Introduction to VMS

Order Number: AA–LA04A–TE

April 1988

This book introduces the new user to VMS. It describes the DIGITAL Command Language (DCL), the Mail and Phone utilities, file manipulation, program development, and basic system concepts.

Revision/Update Information:	This document supersedes the *Introduction to VAX/VMS*, Version 4.0.
Software Version:	VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital™

ZK4510

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	CANADA	INTERNATIONAL
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital$ local subsidiary or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript[TM] printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

[TM] PostScript is a trademark of Adobe Systems, Inc.

Contents

Preface

This manual provides an overview of the VMS operating system. Topics presented include logging in and out, file manipulation, logical names, program development, command procedures, utilities, and the DIGITAL Command Language (DCL).

Intended Audience

This manual is intended for new users of the VMS operating system.

Document Structure

This manual is divided into six chapters and a glossary:

- Chapter 1 describes logging in, using DCL, recognizing system responses, the HELP command, the MAIL and PHONE utilities, and logging out.
- Chapter 2 describes how to create, delete, purge, display, list, print, rename, and protect files.
- Chapter 3 examines the parts of a full file specification and describes how to use logical names.
- Chapter 4 describes how to create, compile, link, and execute programs.
- Chapter 5 explains how to use command procedures.
- Chapter 6 details additional DCL commands.

Associated Documents

The following documents contain additional information about the topics discussed in this manual:

- *Guide to Using VMS*
- *VMS DCL Dictionary*
- *Guide to VMS Text Processing*
- *VAX EDT Reference Manual*
- *VMS Mail Utility Manual*
- *VMS Phone Utility Manual*
- *VMS DCL Concepts Manual*
- *VMS Sort/Merge Utility Manual*
- *Guide to Using VMS Command Procedures*
- *Guide to VMS Files and Devices*
- *Guide to VMS System Security*
- *VMS Access Control List Editor Manual*

- *VMS Networking Manual*
- *VMS Linker Utility Manual*
- *VMS Debugger Manual*

Conventions

Convention	Meaning
$\boxed{\text{RET}}$	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
all *commands* are	All terms shown in italics are defined in the glossary at the back of the book.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

1 Getting Started

This chapter introduces basic system concepts such as interacting with the system and using utilities. It describes logging in, logging out, the DIGITAL Command Language (DCL), system messages, the HELP facility, and the MAIL and PHONE utilities.

1.1 Interaction with the System

VMS is an *interactive operating system*. You and the system conduct a dialogue: you enter a *command*, the system responds, you respond, and so on.

A *batch* user, in contrast, communicates with the system by submitting all commands at one time in a *batch job*. After a batch job is submitted, it executes without interaction from the user. This manual emphasizes how to use VMS interactively.

You use a terminal connected to the computer to tell VMS what to do. When you enter a command on the keyboard, the system responds by executing your command or by displaying an error message at your terminal if it cannot interpret what you entered.

The keyboard you use to enter commands has the same basic configuration as a typewriter keyboard. A terminal keyboard, however, has additional keys called *function keys*. These keys send special signals to the operating system and are discussed later in this chapter.

The terminals that display the information entered at the keyboard fall into two categories: *video display terminals* and *hardcopy terminals*. Video display terminals show your input and system responses on a screen similar to that of a television. Hardcopy terminals print on continuous forms of paper. Figure 1–1 shows the VT240 video display terminal.

1.1.1 Logging In

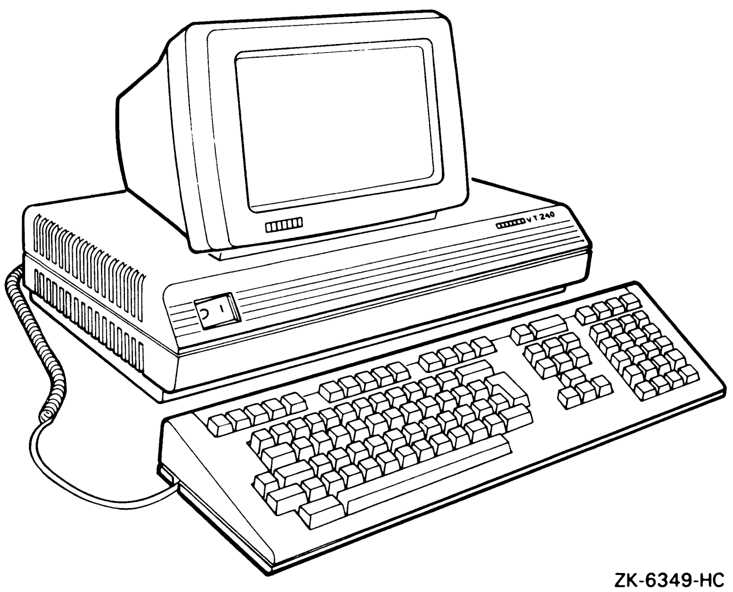
Before you can enter a command or use system resources, you must *log in*. Logging in consists of getting the system's attention and identifying yourself as an authorized user.

In order to log in, you need an account on the system. Your system manager, or whoever authorizes system use at your installation, usually sets up accounts. This person provides you with a *user name* and a *password*.

Your user name is a unique name that identifies you to the system and distinguishes you from other users. In many cases, a user name is your first or last name.

Your password is for your protection. If you maintain its secrecy, other users cannot use system resources under your user name.

Figure 1–1 The VT240 Terminal



Use the following procedure to log in to the system:

1 Make sure your terminal is plugged in and the power is turned on.

2 If your terminal has a LOCAL/REMOTE switch, set the switch to REMOTE.

3 Press RETURN to signal the system that you want to log in. (You may need to press RETURN several times.) The system responds by displaying a *prompt* for your user name.

4 Enter your user name and press RETURN. The system displays your user name on the screen as you type it. After you enter your user name and press RETURN, the system prompts you for your password.

5 Enter your password and press RETURN. The system does not display your password as you type it; this preserves the secrecy of your password.

If you made a mistake entering your user name or password, the system displays the message "User authorization failure." You are not logged in. This message means that you made a typing mistake when entering your user name or password or that your user name or password is incorrect.

A login can also fail if the terminal is not properly connected to the computer, or if the *baud rate* (the speed at which the terminal transmits or receives characters) is not correctly set. If you have any problems with the login procedure, get help from the system operator or system manager.

The following example shows a successful login:

```
RET
Username: CASEY RET
Password: RET
        Welcome to VAX/VMS version 5.0
  Last interactive login on Saturday, 31-DEC-1988 08:41
  Last non-interactive login on Saturday, 31-DEC-1988 11:05
$
```

If your login was successful, you will see a dollar sign symbol ($) in the left margin of your terminal. This prompt indicates that you are at the DIGITAL Command Language (DCL) level and can begin entering commands.

Because DCL uses the dollar sign as the default system prompt, all the examples in this manual use the dollar sign as the prompt character. However, both you and your system manager can change the system prompt to some other character or string. (See the SET PROMPT command in Chapter 6.)

1.1.2 Using the DIGITAL Command Language

Use the DIGITAL Command Language (DCL) to communicate with VMS. The DIGITAL Command Language is made up of DCL commands, which are read and translated by the *command interpreter.*

DCL commands are words, generally verbs, that describe the functions they perform. You can type them in uppercase or lowercase. The following example demonstrates the DCL command SHOW TIME:

```
$ SHOW TIME RET
```

The system responds to this command by displaying the current date and time, as follows:

```
  31-DEC-1988 11:55:40
$
```

SET PASSWORD is another DCL command you may want to use after you log in for the first time. This command allows you to change your password. Enter the command as follows:

```
$ SET PASSWORD RET
```

The system prompts you for the following information, which you must supply:

```
Old password:
New password:
Verification:
```

Enter your old password at the first prompt, and press RETURN. Enter your new password at the next prompt, and press RETURN. Finally enter your new password again, and press RETURN to confirm your choice.

1.1.2.1 Parameters and Qualifiers

DCL, like any language, has its own vocabulary and rules of grammar. The vocabulary consists of commands, *parameters,* and *qualifiers.* The grammar consists of rules for using these terms.

Command parameters define what the command will act upon, and command qualifiers further define how that action will occur. For instance, the following PRINT command requires an object (parameter) to indicate what is to be printed:

```
$ PRINT MYFILE.LIS RET
     Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

In this command, MYFILE.LIS is a parameter for the PRINT command, indicating the name of the *file* to be printed.

Note: **Always separate a command from a parameter with a space.**

You can use command qualifiers to restrict or modify the function the command is to perform. The following example shows how to use the /COPIES qualifier to specify that two copies of the file MYFILE.LIS be printed:

```
$ PRINT/COPIES=2 MYFILE.LIS RET
     Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

The *command string* includes the command and any parameters or qualifiers it may have. The *keywords* (commands, qualifiers, and parameters) that make up DCL have predefined meanings. Therefore, you must use them exactly as defined, and in some cases, supply a value to complete them. For example, the /COPIES qualifier for the PRINT command requires a value: you supply the number of copies you want printed.

The rules of grammar and *syntax* for DCL (that is, the order of the words, the spacing, and the punctuation) are also defined. The *VMS DCL Dictionary* contains a dictionary of DCL commands and discusses the rules of grammar and syntax.

1.1.2.2 Responding to Command Prompts

When you enter a command at the terminal, you do not need to enter the entire command on one line. If you enter a command without specifying required parameters, the system prompts you for the additional information it requires, as the following example shows:

```
$ PRINT RET
$_File: MYFILE.DAT RET
```

If a command requires two or more parameters, it prompts you for each parameter. In response to each prompt, you can enter just the prompted parameter or all the remaining parameters. In the following example, each file name parameter is entered separately:

```
$ COPY RET
$_From:FILE1.DAT RET
$_To:FILE2.DAT RET
```

You could, however, enter both file names after the first prompt, as in the following example:

```
$ COPY RET
$_From:FILE1.DAT FILE2.DAT RET
```

You could also enter the entire command string on one line:

```
$ COPY FILE1.DAT FILE2.DAT RET
```

In each of these examples, the data in FILE1.DAT would be copied into FILE2.DAT.

1.1.2.3 Editing Command Lines

Some keyboard keys provide editing functions you can use to correct mistakes you make while typing commands. Following are the three types of keyboard keys used to edit command lines:

- Right, left, up, and down arrow keys.
- Control keys, which are entered by pressing the CTRL key and, at the same time, another keyboard key. Control keys are referred to as CTRL/x, where x is a keyboard key.
- Function keys, which are located above the main keyboard on VT200 series terminals, and to the right of VT100 series main keyboards. Function keys are referred to as Fx, where x is the number associated with a particular function key.

Use the following keys to edit current DCL command lines:

Table 1–1 Command Line Editing Keys

VT200 Key	VT100 Key	Function
F14,CTRL/A	CTRL/A	Switches between SET TERMINAL/INSERT and SET TERMINAL/ OVERSTRIKE
CTRL/B	CTRL/B	Recalls up to 20 previously entered commands
CTRL/E	CTRL/E	Moves cursor to end of line
F12,CTRL/H	BACKSPACE,CTRL/H	Moves cursor to beginning of line
CTRL/R	CTRL/R	Repeats current command line
CTRL/U	CTRL/U	Deletes characters left of the cursor to the beginning of the line
CTRL/W	CTRL/W	Refreshes the screen
⌫	DELETE	Moves cursor back one character, deleting that character
F13	LINEFEED	Deletes the word left of the cursor
DOWN ARROW	DOWN ARROW	Recalls the command entered after the current command

Table 1–1 (Cont.) Command Line Editing Keys

VT200 Key	VT100 Key	Function
RIGHT ARROW,CTRL/F	RIGHT ARROW,CTRL/F	Moves cursor one character right
LEFT ARROW,CTRL/D	LEFT ARROW,CTRL/D	Moves cursor one character left
UP ARROW, CTRL/B	UP ARROW, CTRL/B	Recalls the command entered before the current command

If you are entering a command and want to cancel it, press CTRL/Y or CTRL/C. Using CTRL/Y or CTRL/C, you can also interrupt the system while it executes a command. After you interrupt the system using CTRL/Y, the DCL prompt ($) returns.

By pressing CTRL/C, you can interrupt the system and still remain in a *utility* (a general-purpose program that does certain common functions). For example, if you begin typing a message in MAIL and change your mind, press CTRL/C. The message is not sent, and you remain in the MAIL utility.

1.1.3 Recognizing System Responses

The system can respond to your command in several ways. It can execute the command. Generally, you know your command has executed successfully when the system prompt returns (by default, the dollar sign). It can execute the command and inform you in a message what it has done. It can, if execution is not successful, inform you of errors. It can even act for you, supplying values (defaults) you have not supplied yourself.

1.1.3.1 What Are Defaults?

A default is the value supplied by the operating system when you do not specify one yourself. For instance, if you do not specify the number of copies as a qualifier for the PRINT command, the system uses the default value of 1. By not explicitly stating a value, the system assumes that you have chosen the default. VMS supplies default values in several areas, including command qualifiers and parameters. The defaults used with individual commands are specified with each command's description in the *VMS DCL Dictionary*.

1.1.3.2 Looking at Informational Messages

The system responds to some commands by giving you information about what it has done. For example, when you use the PRINT command, the system displays the job identification number it assigned to the print job and shows the name of the print queue the job has entered.

```
$ PRINT MYFILE.LIS RET
    Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

Not all commands display informational messages. Successful completion of a command is usually indicated when the dollar sign prompt returns. Unsuccessful completion is always indicated by one or more error messages.

1.1.3.3 Looking at Error Messages

If you enter a command incorrectly, the system displays an error message and prompts you for the correct command string, as the following example shows:

```
$ CAPY RET
%DCL-W-IVVERB, unrecognized command verb - check validity and spelling
 \CAPY\
$
```

The three-part code preceding the text of the message indicates the following:

- The message is from DCL, the default command interpreter
- The message is a warning (W) message
- The message can be identified by the mnemonic IVVERB in the *VMS System Messages and Recovery Procedures Reference Manual*

You can also receive error messages during command execution if the system cannot perform the function you have requested. For example, if you type a PRINT command correctly, but the file you specify does not exist, the PRINT command informs you of the error with a message like the following:

```
$ PRINT NOFILE.DAT RET
%PRINT-E-OPENIN, error opening CLASS1:[MAYMON]NOFILE.DAT; as input
-RMS-E-FNF, file not found
$
```

The first message is from the PRINT command. It tells you it cannot open the specified file. The second message indicates the reason for the first, that is, the file cannot be found. *RMS* refers to the VMS file handling facility, Record Management Services; error messages related to file handling are generally VMS RMS messages.

1.1.3.4 Is the System Still Functioning?

If you suspect that your system is not doing what you think it should be doing, press CTRL/T. CTRL/T displays a single line of statistical information about the current process. When you press CTRL/T during an interactive terminal session, it momentarily interrupts the current command, command procedure, or image in order to display statistics.

Although CTRL/T disrupts the characters on the screen, it does not impact any procedure or editing session. To refresh the screen, press CTR/W. The statistical information includes node and user name, current time, current process, *CPU* usage, number of page faults, level of I/O activity, and memory usage. The following example shows a user named BEAN on node GREEN using the EDT editor:

```
GREEN::BEAN  13:45:02 EDT        CPU=00:00:03.33 PF=778 IO=295 MEM=315
```

If you know that your system is running, and CTRL/T does not display statistical information, enter the SET CONTROL=T at the dollar sign ($) prompt, then press CTRL/T again.

1.1.4 Using the DCL Command HELP

When you use the VMS operating system, you may not always have a reference manual available at your terminal, and you may want to see the format of a command before you enter it. The HELP facility is designed to provide you with this information.

For example, to display a list of commands for which HELP is available, type:

```
$ HELP RET
```

The system responds by displaying the list of HELP commands and prompting you for a choice of topic.

If you want information about a particular command, type that command after the topic prompt. For instance, if you want information about the PRINT command, type the following:

