAX–11 DSM. To determine the optimal value, observe the number of lock entries under heavy load by running the LOCK option of the VAX/VMS MONITOR utility.

REHASHTBL

When you increase LOCKIDTBL (see above), you must increase REHASHTBL by the same proportion.

GBLPAGFIL

This parameter defines the maximum number of system-wide pages allowed for global page-file sections, that is, scratch global sections that can be used without being mapped to a file. VAX–11 RMS creates one such section for each shared file that has a global buffer count associated with it. The size of the section is the bucket size of that file times its global buffer count. If you use global buffering to a significant extent, you will need to raise the value of GBLPAGEFIL. Whenever you increase GBLPAGEFIL, you must also increase the size of the system page file, SYS$SYSTEM:PAGEFIL.SYS, as described in the *VAX/VMS System Management and Operations Guide*. Increase GBLPAGES by the same amount, as described below.

If you are using global buffers, see the descriptions of the GBLSECTIONS parameter and and the PGFLQUOTA process quota.

GBLSECTIONS

This parameter sets the number of global section descriptors allocated in the system header at bootstrap time.

If you are using mapped DSM routine sections, you need one global section header for each mapped section.

VAX–11 RMS also creates a temporary global page-file section for each shared file with a global buffer count. The names of these sections all start with RMS$ followed by eight hexadecimal digits representing the longword address. If you are using global buffers, GBLSECTIONS must be large enough to accommodate one section for each shared, open file that has a global buffer count associated with it.

GBLPAGES

This parameter sets the number of global page table entries allocated at bootstrap time.

If you are using DSM mapped routine sections, you need the number of global pages equal to the sum of the sizes of each section you are mapping. The number of pages required by a DSM mapped routine section is equal to the number of blocks allocated to the corresponding DSM mapped routine file (.VIR). Use the DIRECTORY/SIZE command to find out the size of a DSM mapped routine file.

If you are using global buffers, GBLPAGES must be large enough to accommodate all the sections associated with each shared, open file that has a global buffer count. You must increase GBLPAGES as well as GBLPAGFIL, described above.

If you are using DSM mapped routine files, see the description of the GBLSECTIONS parameter.

NPAGEVIR — This parameter determines the maximum size to which the nonpaged dynamic pool can be increased. If this value is too small, the system could hang.

SRPCOUNTV — This parameter establishes the upper limit to which the number of pre-allocated small request packets can be increased. If this parameter is set too low, system performance can be adversely affected.

IRPCOUNTV — This parameter establishes the upper limit to which the number of pre-allocated intermediate request packets can be automatically increased by the system. If IRPCOUNTV is too low, system performance can be adversely affected.

LRPCOUNTV — This parameter establishes the upper limit to which the number of pre-allocated large request packets can be automatically increased by the system. If LRPCOUNTV is too low, system performance can be adversely affected.

Use the DCL command SHOW MEMORY/POOL/FULL to determine the use of nonpaged dynamic pool, and of small, intermediate, and large request packages.

11.1.8 Guidelines for Estimating RMS File–Sharing Page Count

When you run the RMSSHARE utility, you are asked to specify the maximum number of pages to be allocated for file-sharing structures. A sample RMSSHARE run is shown below. Note that the maximum allocation allowed (78) is limited by the PAGEDYN parameter, and the other values are determined by RMSSHARE. If you do not want to specify a maximum page count, you enter RET, as shown below.

```
$RUN STS$SYSTEM:RMSSHARE
RMS file sharing is currently enabled ...
  Maximum allocation allowed:78
  Number of pages allocated:58
  Max pages used:51
  Current number of pages in use:42
  Enter max pages:RET
```

The number of pages required for VAX–11 RMS file sharing structures on a VMS system depends on the following factors:

- The number of files opened for shared access at any one time
- The global buffer count associated with each shared file
- The multi-buffer count used for accessing each file

The total requirement for RMS file sharing structures is equal to the number of pages required by VAX–11 DSM users running in application mode or with the /SHARED qualifier, plus the requirements of other VMS users using RMS file sharing.

You can calculate an initial value for the number of pages required by VAX–11 DSM by estimating approximately 5 pages for each VAX–11 DSM application user who will be active simultaneously with other users. (Users running VAX–11 DSM in programmer mode do not require any RMS file sharing structures, unless /SHARED is specified.)

Run RMSSHARE to add the number computed above to the current size of the RMS shared file structures.

Then, monitor the system's use of RMS shared file structures periodically by running RMSSHARE, which displays the current and maximum number of pages used, along with the total number of allocated pages. Adjust the total number of allocated pages accordingly.

If you encounter the following error message, the space allocated to the RMS shared file structures has been exhausted:

```
%DSM-W-SYSQUOTA, system quota exceeded
```

You must increase the number of pages as described above. Failure to do so will adversely affect the performance of the system, and will eventually cause VAX–11 DSM applications to terminate with a fatal error.

The values of the following VAX–11 DSM parameters can increase the requirement for RMS shared file structures:

- The index depth of VAX–11 DSM globals (because it determines the value of the multi-buffer count VAX–11 DSM is using for each file)
- The value of the /OPEN_GLOBALS qualifier (default is 7) because it determines the number of simultaneously open global files
- The global buffer count associated with VAX–11 DSM globals

To adjust the number of pages to the value needed, monitor the system's usage as described above.

Whenever you run RMSSHARE to increase the number of pages allocated to RMS file sharing structures, you must also adjust the value of the system working set size, SYSMWCNT. To determine the correct value of SYSMWCNT, invoke the MONITOR utility to record the rate of system page faults under heavy load. If the average system page fault rate exceeds two pages per second, you must increase SYSMWCNT.

11.2 Installing DSM Applications as Global Sections

VAX/VMS allows you to install a DSM mapped routine file or a DSM mapped library file as a permanent global section, either group global or system global. You must always install a DSM mapped library file as a system permanent global section.

Before you can install a permanent global section, you may need to modify the GBLPAGES and GBLSECTIONS sysgen parameters to accommodate all DSM mapped files that are to be installed. GBLSECTIONS must be large enough to accommodate the number of simultaneously installed DSM mapped routine and library sections, as well as all other VAX/VMS installed images and sections. GBLPAGES must be large enough to accommodate the additional total size in pages of all DSM mapped routine and mapped library sections that are to be installed simultaneously.

Refer to Section 5.8 for a general description of mapped routine files and global sections. The following sections describe how to install, delete, and list permanent global sections.

11.2.1 Installing Mapped Routine Files

To install a DSM mapped routine file as either a group or system permanent global section, you include the /INSTALL qualifier along with /SHARE and /MAPPED when you issue the DSM command.

To install a DSM mapped routine file, enter the following sequence of commands:

```
$ DSM/SHARED/MAPPED=file name/INSTALL
>HALT
```

The default name assigned to a global section for DSM mapped routine files is DSM$ROUTINE_SEC.

To install a DSM mapped library file, enter the following sequence of commands:

```
$ DSM/SHARED/MAPPED=file name/LIBRARY/INSTALL
>HALT
```

Where "file name" is the name of the DSM mapped routine file that you built with the ^%RBUILD library utility (described in Chapter 7).

The default name assigned to a global section for a DSM mapped library file is DSM$LIBRARY_SEC. You must include the /LIBRARY qualifier when you install a library file.

You can install a DSM application section system-wide, by specifying /SYSTEM, or change its default name, by specifying /NAME. The following example shows how these qualifiers are used to install a system-wide application named MYAPP$SEC.

```
$ DSM/SHARED/MAP=file name/INSTALL/NAME=MYAPP$SEC/SYSTEM
> HALT
```

However, you cannot change the name of a library section.

When you install any permanent global section, you can specify the page-fault cluster size by including the /CLUSTER_SIZE=n qualifier in the DSM command line. If you do not include this qualifier, VAX/VMS uses the default page-fault cluster size specified in the PCDEFAULT SYSGEN parameter.

To install a DSM mapped routine file as a group permanent global section, you must have the GROUP and PRMGBL privileges. In addition, your UIC group number must correspond with the UIC group number of the DSM application users who are to map to the section.

To install a DSM mapped routine file as a system permanent global section, you must have the SYSGBL, WORLD, and PRMGBL privileges. Because system permanent global sections are available to all users, you need not have the same UIC group number as the application users who map to the section.

11.2.2 Deleting Mapped Routine Files

To delete an application or library that has been installed as a permanent global section, you include the /DELETE qualifier along with /SHARED and /MAPPED when you issue the DSM command. You can only delete a permanent global section if you have the privileges required to install one.

To delete a DSM application installed as a global section with the default name, enter the following sequence of commands:

```
$ DSM/SHARED/MAPPED/DELETE
>HALT
```

To delete a DSM library installed as a permanent global section, enter the following commands:

```
$ DSM/SHARED/MAPPED/LIBRARY/DELETE
>HALT
```

To delete a mapped library, you must include the /LIBRARY qualifier.

To delete a named section, you specify the /NAME qualifier. You must specify /SYS with /DELETE if the section was installed with /SYS. For example, the following line deletes MYAPP$SEC, installed in the example in Section 11.2.1:

```
$DSM/SHARED/MAP=file name/DELETE/NAME=MYAPP$SEC/SYSTEM
>HALT
```

A global section is not actually deleted until all processes have ceased mapping to it.

11.2.3 Listing Permanent Global Sections with INSTALL

The VAX/VMS INSTALL utility allows you to:

- List all group or system-wide permanent global sections
- Determine the number of available global sections and pages

To use INSTALL to list permanent global sections, issue the following series of commands:

```
$RUN SYS$SYSTEM:INSTALL
*/GLOBAL/FULL
```

INSTALL also lists the number of available global sections and global pages. You need this information to determine if there is enough room to install a mapped section. To find out how many pages a file needs to be installed as a section, issue the following DCL command:

DIRECTORY/SIZE filename.VIR

Check that the number of pages displayed is smaller than the number of available global pages. If it is not, adjust the GBLPAGES SYSGEN parameter, described in Section 11.1.7, by performing the AUTOGEN procedure.

11.3 Optimizing VAX–11 DSM Applications

To optimize the throughput of a VAX/VMS system running a VAX–11 DSM application, you should always follow these steps:

1. Optimize the layout of disk volumes and files, particularly VAX–11 DSM global variable files.
2. Install stable DSM applications as mapped virtual memory sections.
3. Tune VMS parameters for special VAX–11 DSM requirements, such as heavy file sharing.

   You do not need any knowledge of the DSM language or DSM applications to perform steps 1–3. These steps use standard VAX/VMS facilities and do not affect the logic of a DSM application.

4. Favor certain DSM application programming techniques over others.

The following sections describe these four activities in detail.

11.3.1 Optimizing Disk Volume and File Layout

Optimizing disk volume and file layout produces the largest performance payoff for a VAX–11 DSM application.

You must:

- Lay out the initial structure of disk volumes and files carefully
- Maintain an optimal arrangement after heavy usage

It is recommended that you re-structure volumes and files periodically, using the appropriate VMS and RMS utilities.

11.3.1.1 Maintaining Disk Volumes — To define the initial structure of disk volumes, use the DCL INITIALIZE command (which erases all information on a disk). In particular, choose an appropriate value for the volume cluster size (INITIALIZE/CLUSTERSIZE = n). A large cluster size allows more efficient access to the file structure and faster extension of files. However, space may be wasted for small files. RMS uses a volume's cluster size as the default extension quantity to a file.

The *VAX/VMS System Management and Operations Guide* gives guidelines for maintaining volumes.

To maintain an optimum structure for disk volumes after heavy usage, use the VAX/VMS BACKUP utility, described in the *VAX–11 Utilities Reference Manual.*

11.3.1.2 Allocating and Maintaining DSM Global Files — It is crucial to the throughput of a VAX–11 DSM application to carefully lay out the structure of large DSM globals. The following attributes of a file have an impact on access time:

- Initial allocation
- Default extension
- Separate areas for index and data buckets
- Bucket size
- Use of global buffers

The RMS utilities EDIT/FDL, CREATE/FDL and CONVERT/FDL are easy-to-use tools to help you layout a global file initially, or restructure it after heavy use. The RMS utility CONVERT/RECLAIM can be used to reclaim space within a global file without copying it. Refer to Chapter 9 and the *VAX–11 Record Management Services Utilities Reference Manual* and the *VAX–11 Record Management Services Tuning Guide* for details about the RMS utilities.

11.3.1.3 Minimizing the Rate of Opening Globals — If your DSM application simultaneously accesses more globals than the default number of globals in the open global list, you may want to adjust the size of this list. You make this adjustment by means of the /OPEN_GLOBALS qualifier on the DSM command, as described in Chapter 9.

11.3.2 Using Mapped Routines

If you are running a stable DSM application (that is, if the application code does not need to be modified frequently), you can gain significant performance in the loading of DSM routines (through the DO command) by installing the most frequently used DSM routines as a mapped virtual memory section.

The VAX–11 DSM ^%RBUILD utility allows you select a set of routines to include in a mapped section file. You do not need to install all routines of an application in a mapped section. The section merely acts as a cache. DSM will load from the DSM routine directory any routine that is not mapped.

Refer to Section 11.2 for instructions on installing a DSM mapped section.

Remember that mapped routine sections are accessible for reading only. If you need to change the code of a DSM routine that is included in a mapped section, you must rebuild the section using ^%RBUILD, and reinstall it.

11.3.3 Adjusting VMS Parameters

To adjust the values of VAX/VMS parameters, you first use the AUTOGEN procedure to let VMS calculate optimum parameters for your configuration.

Then, you adjust a certain number of parameters for a VAX–11 DSM application environment, by recording their value in the parameter file PARAMS.DAT. Finally, you reinvoke the AUTOGEN procedure. Refer to *VAX/VMS System Management and Operations Guide* for details.

VAX–11 DSM applications typically involve heavy sharing of files and a high use of indexed sequential files (DSM globals). Follow these guidelines to adjust parameters according to the characteristics of your application:

1. In an environment that produces heavy file sharing, you must adjust the system working set size, SYSMWCNT, to reduce the system paging rate to no more than an average of two pages a second. A higher rate has a very negative impact on total system performance. Use the VAX/VMS MONITOR utility to monitor the system page fault rate.

2. Be sure to allocate large enough values for the upper limits of the nonpaged pool (NPAGEVIR) the small request packet count (SRPCOUNT) and the intermediate request packet count (IRPCOUNTV). If NPAGEVIR is too small, the system may hang. If SRPCOUNTV or IRPCOUNTV are too small, system performance may be adversely affected. Use MONITOR or SHOW MEMORY/POOL/FULL to determine this usage.

3. Observe the system under heavy load, and note the behavior of modified-page writing, by using the PAGE option of the MONITOR utility. If necessary, adjust the VMS sysgen parameters that affect the writing of modified pages to the page file. These parameters start with the characters MPW_. Consult the *VAX/VMS System Management and and Operations Guide* for descriptions of these parameters.

4. Consult the *VAX/VMS System Management and Operations Guide* for guidelines for adjusting the ACP parameters. These parameters determine the size of various virtual memory caches used by the file system ACP.

5. Make sure every DSM user has a big enough WSQUOTA process quota. This value is minimized by sysgen parameter WSMAX, but the default value of WSMAX is usually adequate for DSM operation.

11.3.4 VAX–11 DSM Programming Techniques

If you are writing a DSM application specifically for VAX–11 DSM, or if you wish to optimize the code of a DSM application developed on another MUMPS system, follow these guidelines:

1. Avoid excessive use of the XECUTE command and indirection.

2. Use local variables rather than global variables for scratch purposes, such as sorting.

3. Consider using sequential files rather than global variables for storing sequential data, such as text or print files. Sequential files, combined with access by RFA, can also be used for large data bases that are never changed, such as archival data. In this case, while the bulk of the data would be kept in a large sequential file, cross-references to pieces of information can be kept in DSM globals, keyed by RFA (record file address). This technique saves disk accesses.

4. Optimize terminal output by using large transfers of data to the terminal rather than many small transfers. Remember that the following constructs produce multiple QIOs:

```
> FOR I=1:1:N WRITE " "
> WRITE A WRITE B WRITE C WRITE ! WRITE D
```

   However, the following construct produces a single QIO:

```
> WRITE A,B,C,!,D
```

5. If you use $TEXT with mapped routines, precede each text line with a double semi-colon (;;) to force the pre-compiler to include this line in the pre-compiled, mapped code. In general, DSM must load the source of a routine to perform the $TEXT function, even if the pre-compiled version of the same routine is mapped in virtual memory.

Chapter 12
Running the VAX–11 DSM Job Controller

This chapter describes the privileged operator interface to the VAX–11 DSM Job Controller. It includes instructions for starting up and shutting down the DSM Job Controller.

The DSM Job Controller manages lock requests for DSM application users and controls operations that must be performed for all users, such as enabling and disabling journaling.

12.1 Overview of the Job Controller

The primary function of the VAX–11 DSM Job Controller is to act as a lock manager, allowing the hierachical locking of a variable or a node of a variable by DSM users running in shared mode. This locking function is required for the operation of the LOCK command (from the MUMPS Standard Language) and the VAX–11 DSM extension commands ZALLOCATE and ZDEALLOCATE.

The VAX–11 DSM Job Controller is a VAX/VMS detached process, with no individual owner associated with it. This process is created by the DSM system manager, who assigns the appropriate privileges and access rights to it and sets up the initial lists of users who are enabled to log into DSM and to use the DSM Journal Process. While the Job Controller Process is running, the DSM operator can communicate with it through a set of utility programs. These programs allow the operator to specify who is enabled to log into DSM and to use the DSM Journal Process, as well as to monitor and shut down the DSM Job Controller Process.

Whenever a DSM user invokes DSM in shared mode (either implicitly, by running DSM in Application Mode, or explicitly, by using the /SHARED qualifier in Programmer Mode), the services of the DSM Job Controller are required. The DSM Job Controller is not needed to run DSM in Programmer Mode without the /SHARED qualifier, or in Application Mode with /NOSHARED specified.

For users running in a shared mode, the DSM Job Controller performs the following functions:

- Manages the processing of LOCK, ZALLOCATE, and ZDEALLOCATE commands.
- Controls which groups of users can log into DSM in Application Mode.
- Keeps a list of DSM users currently running in Application Mode.
- Controls which groups of users can perform journaling using the DSM Journal Process (described in Chapter 13).

Note that since processes started through the DSM ZJOB command always run in Application Mode, the DSM Job Controller must be running for ZJOB to be used.

By default, there is one DSM Job Controller for each VAX–11 DSM system. However, the system manager can set up a separate Job Controller for a group of one or more users with the same group number in their UICs.

The following paragraphs explain what happens when you run a DSM image in shared mode. (Note that each process that runs the DSM image is referred to as a "DSM process," or simply "process," in the remainder of this chapter.)

1. As soon as a user invokes the DSM image in a shared mode, DSM sets up a communication link between that process and the DSM Job Controller. Through this communication link, the user process sends information to the DSM Job Controller, and the DSM Job Controller lets the DSM user image know whether or not it can run in shared mode and whether or not journaling is permitted for the user.
2. Once communication has been established, the Job Controller processes all LOCK, ZALLOCATE, and ZDEALLOCATE requests that the user image makes.
3. In turn, the Job Controller is controlled either by start-up parameters or by utilities that the VAX–11 DSM operator runs.

12.2 Communication between DSM Job Controller and DSM Processes

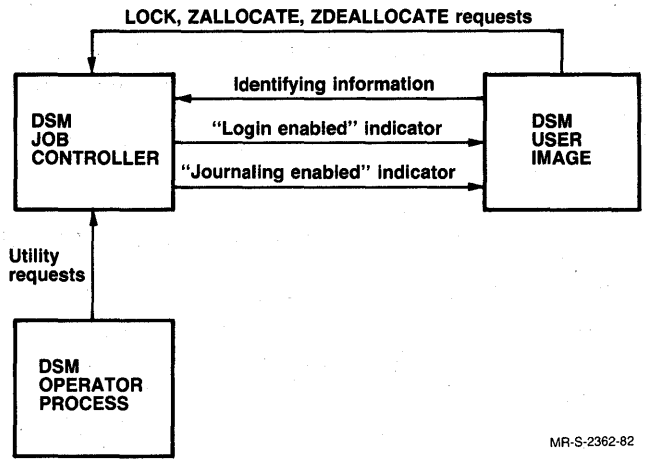
Communication between processes running DSM and the DSM Job Controller is accomplished through the use of mailboxes. As described in Section 6.11, a mailbox is a VAX/VMS pseudodevice that one or more processes can use for interprocess communication.

Communication between user images and the Job Controller is established according to these steps:

1. The DSM user process creates its own LOGIN/LOCK REPLY mailbox.
2. The DSM user image sends its user name, principal I/O device, UIC, and other miscellaneous information to the Job Controller. The Job Controller receives this information through its LOGIN mailbox, as shown in Figure 12–1. The LOGIN mailbox has the logical name DSM$MJC_REQUEST_MBX. This is a system logical name by default, but you can set it up as a group logical name by including the /GROUP qualifier in the start-up option file (see Section 12.3.2).
3. The DSM Job Controller sends back to the user image's mailbox:
   - A "login enabled" indicator, if either of the following is true:
     - DSM logins have been enabled for all users
     - The current user is a member of a group for which DSM logins have been enabled
   - A "journaling enabled" indicator, if either of the following is true:
     - Journaling has been enabled for all users
     - The current user is a member of a group that has been specified for journaling
   - Information on how to reach the DSM Job Controller's LOCK mailbox, which receives LOCK, ZALLOCATE, and ZDEALLOCATE requests
4. Once the user process receives this information, it can issue the LOCK, ZALLOCATE, and ZDEALLOCATE commands. The DSM Job Controller locks variables for user processes as requested. If another user has previously locked the same variable, the first user must either wait for the variable to be unlocked, or wait for a specified timeout to occur.
5. The DSM Job Controller also sets up an Operator Request Mailbox with the group logical name DSM$MJC_OPERATOR_MBX, accessible only to the DSM operator, and only if the operator has the same UIC as the DSM Job Controller Process. The operator can then send utility requests to this mailbox, as shown in Figure 12–1.

Figure 12–1 illustrates the interaction between a DSM user process, a DSM operator, and the DSM Job Controller.

Figure 12–1: Communicating with the DSM Job Controller



12.3 Starting the Job Controller

The VAX–11 DSM distribution kit includes a command file that starts the Job Controller. This file is called DSMMJCSTA.COM; it resides in the VAX/VMS directory SYS$LIBRARY.

DSMMJCSTA.COM contains the DCL RUN command and a number of qualifiers. The image that is executed by the RUN command is SYS$SYSTEM:DSMMJC.EXE. Thus, the format of the start-up command file is:

```
$RUN-
/INPUT='P1'-
/UIC='P2'-
/PROCESS_NAME=DSM_CONTROL-
/OUTPUT=NL:-
/ERROR='P3'-
/PRIVILEGES=(TMPMBX, PRMMBX, GROUP, WORLD)-
SYS$SYSTEM:DSMMJC.EXE
```

Section 12.3.1 describes the contents of DSMMJCSTA.COM in greater detail.

12.3.1 The Job Controller Start–up Command File

The RUN command causes VAX/VMS to create the DSM Job Controller as a detached process and to execute an image called DSMMJC.EXE.

VAX/VMS applies certain defaults to a detached process. For example, the DSM Job Controller's default directory is the same as the default directory of the system manager who starts up the DSM Job Controller.

The qualifiers associated with the RUN command provide information about the files that you wish to use and the privileges that you wish to assign to the Job Controller. The qualifiers can be specified in any order. Table 12–1 lists the qualifiers associated with the RUN command, in the order used in DSMMJCSTA.COM.

Table 12–1: Command Qualifiers in DSMMJCSTA.COM

Qualifier	Function
/INPUT (P1)	Specifies the name of the start-up option file that the Job Controller uses initially. This file specifies the start-up options that the Job Controller uses until it is instructed to change them (by a VAX–11 DSM operator). If you do not specify P1, the Job Controller uses the default file SYS$LIBRARY:DSMMJCPAR.OPT, described in Section 12.3.2.
/UIC (P2)	Specifies the group and member numbers of the DSM Job Controller. The default is your current UIC. (The DSM operator must have the same UIC as the DSM Job Controller.) **NOTE** If you invoke the DSM Job Controller from SYSTARTUP.COM, you must specify the UIC [1,4]. Otherwise, the DSM Job Controller is set up with the UIC [10,40].
/PROCESS_NAME	Specifies the name of the VAX–11 DSM Job Controller. The default name is DSM_CONTROL. If the DSM operator assigns a new name to the Job Controller, the name must be unique in the VAX/VMS system within the group.
/OUTPUT	Specifies the name of the Job Controller's current output file or device. The /OUTPUT qualifier assigns an equivalence name for the logical name SYS$OUTPUT. The DSM Job Controller writes internal warning, error, and information messages to this device or file. Since the Job Controller duplicates all messages other than success messages in the file name you specify with /ERROR, you specify /OUTPUT=NL: to ensure a single copy of messages.
/ERROR (P3)	Specifies the file name to which the Job Controller writes internal warning, error, and information messages. The error file is important as a record of problems if the Job Controller stops unexpectedly. The /ERROR qualifier assigns an equivalence name for the logical name SYS$ERROR. The text of messages is stored in the file specified. If the Job Controller does not issue any messages, it does not create the file. The default name is DSMMJC.ERR.

Table 12–1 (Cont.): Command Qualifiers in DSMMJCSTA.COM

Qualifier	Function
/PRIVILEGES	Assigns privileges to the Job Controller, as follows: • TMPMBX is required by the Job Controller. This privilege allows the Job Controller to create the temporary Operator Request mailbox and LOCK mailbox. • PRMMBX is required by the Job Controller. This privilege allows the Job Controller to create a permanent LOGIN mailbox if the Job Controller start-up option file specifies /SYSTEM. (If the option file specifies /GROUP, the LOGIN mailbox is temporary.) • GROUP and WORLD privileges are only needed internally if the DSM operator uses the Job Controller to execute the KILL A DSM USER command or the VERIFY DSM USERS command. GROUP allows the operator to control other users within its own group. WORLD allows any user in the system to control all other users in the system. Specifying NOSAME provides the Job Controller with only the privileges listed in the command file.

As shown in Table 12–1, there are three optional parameters that you can specify when you start the Job Controller by executing DSMMJCSTA.COM:

- P1, applied to the /INPUT qualifier, specifies the name of the start-up option file.
- P2, applied to the /UIC qualifier, specifies the UIC of the DSM Job Controller.
- P3, applied to the /ERROR qualifier, specifies the name of an error log file that is created if the Job Controller generates any warning, error, or information messages.

Note the three parameters in the following example of a command line that invokes the Job Controller attribute command file:

```
$@SYS$LIBRARY:DSMMJCSTA MYMJCPAR.OPT [1,4] MYMJC.ERR
```

In this example, the DSM Job Controller starts up with UIC [1,4]. The Job Controller uses a start-up option file called MYMJCPAR.OPT in the current default directory. Any messages that the Job Controller generates will be stored in a file called MYMJC.ERR in the current default directory.

A system manager can modify the site-independent start-up file so the DSM Job Controller starts automatically when the system is booted. The command line shown in the above example is typical of the kind of addition you have to make to the file to do this. Refer to the *VAX/VMS System Management and Operations Guide* for details on tailoring the site-independent start-up file.

12.3.2 The Job Controller Start–up Option File

The /INPUT qualifier in the DSMMJCSTA.COM file identifies the option file to be used in starting up the Job Controller. If you do not specify a name, the Job Controller uses the file SYS$LIBRARY:DSMMJCPAR.OPT. You can construct your own start-up file (with the default name or any name that you choose), or use the sample file supplied with the distribution kit.

The DSMMJCSTA.COM file contains qualifiers and the values associated with them, in the following general format:

/[NO]QUALIFIER/SUBQUALIFIER=[argument][!COMMENTS]

VAX/VMS interprets any text preceded by an exclamation mark (!) as a comment and ignores it. There can be unlimited comments in an option file.

Table 12–2 shows the qualifier options and their defaults. Table 12–3 describes the qualifiers in greater detail.

Table 12–2: Qualifiers Used in DSMMJCPAR.OPT

Qualifier	Default
/[SYSTEM] [GROUP]	/SYSTEM
/[NO]LOGIN[/ALL] [/SELECT_GROUPS=(list)]*	/LOGIN/ALL
/[NO]JOURNAL[/ALL] [/SELECT_GROUPS=(list)]*	/NOJOURNAL/SELECT_GROUPS

The "list" is a series of group numbers separated by commas, for example: (100,101,102).

Table 12–3: Qualifier Summary

Qualifier	Function
/GROUP or /SYSTEM	Specifies that the DSM Job Controller is used by a select group of users (such as program developers) or for all DSM users in the system.
/[NO]LOGIN	Specifies whether or not DSM user images can run. If, however, you defined the Job Controller for only a group, the /[NO]LOGIN qualifier applies only to that group. Therefore, if you specify /NOLOGIN, and the DSM operator later runs the ALLOW FUTURE DSM LOGINS utility, only that group of DSM user images can run. • /ALL indicates that any DSM user image in any group can run. If the option file includes /NOLOGIN/ALL, and the DSM operator later runs the ALLOW FUTURE DSM LOGINS utility, any DSM user image in any group can run. • /SELECT_GROUPS=(list) specifies which groups of DSM user images can run. If the command file includes /NOLOGIN/SELECT_GROUPS, no DSM user images can run. If, however, the DSM operator later runs the ALLOW FUTURE DSM LOGINS utility, only the listed groups of DSM user images can run. (SELECT_GROUPS does not apply if you are running the Job Controller on a group basis.) While the DSM Job Controller is running, you can use operator utilities to change the list of users for whom login is enabled. These utilities are described in Section 7.6.2.
/[NO]JOURNAL	Specifies whether or not users can perform journaling (described in detail in Chapter 13). If the status of the "journaling enabled" indicator changes while a DSM user image is running, that image is not affected by the change. A DSM user image only receives an indicator at start-up time. • /ALL indicates that all groups can use journaling. If the option file includes /NOJOURNAL/ALL, and the DSM operator later runs the START JOURNALING FOR DSM USERS utility, all groups can use journaling. • /SELECT_GROUPS=(list) specifies which groups can perform journaling. If the option file includes /NOJOURNAL/SELECT_GROUPS, no users can use journaling. If, however, the DSM operator later runs the START JOURNALING FOR DSM USERS utility, only the listed groups can use journaling. (SELECT_GROUPS does not apply if you are running the Job Controller on a group basis.) While the DSM Job Controller is running, you can use operator utilities to change the list of users who can use journaling. These utilities are described in Section 7.6.1.

Remember that before you enable journaling for all users or selected users, you must start a DSM Journal Process for those users (one process for all users or one process for each group, as appropriate). Otherwise, these users will receive a journal-initialization failed error message (%DSM–E–JRNINIFAIL) when they try to run in Application Mode (or in Programmer Mode with /SHARED).

12.4 Job Controller Operator Utilities

The DSM operator uses the VAX–11 DSM utility routines on the ^%OPER menu to communicate with the Job Controller. Table 12–4 lists these utilities and their entry points. For details on these utilities, see Sections 7.6.2 and 7.6.3, or the on-line documentation.

WARNING

You must have the UIC of the DSM Job Controller to invoke these utilities.

Table 12–4: DSM Job Controller Utilities

Utility	Entry Point
Add Group to Login List	ADDLOG^%MJC
Allow Future DSM Logins	LOG^%MJC
Change DSM Login–Enable Mode	LOPT^%MJC
Delete Group from Login List	DELLOG^%MJC
Job Table Display	^%JOBTAB
Kill a DSM User	KILLUSE^%MJC
Lock Table Display	^%LCKTAB
Prevent Future DSM Logins	NOLOG^%MJC
Show Groups	DISLOG^%MJC
Shutdown DSM	SHUTUP^%MJC
Status of DSM Job Controller	STATUS^%MJC
Verify DSM Users	VERIFY^%MJC

Table 12–5 lists the journaling utilities that interact with the DSM Job Controller.

Table 12–5: Journal Utilities that Use DSM Job Controller

Utility	Entry Point
Add Group to Journal List	ADDJRN^%MJCJRN
Change Journal–Enable Mode	JOPT^%MJCJRN
Delete Group from Journal List	DELJRN^%MJCJRN
Show Groups Journal List	SHOJRN^%MJCJRN
Start Journaling for DSM User	JRNON^%MJCJRN
Stop Journaling for DSM User	JRNOFF^%MJCJRN

Chapter 13
Running the VAX–11 DSM Journal Process

This chapter describes the VAX–11 DSM journaling facility. It explains the operator's interface to Journal Processes and the relationship between DSM Journal Processes and users running VAX–11 DSM.

13.1 Overview of Journaling

Journaling is a method of keeping a record on a sequential storage device (disk or magnetic tape) of the SET and KILL transactions that a DSM user performs on the global data base.

Journaling is done through a VAX/VMS detached process called a DSM Journal Process. A Journal Process can also be started a batch process, as described in Section 13.7.

VAX–11 DSM offers a number of journaling options to meet the needs of a system running multiple applications. Each group of related DSM user images running the same application can have its own Journal Process, or there can be a single Journal Process for all DSM applications on the system.

Journaling is performed on a per-global basis. Within an application for which journaling is enabled, only those globals that are marked for journaling are actually journaled.

A VAX–11 DSM operator can interact with a Journal Process at run time to control the journaling operation. This interaction is done through utilities supplied with the VAX–11 DSM system (see Section 13.6 for details).

13.1.1 Enabling Journaling

Journaling can be enabled and disabled only by the DSM operator, not by individual DSM users. The operator must do the following before the SETs and KILLs of global variables can be journaled:

1. Start the DSM Job Controller, as described in Chapter 12.
2. Mark specific globals to be journaled within each application that is to use journaling. To mark globals for journaling, you use the ^%GBLATR utility or the utilities on the ^%JOURNAL menu.
3. Start one DSM Journal Process to perform journaling for the users of each application for which you plan to enable journaling. You can use a command file supplied by the VAX–11 DSM system to start DSM Journal Processes.
4. Interact with the DSM Job Controller to enable groups of DSM user images to use the Journal Process. When users invoke DSM in Application Mode, the DSM Job Controller tells the DSM image if journaling is enabled or disabled for that process.

A Journal Process can accept parameters to tailor the journaling operation either at start-up or while a Journal Process is running.

During start-up, a Journal Process acquires parameters through a start-up parameter file.

At run-time, a Journal Process acquires parameters through DSM operator intervention with Journal Process utilities. The communications link between the DSM operator and a Journal Process is the Operator Request mailbox associated with that Journal Process. The Journal Process reads the contents of this mailbox and sends replies back to the operator if requested to do so.

13.1.2 Defining Users of Journaling

To specify which DSM applications are enabled to use journaling, the operator uses either the DSM Job Controller's start-up option file (described in Section 12.3.2) or a Job Controller operator utility (described in Section 12.4).

Journaling can be enabled system-wide or for members of specified groups. Users running the same DSM application (that is, accessing the same globals and routines) must have the same group number. (Conversely, users with the same group number are assumed to be sharing resources; there can be no more than one Journal Process per group.)

If journaling is enabled system-wide, all users running DSM applications record SET and KILL transactions on the same disk or magnetic tape files through the same Journal Process. If journaling is enabled on a group basis, only members of the same group record transactions on the same device or file through a single Journal Process.

13.1.3 Journal Output Files and Journal Records

The Journal Process writes the information it receives from user processes as *journal records* in a set of output files. As described in Section 13.3, you can specify output files when you start the DSM Journal Process, or a DSM operator can specify them through utilities. The operator can then maintain some control over these files through utilities while a Journal Process is running.

Each DSM Journal Process keeps a list of available output file names. The Journal Process writes journal records to the current output file. It keeps writing to the current file until one of the following events occurs:

- A pre-defined number of blocks have been written to the file
- An error occurs while writing to the file
- The operator intervenes

When one of these occurs, the Journal Process closes the current file, opens the next file on the list, and continues writing journal records.

The journal record for a data base transaction contains the following information about a transaction:

- The full file specification of the global variable
- The assigned value, if SET
- The user's Process ID (PID)
- The user's UIC
- The date and time of the transaction

Whenever the Journal Process uses a new output file, the Journal Process writes its file specification in a Journal Log file (file type .LOG), which maintains a list of all files actually used by the Journal Process.

13.1.4 Dejournaling

The opposite of writing journal records about a data base transaction is restoring the journal records from the journal files. This process is called *dejournaling*.

To dejournal, you can use a VAX–11 DSM utility called %DEJRNL. This utility asks you to specify either of the following:

- The Journal Log file which contains the specifications of the output files used by the Journal Process
- A list of the journal output files (file type .JRN) from which to restore your globals

Then, the utility gives you the option of restoring all or specific globals.

You can also write your own dejournaling utility, using the description of a journal record given in Appendix C.

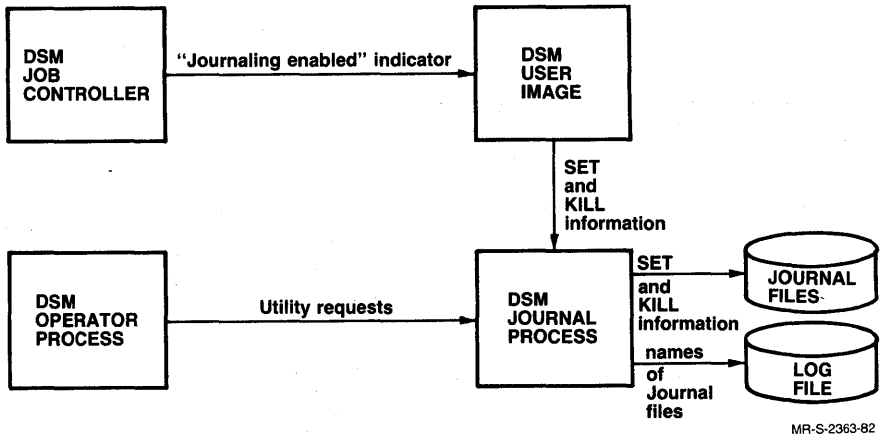
13.2 Communication between Journal Process and User Image

Communication between a user running the DSM image and a Journal Process is accomplished through the use of mailboxes. Each Journal Process has two mailboxes:

- The Journal Input mailbox (DSM$JOURNAL) — This is where DSM user images send information when they SET or KILL a global variable that is marked for journaling.
- The Operator Request mailbox (DSM$JOURNAL$COMMAND) — Accessible to a VAX–11 DSM operator through the DSM utilities on the ^%JOURNAL menu.

Figure 13–1 illustrates the interaction between DSM user images, the DSM Job Controller, and a Journal Process.

Figure 13–1: Journal Process Job Controller/User Image Interaction



Journaling proceeds according to the following sequence of events:

1. When DSM is invoked in Application Mode, a link is established between the user image and the DSM Job Controller, as described in Chapter 12.
2. If journaling has been enabled for the system or the group of which the user is a member, the DSM Job Controller sends a "journaling enabled" indicator to the DSM user image.
3. The DSM user image looks for the logical name DSM$JOURNAL. If the Journal Process was started system-wide, DSM$JOURNAL is a system logical name. If the Journal Process was started for the current group, DSM$JOURNAL is a group logical name. This name (if present) corresponds to the input mailbox of a Journal Process started for the user's group (or for the system). If this logical name is not present, a journal initialization failure occurs.

4. When the DSM user image performs a SET or KILL operation on a marked global, it writes the corresponding information to the mailbox DSM$JOURNAL.

5. The Journal Process reads this information from the mailbox and writes it to its current journal output file.

13.3 Defining Journal Output Files

You must give a Journal Process a list of the names of files to which it can write journal information. You set up this list using multiple /OUTPUT qualifiers in the Journal Process's start-up option file, described in Section 13.5.2. The DSM operator can then add to and maintain the list through utilities.

The Journal Process writes journaling information to the current output file. It switches to another file when:

1. The current file exhausts its specified or predefined allotment of space.

2. The DSM operator closes the current file and opens the next one from the list of names, or by using the REPLY/ABORT command described below.

When the Journal Process starts writing to a new file, it sends the full file specification to a Journal Log File.

You set the size of journal output files by specifying the /BLOCK=n subqualifier with the /OUTPUT qualifier when you start up the Journal Process, as described in Section 13.5.2.1.

Using multiple output files has several advantages over using one file. After a Journal Process has filled and closed a file, you can dump the contents of that file onto another storage device, such as a magnetic tape. The Journal Process can then reuse the space freed.

13.4 Marking Globals to be Journaled

To indicate which globals in an application should be journaled, use the VAX–11 DSM utilities SETGLO^%JRNL or ^%GBLATR.

Note that if you KILL an entire global marked for journaling, the journal indicator associated with that global disappears. Thus, if you subsequently recreate that global, you must remark it for journaling.

You can cause records to be written to the journal output file without causing an actual global access. You do this by setting a node in the virtual global ^%JOURNAL. When you SET ^%JOURNAL, a journal record is written to the journal output file in the same format as a record for a global transaction. There is no set use for the information that you store in ^%JOURNAL records. You can use these records to place "comments" in the output file. For example, you can note the beginning and end of a series of transactions for a particular process. (Note that the Process ID of the process is recorded in the journal record.)

If you wish to make use of the ^%JOURNAL records that you have placed in an output file, you must provide your own dejournaling utility. The dejournaling utility provided with VAX–11 DSM (^%DEJRNL) ignores records created by setting nodes of ^%JOURNAL. See Appendix C for the format of a journal record.

13.5 Starting a Journal Process

The VAX–11 DSM distribution kit includes a command file that starts up a DSM Journal Process. This command file is called DSMJRNSTA.COM; it resides in the VAX/VMS directory SYS$LIBRARY. It accepts an input parameter file called DSMJRNPAR.OPT that also resides in SYS$LIBRARY.

NOTE

If you are going to create more than one Journal Process, you may want to make copies of DSMJRNSTA.COM and DSMJRNPAR.OPT with different names. These copies should retain (respectively) the .COM and .OPT file types, however.

The DSMJRNSTA.COM file contains the DCL RUN command and a number of qualifiers. The image that is executed by the RUN command is SYS$SYSTEM:DSMJRN.EXE. Thus the format of the start-up command file is:

```
$RUN–
/UIC='P2' –
/PROCESS_NAME=DSM_JOURNAL
/OUTPUT=NL:–
/ERROR='P3' –
/PRIVILEGES=(NOSAME, TMPMBX, GRPNAM, SYSNAM)–
/INPUT='P1' –
SYS$SYSTEM:DSMJRN.EXE
```

The following sections describe the start-up command file and the start-up option file that is used with it.

13.5.1 The Journal Process Start–up Command File

The RUN command causes VMS to create a Journal Process as a detached process and to execute an image called DSMJRN.EXE.

The qualifiers associated with the RUN command provide information about the files that you wish to use and the privileges that you wish to assign to the Journal Process. The qualifiers can be specified in any order. Table 13–1 lists the qualifiers associated with the RUN command in the order used in DSMJRN.EXE.

Table 13–1: Command Qualifiers in DSMJRNSTA.COM

Component	Function
/UIC (P2)	Specifies the UIC group and member numbers of a Journal Process. To interact with this Journal Process at a later time, the DSM operator must have the same UIC as the Journal Process. The default value of this qualifier is your current UIC.
/PROCESS_NAME[=name]	Specifies the name of the Journal Process for a particular DSM image or group. The default name is DSM_JOURNAL. If you change the name of a Journal Process, make sure the name you choose is unique within the group using this Journal Process.
/OUTPUT[=name]	Specifies the name of the Journal Process's output file or device. /OUTPUT assigns an equivalence name for the logical name SYS$OUTPUT. The Journal Process writes internal warning, error, and information messages to this device or file. In addition, the Journal Process duplicates all messages other than success messages at the device or file you specify with /ERROR (SYS$ERROR). Therefore, /OUTPUT should be null (NL:) to ensure a single copy of messages.
/ERROR (P3)	Specifies the file name to which the Journal Process sends internal warning, error and information messages. /ERROR assigns an equivalence name for the logical name SYS$ERROR. The text of messages is stored in the specified file. If the Journal Process does not issue any messages, it does not create the file. The default name for this file is DSMJRN.ERR. The error file is important as a record of possible problems. This file is created on the default device and directory that are in effect when you start the Journal Process, unless you specify a device and directory.
/PRIVILEGES[=list]	Lists the privileges a Journal Process must have to run. The privileges are: • NOSAME, a convention that provides the Journal Process with only the privileges listed in the command file. • TMPMBX allows the Journal Process to create temporary mailboxes. • GRPNAM allows the Journal Process to create a logical name for the Journal Input mailbox (DSM$JOURNAL) on a group basis. This privilege is required if the /GROUP subqualifier is specified in the Journal Process option file. • SYSNAM allows the Journal Process to create a logical name for the Journal Input mailbox (DSM$JOURNAL) on a system basis. Any DSM user image in the system can write to a mailbox with a system logical name. This privilege is required if the /SYSTEM subqualifier is specified in the Journal Process option file.

(continued on next page)

Table 13–1(Cont.): Command Qualifiers in DSMJRNSTA.COM

Component	Function
/INPUT (P1)	Directs the DSM Journal Process to examine an option file. The default name for this option file is SYS$LIBRARY:DSMJRNPAR.OPT. However, you can tailor copies of this file to suit your needs, assigning a different name to each process's copy. different file name for each process.

As shown in Table 13–1, the command string includes three parameters for which you can accept the default values or specify your own values. These parameters are:

- P1, applied to the /INPUT qualifier, specifies the name of the start-up option file that the Journal Process uses initially to run. If you do not specify a name, the Journal Process uses the default file, SYS$LIBRARY:DSMJRNPAR.OPT, for which a sample file is supplied with the distribution kit.
- P2, applied to the /UIC qualifier, specifies the UIC of the Journal Process. If you do not specify this parameter, the Journal Process uses your current UIC.
- P3, applied to the /ERROR qualifier, specifies the name of an error log file that is created if the Journal Process generates any warning, error, or information messages. If you do not specify a name, the file is created in your current directory under the name DSMJRN.ERR.

Note the three parameters in the following example of a command line that invokes the Journal Process start-up command file:

```
$ @SYS$LIBRARY:DSMJRNSTA MYJRNPAR.OPT [100,100] MYJRN.ERR
```

In this example, the Journal Process starts up with UIC [100,100] using a start-up option file called MYJRNPAR.OPT in the current default directory. It then writes messages to a file called MYJRN.ERR that the Journal Process creates in the default directory.

Note that you can modify your site-independent start-up file so your DSM Journal Process(es) start automatically when you boot the system. Refer to the *VAX/VMS System Management and Operations Guide* for details on tailoring the site-independent start-up file.

13.5.2 The Journal Process Start–up Option File

The /INPUT qualifier in the DSMJRNSTA.COM file identifies the option file to be used in starting up the Journal Process. If you do not specify a name, the Job Controller uses the file SYS$LIBRARY:DSMJRNPAR.OPT. You can construct your own start-up option file (with the default name or any name that you choose), or use the sample file supplied with the distribution kit.

The DSMJRMPAR.OPT contains qualifiers and the parameters associated with them, in the following general format:

/[NO]QUALIFIER/SUBQUALIFIER=[PARAMETER][!COMMENTS]

Table 13–2 shows the qualifier options and their defaults. Table 13–3 describes the qualifiers in detail.

Table 13–2: Qualifiers Used in DSMJRNPAR.OPT

Qualifier	Default
/JOURNAL_INPUT [/GROUP] [/SYSTEM]	/GROUP
/COMMAND [/GROUP] [/SYSTEM]	/GROUP
/LOG=file name.LOG	DSMJOURNA.LOG
/OPERATOR=[CENTRL] [OPER1...OPER12]	CENTRL
/MAILBOX_SIZE=n	603 bytes
/OUTPUT=name.JRN[/BLOCK=n] [/MESSAGE][/REPLY="text"]	n is extended to VAX/VMS limit if not given or 0

Table 13–3: Qualifier Summary

Qualifier	Function
/JOURNAL_INPUT	Determines whether the Journal Input mailbox is created as a group or a system mailbox. You can specify a subqualifier with /JOURNAL_INPUT as follows: • /GROUP creates a group logical name for the Journal Input mailbox. • /SYSTEM creates a system logical name for the Journal Input mailbox
/COMMAND	Determines whether the Operator Request mailbox is created as a group or a system mailbox. You can specify a subqualifier with /COMMAND as follows: • /GROUP creates a group logical name for the Operator Request mailbox. • /SYSTEM creates a system logical name for the Operator Request mailbox
/LOG	Defines the name of the log file into which the Journal Process logs the full file specifications of output files used. The default name is DSMJOURNA.LOG.

(continued on next page)

Table 13–3 (Cont.): Qualifier Summary

Qualifier	Function
/OPERATOR	Indicates to the Journal Process which terminal to send messages to if the /MESSAGE qualifier is used. If /OPERATOR is omitted, the default operator terminal is CENTRL, the VAX/VMS operator console. There can be up to 12 other operator terminals, however, each with a specialized purpose, designated OPER1 through OPER12. You can designate any terminal as an operator terminal by using the DCL command $REPLY/ENABLE=(OPER1,...OPER12), as described in Section 13.5.2.2.
/MAILBOX_SIZE	Determines the size of messages that you can write to the Journal Input mailbox. The default size is 603 bytes, corresponding to the default value of the DSM command qualifier /KEY_SIZE, which is 64 bytes. You must increase mailbox size if you increase /KEY_SIZE at DSM image start-up. (See Section 9.7.3.1 for a description of the /KEY_SIZE qualifier.)
/OUTPUT	Specifies the file name of an output file in which the Journal Process stores journal transactions. You specify multiple file names by repeating this qualifier. If you do not specify the file type, the default is .JRN. You can also specify subqualifiers with /OUTPUT as follows: • /BLOCK=n limits the size of the output file to n blocks, as described in Section 13.5.2.1. • /MESSAGE=text sends the specified text to the operator, as described in Section 13.5.2.2. • /REPLY indicates that the operator must respond to the message before the Journal Process can use the file. See Section 13.5.2.2 for information on replying to the message.

13.5.2.1 Use of the /BLOCK Subqualifier — The /BLOCK subqualifier of the /OUTPUT qualifier defines the maximum number of blocks (n) to be written to an output file. If the subqualifier is omitted or if n=0 and the file is not predefined, the size of the file is determined by VAX/VMS limits. If /BLOCK is omitted or if n=0 and the file is predefined, the size of the file is limited to the preallocated space.

When the maximum number of blocks is reached, the Journal Process uses the next file that you specified with /OUTPUT. If you did not specify more than one /OUTPUT qualifier, you must add another file with the ADD JOURNAL FILE NAME and OPEN CURRENT JOURNAL FILE utilities described in Section 7.7. The Journal Process delays further execution until a new output file is available.

The switch from one file to the next or from one output device to the next is handled internally and (in most cases) transparently. However, if you want to know when the transfer occurs, you can include The /MESSAGE subqualifier with /OUTPUT, as described in the following section.

13.5.2.2 Use of the /MESSAGE and /REPLY Subqualifiers — If a Journal Process's start-up option file contains the /MESSAGE and /REPLY subqualifiers with the /OUTPUT qualifier, the Journal Process communicates with the VAX/VMS operator to issue such messages as "OPENING NEXT FILE" when the file specified with /OUTPUT is opened. The /MESSAGE=xxx subqualifier specifies the message (here, xxx) sent to the terminal. /REPLY indicates that the DSM Journal Process waits for a reply from the VAX/VMS operator.

You specify the terminal to which the message specified is sent by including the /OPERATOR=OPERn qualifier in the start-up option file, as described in Table 13–3. The default terminal (if you do not include /OPERATOR) is the VAX/VMS console terminal.

You designate a terminal to be a VAX/VMS operator's terminal when you issue the DCL REPLY/ENABLE=OPERn command at the terminal, with the value of OPERn equal to the value of OPERn included in the /OPERATOR qualifier. To reply to a message, you can use any of the following forms of the DCL REPLY command:

$ REPLY/TO=id	Causes the Journal Process to open the output file and resume operation.
$ REPLY/ABORT=id	Causes the Journal Process to switch to the next output file.
$ REPLY/PENDING=id	Causes the Journal Process to wait for an additional reply before it resumes operation.

The argument "id" is the message number of the message from the DSM Journal Process. If there is only one message requiring a reply, id is 0. For details about the REPLY command, refer to the *VAX/VMS System Management and Operations Guide.*

13.5.3 Journal Process Examples

You can invoke the DSMJRNSTA command file with different sets of parameters by specifying different start-up option files. For example, you can set start one Journal Process by issuing the command:

```
@SYS$LIBRARY:DSMJRNSTA JRN1PAR.OPT [100,100] JRN1.ERR
```

This command specifies the start-up option file JRN1PAR.OPT, which contains the following qualifiers:

```
/LOG=A.LOG
/OPERATOR=OPER1
/MAILBOX_SIZE=1024
/OUTPUT=J1A.JRN/BLOCK=1000
/OUTPUT=J1B.JRN/BLOCK=1000
/OUTPUT=J1C.JRN/BLOCK=1000/MESSAGE="OPENING LAST FILE"
```

You can start another Journal Process (for a different group) by invoking DSMJRNSTA.COM with different parameters, for example:

@SYS$LIBRARY:DSMJRNSTA JRN2PAR.OPT [120,120] JRN2.ERR

This command specifies the start-up option file JRN2PAR.OPT, which contains the following qualifiers:

```
/INPUT/SYSTEM
/COMMAND/SYSTEM
/LOG=B.LOG
/OPERATOR=CENTRL
/MAILBOX_SIZE=512
/OUTPUT=J2A.JRN/BLOCK=1000
/OUTPUT=J2B.JRN/BLOCK=1000
/OUTPUT=J2C.JRN/BLOCK=1000/MESSAGE="OPENING LAST FILE"
```

13.6 Utilities that Interact with the Journal Process

Sections 7.6.1, 7.7.2, 7.7.3, and 7.7.4 describe the VAX–11 DSM utilities that the DSM operator needs to control the DSM Journal Process. All of these utilities can be invoked through the ^%JOURNAL menu. Table 13–4 lists these sixteen utilities and their entry points.

WARNING

You must have the same UIC as the DSM Journal Process to use these utilities.

Table 13–4: Journaling Utilities

Utility	Entry Point
Add Journal File Name	ADDFIL^%JRNL
Clear Journaling for Global	CLRGLO^%JRNL
Close Current Journal File	CLOSEFI^%JRNL
Delete Journal File Name	DELFIL^%JRNL
Disable Journal Process	DISABLE^%JRNL
Enable Journal Process	ENABLE^%JRNL
Kill Journal Process	KILL^%JRNL
New Log File	NEWLOG^%JRNL
Open Current Journal File	OPENFIL^%JRNL
Pause Journal Process	PAUSE^%JRNL
Purge Journal File Names	PURGE^%JRNL
Resume Journal Process	RESUME^%JRNL
Set Journaling for Global	SETGLO^%JRNL

Table 13–4 (Cont.): Journaling Utilities

Utility	Entry Point
Show Journal File Name	SHOW^%JRNL
Status of Journal Process	STATUS^%JRNL
Test Journaling for Global	TSTGLO^%JRNL

Chapter 12 lists the ^%OPER utilities that interact with the DSM Job controller to enable journaling.

13.7 Journaling to Magnetic Tape

If you wish to journal to magnetic tape, the DSM Journal Process must be started as a batch job, not as a detached process. You cannot use DSMJRNSTA.COM, because it starts the Journal Process as a detached process.

The following is a sample batch command procedure, MAGJRNSTA.COM, that can be used to start a Journal Process as a batch job:

```
$ ! Batch command procedure to start Journal Process
$ SET NOON
$ MOUNT <magtape> <label>
$ RUN SYS$SYSTEM:DSMJRN
/OUTPUT=<magtape>AJRN1.JRN/BLOCK=1000
/OUTPUT=<magtape>AJRN2.JRN
$ DISMOUNT <magtape>
```

Before submitting the above procedure as a batch job, you must be logged into a user account that has the following attributes:

- The privileges required by a DSM Journal Process
- A UIC corresponding to that of the DSM application for which you want to start the Journal Process

Remember that a batch job goes through a complete VMS log-in procedure, including the execution of the log-in command file, before executing the commands in the batch command file.

The commands in the file shown above have the following functions:

1. The first command, SET NOON, is necessary because the DSM Journal Process always exits with an error (normally, the status corresponding to "Terminated by operator"). SET NOON forces the batch command procedure to execute the last statement, that is, dismount the tape.
2. The second statement mounts the tape on which the Journal Process will create the journal files AJRN1.JRN and AJRN2.JRN.

3. The RUN command runs the journal image within the context of the current process.

4. The statements that follow RUN are the parameters to the Journal Process normally included in the Journal Startup Options file. In case of a batch job, SYS$INPUT is the batch control file. Therefore, the journal image reads options until it encounters a line starting with a dollar sign ($).

5. After the Journal image exits (for instance, by operator shutdown) the next statement, DISMOUNT, is executed to dismount the tape.

You should set up a separate batch queue for the DSM Journal Process. See the *VAX/VMS System Management and Operations Guide* for information of batch queues.

Assuming that you have created a batch queue called DSM$JBATCH, you can submit the above command procedure by entering the following statement:

```
$ SUBMIT MAGJRNSTA/QUEUE=DSM$JBATCH
```

Appendix A
VAX–11 DSM Error Messages

This appendix describes the error messages generated by VAX–11 DSM. It does not describe VAX/VMS system error messages or VAX–11 RMS error messages. For a detailed description of these messages, refer to the *VAX/VMS System Messages and Recovery Procedures Manual* and the *VAX–11 Record Management Services Reference Manual.*

A.1 Error Message Types

VAX–11 DSM error messages fall into two general categories:

1. Those VAX–11 DSM reports interactively at the terminal at run-time.
2. Those VAX–11 DSM reports in the Job Controller error log file and the Journal Process error log file.

VAX–11 DSM reports the following types of errors at the terminal:

- Compiler/Interpreter errors
- Global handler errors
- I/O errors
- Routine manipulation errors
- General information and warnings

VAX–11 DSM reports most Job Controller-related and journaling-related errors in the Job Controller and Journal Process error log files.

A.2 Error Message Format

VAX–11 DSM reports error messages in the following format:

%DSM–SEVERITY–MESSAGE ID, MESSAGE

SEVERITY represents the severity level of the error. Errors can have one of four severity levels, as follows:

WARNING

INFORMATION

ERROR

SEVERE

Errors having warning-level or information-level severity values do not inhibit the execution of a routine. The severity code for Warning is W; the severity code for Information is I.

Errors having error-level severity values are fatal to a DSM routine (unless you set $ZT), that is, they prevent a routine from executing to completion. Such errors must be corrected for a DSM routine to execute properly. The severity code for Error is E.

Severe errors are fatal to the DSM image, that is, they cause DSM image rundown. The severity code for Severe errors is F.

Unless you run a routine in Application Mode, DSM errors are generally not severe errors. However, DSM Job Controller and Journal Process startup errors are always severe errors. DSM command line-related errors are also severe errors.

A.3 Description of VAX–11 DSM Error Messages

The error messages VAX–11 DSM reports at the terminal are described below in alphabetical order. Section A.3.1 describes most Job Controller and Journal Process error messages. Unless otherwise noted, all errors described this section are of error-level severity.

ALLOC

Indicates a symbol table allocation failure. This error occurs when you try to set more local variables than you have room to store in the local symbol table.

ATOM

Indicates a bad expression atom. An expression atom is an element of a string of characters that yields a value when evaluated. An expression atom can be a variable, a literal, a function, an expression atom preceded by a unary operator, or an expression enclosed in parentheses.

ATLABEL

Indicates the routine line and name of the routine in which the primary DSM error occurred (Information). ATLABEL always follows the primary error message in the following format:

–DSM–I–ATLABEL, Routine Line^Routine Name Source Line

BLOCKERR

Indicates an attempt to read a magnetic tape block that is too large for the input buffer.

BREAK

Indicates that a breakpoint was reached, either through the BREAK command or through a setting of $ZBREAK. (Information)

BUG

Indicates the occurrence of an unexpected internal error, such as a bug in the system software. If VAX–11 DSM generates a BUG error, consult your DIGITAL Software Specialist for diagnostic assistance.

CLOSEP

Indicates a missing right parenthesis.

CLOSERR

Indicates an error closing a device or file.

CODEFULL

Indicates a precompiled routine buffer overflow.

COMAND

Indicates a bad DSM command was detected. This error can occur if you enter more than one or two characters of the command name, but less than the total number of characters in the command name. For example, if you enter the following, it generates a COMAND error:

```
>ZLO
```
For ZLOAD (or ZL)

```
>EL
```
For ELSE (or E)

COMEND

Indicates bad characters at the end of a statement. For example:

```
>S A=B+C,1
```
CTLC

Indicates a CTRL/C interrupt was typed on the principal device.

CTRAP

Indicates that a trap character (declared using USE 0:CTRAP) has been entered on the principal device.

CTRAPERR

Indicates an unsuccessful attempt to trap a CTRL/C interrupt. This error generally occurs when you exceed your AST limit.

DEALLOC

Indicates an unexpected memory deallocation failure.

DELETERR

Indicates an unsuccessful attempt to delete the record specified in the current I/O request. This error occurs after an unsuccessful USE:DELETE command.

DEVALLOC

Indicates a physical device or file is unavailable for use because another user has allocated it. This error can also occur if a device does not yet exist, specifically, a mailbox.

DEVNOTOPN

Indicates an attempt to use a device (with the USE command) that was not opened with the OPEN command.

DISCONERR

Indicates an error disconnecting the file on USE:DISCONNECT.

ENDOFILE

Indicates the last record in the file has been retrieved and that no more records exist.

EQUALS

Indicates the equals sign (=) is missing from the current FOR or SET statement.

ESC_APPL

Indicates an attempt to use the ZESCAPE command in Programmer Mode.

FINDERR

Indicates an unsuccessful attempt to find the record specified in the current I/O request. This error occurs on unsuccessful USE:KEY and USE:RFA commands.

FORCEXIT

Indicates forced DSM image rundown by the DSM operator (via DSM Job Controller utilities).

FRAME

Indicates an unexpected interpreter frame error. A frame error is an internal system error and indicates a bug in the VAX–11 DSM software. If VAX–11 DSM still generates a FRAME error after you recompile your routines with ^%REPLACE, consult your DIGITAL Software Specialist for diagnostic assistance.

FUNC

Indicates faulty use of a function in the DSM language set or that the function is not defined.

GBLCLOSER

Indicates an unsuccessful attempt to close a global.

GBLDATERR

Indicates an attempt to read a global record in an invalid format; this error can occur if the key does not end with two zero bytes.

GBLDATOVF

Indicates an attempt to read a global record that exceeds 255 characters in length. This error can occur if you try to read an improperly formatted indexed file as a global.

GBLDELERR

Indicates an unsuccessful attempt to delete a global node or an entire global array (with the KILL command).

GBLOPENER

Indicates an unsuccessful attempt to open a global file.

GBLREADER

Indicates a read error on the current global access.

GBLWRITERR

Indicates a write error on the current global access.

HALT

Indicates the DSM HALT command was executed. (Information)

ILLBRK

Indicates the use of an illegal command in BREAK mode. Illegal commands are ZINSERT, ZREMOVE, and ZLOAD.

ILLOPT

Indicates an illegal I/O option or an I/O command syntax error on OPEN, CLOSE, or USE.

INTVER

Indicates a mismatch between the current version of the VAX–11 DSM interpreter and the version of the interpreter that generated the precompiled code you are trying to execute. To generate compatible code, use the ^%REPLACE utility.

INVCOL

Indicates that a global variable has been marked with an invalid collating sequence indicator.

INVMAPHDR

Indicates the file specified as the object of the /MAPPED qualifier has an invalid format, and thus cannot be mapped as a DSM routine section. (Severe)

INVSUBSCR

Indicates a null string ("") in the subscript field of a global variable; the null string is an invalid global subscript.

JRNLDATOVF

Indicates the journal mailbox is too small to accept journal records with the keysize of the global being accessed.

JRNLINFAIL

Indicates DSM cannot find the logical name DSM$JOURNAL at startup, and thus cannot start in Application Mode. This error indicates that though journaling has been enabled for the current user, but that no Journal Process has been started for this user's group or for the system. (Severe)

JRNLINSBUF

Indicates the journal mailbox is too small to accept journal records with the current global keysize. (Severe)

JRNWRITER

Indicates an unsuccessful attempt by a DSM user process to write to the journal mailbox.

LABEL

Indicates a bad routine line label was detected.

LCKBUFOVF

Indicates the current lock request caused a lock buffer overflow. Generally, this error occurs when you try to lock (or unlock) too many variables in a single lock statement (LOCK, ZALLOCATE, ZDEALLOCATE).

LCKINTFAIL

Indicates an unsuccessful attempt to establish communication with the DSM Job Controller at DSM image startup. This error can occur if the logical name DSM$GLOBAL_DIR does not translate properly. (Severe)

LITERAL

Indicates a bad ASCII literal was encountered.

LOCALSS

Indicates the length of a local variable subscript exceeds 59 characters; 59 characters is the maximum permissible length of the local variable subscript field.

MAPERR

Indicates an unsuccessful attempt to map a file in virtual memory with the /MAPPED=file–spec command qualifier. This can happen if the file is in the wrong format. (Severe)

MBXBUFFUL

Indicates the mailbox output buffer is full, or an attempt to send a message larger than the mailbox's capacity. After this error occurs, VAX–11 DSM clears the output buffer.

MJCINTFAIL

Indicates the communication link between the DSM image and the Job Controller was not established. A secondary DSM error message always follows MJCINTFAIL to further explain the error, typically, NOLOGINS or MJCNOTRUN. (Severe)

MJCINVFMT

Indicates the Job Controller cannot read the message sent to it at login by the DSM user process. (Severe)

MJCNOTRUN

Indicates an attempt to log into the Job Controller before is has been started. (Severe)

NAKEDERR

Indicates an attempt to reference a global variable using naked syntax before a full syntax reference to the global or when the naked reference is undefined, for example, after referencing a root node.

NAKEDOVF

Indicates a global reference exceeds the maximum permissible key size.

NAME

Indicates a bad variable name was encountered, for example, a variable name that uses graphics characters (such as $ or *).

NEXTERR

Indicates an unsuccessful attempt to request the next volume of a volume set for magtape I/O. This error occurs after an unsuccessful USE:NEXT command.

NODUNDEF

Indicates a reference to a global variable node that is not defined.

NOFILMAP

Indicates the absence of a file name in the current mapping request. This error is fatal to the mapping operation.

NOLOGINS

Indicates that DSM logins have been disabled for application users (via Job Controller utilities).

NOPRIV

Indicates insufficient privilege to execute the specified operation.

NOSUCHPGM

Indicates a reference to a routine name that does not exist in the current routine directory or is not in the library routine directory.

NOTCREATED

Indicates the specified file was not mapped in virtual memory because the section already exists. (Warning)

NUMBER

Indicates the use of an illegal number, such as MAXINT+1.

OPENERR

Indicates an error opening a device or file. This error can occur if you try to open a device to which you have insufficient access privileges or if you specify an illegal file specification.

PARSERR

Indicates an unsuccessful attempt to construct the file specification for a file at start-up. This error occurs when VAX–11 DSM cannot apply the necessary defaults to the file specification for the DSM routine or global directories. (Severe)

PATTERN

Indicates an invalid pattern match operator code.

PGMDELERR

Indicates an unsuccessful attempt to delete a routine.

PGMERR

Indicates an unsuccessful attempt to load a routine into your routine buffer. This error is the same as PGMLOADER. However, VAX–11 DSM returns this error message in $ZE when error trapping is enabled.

PGMLOADERR

Indicates an unsuccessful attempt to load a routine into your routine buffer. This error is the same as PGMERR. However, VAX–11 DSM returns this error when error trapping is disabled.

PGMMAPPED

Indicates that saving the routine (with ZSAVE) did not update the mapped version of the routine. (Warning)

PGMOPNERR

Indicates an unsuccessful attempt to open a DSM routine directory file. This error can occur if you try to open a file to which you have insufficient access privileges or if DSM$ROUTINE_DIR translates to an invalid device and file specification.

PGMSAVERR

Indicates an unsuccessful attempt to save a routine (with the ZSAVE command). This error can occur if you have insufficient privilege to perform a save operation on the specified file. For example, if you try to save a library utility routine, VAX–11 DSM returns a PGMSAVERR error.

PNAME

Indicates an invalid routine name. For example, a routine name that includes graphics characters.

PROCQUOTA

Indicates that insufficient process quotas (such as FILLM) exist for a global access. When this occurs, VAX–11 DSM automatically closes globals in the Open Globals List to compensate. However, if this error occurs repeatedly, process quotas should be increased.

READERR

Indicates an error during a read operation.

RENAMERR

Indicates an unsuccessful attempt to rename a file with the CLOSE:RENAME command.

SEARCHERR

Indicates an error on the current $ZSEARCH operation.

SPACERR

Indicates an unsuccessful attempt to position a magnetic tape for a Block I/O operation. This error occurs after an unsuccessful USE:SPACE command.

SRCFULL

Indicates a source routine buffer overflow.

STKOVR

Indicates an interpreter stack overflow. This error is generally caused by faulty programming techniques, such as:

```
A1 F I=1:1 D A1+1
D A1
```

Or

```
>S X="X X"
>X X
```

STRLEN

Indicates the string length exceeds 255 characters.

SYNTAX

Indicates a DSM command line syntax error. (Severe)

SYSQUOTA

Indicates that VAX/VMS system quotas, such as the RMS shared file pool, are low and may be exceeded. When this occurs, VAX–11 DSM automatically closes globals in the Open Globals List to compensate. However, when this message appears, degradation of system performance is inevitable, and VAX–11 DSM should be shut down and system quotas increased. (Warning)

UNDEF

Indicates a reference to an local variable that does not exist or to a line label that does not exist.

WRITERR

Indicates an error in a write operation.

ZCARG

Indicates an error in specifying the $ZCALL function arguments.

ZCERROR

Indicates than an error was returned by $ZCALL.

ZCFATAL

Indicates a VAX/VMS error occurred that caused the current $ZCALL operation to be aborted. VAX–11 DSM reports this error immediately following the ZCFATAL message. A ZCFATAL error can also be generated if a user-written program engages in faulty programming practices, for example, returning a fatal status in R0.

ZCINPUT

Indicates that the input passed to the $ZCALL function is invalid.

ZCNAME

ZCOUTPUT

Indicates that the output passed from the $ZCALL function is invalid.

ZCPARSE

Indicates that the $ZCALL function name or arguments specified are invalid.

ZCSIGNAL

Indicates that a condition was signalled by the $ZCALL routine.

ZCTABLE

Indicates an error in the ZCALL Table.

ZCWARN

Indicates that a warning was returned by $ZCALL.

ZGO

Indicates an attempt to use the ZGO command before the BREAK command was executed.

ZJOBERR

Indicates an unsuccessful attempt to create a ZJOB process. This error can occur if you exceed your process quotas or if you do not have the appropriate privileges to start a detached process.

ZJOBINTERR

Indicates the ZJOB process generated an error at startup.

ZJOBREPLY

Indicates an unsuccessful attempt by a ZJOB process to send an acknowledgement message to the creating process.

ZJOBRPYERR

Indicates the ZJOB process generated an error after startup but prior to executing the specified routine. In other words, an error occurred between the following two events:

1. ZJOB process initialization
2. The point at which the specified routine is loaded, including the routine load itself

ZQUIT

Indicates an attempt to use the ZQUIT command outside a declared error processing routine.

ZTRAP

Indicates that a software trap has been generated by the ZTRAP command.

A.3.1 Job Controller and Journal Process Error Messages

The error messages that VAX–11 DSM reports in the Job Controller and Journal Process error log files are described below in alphabetical order. VAX–11 DSM reports messages followed by a single asterisk (*) in the Job Controller error log file and messages followed by a double asterisk (**) in the Journal Process error log file. Messages followed by a percent sign (%) can appear in either error log file.

INTOPTION%

Indicates that the DSM Job Controller cannot locate the DSMMJCPAR.OPT file or the Journal Process cannot locate the DSMJRNPAR.OPT file. This error generally occurs as a result of an improper entry in either the DSMMJCSTA.COM or DSMJRNSTA.COM files for /INPUT. This error is fatal to both the Job Controller and Journal Process, that is, it causes process rundown. (Severe)

INVLCKFMT*

Indicates a lock request in the Job Controller mailbox that cannot be processed because it is in an invalid format.

INVOPRFMT**

Indicates an operator request in the Journal Process mailbox that cannot be processed because it is in an invalid format.

INVOPRREQ**

Indicates an invalid operator request, such as an out of range request number.

JRNCLOERR**

Indicates an unsuccessful attempt to close a journal output file (.JRN).

JRNEXIT**

Indicates Journal Process rundown. (Information)

JRNINIFAIL**

Indicates a Journal Process initialization failure. (Severe)

JRNINVFMT**

Indicates that the Journal Input mailbox contains a journal record in an invalid format.

JRNLOGCLO**

Indicates an unsuccessful attempt to close a journal log file.

JRNLOGERR**

Indicates a journal log file write error.

JRNLOGOPN**

Indicates an unsuccessful attempt to open a new journal log file.

JRNOPNERR**

Indicates an unsuccessful attempt to open a journal output file (.JRN).

JRNOPRRPY**

Indicates an unsuccessful attempt by the Journal Process to send the DSM operator a message.

JRNPARSE**

Indicates a formatting error in the DSMJRNPAR.OPT file. This error is fatal to the Journal Process, that is, it causes process rundown. (Severe)

JRNWRITE**

Indicates an unsuccessful write to the journal output file (.JRN).

MJCEXIT*

Indicates Job Controller process rundown. A secondary DSM error message always follows MJCEXIT that further explains the error, for example, OPERSTOP.

MJCINTPARS*

Indicates a formatting error in the DSMMJCPAR.OPT file. This error generally occurs as the result of a typographical error in the file. The MJCINTPARS error is fatal to the Job Controller process; it causes process rundown. (Severe)

MJCLCKRPY*

Indicates an unsuccessful attempt to respond to a DSM user's lock request.

MJCLOGRPY*

Indicates the Job Controller cannot reply to a DSM user at login.

MJCOPRRPY*

Indicates an unsuccessful attempt by the Job Controller to send an operator reply message to a DSM user.

MSGINVFMT*

Indicates an operator request in the Job Controller mailbox cannot be processed because it is in an invalid format. This error can occur as the result of an accidental write to the Job Controller mailbox.

OPRBUFOVF*

Indicates a Job Controller operator reply mailbox overflow. This error occurs when the length of the message sent by the Job Controller to the DSM operator exceeds the space remaining in mailbox.

OPREQERR*

Indicates an invalid request by the DSM Job Controller operator.

OPERSTOP*

Indicates the Job Controller has been stopped by the operator via DSM utility routines. (Information)

RPYMBXERR%

Indicates the DSM user mailbox has insufficient space to receive a message from either the Job Controller, if a Job Controller–related error, or the Journal Process, if a journaling-related error.

RPYSENDERR%

Indicates an unsuccessful attempt to send a message to a DSM user mailbox from either the Job Controller or the Journal Process. A secondary message always follows RPYSENDERR that further explains the error, for example, RPYMBXSIZ.

VMSOPRERR**

Indicates an unsuccessful attempt by the Journal Process to send a message to the VAX/VMS operator.

Appendix B
The VAX–11 DSM Editor

The VAX–11 DSM editor consists of seven prompts that are displayed sequentially. After the editor displays a prompt, you enter the requested information and the editor performs the specified operation. The following chart describes each editor prompt in detail. It lists the valid responses to each prompt and the results of their entry.

PROMPT 1: LINE

Response	*Result*
Carriage Return	The editor interprets a carriage return to mean that you want to edit the last line edited. If this is the first line request, or the line does not exist for the label reference, the editor writes "LINE NOT FOUND" and returns to the LINE prompt. Otherwise, it proceeds to the prompt 2.
. (period)	Ends the edit session.
; (semicolon)	Proceeds to the fourth editor prompt, CHANGE EVERY.
; followed by any DSM code	Temporarily leaves edit mode and executes the DSM code, then reenters the editor at the LINE prompt.
Label or label + offset	Attempts to obtain the $TEXT of the label reference and, if unsuccessful, writes "LINE NOT FOUND" and returns to the LINE prompt. Otherwise, it proceeds to prompt 2.
+ or +N, where N is a positive integer	Interprets this entry as a positive offset from the last line label entered. Treatment is as described in response 1: Carriage Return. Note that a +0 works exactly like a Carriage Return.

PROMPT 2: REPLACE

Response	*Result*
Carriage Return	Writes the line being edited and returns to prompt 1.
END	This entry has two meanings. If the line you are editing contains the string "END," the editor proceeds to prompt 3 to replace the occurrence of "END" with the response to prompt 3. Otherwise, the editor proceeds to prompt 3 to add something to the end of the line.
A string containing either code, a comment, or string literal	If the line contains the specified string, the editor proceeds to prompt 3 to replace it with the response to prompt 3.
SUBSTRING...	This entry has two meanings. If the line contains the string literal "SUBSTRING...," the editor proceeds to prompt 3 to replace that string with the response to prompt 3. If it does not contain this string literal, the editor proceeds to prompt 3 to replace all text from the occurrence of SUBSTRING to the end of the line with the response to prompt 3.
SUBSTRING 1 ...SUBSTRING 2	If the line contains the string literal "SUBSTRING 1...SUBSTRING 2," the editor proceeds to prompt 3 to replace the occurrence of the string with the response to prompt 3. If the line does not contain the string literal, the editor prints a "LINE NOT FOUND" error and returns to prompt 2.

PROMPT 3: WITH

Response	*Result*
Carriage Return	Replaces the string specifed in prompt 2 with nothing; that is, it removes it from the line and returns to prompt 2.
A string containing either code, a comment, or string literal	Replaces the previously specified string from prompt 2 with this response, and returns to prompt 2.

PROMPT 4: CHANGE EVERY

Response	*Result*
Carriage Return	Returns to prompt 1.
A string you wish to search for	Accepts whatever is typed, (including a ?) as the string to be searched for, and proceeds to prompt 5.

PROMPT 5: FROM LINE

Response	*Result*
Carriage Return	Assumes that you want to start searching from the the first line in the routine; the editor writes "FIRST," and then proceeds to prompt 6.
A line label and optional offset	If the line does not exist, the editor writes "LINE NOT FOUND," and repeats the prompt. Otherwise, the editor proceeds to prompt 6.

PROMPT 6: TO LINE

Response	*Result*
Carriage Return	Assumes that you want to search to the last line in the routine buffer, writes "LAST," and proceeds to prompt 7.
A line label and optional offset	If the line does not exist, the editor writes "LINE NOT FOUND," and repeats the prompt. Otherwise, the editor proceeds to prompt 7.

PROMPT 7: CHANGE TO

Response	*Result*
Carriage Return	Every occurrence of the "CHANGE EVERY" string is removed from the routine buffer. Responding to this question begins the search. Each time the editor finds a line with the "CHANGE EVERY" string in it, it removes the string and prints the line in its edited form. When all lines specified have been searched, the editor returns to prompt 1.
' (single quote)	Leaves all occurrences of the "CHANGE EVERY" string as they are; that is, the editor just performs a search. Each line that contains the "CHANGE EVERY" string is written out unchanged. When the editor is finished, it returns to prompt 1.
The string you want to insert	Proceeds with the search. Each time the editor finds an occurrence of the "CHANGE EVERY" string, it is changed to the "CHANGE TO" string and the edited line is written out. Note that the line is written out only once. Multiple occurrences of the "CHANGE EVERY" string are all edited before the line is written out. Editor returns to prompt 1.

Appendix C
Data Structures and Related Information

C.1 Data Structures

Figure C–1: Global Attribute Record

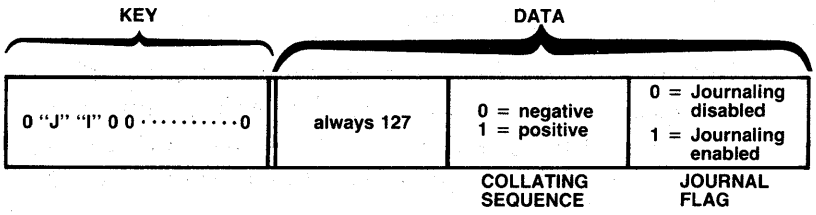
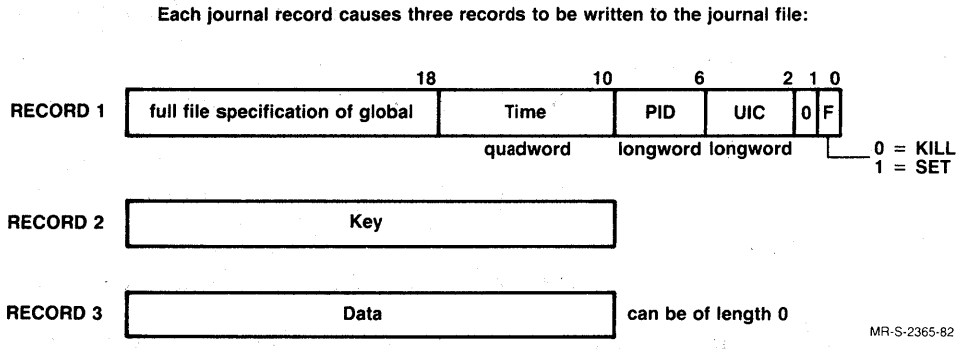


Figure C–2: Journal Record



C.2 Internal Subscript Format for a VAX–11 DSM Global

Each subscript in the node reference for a VAX–11 DSM global is constructed of a variable-length string of bytes, as follows:

coll. desc.	1st byte	2nd byte	...	last byte	null byte

Each key is a concatenation of subscripts, each with a trailing zero byte, as follows:

C	B	B	B...B	0	C	B	B	B...B	0	...C	B	B...B	0	0

The collation descriptor byte, the first byte in the subscript, has a value that depends upon the type of subscript. These values are:

0 – reserved
1 – null subscript
2–127 – negative canonic number having 124–0 integer digits
128 – canonic zero
129–253 – positive canonic number having 0–123 integer digits
254 – ASCII string
255 – reserved

When the subscript is an ASCII string, the bytes of the subscript following the collation descriptor byte are the bytes of the string.

When the subscript is a positive canonic number, the bytes after the collation descriptor byte are the ASCII values of the digits (including decimal point, if any) of the number.

When the number is a negative canonic number, the bytes following the collation descriptor byte are the two's complement of the ASCII values of the digits (and possible decimal point). In this case, the minus character is not included, but a byte of value 254 is appended to the end of the subscript.

All types of subscript have a trailing null byte, which acts as a separator between subscripts. After the last subscript, null bytes are added to fill out the remainder of the key.

The following examples show how particular global nodes are represented:

^("ABCD")
254 "A" "B" "C" "D" 0 plus extra nulls if any

^(12.5)
131 "1" "2" "." "5" 0 plus extra nulls if any

^(–2.5)
126 '2' '.' '5' 254 0 plus extra nulls if any

A number written without quotes is the actual decimal value of the byte. Double quotes indicate that the value of the byte is the ASCII value of the quoted character. Single quotes indicate that the value of the byte is the two's complement of the ASCII value of the quoted character. For example, '2' = 206.

Index

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) _____

Name _____ Date _____

Organization _____ Telephone _____

Street _____

City _____ State _____ Zip Code _____
or Country

---- Do Not Tear – Fold Here and Tape ----

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1–2/L12
MARLBOROUGH, MA 01752

--- Do Not Tear – Fold Here and Tape ---