```
Topic? PRINT RET
```

The information displayed includes a synopsis of what the PRINT command does, the parameters it requires, and the qualifiers it can take.

If you want to know more about one of the PRINT command qualifiers, respond to the prompt "PRINT subtopic?" with that qualifier. (Remember to type the slash character (/) before the qualifier.) For example, to display information about the /COPIES qualifier of the PRINT command, type:

```
PRINT subtopic?/COPIES RET
```

If you know the subtopic on which you need help, type the following:

```
$ HELP PRINT/COPIES RET
```

When you have finished using HELP, press CTRL/Z. The dollar sign prompt is displayed in the left margin, indicating that VMS is ready to receive a command.

1.1.4.1 Exploring Several HELP Topics

The topics and commands you see when you enter the HELP command cover a wide range of subjects:

- To manipulate files, you can get information about the following DCL commands: COPY, DELETE, PURGE, RENAME, and TYPE. (Files are discussed in detail in Chapter 2.)
- For programming needs, you can get information about the LINK and RUN commands.
- For information about the available language *compilers*, enter the HELP command followed by the name of the compiler (for example, HELP BASIC, or HELP FORTRAN).
- For text processing needs, you can get information about available editors (HELP EDIT) and a text formatter (HELP RUNOFF).
- For operator needs, you can get information about the ALLOCATE, DEALLOCATE, MOUNT, and DISMOUNT commands.
- For information by category, enter the HELP HINTS command.

1.1.5 Logging Out

When you have finished your session at the terminal, log out as follows:

```
$ LOGOUT RET
```

The system responds with the following:

```
MAYMON  logged out at 31-DEC-1988 12:43:10.38
```

To make sure that you have logged out, always use the LOGOUT command to end a terminal session.

Note: **Shutting off your terminal or setting the REMOTE/LOCAL switch to LOCAL does not automatically cause you to log out. If you shut a terminal off without logging out properly, another user may be able to turn the terminal on and use your account.**

1.2 Using Utilities

VMS supports many different utilities. Each utility performs a task or set of tasks. A utility is an environment in which you can use a specific set of commands and qualifiers to perform a desired task.

To invoke a utility, enter the corresponding DCL command. For example, to invoke the VMS Phone Utility (PHONE) you enter the DCL command PHONE. After you enter the DCL command and press RETURN, you see the utility prompt. Some utilities have a prompt that matches their name. For example, the VMS Mail Utility (MAIL) displays the following prompt:

```
MAIL>
```

Other utilities prompt you for a file name. For example, when you enter the DCL command SORT to invoke the VMS Sort Utility, you see the following prompt:

```
_Input:
```

When you are using the type of utility that prompts you for a file name (for example, BACKUP, MESSAGE, PATCH, AND SORT), you can add qualifiers to the DCL command string in order to tailor the utility to your own needs. The following example shows how to tailor the VMS Backup Utility (BACKUP) by specifying qualifiers on the command string when invoking the utility:

```
$ BACKUP/RECORD/IMAGE/LOG  RET
_From:
```

The following table lists four utilities that prompt you for a file name, the DCL command you enter to invoke each utility, and the first DCL prompt displayed.

Utility	DCL Command	DCL Prompt
BACKUP	BACKUP	_From:
MESSAGE	MESSAGE	_File:
PATCH	PATCH	_File:
SORT	SORT	_Input:

When you are using the type of utility that displays a utility prompt (for example, MAIL>), you can enter utility commands at the utility prompt. Each utility responds to a different set of commands. For example, the VMS Phone Utility recognizes the following commands because they are PHONE commands:

- DIAL
- HOLD
- REJECT

If you invoke the VMS Mail Utility (MAIL) and enter these commands, you see error messages displayed because PHONE commands are not recognized by MAIL.

The following table lists three utilities that display a utility prompt, the DCL command you enter to invoke each utility, and the utility prompt:

Utility	DCL Command	Utility Prompt
MAIL	MAIL	MAIL>
PHONE	PHONE	%
MONITOR	MONITOR	MONITOR>

The following sections discuss the MAIL and PHONE utilities in more detail. For more information about these utilities, see the *VMS Mail Utility Manual* and the *VMS Phone Utility Manual.*

1.2.1 What Is MAIL?

The VMS Mail Utility (MAIL) allows you to send messages to other users on your system or to users on another VAX computer connected to your system by means of DECnet-VAX. (For information about DECnet-VAX, see the *VMS Networking Manual.*) To invoke the MAIL utility, enter the MAIL command at the DCL prompt. The twelve MAIL commands introduced in this section enable you to perform various functions within MAIL.

This section discusses the following twelve MAIL commands:

SEND	DIRECTORY	EXTRACT
READ[/NEW]	DELETE	PRINT
FORWARD	MOVE	HELP
REPLY	SELECT	EXIT

1.2.1.1 Sending Mail

After you invoke MAIL, try the SEND command. You can send a message to anyone on the system by entering their *node specification* and user name at the "To:" prompt. A node is one of several computer systems that are connected together to form a *network*. If your computer system is not part of a network, you do not need to use a node specification.

If your computer system is part of a network, and you are sending mail to someone who is not on your node, include a node specification in the following format:

nodename::username

If the person you are sending mail to is on your node, only specify a user name at the "To:" prompt. Refer to Chapter 3 for additional information about nodes. The following example shows how to send a message to a user named DAILEY on node BRUSH:

```
MAIL> SEND RET

To: BRUSH::DAILEY RET
```

Try sending a message to yourself. Enter the SEND command at the mail prompt (MAIL>) and press RETURN. Enter your own user name at the prompt "To:", and press RETURN. Enter a subject when prompted, and press RETURN again. The following example shows how to use the SEND command:

```
MAIL> SEND RET

To:     MITCHELL RET

Subj:   Testing...  RET
Enter your message below.  Press CTRL/Z when complete, or CTRL/C to quit:
```

When you finish entering the text of your message, press CTRL/Z. Because you are sending the message to yourself, MAIL will display a message on your screen announcing that you have new mail, like the following:

```
New mail on node WHIP from MITCHELL

MAIL>
```

1.2.1.2 Reading Mail

Next, use the READ command. To read the message you just sent to yourself, enter the READ command with the /NEW qualifier and press RETURN, as follows:

```
MAIL> READ/NEW
```

The only time you must specify the /NEW qualifier with the READ command is when you want to read new mail that arrives while you are in MAIL. If you are not in MAIL and you receive new mail, invoke MAIL, and enter the READ command (without the /NEW qualifier) to read the new message. Also, when you want to reread old messages (messages that you have already read), enter the READ command. If you have just read a new message, and you want to reread an old message, enter the following:

```
MAIL> SELECT MAIL
```

(The MAIL command SELECT is discussed later in this section.) Now you can use the READ command to reread the old message.

1.2.1.3 Forwarding Mail

You can forward a copy of a mail message to another user by entering the FORWARD command. MAIL prompts you for the name of the user to receive the message. If your system is part of a network and you are trying to forward mail to a user on another node, you must include a node specification on this line. The user name and node specification are separated by two colons (::). Refer to Chapter 3 for additional information about nodes.

Try forwarding a copy of the message you just received back to yourself. Enter your user name, and press RETURN. Supply a subject when prompted, and press RETURN. MAIL signals that you have just received a new message. Enter the READ/NEW command to read the forwarded message.

1.2.1.4 Replying to Mail

When you receive a message and want to respond to it, enter the REPLY command, and press RETURN. MAIL displays the header information as follows:

```
MAIL> REPLY
To:         SHRED::GEEZER
Subject:    Re:Official reprimand
Enter your message below.  Press CTRL/Z when complete, CTRL/C to quit:
```

When you finish typing your response, press CTRL/Z. Again, MAIL signals that you have just received a new message. To read the message, enter the READ/NEW command.

1.2.1.5 Listing Mail Messages

When you want to see a list of all new mail messages you have collected, enter the DIRECTORY command, and press RETURN. MAIL displays a list or directory like the following:

```
# From              Date                  Subject

1 MURPHY            31-DEC-1988            How to Write a Memo
2 PIT::HORACE       31-DEC-1988            Using the Printer
3 NOREEN::CASEY     31-DEC-1988            Party
```

To see a list of all the old messages, enter the SELECT MAIL command string followed by the MAIL command DIRECTORY.

1.2.1.6 Organizing Mail into Folders

MAIL allows you to organize your messages by moving them into *folders*. By default, MAIL provides the following folders:

- MAIL–Contains messages that have been read but not deleted. This folder cannot be deleted.
- NEWMAIL–Contains all messages that have not been read.
- WASTEBASKET–Contains messages that have been deleted. This folder and its contents are deleted when you exit MAIL.

MAIL also allows you to create your own folders. You can create as many folders as you want.

To move a message to a folder, enter the MOVE command (while you are reading the message), and press RETURN. MAIL prompts you for a folder name. Type any name, such as REVIEWS, JOKES, or STATUS_REPORTS. When you enter the name of a new folder (one you have not created before), MAIL asks whether or not you want to create it. Answer "Y". MAIL also prompts you for a file name. You can specify the default mail file by pressing RETURN. A sample session demonstrating the MOVE command follows. (The name of the new folder is WINNERS and the default mail file is specified.)

```
MAIL> 2
MAIL> MOVE
_Folder: WINNERS
_FILE: RET

Folder WINNERS does not exist.
Do you want to create it (Y/N, default is N)?y
%MAIL-I-NEWFOLDER, folder WINNERS created
```

To move from one folder to another, use the SELECT command. If you want to move to the WINNERS folder, enter the following command string. (MAIL displays a message indicating the number of messages in the folder.)

```
MAIL> SELECT WINNERS

%MAIL-I-SELECTED, 1 message selected
```

To move to a folder named JOKES, enter the following command string:

```
MAIL> SELECT JOKES

%MAIL-I-SELECTED, 3 messages selected
```

To move to your default mail folder (MAIL), enter the command string SELECT MAIL.

You can enter the DIRECTORY command to see a list of the messages in the folder you just selected. Enter the DIRECTORY/FOLDERS command to see a listing of your folders.

1.2.1.7 Deleting Mail

When you want to remove a message, use the DELETE command. You can either enter the DELETE command while you are reading the message, or you can enter the DELETE command followed by the number (or range of numbers) of the message you want to remove. To remove messages 2, 4, 5, 6, 8, 9, and 10 in the list, enter the following command string:

```
MAIL> DELETE 2,4-6,8:10
```

Note that you can use either a colon (:) or a hyphen (-) to define the range of messages to be deleted.

If you enter the DIRECTORY command after you have deleted messages, you will see the messages marked for deletion, as in the following example:

```
# From             Date               Subject

1 MURPHY           31-DEC-1988         How to Write a Memo
2 (Deleted)
3 RHUMBA::CASEY    31-DEC-1988         Party
4 (Deleted)
```

When you exit from MAIL, the messages marked for deletion are removed.

1.2.1.8 Extracting Mail

When you want to move a mail message from your mail file to a file that you can access from the DCL command level, use the EXTRACT command. Enter the EXTRACT command (while you are reading the message), and press RETURN. MAIL prompts you for the name of a file. Then, when you exit from MAIL, the file is listed in your main directory. The following example shows how to use the EXTRACT command to move a mail message to a file named GAMES.DAT:

```
MAIL> EXTRACT

_File: GAMES.DAT

%MAIL-I-CREATED, DISK:[BERGERON]GAMES.DAT;1 created

MAIL>
```

1.2.1.9 Printing Mail

To make a hard copy of a mail message, enter the PRINT command while you are reading the message, and press RETURN. (When you exit from MAIL, the system responds by telling you that the message has entered the print queue.) The following example shows how to make a hard copy of message number 4 using the PRINT command:

```
MAIL> 4

    #4          31-DEC-1988 09:39:20          MAIL
From:  WHIP::MITCHELL
To:    GEEZER
Subj:  lunch
Hey Geezer, WAKE UP!!!! Is our lunch date still on?...

MAIL> PRINT
MAIL> EXIT
     Job MYFILE (queue SCALE_PRINT, entry 210) started on SYS$PRINT
```

1.2.1.10 Getting Help in Mail

To see detailed information about MAIL, enter the HELP command at the mail prompt (MAIL>). There are several topics to choose from. For example, the HELP topic "Folders" describes the organization of the Mail Utility in detail. For more information about each MAIL command, use the HELP facility provided in MAIL, or refer to the *VMS Mail Utility Manual.*

1.2.1.11 Exiting from MAIL

When you are ready to leave MAIL, enter the EXIT command, and press RETURN. Any messages marked for deletion are removed. Any messages marked for printing enter the print queue, and a message similar to the following is displayed:

```
MAIL> EXIT
  Job 790 entered on queue ATLAS_PRINT
```

1.2.2 What Is PHONE?

The VMS Phone Utility (PHONE) allows you to "talk" with other users on your system or on any other VAX computer connected to your system by means of DECnet-VAX. It is designed to closely simulate the features of a real telephone.

Note: **PHONE can be used only on video terminals with direct cursor positioning, such as the DIGITAL VT52, VT100, and VT200 Series terminals.**

To invoke the PHONE utility, enter the PHONE command at the system prompt. The five commands discussed in this section introduce you to PHONE. For more information about PHONE, see the *VMS Phone Utility Manual.*

This section discusses the following PHONE commands:

ANSWER	DIRECTORY
REJECT	HELP
EXIT	

1.2.2.1 How to Phone Another User

After you invoke PHONE, you can place a call to anyone who is currently logged in. Enter the person's node specification and user name at the *switch hook character,* which by default is the percent sign (%). For example, to phone user JACKSON on node FENDER enter the following:

```
% FENDER::JACKSON
```

If the person you are calling is on your node, or if your computer system is not part of a network, type only a user name.

Try phoning a friend on the system, or try phoning yourself. Enter your user name at the switch hook character (%), and press RETURN. Because you are calling yourself, do not include a node name. Your call is automatically answered. PHONE displays the following message:

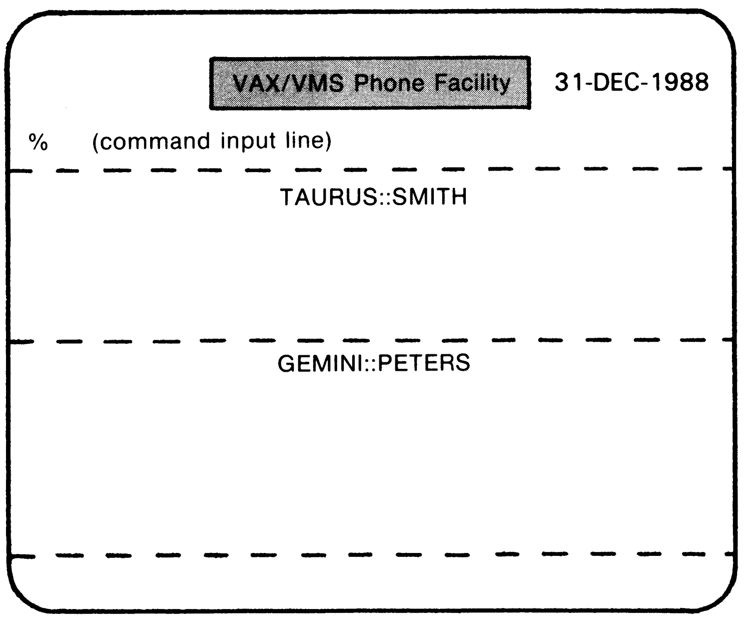
```
That person has answered your call.
```

Your screen display divides, showing the name of the person placing the call in the top section of the screen and the name of the person receiving the call in the bottom section of the screen. Figure 1–2 shows a PHONE screen display.

1.2.2.2 Answering a Phone Call

To answer a call from another user, invoke PHONE, and enter the ANSWER command. Your screen display divides, and you can begin typing your conversation. Every character you type is displayed as part of the conversation (except the percent sign (%), which is the default switch hook character).

Figure 1–2 Looking at a PHONE Screen Display



Note: The switch hook character is always displayed in column 1 of the command input line.

ZK-898-82

While you are typing text, you can use the following key combinations to clear parts of the screen:

Key Combination	Function
CTRL/L	Clears all the text in your part of the screen
CTRL/U	Clears the current line
CTRL/W	Restores the entire screen

To signal PHONE that you want to enter a PHONE command during a conversation, type the switch hook character (%) followed by the command. After PHONE executes the command, the cursor returns to its position before you typed the switch hook character (%).

If you type the switch hook character, then decide not to enter a command, press RETURN to continue your conversation.

1.2.2.3 Rejecting a Phone Call

If you decide not to answer a call, PHONE keeps broadcasting the message on your screen until either the person calling cancels the call, or you enter the REJECT command. When you enter the REJECT command, the user placing the call receives a terminal message that the call has not been accepted.

1.2.2.4 Displaying a List of Users You Can Call

To see a list of users you can call, enter the DIRECTORY command. If you enter this command with a node name (as the following example shows), it lists the users on that system:

```
%DIRECTORY ATLAS

Press any key to cancel the directory listing and continue.
------------------------------------------------------------
Process Name     User Name        Terminal        Phone Status

Mike              MROSOSKY         TTE1:           available
Biff Riff         RIFF             VTA3:           available


2 persons listed.
```

1.2.2.5 Getting Help in PHONE

When you want general information about PHONE, including descriptions for all the available PHONE commands, enter the HELP command, and press RETURN.

1.2.2.6 Exiting from PHONE

When you want to leave PHONE, enter the EXIT command at the switch hook character, and press RETURN. Your entire screen clears, and the DCL command prompt appears.

2 Working with Files

This chapter explains how to use DCL commands to manipulate files. It describes how to identify, create, delete, and purge files; how to create and list directories; and how to print, copy, protect, and rename files.

2.1 What Is a File?

A file contains information. One type of information a file can contain is text. For example, to organize a book, you can create a file for each chapter. To edit each chapter, you enter and manipulate text within each file. You might also create a file containing a memo. You can use MAIL to send the file containing the memo to other users, or edit the file to change the memo.

You can create a file and fill it with instructions using a language like FORTRAN or Pascal. Then you use a language compiler to generate a machine language the computer understands. This list of instructions is called a program. By entering the appropriate DCL commands and specifying the name of the file containing the program, you can instruct the computer to perform the task.

If you often use the same series of DCL commands, you can create a file containing this series. A series of DCL commands is called a command procedure. When you want to use this series of commands, you enter the appropriate DCL command and the name of the file containing the series. For information about command procedures, see Chapter 5.

2.1.1 Looking at File Names, Types, and Versions

You can identify a file by specifying its file name and *file type* in the following format:

```
filename.type
```

The file name can be from 0 through 39 characters chosen from the letters A through Z, the numbers 0 through 9, an underscore (_), a hyphen (-), or a dollar sign ($). Do not use a hyphen (-) as the first character in the file name.

When you create files, you can give them any names that are meaningful to you. A list of legitimate file names follows:

2409CHAP2
2409CHAP13
APPRAISE
BASIC-EXAMPLES
FORTRAN_EXAMPLES
GETTING-STARTED
MAIL
ORCHESTRA_MEMBERS
THINGS_TO_DO

The file type can be from 0 through 39 characters and must be preceded by a period. Again, you can choose any of the letters A through Z, the numbers 0 through 9, an underscore, a hyphen or a dollar sign for the file type. Do not use the hyphen as the last character in the file type.

The file type usually describes the kind of data in the file. The system recognizes many default file types used for special purposes. Some of the more common default file types follow:

File Type	Use
COM	Command procedure
DAT	Data file
DIS	Distribution list for MAIL
DIR	Directory file
EDT	Start-up command file for EDT editor
EXE	Executable program image file
JOU	Journal file used by the EDT editor
LIS	Output listing file
LN01	Output file for LN01 laser printer
LN03	Output file for LN03 laser printer
LOG	Batch job output file
MAI	MAIL message file
MEM	Output file for DIGITAL Standard Runoff (DSR)
OBJ	Object module file output from a compiler or assembler
OLB	Object library file output from the Librarian Utility
RNO	Input file for DIGITAL Standard Runoff (DSR)
TMP	Temporary file
TXT	Input file for text libraries or MAIL output command

Each high-level language has a default file type for source programs. Some of these file types are listed in the following table:

File Type	Contents
BAS	Input source file for the VAX BASIC compiler
B32	Input source file for the VAX BLISS-32 compiler
C	Input source file for the VAX C compiler
COB	Input source file for the VAX COBOL compiler
COR	Input source file for the VAX CORAL-66 compiler
FOR	Input source file for the VAX FORTRAN compiler
MAR	Input source file for the VAX MACRO assembler
PAS	Input source file for the VAX Pascal compiler
PLI	Input source file for the VAX PL/I compiler

The following list combines several previously mentioned file names with file types:

2409CHAP2.RNO
BASIC-EXAMPLES.BAS
MAIL.MAI
ORCHESTRA_MEMBERS.DIS
APPRAISE.REAL-ESTATE

In addition to a file name and type, every file has a version number the system assigns to a file when the file is created or revised. When you initially create a file, the system assigns it a version number of 1. Subsequently, when you edit a file or create additional versions of it, the version number is automatically increased by one.

Version 12 of a file named APPRAISE.MEM follows:

```
APPRAISE.MEM;12
```

You rarely need to specify the version number with a file specification. The system assumes default values for version numbers. Version number defaults are determined as follows:

- For an input file, the system uses the highest existing version number of the file. (Many system utilities take existing files, alter them, and produce new files. The existing file is called an input file and the newly produced file is called an output file.)
- For an output file, the system adds 1 to the highest existing version number.

When you specify a version number in a file specification, precede the version number with a semicolon (;).

2.1.2 Using Wildcard Characters

A *wildcard character* can be used with many DCL commands to apply the command to several files at once, rather than specifying each file individually. Two wildcard characters, the asterisk (*) and the percent sign (%), can be used when you specify a file name and a file type. The asterisk can also be used to specify version numbers.

For example, you can specify all versions of a file by using an asterisk in place of the version number in the file specification. If you want to print all versions of the file TESTFILE.DAT without specifying each version number separately, enter the following command string:

```
$ PRINT TESTFILE.DAT;*
```

If there were no wildcard character in the preceding example, the PRINT command by default would apply only to the most recent version of the file TESTFILE.DAT.

The following command string displays a listing of all versions of all files with the file type of DAT:

```
$ DIRECTORY *.DAT;*
```

To print all versions of all files named TEST, enter the following command string:

```
$ PRINT TEST.*;*
```

The percent sign character can be substituted for any single character in the file specification (except the version number). For example, to print the latest version of several files with a file type of TXT and a file name that begins with CHAP but ends in a series of different numbers, as in CHAP1.TXT, CHAP2.TXT, and CHAP3.TXT, enter the following command string:

```
$ PRINT CHAP%.TXT
```

Note that in this example the percent sign specifies only one character. Therefore, this PRINT command would not affect a file named CHAP.TXT or CHAPIX.TXT.

The following example shows how to display a listing of the files beginning with the letters CHAP and having a file type of TXT:

```
$ DIRECTORY CHAP*.TXT
```

2.2 Creating Files

To create a file, enter one of the following commands:

- EDIT
- CREATE

The EDIT command invokes an editor to create a file. The default VMS editor is EDT. To learn how to use the EDT editor, see the *Guide to VMS Text Processing.*

You can also use the CREATE command to make a new file, and specify the file name as a parameter. You can insert text immediately, and terminate the insertion by pressing CTRL/Z.

```
$ CREATE MYFILE.DAT
This is the only line.
CTRL/Z
```

Unlike the EDIT command, the CREATE command does not modify an existing file.

2.3 Deleting Files

Because files take up disk space, periodically delete files you no longer need.

The DELETE command deletes specific files. When you use the DELETE command, you must specify a file name, file type, and version number. Having to specify a version number provides some protection against accidental deletion. However, any of these file components can be specified as a wildcard character. You can also enter more than one file specification in a command string by separating the file specifications with commas. The following table shows some examples of the DELETE command.

Command	Result
$ DELETE AVERAGE.OBJ;1	Deletes the file named AVERAGE.OBJ;1
$ DELETE *.LIS;*	Deletes all files with file types of LIS (This command deletes all versions of all program listings)
$ DELETE A.DAT;1,A.DAT;2	Deletes the first two versions of the same data file

2.4 Purging Files

When you want to delete many versions of many files, you can use the PURGE command.

The PURGE command deletes all but the most recent version of a file. Therefore, you cannot enter a version number with the PURGE command. The following command string deletes all files named AVERAGE.FOR except the file with the highest version number:

```
$ PURGE AVERAGE.FOR RET
```

Use the /KEEP qualifier with the PURGE command to specify that you want to keep more than one version of a file. The following command string deletes all but the two most recent versions of the file TEST.DAT:

```
$ PURGE/KEEP=2 TEST.DAT RET
```

To clean up your entire directory, enter the PURGE command without any parameters or qualifiers.

2.5 Displaying Files at Your Terminal

The TYPE command displays a file at your terminal. To display a file named TEST.DAT, enter the following command string:

```
$ TYPE TEST.DAT RET
This is the first line of a file created with the EDT editor.
```

While a file is being displayed, you can suspend and resume the upward movement, or *scrolling*, of the terminal display by using CTRL/S and CTRL/Q. To temporarily stop the display from scrolling, press CTRL/S; to continue the scrolling, press CTRL/Q.

The NO SCROLL key on VT100 terminals and the F1 key (Function Key 1) on VT200 series terminals perform the same functions. Pressing NO SCROLL once suspends scrolling; pressing it again resumes scrolling.

2.6 Listing Files in a Directory

A directory is a file that contains a list of other files. The directory list includes the name, type, and version number of each file in that directory. (See Chapter 3 for more information about directories.)

To list the names of files in a particular directory, use the DIRECTORY command. When you enter the DIRECTORY command with no parameters or qualifiers, files listed in your default directory are displayed on the terminal, as in the following example:

```
$ DIRECTORY RET

Directory BOOK2:[MALCOLM] ❶

AVERAGE.EXE;2      AVERAGE.EXE;1      AVERAGE.FOR;2
AVERAGE.FOR;1      AVERAGE.OBJ;2      AVERAGE.OBJ;1 ❷

Total of  6 files. ❸
```

❶ The disk and directory name (see Section 3.1.2 for information about disks)

❷ The file names, file types, and version numbers of each file in the directory

❸ The total number of files in the directory

When you enter the DIRECTORY command, you can provide one or more file specifications to obtain a listing of particular files. For example, to find out how many versions of the file AVERAGE.FOR currently exist, enter the DIRECTORY command as follows:

```
$ DIRECTORY AVERAGE.FOR RET

Directory BOOK2:[MALCOLM]

AVERAGE.FOR;2   AVERAGE.FOR;1

Total of 2 files.
```

You can use the wildcard character (*) to display selected groups of files. If you want to see a list of all the files in your directory beginning with the letters "TR", enter the following command string:

```
$ DIRECTORY TR*  RET

Directory CIRCUS:[PERT]

TRAINS.DAT;16  TRAPEZE.BAR;2  TRAVEL.FUN;5

Total of 3 files.
```

2.7 Printing Files

To make a hard copy of a file, use the PRINT command. The system cannot always print a requested file immediately since there may be only one or two printers for all users to share. The system enters the name of the file you want to print in a queue, and prints the file at the first opportunity.

A printed file is preceded by a *header page* describing the file so you can identify your own listing. For example, if you enter the following command string, the header page shows your user name and the file's name, type, and version number:

```
$ PRINT WRITERS:[JONES]AVERAGE.LIS RET
  Job AVERAGE (queue GROUP_PRINT, entry 1995) started on BIG$LPA0
```

When you use the PRINT command, the system responds with a message indicating the job number it assigned to the print job.

The PRINT command also has qualifiers that allow you to control the number of copies of the file to print, the type of forms to print the file on, and other similar functions. See the *VMS DCL Dictionary* for detailed information about these qualifiers.

2.8 Renaming Files

To change the identification of one or more files, use the RENAME command. For example, the following command string changes the file name and type of the most recent version of the file PAYROLL.DAT to TEST.OLD:

```
$ RENAME PAYROLL.DAT TEST.OLD RET
```

After you enter this command string, the file name PAYROLL.DAT no longer exists. Its contents now reside in the file named TEST.OLD.

You can use the RENAME command to move a file from one directory to another. For example, the following command moves TEST.OLD from the directory [MALCOLM] to the subdirectory [MALCOLM.TESTFILES]:

```
$ RENAME [MALCOLM]TEST.OLD [MALCOLM.TESTFILES]
```

You can use wildcard characters if you want to change a number of files that have either a common file name or file type. The following command string changes the directory name for all versions of all files with file names of PAYROLL:

```
$ RENAME PAYROLL.*;* [MALCOLM.TESTFILES]*.*;* RET
```

The files are now cataloged in the subdirectory [MALCOLM.TESTFILES].

2.9 Protecting Files

To prevent others from gaining unauthorized or undesired access to a file, enter the SET FILE/PROTECTION command. The command allows you to define the type of file access you want to allow other users. The following tables list the four user categories and the four access types:

User Category	Type of User
OWNER	The user who created the file
GROUP	All users, including the owner, who have the same group number (or group identifier) in their user identification codes (UIC) as the owner of the file
WORLD	All users
SYSTEM	All users who have system privilege (SYSPRV) or low group numbers (from 0 to the value of the system parameter MAXSYSGROUP)

Access Type	Type of Access
READ	The right to examine, print, or copy a file
WRITE	The right to modify or write a file
EXECUTE	The right to execute a file that contains executable program images
DELETE	The right to delete a file

A user identification code (UIC) is a specification the system uses to determine if a user can access a file. For example, a UIC can resrict access to a file to a certain group. Only users with the proper UIC can access that file.

A UIC can be either a pair of numbers or a name (or optionally, a pair of names). When a DCL command requires a UIC specification, you can use either format. The system translates all UICs to numbers when it determines a user's access.

A numeric UIC consists of a group number (g) and a member number (m) in the following format:

```
[g,m]
```

An example of two numeric UICs follows (notice that they both have the same group number):

```
[360,055]
[360,223]
```

A UIC can also be either one or two names. These names are called a member identifier and a group identifier), as follows:

```
[member-identifier]
        or
[group-identifier,member-identifier]
```

An example of three UICs consisting of names follows. Notice that the first two UICs have the same group identifier:

```
[PATS,BELLINI]
[PATS,FRANKLIN]
[RAMS,CHOO]
```

For detailed information about UICs, group numbers, member numbers, group identifiers, and member identifiers, see the *VMS DCL Dictionary* and the *Guide to Using VMS Command Procedures*.

You can abbreviate user categories and access types to one letter, as the following table shows:

User Category	Access Type
O - OWNER	R - READ
G - GROUP	W - WRITE
W - WORLD	E - EXECUTE
S - SYSTEM	D - DELETE

When you use the SET FILE/PROTECTION command, separate the category from the type of access by a colon (:). When you specify more than one category, use a comma (,) to separate one category from another, and enclose the list of categories in parentheses ( ). For example, the following command string contains three categories separated from each other by commas and enclosed in parentheses:

```
$ SET FILE/PROTECTION RESUME.TXT/PROTECTION=(O:RWE,G:RWE,W:R)
```

You can use the SET FILE/PROTECTION command, specifying a different type of access for each category of user. For example, if you want all users (WORLD category) to be allowed read access only on a file named FRIENDS.DIS, enter the following command string:

```
$ SET FILE/PROTECTION FRIENDS.DIS/PROTECTION=W:R
```

All other access is denied. Therefore, users are not allowed to WRITE, EXECUTE, or DELETE the file named FRIENDS.DIS.

If you want users in the GROUP category to be able to READ but not DELETE a file named JOKES.COM, enter the following command string:

```
$ SET FILE/PROTECTION JOKES.COM/PROTECTION=G:R
```

If you want all users (WORLD category) to have total access to a file named GAMES.COM, enter the following command string:

```
$ SET FILE/PROTECTION GAMES.COM/PROTECTION=W:RWED
```

By default, when you create a file, it is protected in the following way:

- Both you (OWNER) and system users (SYSTEM) have total access, or the ability to READ, WRITE, EXECUTE, and DELETE the file. This protection information is coded in the following way:

  ```
  (SYSTEM:RWED,OWNER:RWED)
  ```

- Users in the GROUP category have READ and EXECUTE access, coded as follows:

  ```
  (GROUP:RE)
  ```

- Other users who are not in your group are considered part of the WORLD category and have no access, coded as follows:

  ```
  (WORLD: )
  ```

3 Understanding Directory Structure

This chapter describes the components of a complete file specification.

3.1 Dissecting a Complete File Specification

A complete file specification contains all information the system needs to locate and identify a file. A complete file specification has the following format:

```
node::device:[directory]filename.type;version
```

You must enter the punctuation marks exactly as shown (colons, brackets, period, semicolon) to separate the components of the file specification.

Figure 3–1 shows the relationship between the parts of a full file specification. Notice how the file is listed in a directory, that the directory resides on a device, and that the device is located on a node.

Figure 3–2, a full file specification, contains all necessary information to enable the system to locate and identify the file STORIES.TXT in the [MCNALLY] directory, on device VAMP, located on node DRACUL.

Figure 3–3 shows the relationship between the parts of the previous file specification.

3.1.1 Looking at Nodes

When computer systems are linked together, they form a network. Each system in the network is called a node and is identified within the network by a unique node name. Your system may or may not be part of a larger network. For detailed information about networks and nodes, see the *VMS Networking Manual.*

If your system is a network node, you may be able to gain access to a file located at another node on the network by adding a node specification to the first part of the file specification. (This specification allows you access to the file only if the user owning the file has permitted other users access to it.)

For example, to access a file named ZAP.LIS, which is listed in the [LAWRENCE] directory, stored on device DEVO on node LOTUS, enter the following command string:

```
$ TYPE LOTUS::DEVO:[LAWRENCE]ZAP.LIS
```

Figure 3–1 Relationship Between Parts of Full File Specification

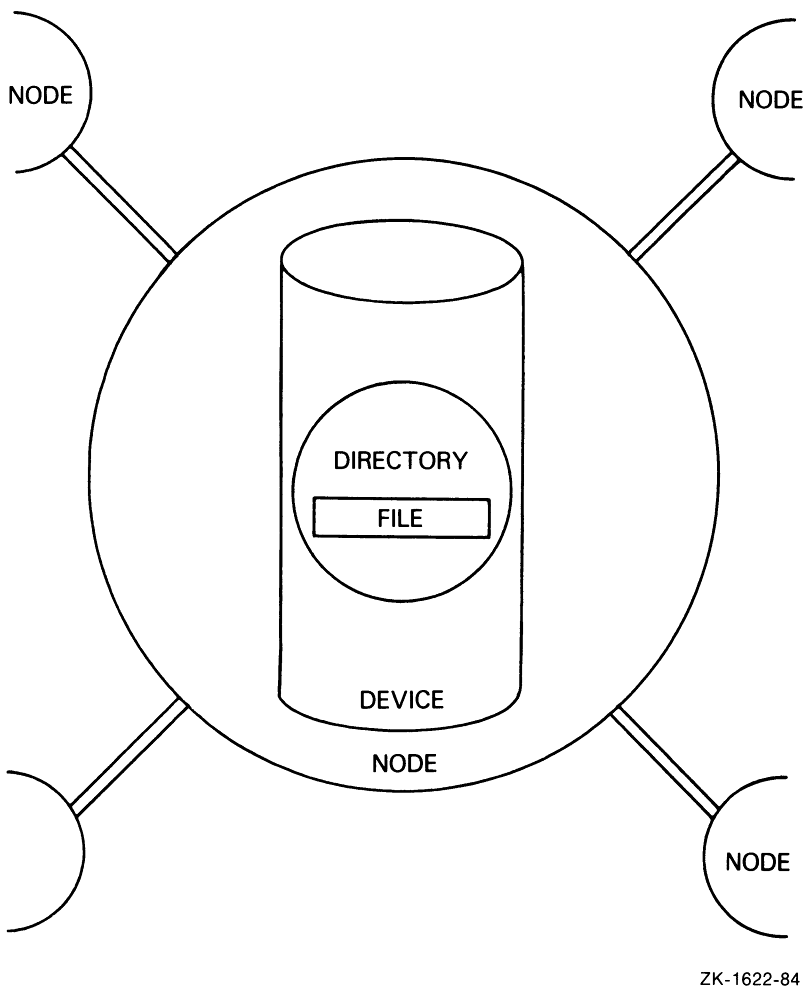
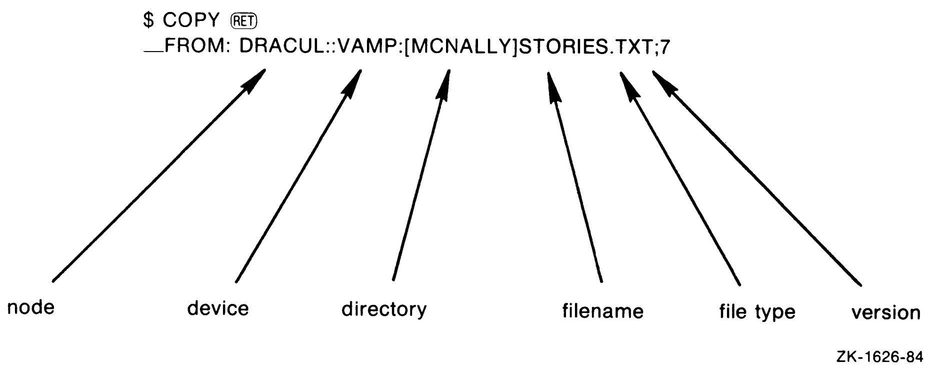
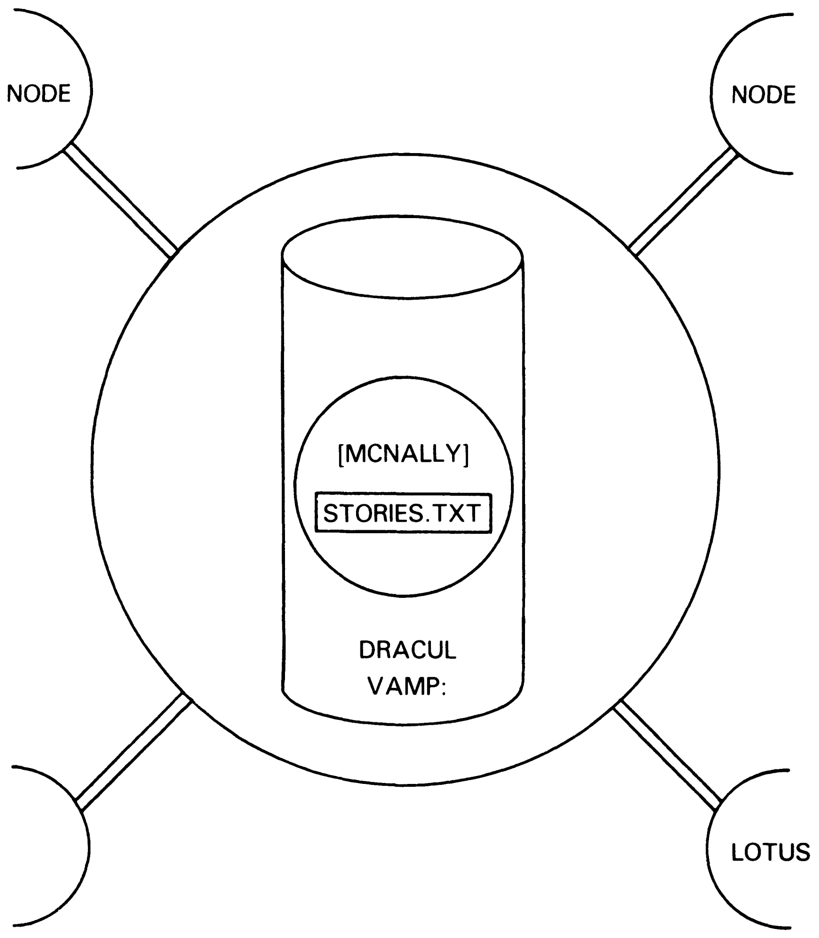


Figure 3–2 Full File Specification



```
$ COPY (RET)
_FROM: DRACUL::VAMP:[MCNALLY]STORIES.TXT;7
```

Figure 3–3 Relationship Between Parts of DRACUL::VAMP:[MCNALLY]STORIES.TXT



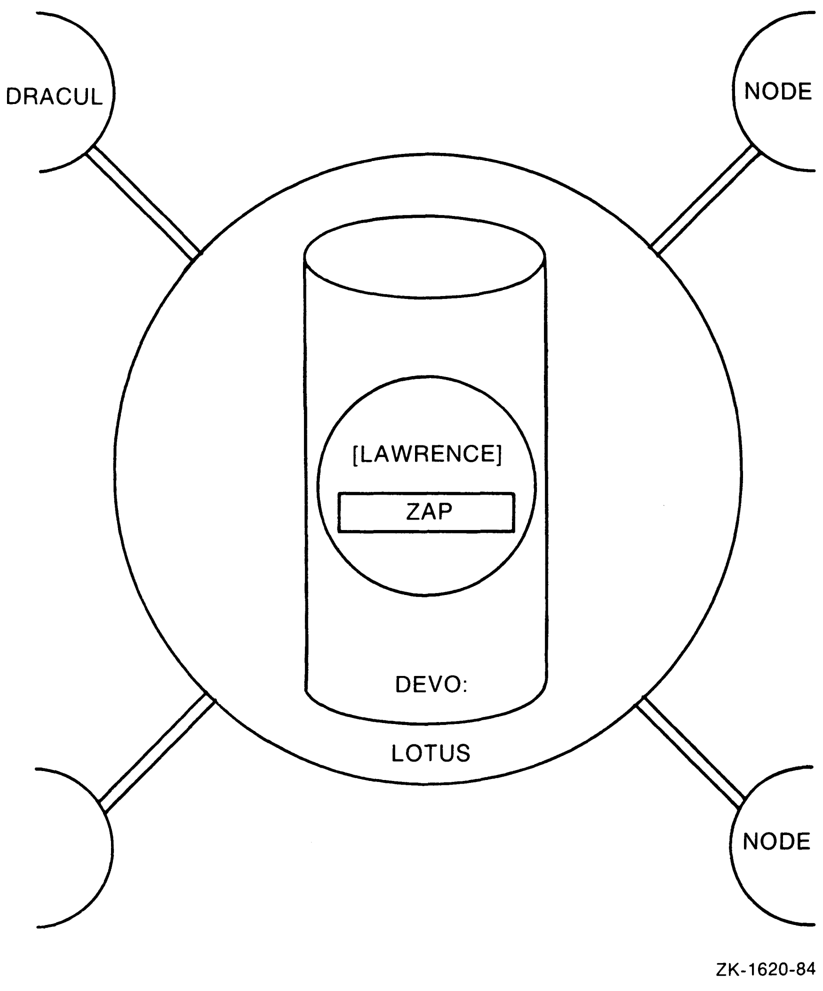
ZK-1621-84

Figure 3–4 shows the relationship between the parts of the previous file specification.

If you do not specify a node, the system assumes by default that the file belongs to your own, or local, node.

You should use network defaults when possible. For example, when you access a file on your local node, you do not need to specify the node name. By not specifying the node name, you avoid the unnecessary overhead of invoked network routines.

Figure 3–4 Relationship Between Parts of LOTUS::DEVO:[LAWRENCE]ZAP.LIS



3.1.2 Looking at Devices

The second part of a file specification, the device name, identifies the physical device on which a file is stored. Tapes and disks are examples of devices. A device name has the following three parts:

- The device type, which identifies the hardware device (for example, an RP06 disk is DB and a TE16 magnetic tape is MT).
- A controller designator, which identifies the hardware controller to which the device is attached
- The unit number, which uniquely identifies a device on a particular controller

Several examples of device names follow:

Name	Device
DBA2	RP06 disk on controller A, unit 2
MTA0	TE16 magnetic tape on controller A, unit 0
TTB3	Terminal on controller B, unit 3

You may notice that your device names are words instead of four-digit codes. These words are logical names. Logical names are character strings used to refer to files or devices. (For additional information about logical names, see Sections 3.2.1 and 4.6.)

If you enter the DCL command SHOW DEFAULT and see your default device displayed like the following example, you know that your system manager has set up logical names to indicate the devices available to you:

```
$ SHOW DEFAULT
 BOOK1:[MYRON]
```

In this example, the logical device name is BOOK1.

You can use logical names when referring to files to avoid typing long file and device names. If you specify a file using a logical device name, you can access the file regardless of which physical device holds the disk or tape containing your file. Your system manager ensures that the logical device names are always equated to the correct physical devices.

If you want to access a file located on the same node as your own but stored on a different device, you must specify the device name as part of the file specification. (When you use a logical name in a file specification, you must end the logical name with a colon.) The following example shows how to access a file named TREES.DAT, which is stored on a magnetic tape labeled TAPE1:

```
$ TYPE TAPE1:TREES.DAT
```

You do not need to specify a node name in the previous command string because the file TREES.DAT is located on your own node.

If you omit a device name from a file specification, the system supplies the default value; that is, it assumes the file is on the disk assigned to you when the system manager set up your account. This disk is your *default disk.*

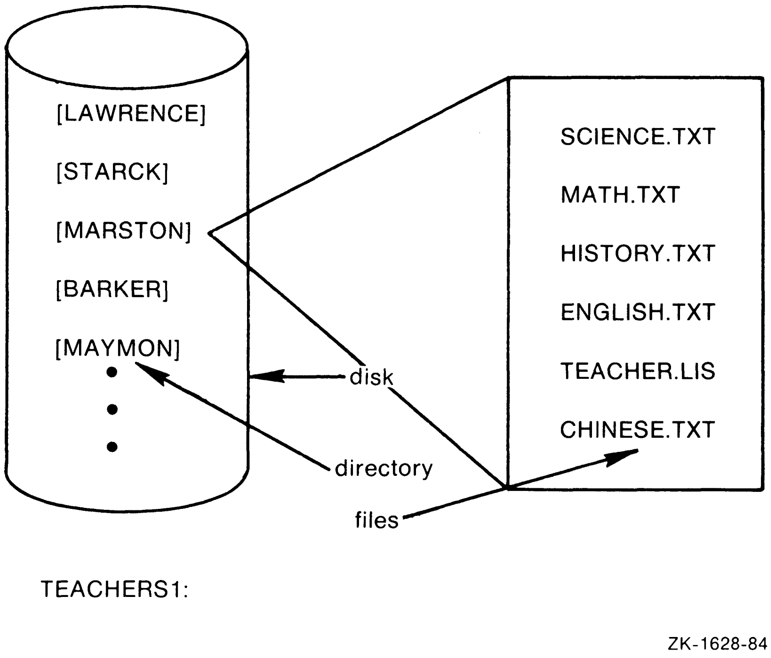
To list all the devices on the system, enter the SHOW DEVICES command.

3.1.3 Looking at Directories

Since a disk can contain files belonging to many different users, each user of a given disk has a directory that catalogs all of that user's files on the disk. A directory is a file that catalogs other files. A directory file contains the names and locations of files in a format that the system understands.

Figure 3–5 shows a disk (TEACHERS1) containing a list of five directories (for users LAWRENCE, STARCK, MARSTON, BARKER, and MAYMON) and the files listed in the [MARSTON] directory.

Figure 3–5 Files in [MARSTON] Directory



To access the file SCIENCE.TXT, you would enter the following command string:

```
$ TYPE TEACHERS1:[MARSTON]SCIENCE.TXT
```

As with the default disk, if you do not specify another directory, or if you do not specify any directory, the system applies the default; it assumes the files to which you refer are cataloged in your default directory. You can find out what your current default disk and directory are by entering the SHOW DEFAULT command, as follows:

```
$ SHOW DEFAULT
TEACHERS1:[MALCOLM]
```

The system's response to the SHOW DEFAULT command indicates the user's default device is TEACHERS1, and the default directory is [MALCOLM].

To gain access to files in other directories (including directories that catalog files belonging to other users), specify the directory name in a file specification. For example, to display the contents of a file named CONTENTS.DAT belonging to a user whose directory is [JONES], enter the TYPE command, as follows:

```
$ TYPE [JONES]CONTENTS.DAT [RET]
   .
   .
   .
```

Note that the file specification does not include a device name. For the TYPE command to execute successfully, the directory [JONES] must be on your default disk device. This is because the system always applies a default when you omit a device name. If user JONES's directory is on the disk RESEARCH3, you would enter the TYPE command as follows:

```
$ TYPE RESEARCH3:[JONES]CONTENTS.DAT RET
    .
    .
    .
```

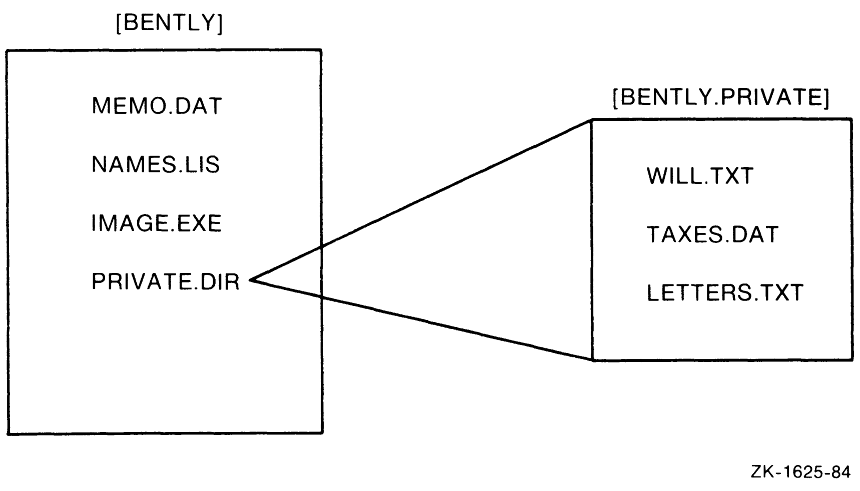
In both of these examples, it is assumed that user JONES has given other users access to files in the directory. You can explicitly allow or restrict access to your own files, either generally or on a file-by-file basis, with the SET PROTECTION command. See the *VMS DCL Dictionary* for more information about directory and file protection and for a detailed description of the SET PROTECTION command.

3.1.4 Looking at Subdirectories

Files can also be cataloged in *subdirectories.* A subdirectory is a file (cataloged in a higher level directory) that contains additional files.

Figure 3–6 shows a directory [BENTLY] containing one subdirectory [BENTLY.PRIVATE]:

Figure 3–6 Files in [BENTLY.PRIVATE] Subdirectory



A subdirectory name is formed by concatenating (or joining) its name to the name of the directory that lists it and separating the names with a period. The following TYPE command requests a display of the file MEMO.SUM, which is cataloged in the subdirectory [JONES.DATAFILES]. The subdirectory file name is DATAFILES.DIR and is cataloged in the directory [JONES]:

```
$ TYPE [JONES.DATAFILES]MEMO.SUM
```

Use subdirectories to organize and separate your files.

3.1.4.1 Creating Subdirectories

Normally, the system manager provides each system user with one directory in which to maintain files. If you are a frequent user of the system and work with several applications, you may find it convenient to create several subdirectories, which are cataloged in your main directory. You can create subdirectories in any directory in which you can create files.

Use the CREATE/DIRECTORY command to create a subdirectory. The following command creates the subdirectory file TESTFILES.DIR in the directory [MALCOLM], resulting in a subdirectory with the name [MALCOLM.TESTFILES]:

```
$ CREATE/DIRECTORY [MALCOLM.TESTFILES]
```

You can specify the subdirectory name [MALCOLM.TESTFILES] in commands or programs.

3.1.4.2 Changing Your Default Directory

To create a file in a subdirectory, you must be located at that directory, making it your new default directory. To change your default directory, use the SET DEFAULT command. The following example shows how to create a file in the subdirectory [MALCOLM.TESTFILES] by changing your default directory and then creating the file with the EDIT command:

```
$ SET DEFAULT [MALCOLM.TESTFILES]
$ EDIT NEWFILE.TXT
Input file does not exist
[EOB]
*
```

The new file is cataloged in the subdirectory [MALCOLM.TESTFILES].

You can also use the SET DEFAULT command to change your default disk. The following example shows how to specify PILOT as your default disk:

```
$ SET DEFAULT PILOT:
```

After you enter this command, the system uses the disk PILOT as the default disk for all files that you access or create.

You can change your default disk and directory as often as is convenient. The changes you make with the SET DEFAULT command remain in effect until you enter another SET DEFAULT command or log out of the system.

3.2 Using Logical Names

A *logical name* is a name equated to an equivalence string or to a list of equivalence strings. An equivalence string can be any group of characters. Most often, an equivalence string is a file specification, a device name, or another logical name. You use logical names for the following purposes:

- To reduce typing by using logical names as a short way of specifying files or directories you refer to frequently
- To avoid confusion about the location of volumes
- To keep your programs and command procedures independent of physical file specifications

You can use either the DEFINE command or the ASSIGN command to create a logical name by associating it with another name, the equivalence name. (The DEFINE and ASSIGN commands require different syntax. This chapter shows logical names with the DEFINE command. For information about the ASSIGN command, see the *VMS DCL Dictionary*.)

The following command assigns the logical name ZAP to the file specification DRACUL::DOC1:[MALVOLIO]ZAPISTS.DAT;21:

```
$ DEFINE ZAP DRACUL::DOC1:[MALVOLIO]ZAPISTS.DAT;21
```

After you have entered this command string, you can use the logical name ZAP in place of the longer file specification. The following command displays the contents of the file:

```
$ TYPE ZAP
```

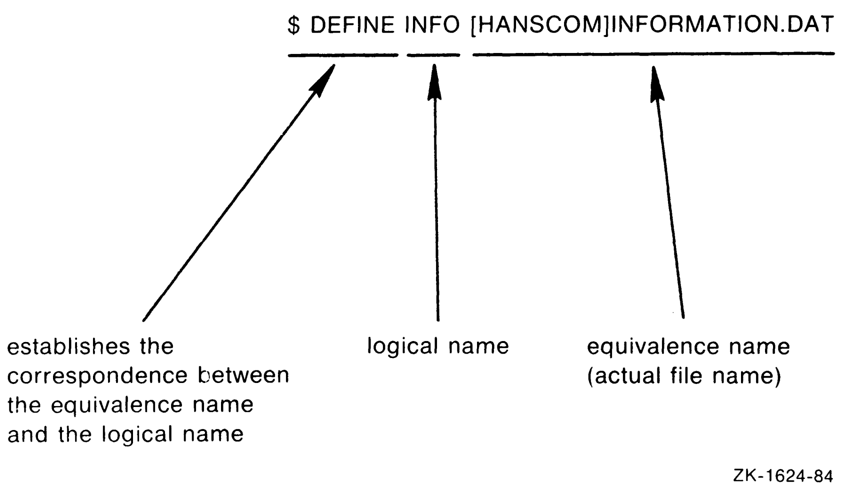
3.2.1 How to Use Logical Names

You can use the following syntax to assign a logical name:

```
$ DEFINE logical-name equivalence-name
```

Figure 3–7 assigns the logical name INFO to a file named INFORMATION.DAT, which is located in a directory named [HANSCOM]:

Figure 3–7 Assigning a Logical Name


$ DEFINE INFO [HANSCOM]INFORMATION.DAT
establishes the correspondence between the equivalence name and the logical name
logical name
equivalence name (actual file name)
ZK-1624-84

When you want to access the file INFORMATION.DAT (in the [HANSCOM] directory), you can use the logical name INFO. To display the file, enter the following command:

```
$ TYPE INFO
   .
   .
   .
```

When you create a logical name, it is maintained in a *logical name table.* A logical name table contains a set of logical names and their equivalence names. For detailed information about logical name tables, see the *VMS DCL Dictionary.*

You can use the SHOW LOGICAL command to display a logical name and its equivalence name. For example, to display the logical name HOME, you would enter the following command:

```
$ SHOW LOGICAL HOME
    .
    .
    .
```

The system searches the logical name tables for the logical name HOME. If it finds an entry, it displays the logical name and its equivalence name, and identifies the logical name table in which it found the logical name. In this example, the logical name HOME occurs in the process logical name table with the equivalence name of BOOK2:[JACK]. The system displays the following information:

```
  "HOME" [SUPER] = "BOOK2:[JACK]" (LNM$PROCESS_TABLE)
```

The following example shows how the DEFINE command equates the logical name MYFILE to the file PERSONNEL.REC listed in the directory CHUCK. To display this file on the terminal, enter the TYPE command, as follows:

```
$ DEFINE MYFILE [CHUCK]PERSONNEL.REC
$ TYPE MYFILE
```

A logical name can also define the first portion of a file specification. The following example shows how the DEFINE command equates the logical name TEST to the disk, device, and directory SCIENCE4:[MALCOLM.TESTFILES]. Subsequently, the RUN command executes the program image MEMO.EXE cataloged in this subdirectory, and the PRINT command prints another file:

```
$ DEFINE TEST SCIENCE4:[MALCOLM.TESTFILES]
$ RUN TEST:MEMO
$ PRINT TEST:MEMO.LIS
```

The system always examines file specifications to see if the portion of the file specification that precedes the colon (:) is a logical name; if it is (as in this example), the system substitutes the equivalence name.

3.2.2 System Default Logical Names

When you log in to the system or submit a batch job, the system provides several default logical names. These names are used by the command interpreter to read your commands and to print responses or error messages.

Most system default logical names have the following format:

```
xxx$name
```

The three-character prefix, xxx, identifies the system component that uses the logical name. The use of a dollar sign ($) within logical names is reserved for DIGITAL.

Several system default logical names follow:

Name	Use
SYS$COMMAND	The default device name of your terminal.
SYS$INPUT	The default input stream from which the system reads commands and your programs read data. The default interactive assignment for SYS$INPUT is your terminal. The default batch assignment for SYS$INPUT is the command procedure or batch stream.
SYS$OUTPUT	The default output stream to which the system writes responses to commands and your programs write data. The default interactive assignment for SYS$OUTPUT is your terminal. The default batch assignment for SYS$OUTPUT is the batch job log file.
SYS$ERROR	The default device to which the system writes all error and informational messages. The default interactive assignment for SYS$ERROR is your terminal. The default batch assignment for SYS$ERROR is the batch job log file.
SYS$DISK	Your default disk device. The default assignment is initially set in your User Authorization File (UAF) and can be changed with the DCL command SET DEFAULT.

Enter the following command to find out the equivalence names for these and other logical names created for your process:

```
$ SHOW LOGICAL
```

You may want to redefine SYS$OUTPUT to redirect output from your default device to a file. For example, if you want a hard copy of an online HELP file, you can assign an equivalence name (for example, HELP_LOGICAL_EXAMPLES.DAT) to the logical name SYS$OUTPUT, rechanneling output from your terminal to a file. Then, as you enter DCL commands interactively, the output goes to the specified file (equivalence name). To return output back to your terminal (away from the file), enter the DEASSIGN command. The following example demonstrates how to make a hard copy of the online HELP examples for the DCL command SHOW LOGICAL:

```
$ DEFINE SYS$OUTPUT HELP_LOGICAL_EXAMPLES.DAT
$ HELP SHOW LOGICAL EXAMPLES

Topic? RET
$ DEASSIGN SYS$OUTPUT
$ PRINT HELP_LOGICAL_EXAMPLES.DAT
```

If you want to capture the output from only one DCL command, use the /USER_MODE qualifier, as follows:

```
$ DEFINE/USER_MODE SYS$OUTPUT HELP_LOGICAL_EXAMPLES.DAT
$ HELP SHOW LOGICAL EXAMPLES

Topic? RET
$ PRINT HELP_LOGICAL_EXAMPLES.DAT
```

When you use the /USER_MODE qualifier, you do not need to enter the DEASSIGN command to return the output back to your terminal. (For more detailed information, see the description of the DEFINE command in the *VMS DCL Dictionary*.)

You can also use logical names in programs. For example, if you code a program to write a file to a device named SYS$OUTPUT, the output file goes to your terminal if you execute the program interactively, or to the batch job log file if you execute the program in a batch job.

4 Program Development

This chapter describes the following four steps required to develop a program in the VMS environment:

- Creating the *program* file
- Compiling or assembling the source program file to produce an *object module* file
- Linking the object module file to produce an image
- Executing and debugging the program

4.1 Creating the Program

To run your program, you must first create a file of the program source statements. Use a text editor to create the program. The default editor for VMS is EDT. To invoke the EDT editor, enter the DCL command EDIT. The *Guide to VMS Text Processing* describes how to use EDT. For complete descriptions of all available EDT commands, see the *VAX EDT Reference Manual*.

The default file type of the source program corresponds to the language in which the program is written. For instance, if your program is written in VAX BASIC, its file type default is BAS. Table 4–1 lists the default file types for source program files written in several VAX languages:

Table 4–1 Default File Types for Source Program Files

File Type	Input Source File for:
BAS	VAX BASIC compiler
B32	VAX BLISS-32 compiler
C	VAX C compiler
COB	VAX COBOL compiler
COR	VAX CORAL-66 compiler
FOR	VAX FORTRAN compiler
MAR	VAX MACRO assembler
PAS	VAX Pascal compiler
PLI	VAX PL/I compiler

4.2 Compiling or Assembling the Program

To prepare your source program for execution by the computer, a language processor must translate it into a format the computer can read. That is, your program must be either assembled or compiled, depending upon whether it is written in assembly language or in one of the high-level languages supported by VMS.

Both compilers and assemblers are programs that translate source programs into binary machine code that can be interpreted by the computer. An assembly language is usually designed for a specific computer, and it generally assembles each line of source code into a line of machine code. Most high-level languages, on the other hand, are designed to be universal and usually compile one line of source code into several lines of machine code. If your source program is written in assembly language (in this case, VAX MACRO), you invoke the VAX MACRO assembler to translate it. If it is written in a high-level language (such as BASIC, C, COBOL, FORTRAN, Pascal, or PL/I), you invoke the appropriate VAX language compilers to compile the program.

Table 4–2 lists the DCL commands you use to invoke various language processors.

Table 4–2 DCL Commands to Invoke Language Processors

Command	Language Processor Invoked
BASIC	VAX BASIC compiler
BLISS	VAX BLISS-32 compiler
CC	VAX C compiler
COBOL	VAX COBOL compiler
CORAL	VAX CORAL-66 compiler
FORTRAN	VAX FORTRAN compiler
MACRO	VAX MACRO assembler
PASCAL	VAX Pascal compiler
PLI	VAX PL/I compiler

Each of these commands invokes a compiler (or assembler) to translate the source program named in the file that follows the command. Although each command differs slightly in its parameters and qualifiers, the command format is essentially the same:

```
$ FORTRAN MYFILE
```

This command invokes the FORTRAN compiler to translate the file MYFILE into machine code, writing it to an output file called an object module. Since no file type is specified, the compiler assumes the default file type of FOR.

4.3 Linking the Object Module

An object module is not executable; generally, it contains references to other programs or routines that must be combined with the object module before it can be executed. It is the function of the *linker* to do the combining.

The LINK command invokes the VAX Linker. (For detailed information about the VAX Linker, see the *VMS Linker Utility Manual.*) The linker searches system libraries to resolve references to routines or symbols that are not defined within the object modules it is linking. You can request the linker to include more than one object module as input, or specify your own libraries of object modules for it to search. Following is the format of the LINK command:

```
$ LINK MYFILE
```

Since no file type is specified, the linker supplies a default file type of OBJ for object modules.

The linker creates an image, which is a file containing your program in an executable format. An image file has a default file type of EXE.

4.4 Executing the Program

The RUN command executes an image; that is, it places the image created by the linker into memory so that it can run. Following is the format of the RUN command:

```
$ RUN MYFILE
```

Since no file type is specified, the RUN command uses the default file type EXE for executable images.

The first time you run a program, it may not execute properly; if it has a bug or programming error, you may be able to determine the cause of the error by examining the output from the program. When you have determined the cause of the error, correct your source program, and repeat the compile, link, and run steps to test the result. Figure 4–1 illustrates these steps in program development.

Figure 4–2 lists the four steps you follow to develop a program using the BASIC language processor. (The name of the file containing the program is PROG.BAS.)

4.5 Looking at Sample Programs

The following sections illustrate the steps of program development with three sample programs: a BASIC example for novice users, a MACRO example for assembly language users, and a FORTRAN example for high-level language users. These sections describe the input and output files used in each step and the naming conventions for the files. They also present optional command qualifiers you can use to create additional output files, including program listings.

Figure 4–1 Program Development

Use the *editor* to create a disk file containing your source program statements. Specify the name of this file when you invoke the compiler or assembler.

Commands invoke language processors that check syntax, create object modules, and if requested, generate program listings.

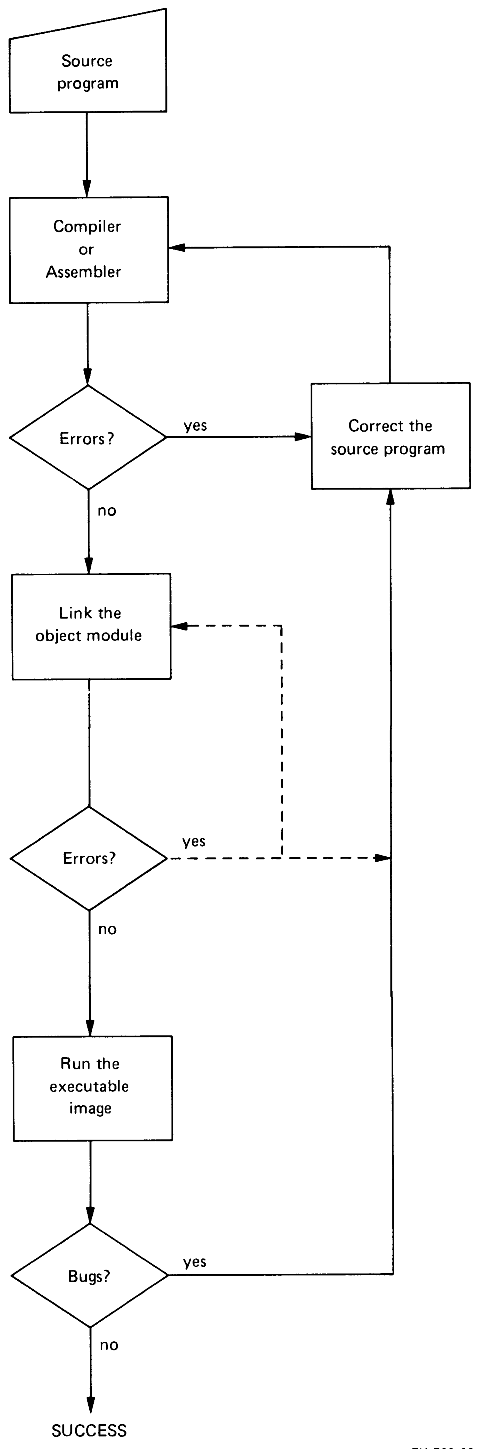
If a processor signals any errors, use the editor to correct the source program.

The *linker* searches the system libraries to resolve references in the object module and create an executable image. Optionally, you can specify private libraries to search, and request the linker to create a storage map of your program.

The linker issues diagnostic messages if an object module refers to subroutines or symbols that are not available or undefined. If the linker cannot locate a subroutine, you must reissue the *LINK* command specifying the modules or libraries to include. If a symbol is undefined, you may need to correct the source program.

The *RUN* command executes a program image. While your program is running, the system may detect errors and issue messages. To determine if your program is error-free, check its output.

If there is a bug in your program, determine the cause of the error and correct the source program.



ZK-763-82

Figure 4–2 Four Steps in Program Development

What You Do	Command Line You Enter	Input File You Supply	Resulting Output File
Create a source program file	$ EDIT PROG.BAS	PROG.BAS	PROG.BAS
Compile the source program to produce an object module file	$ BASIC PROG.BAS	PROG.BAS	PROG.OBJ
Link the object module file to produce an image	$ LINK PROG.OBJ	PROG.OBJ	PROG.EXE
Run the executable image	$ RUN PROG.EXE	PROG.EXE	—

ZK-6372-HC

4.5.1 An Introductory BASIC Program

The following section describes four steps you would perform to develop a BASIC program that adds three integers:

1. Use an editor to create a file named ADD.BAS containing the following six lines:

```
10      INPUT "What is the first integer" ;B
20      INPUT "What is the second integer" ;C
30      INPUT "What is the third integer" ;D
40      A = B + C + D
50      PRINT "Their sum is" ;A
60      END
```

2. Compile the source program file (ADD.BAS) using the following command to produce an object module file (ADD.OBJ):

```
$ BASIC ADD.BAS
```

3. Link the object module file (ADD.OBJ) using the following command to produce an image file (ADD.EXE):

```
$ LINK ADD.OBJ
```

4. Run the executable image using the following command:

```
$ RUN ADD.EXE
```

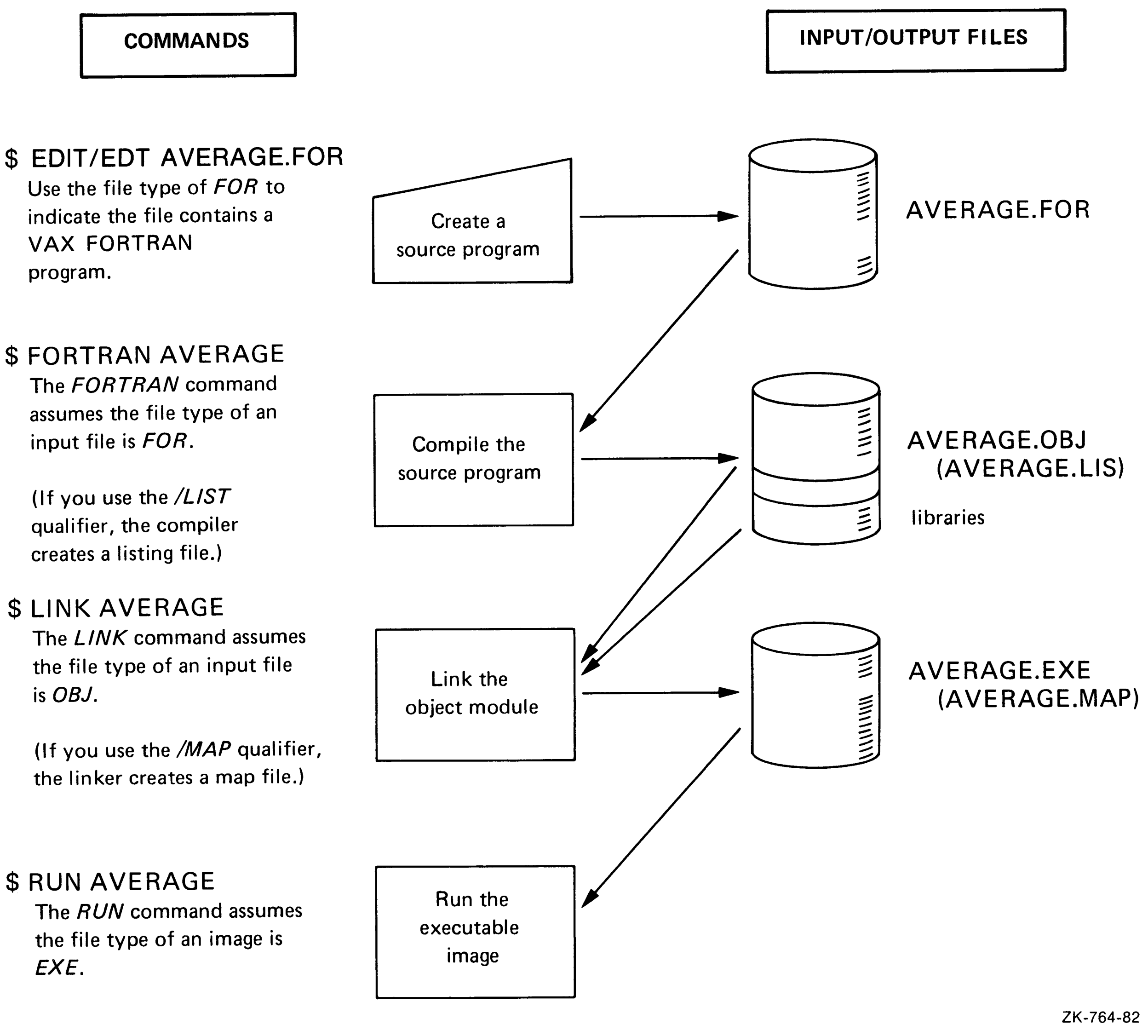
When you enter the RUN command, the ADD program prompts you for three integers and gives you their sum.

4.5.2 A FORTRAN Program

The steps required to prepare a VAX FORTRAN program to run on VMS are illustrated in Figure 4–3. Figure 4–3 also notes the default file types used by the FORTRAN, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the defaults when you name an input or output file.

Note: **The VAX FORTRAN compiler is referred to as FORTRAN throughout this manual.**

Figure 4–3 Commands for FORTRAN Program Development



COMMANDS

$ EDIT/EDT AVERAGE.FOR
Use the file type of *FOR* to indicate the file contains a VAX FORTRAN program.

$ FORTRAN AVERAGE
The *FORTRAN* command assumes the file type of an input file is *FOR*.

(If you use the */LIST* qualifier, the compiler creates a listing file.)

$ LINK AVERAGE
The *LINK* command assumes the file type of an input file is *OBJ*.

(If you use the */MAP* qualifier, the linker creates a map file.)

$ RUN AVERAGE
The *RUN* command assumes the file type of an image is *EXE*.

4.5.2.1 Creating the Source Program

Use an editor to create a source program interactively. For example, to create the FORTRAN program called AVERAGE, enter the DCL command EDIT. The following EDIT command invokes EDT, which is the default editor for VMS:

```
$ EDIT AVERAGE.FOR
```

The program AVERAGE follows. This program includes a syntax error and a bug to show you how to use VMS to debug a program. When you type the input statements, you can use the TAB key to align the statement and comments columns.

```
        PROGRAM AVERAGE

C       COMPUTES THE AVERAGE OF NUMBERS ENTERED AT TERMINAL
C       TO TERMINATE THE PROGRAM, ENTER 9999

        TOTAL = 0                   ! INITIALIZE ACCUMULATOR
        N = 0                       ! INITIALIZE COUNTER

5       N = N + 1
        WRITE (6,10)                ! PROMPT TO ENTER NUMBER

10      FORMAT (' ENTER NUMBER, END WITH 9999')
        READ (5,20) K               ! READ NUMBER FROM TERMINAL

20      FORMAT I10
        IF (K .EQ. 9999) GOTO 40    ! 9999 MEANS NO MORE INPUT
        TOTAL = TOTAL + K           ! COMPUTE TOTAL WITH NUMBER
        GOTO 5

C       NOW, COMPUTE AVERAGE BY DIVIDING TOTAL BY THE NUMBER OF
C       TIMES THROUGH THE LOOP

40      AVERAG = TOTAL/N
        WRITE (6,50) AVERAG         ! DISPLAY THE RESULT

50      FORMAT ('       AVERAGE IS  ',F10.2)

        STOP
        END
```

When properly debugged, the program AVERAGE reads and writes lines to the current input and output devices; it prompts you to enter numbers and then computes the average of the numbers entered.

4.5.2.2 Compiling the Source Program

When you enter the FORTRAN command from the terminal, the FORTRAN compiler does the following by default:

- Produces an object module that has the same file name as the source file and a file type of OBJ
- Uses FORTRAN compiler defaults when it creates the output files (qualifiers in the FORTRAN command string can override these defaults)

To compile the source program AVERAGE, enter the following command:

```
$ FORTRAN AVERAGE
```

Since the FORTRAN command assumes a file type of FOR, you do not need to specify the file type when you name the file to be compiled.

If the compilation is successful (that is, if the compiler did not detect any errors) the system displays the DCL prompt for the next command as follows:

```
$
```

If there are any errors, the FORTRAN compiler displays them on the terminal. If you entered the source program AVERAGE exactly as it appeared in Section 4.5.2.1, you received the following messages:

```
%FORT-F-ERROR 33, Missing operator or delimiter symbol
    [FORMAT I] in module AVERAGE at line 15
%FORT-F-ENDNOOBJ, TEST2:[MALCOLM]AVERAGE.FOR;1,
      completed with 1 diagnostic-object deleted
```

These fatal error messages indicate the FORMAT statement was incorrectly coded; you must put parentheses around the format specification.

To correct the error, edit the following line in the source file:

```
20   FORMAT I10
```

The corrected line, which contains parentheses, follows:

```
20   FORMAT (I10)
```

Now you can recompile the program using the following command:

```
$ FORTRAN AVERAGE
```

By default, the FORTRAN command always uses the version of the file with the highest version number. If the program compiles successfully this time, you can go on to the next step. Otherwise, repeat the procedure of correcting the source file and compiling it.

When you compile a source program, use the /LIST qualifier with the FORTRAN command to request the compiler to create a program listing, as in the following example:

```
$ FORTRAN/LIST AVERAGE
```

In addition to an object module, the FORTRAN compiler creates a file named AVERAGE.LIS. To obtain a printed copy of the program, use the PRINT command, as follows:

```
$ PRINT AVERAGE
```

The PRINT command uses the default file type of LIS.

4.5.2.3 Linking the Object Module

To link the program AVERAGE, enter the LINK command, as follows:

```
$ LINK AVERAGE
```

This LINK command creates a file named AVERAGE.EXE, which is an executable program image. The linker automatically includes in the executable image any library routines the compiler requested for input/output handling and error routines.

4.5.2.4 Running the Program

To execute the program AVERAGE, use the RUN command. When you enter the RUN command, you provide the name of an executable image. By default, the RUN command assumes a file type of EXE. Thus, to run the program AVERAGE, type the RUN command as follows:

```
$ RUN AVERAGE
```

AVERAGE is interactive; it prompts you to continue entering numbers and keeps a cumulative sum of the numbers you enter. When you enter 9999, it computes the average of all the numbers you entered. A typical run of this program might appear as follows:

```
ENTER NUMBER, END WITH 9999
33 RET
ENTER NUMBER, END WITH 9999
66 RET
ENTER NUMBER; END WITH 9999
99 RET
ENTER NUMBER, END WITH 9999
9999 RET
AVERAGE IS   49.50
FORTRAN STOP
$
```

As you can see, the program is not computing the average correctly. By looking at the program listing, you can see that the error occurs because the loop counter (N) is incremented a final time when you enter 9999 to terminate entering numbers. The value *N* must be decremented by 1.

To correct the error, edit the following line in the source file:

```
40   AVERAG = TOTAL/N
```

The corrected line follows:

```
AVERAG = TOTAL/(N-1)
```

Now repeat the compile, link, and run steps:

```
$ FORTRAN AVERAGE
$ LINK AVERAGE
$ RUN AVERAGE
ENTER NUMBER, END WITH 9999
33
ENTER NUMBER, END WITH 9999
66
ENTER NUMBER; END WITH 9999
99
ENTER NUMBER, END WITH 9999
9999
AVERAGE IS   66.00
FORTRAN STOP
$
```

In this example, the bug was easy to spot. This is not usually the case, however, and you may need to investigate a program further to debug it.

4.5.2.5 Debugging the Program

The VMS operating system has a *debugger,* which is a program that permits you to find and correct errors in your programs interactively. When you want to use the debugger, you must first compile the source program with the /DEBUG and /NOOPTIMIZE qualifiers, as follows:

```
$ FORTRAN/DEBUG/NOOPTIMIZE AVERAGE
```

The /NOOPTIMIZE qualifier prevents the debugger from rearranging your source code.

When the compilation completes, use the /DEBUG qualifier when you link the object module, as follows:

```
$ LINK/DEBUG AVERAGE
```

Now when you use the RUN command to execute the program image AVERAGE.EXE, the debugger takes control, and you can use debugging commands to stop the execution of the program at a particular statement to examine or modify a variable.

For online information about the debugger, enter the HELP command at the debugger prompt, as follows:

```
DBG>HELP
```

For detailed information about the debugger, see the *VMS Debugger Manual.*

4.5.3 A MACRO Program

The steps required to prepare a VAX MACRO program to run with VMS are illustrated in Figure 4–4. Figure 4–4 also notes the default file types used by the MACRO, LINK, and RUN commands. For any of these commands, you can specify an explicit file type to override the default when you name an input or output file.

Note: **The VAX MACRO assembler is referred to as MACRO throughout this manual.**

4.5.3.1 Creating the Source Program

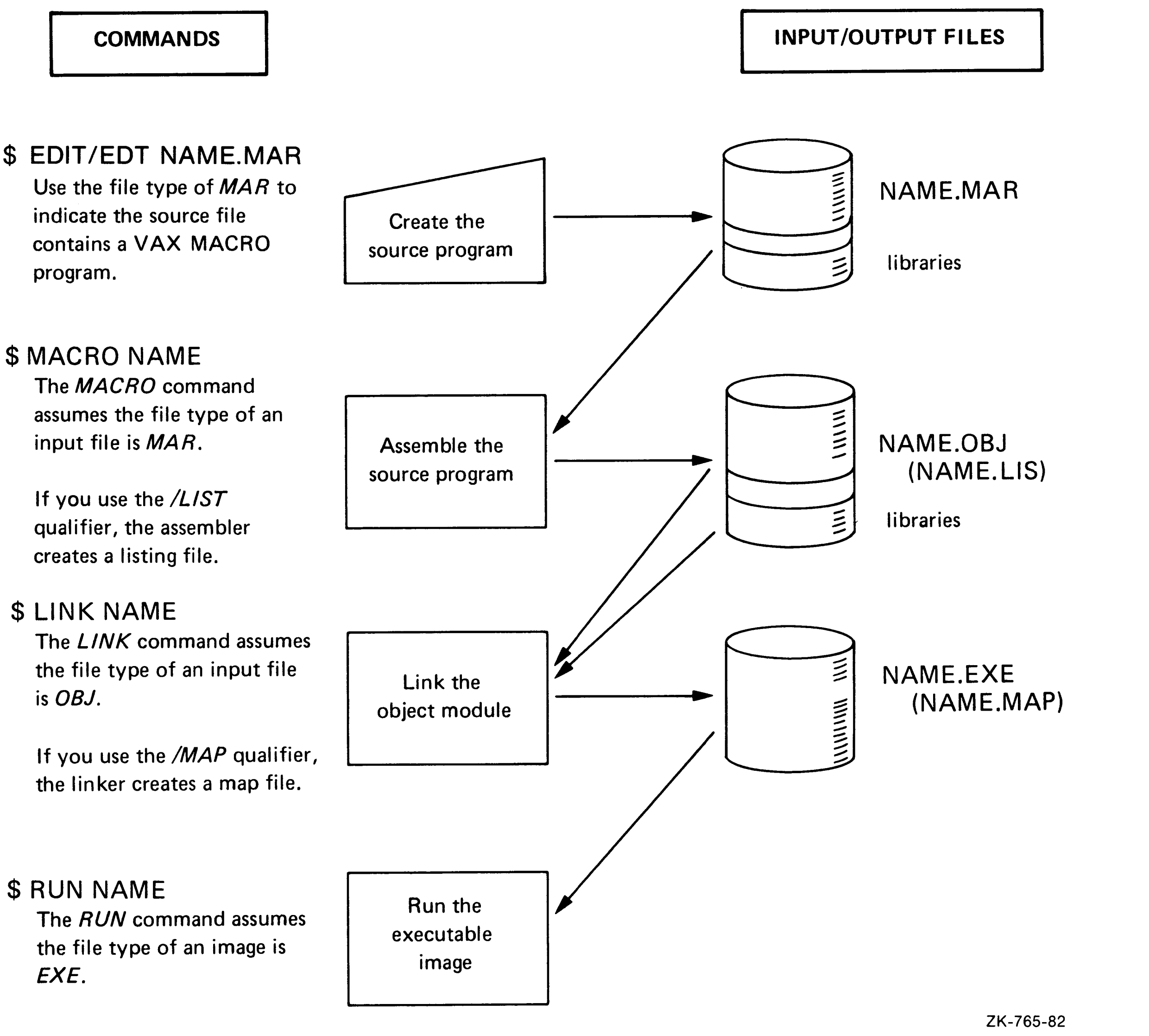
Use an editor to create a source program interactively. For example, to create the MACRO program called NAME, you can enter the DCL command EDIT. The following EDIT command invokes EDT, which is the default editor for VMS:

```
$ EDIT NAME.MAR
```

The program NAME is shown below Figure 4-4. When you type the input statements, you can use the TAB key to align the operand and comments columns.

The program uses VMS RMS to read and write lines to the current terminal; it issues a prompting message asking for the user's name and redisplays whatever is entered in response. The program NAME includes a syntax error and a bug to show you how to use VMS to correct programming errors.

Figure 4–4 Commands for MACRO Program Development


COMMANDS
INPUT/OUTPUT FILES
$ EDIT/EDT NAME.MAR
Use the file type of *MAR* to indicate the source file contains a VAX MACRO program.
Create the source program
NAME.MAR
libraries
$ MACRO NAME
The *MACRO* command assumes the file type of an input file is *MAR*.
If you use the */LIST* qualifier, the assembler creates a listing file.
Assemble the source program
NAME.OBJ
(NAME.LIS)
libraries
$ LINK NAME
The *LINK* command assumes the file type of an input file is *OBJ*.
If you use the */MAP* qualifier, the linker creates a map file.
Link the object module
NAME.EXE
(NAME.MAP)
$ RUN NAME
The *RUN* command assumes the file type of an image is *EXE*.
Run the executable image
ZK-765-82

A Sample Program: NAME

```
          .TITLE NAME
          .IDENT /01/
          .PSECT  RWDATA,WRT,NOEXE

; DEFINE CONTROL BLOCKS FOR TERMINAL INPUT AND OUTPUT

TRMFAB: $FAB    FNM=TT:,RAT=CR,FAC=<GET,PUT> ;FAB FOR TERMINAL

TRMRAB: $RAB    FAB=TRMFAB,UBF=BUFFER,USZ=BUFSIZ, -
                ROP=PMT, PBF=PMSG1, PSZ=P1SIZ

BUFFER: .BLKB    132                    ; INPUT READ BUFFER
BUFSIZ= .-BUFFER                        ; BUFFER LENGTH
```

```
PMSG1:  .ASCII   /ENTER YOUR NAME:         ; PROMPT MESSAGE
P1SIZ=  .-PMSG1                            ; MESSAGE SIZE

OUTMSG: .ASCII   /HELLO, YOUR NAME IS/     ; OUTPUT MESSAGE
OUTBUF: .BLKB    30                        ; MOVE NAME HERE
OUTLEN: .LONG    OUTBUF-OUTMSG
MSGSIZ: .BLKL    1                         ; ADD LENGTHS HERE

        .PSECT   NAME,EXE,NOWRT
        .ENTRY   BEGIN,0                   ; ENTRY MASK

        $OPEN    FAB=TRMFAB                ; OPEN TERMINAL FILE
        BLBC     R0,ERROR                  ; EXIT IF ERROR
        $CONNECT RAB=TRMRAB                ; ESTABLISH RAB
        BLBC     R0,ERROR                  ; EXIT IF ERROR

        $GET     RAB=TRMRAB                ; ISSUE PROMPT
        BLBC     R0,ERROR                  ; EXIT IF ERROR

; MOVE NAME ENTERED INTO OUTPUT MESSAGE, AND FIX UP LENGTH

        MOVC3    TRMRAB+RAB$W_RSZ,BUFFER,OUTBUF
        MOVZWL   TRMRAB+RAB$W_RSZ,MSGSIZ
        ADDL     MSGSIZ,OUTLEN
; AFTER CONSTRUCTING OUTPUT MESSAGE, OUTPUT IT

        MOVAL    OUTMSG,TRMRAB+RAB$L_RBF ; UPDATE RAB: ADDRESS
        MOVW     MSGSIZ,TRMRAB+RAB$W_RSZ ; UPDATE RAB: SIZE
        $PUT     RAB=TRMRAB
        BLBC     R0,ERROR                  ; EXIT IF ERROR

; ALL DONE, CLOSE THE FILE

        $CLOSE   FAB=TRMFAB

ERROR:

         RET
         .END    BEGIN
```

4.5.3.2 Assembling the MACRO Source Program

When you enter the MACRO command, the MACRO assembler does the following by default:

1. Produces an object module that has the same file name as the source file and a file type of OBJ
2. Uses MACRO assembler defaults when it creates output files (qualifiers on the command string can override these defaults)
3. Searches the system macro library for definitions for system macros, such as the VMS RMS macros $FAB and $RAB used in the sample program NAME.MAR

To assemble the source program NAME, enter the following command:

```
$ MACRO/LIST NAME
```

Since the MACRO command assumes a file type of MAR, you do not need to specify the file type when you name the file to be assembled. The /LIST qualifier indicates that you want a listing of the program; if there are any errors in the assembly, you may need the listing to determine the errors.

If the assembly is successful (that is, if the assembler did not detect any errors), the system displays the DCL prompt ($).

If errors occur, a message is displayed at the terminal. If you entered the source program NAME exactly as it appeared in Section 4.5.3.1, you received the following error message:

```
     45 47 41 53 53 45 40 20 54 50  0130
%MACRO-E-UNTERMARG, Unterminated argument
There were 1 error, 0 warnings, and 0 information messages on lines:
15(1)

MACRO/LIST NAME
```

This message indicates the ASCII string argument coded on line 15 is incorrect; you must terminate the string with a slash (/) character.

To correct the error, edit the following line in the source file:

```
PMSG1:  .ASCII   /ENTER YOUR NAME:      ; PROMPT MESSAGE
```

The corrected line, which contains the slash character, follows:

```
PMSG1:  .ASCII   /ENTER YOUR NAME:/     ; PROMPT MESSAGE
```

Now reassemble the program by entering the following command string:

```
$ MACRO/LIST NAME
```

If the program assembles successfully, go on to the next step. Otherwise, repeat the procedure of looking at the listing, correcting the source file, and assembling it.

4.5.3.3 Linking the Object Module

To link the program NAME, enter the LINK command as follows:

```
$ LINK NAME
```

The LINK command creates a file named NAME.EXE, which is an executable program image. The linker automatically includes in the executable image any library procedures required by the VMS RMS routines used.

4.5.3.4 Running the Program

To execute the program NAME, use the RUN command. When you enter the RUN command, you provide the name of an executable image. By default, the RUN command assumes a file type of EXE. Thus, to run the program NAME, type the RUN command, as follows:

```
$ RUN NAME
```

NAME is interactive; it prompts you to enter your name, then creates an output string from the string you entered and outputs it. A typical run of this program might appear as follows:

```
ENTER YOUR NAME: YORICK
HELLO,
$
```

The program is writing only the first 6 characters of the output message. If you examine the listing, you can see that on line 43 the MOVW instruction places the wrong length in the buffer size field of the RAB; it uses the MSGSIZ field (that is, the length of the string you entered) rather than the sum of the string you entered and the OUTMSG string.

To correct the error, edit the source file again using the following commands:

```
$ EDIT NAME.MAR
```

Edit the following line in the source file :

```
MOVW     MSGSIZ, TRMRAB+RAB$W_RSZ;  UPDATE RAB: SIZE
```

The corrected line follows:

```
MOVW     OUTLEN, TRMRAB+RAB$W_RSZ;  UPDATE RAB:SIZE
```

Now repeat the assembling, linking, and running using the following commands:

```
$ MACRO NAME
$ LINK NAME
$ RUN NAME
ENTER YOUR NAME: YORICK
HELLO, YOUR NAME IS YORICK
$
```

In this example, the bug was easy to spot. This is not always the case, however, and you may need to investigate a program further to debug it.

4.5.3.5 Debugging the Program

The VMS operating system has a debugger, a program that permits you to debug your programs interactively. When you want to use the debugger, assemble the source program with the /ENABLE=DEBUG qualifier, as follows:

```
$ MACRO/ENABLE=DEBUG NAME
```

This qualifier requests the assembler to include, in the object module, special information the debugger can use. When you link the object module you must specify the /DEBUG qualifier to link the debugger program with your program, as the following command string shows:

```
$ LINK/DEBUG NAME
```

Now when you use the RUN command to execute the program image NAME.EXE, the debugger takes control, and you can use debugging commands to stop the execution of the program at a particular instruction to examine or modify a variable.

For information on how to use the debugger, see the *VMS Debugger Manual.*

4.6 Using Logical Names for Programming Needs

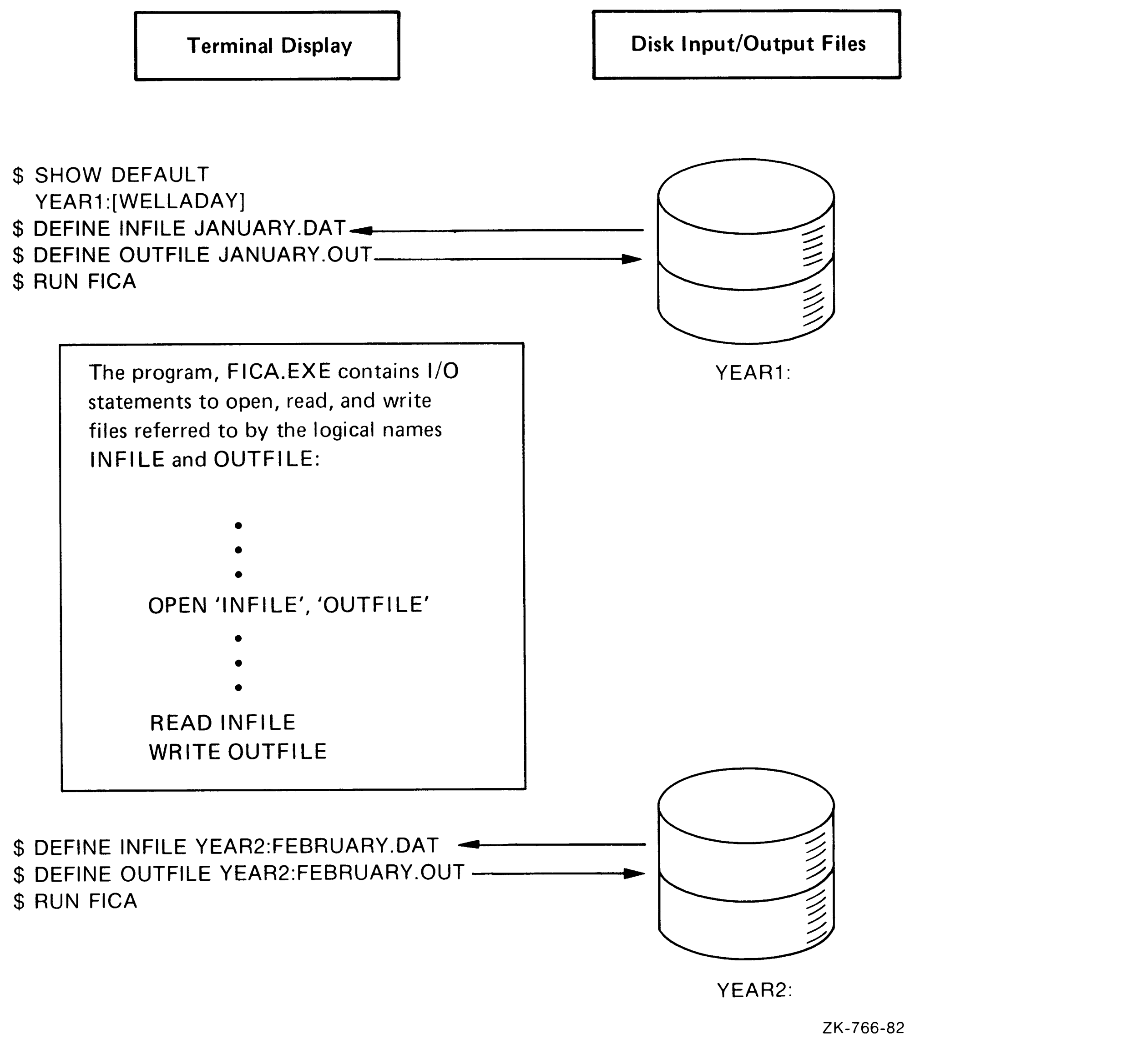
This section discusses how to make a program more efficient by using logical names.

When you design programs to read and write data, you can code the programs to read or write different files each time you run them. This is called device and file independence. In the VMS operating system, device independence is accomplished through the use of logical names.

When you code a program, you refer to an input or output file according to the syntax requirements of the language you are using. After the program is compiled and linked, but before you run it, you can use the DEFINE command to make a connection between the logical names you used in the program and the actual files or devices you want to use when you run the program.

Figure 4–5 shows how logical names are used. The program FICA contains OPEN, READ, and WRITE statements in a general form; the program reads from a file referred to by the logical name INFILE and writes to a file referred to by the logical name OUTFILE.

Figure 4–5 Using Logical Names



```
$ SHOW DEFAULT
  YEAR1:[WELLADAY]
$ DEFINE INFILE JANUARY.DAT
$ DEFINE OUTFILE JANUARY.OUT
$ RUN FICA
```

The program, FICA.EXE contains I/O statements to open, read, and write files referred to by the logical names INFILE and OUTFILE:

```
        •
        •
        •
    OPEN 'INFILE', 'OUTFILE'
        •
        •
        •
    READ INFILE
    WRITE OUTFILE
```

```
$ DEFINE INFILE YEAR2:FEBRUARY.DAT
$ DEFINE OUTFILE YEAR2:FEBRUARY.OUT
$ RUN FICA
```

For different runs of the program, the DEFINE command establishes different equivalence names for INFILE and OUTFILE. In the first example, the program reads the file JANUARY.DAT from the device YEAR1 and writes to the file JANUARY.OUT on the same device. In the second example, it reads the file FEBRUARY.DAT from the device YEAR2 and writes the file FEBRUARY.OUT to that device.

5 Using Symbols and Command Procedures

This chapter provides information on adapting the DCL command language to your individual needs. For example, you can:

- Establish synonyms to use in place of command names and entire command strings
- Establish default qualifiers for commands
- Create command procedures to perform a series of DCL commands
- Submit command procedures as batch jobs

You can simplify the command language to save time during interactive terminal sessions. You can establish your own default commands and command qualifiers and can also create command procedures. Command procedures enable you to execute a series of DCL commands by entering one command.

5.1 Abbreviating DCL Commands with Symbols

When using long DCL commands on a regular basis, you can save time by equating them to symbols. For example, you can equate the symbol ST to the DCL command SHOW TIME:

```
$ ST = "SHOW TIME"
```

After you equate a symbol to an expression (which can be a DCL command), the symbol assumes a new identity or value. In the previous example, the symbol ST assumes a new identity as the DCL command SHOW TIME. Now you can use the symbol ST in place of the DCL command SHOW TIME, as follows:

```
$ ST
   31-DEC-1988  10:45:19
```

The three parts of a symbol equation follow:

- The symbol (for example, ST)
- An equal sign (=)
- The expression (for example, "SHOW TIME")

The symbol part of the equation can be any alphanumeric string that you provide. Follow the symbol with an equal sign (=) that assigns the value of the expression to the symbol. You also need to supply an expression. An expression can be either a character string (for example, the command SHOW PROCESS) or a number (for example, 8). Enclose the expression in quotation marks (") if it is a character string.

You can simplify the use of the DCL command SHOW USERS by equating it to the symbol LOOK, as follows:

```
$ LOOK = "SHOW USERS"
```

Instead of typing the DCL command SHOW USERS, you can now type the word LOOK for a listing of all the current users on the system. In this example, LOOK is the symbol, and SHOW USERS is the expression.

You can use the DCL command SHOW SYMBOL to see the value of any symbol you create. To see the value of the symbol LOOK, enter the following command string:

```
$ SHOW SYMBOL LOOK
 LOOK = "SHOW USERS"
$
```

The following example shows how to equate the symbol WEIGHT to the arithmetic expression 125:

```
$ WEIGHT = 125
```

Notice that when equating a symbol to a number, you do not enclose the number in quotation marks ("). (For detailed information about syntax and grammar rules, see the *VMS DCL Dictionary*.)

Now the symbol WEIGHT has the value of 125. You can substitute the symbol WEIGHT for the number 125. To see the value of the symbol WEIGHT, enter the following command string:

```
$ SHOW SYMBOL WEIGHT
 WEIGHT = 125   Hex = 0000007D  Octal = 00000000175
```

Notice that the system displays the numeric value of the symbol in three forms: decimal, hexadecimal, and octal.

You can save time by equating long command strings to symbols. For example, a user named BERGMAN would enter the following command string to move from a subdirectory to the main directory:

```
$ SET DEFAULT WORK6:[BERGMAN]
```

To save time, BERGMAN can equate this command string to a symbol (HOME), as follows:

```
$ HOME = "SET DEFAULT WORK6:[BERGMAN]"
```

To move to the main directory, the user BERGMAN only needs to type the symbol HOME, as follows:

```
$ HOME
```

Symbols can also be defined for command strings containing qualifiers. For example, to define a synonym for the DIRECTORY command that automatically includes the /FULL qualifier, you can define the symbol LIST, as follows:

```
$ LIST = "DIRECTORY/FULL"
```

If you enter the following command string, the system substitutes the command DIRECTORY/FULL for the symbol LIST:

```
$ LIST MYFILE.DAT
```

The system executes the command string DIRECTORY/FULL MYFILE.DAT, displaying detailed information about the files in your directory.

Some symbols must be enclosed in apostrophes (') to identify them as symbols to the system. When the system detects the apostrophes, it knows to perform symbol substitution. For example, you can equate a symbol to a long file name, as follows:

```
$ BEST = "3145CHAPTER_ON_SONGS_OF_1990.DAT"
```

To type the file, all you need to enter is the TYPE command with the symbol name enclosed in apostrophes ('), as follows:

```
$ TYPE 'BEST'
```

See the *VMS DCL Dictionary* and the *Guide to Using VMS Command Procedures* for detailed information about using apostrophes with symbols.

Symbols can be concatenated (linked) with other symbols or items on a command string. In this case, you must enclose the symbol in apostrophes (') to indicate to the system that it must perform symbol substitution. For example, you can assign the symbol PQUALS to the following qualifiers for the PRINT command:

```
$ PQUALS = "/COPIES=2/FORMS=4/NOBURST"
```

Then, to use the symbol with the PRINT command, you must enclose it in apostrophes, as follows:

```
$ PRINT REPORT.DAT'PQUALS'
```

The system recognizes the apostrophes and substitutes the appropriate value (in this case the following string of qualifiers) for the symbol PQUALS:

```
/COPIES=2/FORMS=4/NOBURST
```

For more information about the effect of these qualifiers on the PRINT command, as well as rules about when to use apostrophes for symbol substitution, see the *VMS DCL Dictionary*.

Note that any symbols you assign will disappear when you log out unless you put them in a LOGIN.COM file. Section 5.2.4 explains how to create and use a LOGIN.COM file.

5.2 Creating and Executing a Command Procedure

A command procedure is a file that contains a sequence of DCL commands. You create a command procedure by using a text editor (such as EDT) to create a file, then fill the file with DCL commands. When you invoke the command procedure, the commands are executed beginning with the first command and continuing consecutively to the end of the procedure. Each command is executed as if you had actually typed it in on the command line.

The default file type for a command procedure file is COM.

The following example shows how to create a command procedure named CLEAN.COM that purges your default directory and then displays the remaining files in that directory. The exclamation mark introduces a comment, which is text that is ignored when the command procedure is executed.

Use a text editor to create a file named CLEAN.COM. Copy the following three lines of text into the file:

```
$ ! Purge files and look
$ PURGE
$ DIRECTORY
```

Exit from the editor. Execute the command procedure by typing an at sign (@) followed by the name of the file containing the commands. For example, to execute the command procedure CLEAN.COM, enter the following command string:

```
$ @CLEAN
```

You can invoke a command procedure from any directory. But if the command procedure is not located in your current directory, you must precede the command procedure name by the name of the directory in which it is located. For example, if your current directory is [BASIL.FOREIGN], but the command procedure you want to invoke (CLEAN.COM) is in a different directory named [BASIL.DOMESTIC], you would enter the following command string to invoke CLEAN.COM:

```
$ @[BASIL.DOMESTIC]CLEAN.COM
```

Following are the rules for formatting a command procedure:

- Begin each command string with a dollar sign. If a command in your command procedure requires information you would normally type in, put that information on lines without dollar signs (data lines) following the command. For example, to use MAIL from a command procedure, put your responses to the prompts on data lines following the MAIL command.
- Do not abbreviate commands. Although abbreviation is allowed, the command procedure is easier to read if all commands are spelled out.
- Begin comments with an exclamation point, and use them often. Comments explain what the procedure is doing and are especially helpful in complex command procedures.

The following command procedure (SEND.COM) sends a message to node EBONY, with a note to the accounting department. (The CTRL/Z that usually ends the message is not necessary because the end of the command procedure indicates the end of the message.)

```
                 +------------------SEND.COM----------------+
       comment ->| $ ! Send a message to EBONY               |
command string ->| $ MAIL                                    |
             +---| SEND                                      |
             |   | EBONY::USER                               |
data lines ->|   | Attention Accounting                      |
             |   | Please forward the RAZORON transactions.  |
             +---| Thanks.                                   |
                 +-------------------------------------------+
```

Do not put comments on data lines. If you do, DCL treats the comments as data when it reads the information from the data lines. To change the information on the data lines, you must edit the command procedure.

5.2.1 Passing Information

When you want to pass information (for example, a file name) to a command procedure, you can have the procedure request a value for a symbol. You type in a value, and your command procedure uses the symbol equated to that value in subsequent commands. When you want your command procedure to pass information back to you, you can have the command procedure display information to the terminal.

5.2.1.1 Requesting Information with the INQUIRE Command

If your command procedure needs information from you, use the INQUIRE command to request it. The INQUIRE command prompts you for information, then equates your response to a symbol. This command requires two parameters: the symbol name and the prompt. Use descriptive prompts (for example, "Enter a file name") to keep your command procedures clear.

The following example shows how the INQUIRE command displays the prompt "File:" (DCL automatically adds the colon and space to the prompt that you specify) and puts the user's response in the symbol OLD_FILE. The command procedure uses OLD_FILE in the PRINT and PURGE commands. (See the *VMS DCL Dictionary* and the *Guide to Using VMS Command Procedures* for detailed information about using apostrophes.)

Use an editor to create a file named NEW_CLEAN.COM. Copy the following four lines of text into the file:

```
$ ! Print all versions and purge
$ INQUIRE OLD_FILE "File"
$ PRINT 'OLD_FILE';*
$ PURGE 'OLD_FILE'
```

To execute the command procedure, enter the following command string:

```
$ @NEW_CLEAN
```

You will be prompted for a file name as the following example shows:

```
File: BILLS.DAT
```

The system executes the command procedure NEW_CLEAN, which prints and purges all versions of the file BILLS.DAT.

5.2.1.2 Displaying Information with the WRITE Command

When you want your command procedure to display information on your screen, use the WRITE command. The WRITE command takes two parameters: the first parameter tells DCL where to display the text; the second parameter tells DCL what text to display. To tell DCL that you want the text displayed on the terminal, use the logical name SYS$OUTPUT as the first parameter. (When you log in, the system automatically equates SYS$OUTPUT to the output stream for your terminal. The system uses this output stream for prompting and informational messages.)

If you want to display a line of text, enclose the text in quotation marks, and include it as the second parameter. For example, create a file named FUN.COM and fill it with the following line:

```
+----------------------FUN.COM---------------------------+
|                                                        |
| $ WRITE SYS$OUTPUT "Hello Dolly"                       |
|                                                        |
+--------------------------------------------------------+
```

Now enter the command @FUN to see the text "Hello Dolly".

The following command procedure prints and purges a file, then displays the text "All versions printed; file purged."

```
+-------------------------CLEAN.COM------------------------+
| $ PRINT 'P1';*                                           |
| $ PURGE 'P1'                                             |
| $ WRITE SYS$OUTPUT "All versions printed; file purged."  |
+----------------------------------------------------------+
```

When you want to display the value of a symbol, include the symbol as the second parameter. For example, the third line of the following command procedure (CLEAN1.COM) contains the WRITE command followed by the first parameter (SYS$OUTPUT) and the second parameter (P1):

```
+-------------------------CLEAN1.COM-----------------------+
| $ PRINT 'P1';*                                           |
| $ PURGE 'P1'                                             |
| $ WRITE SYS$OUTPUT P1                                    |
+----------------------------------------------------------+
```

After you enter the following command string, the command procedure CLEAN1.COM displays the value of P1 as BILLS.DAT:

```
$ @CLEAN1 BILLS.DAT
```

If you want to display a line of text and the value of one or more symbols, include the symbols in the text, and specify the entire line as the second parameter. Each symbol in the text must be preceded by two apostrophes and followed by one apostrophe. For example, when the following command procedure is invoked with the command string @CLEAN BILLS.DAT, the procedure displays the message "All versions of BILLS.DAT were printed; file was purged."

```
+-------------------------CLEAN.COM------------------------+
| $ PRINT 'P1';*                                           |
| $ PURGE 'P1'                                             |
| $ WRITE SYS$OUTPUT -                                     |
| "All versions of ''P1' were printed; file was purged."   |
+----------------------------------------------------------+
```

5.2.2 Using Logic

When you want your command procedure to choose a course of action depending on a piece of information you provide, use the IF command. When you want your command procedure to skip a number of steps, use the GOTO command.

The IF command conditionally executes a command. You specify both the condition and the command in the following format:

```
$ IF condition THEN command
```

When the command procedure reads an IF statement, it evaluates the condition. If the condition is true, the command executes; if it is false, the system ignores the command and executes the statement following the IF command. A condition must be stated using the conditional operators listed in Table 5–1.

Table 5–1 Conditional Operators

Operator	Operator	Function
#.EQ.	.EQS.	Equal to
#.NE.	.NES.	Not equal to
#.LE.	.LES.	Less than or equal to
#.LT.	.LTS.	Less than
#.GE.	.GES.	Greater than or equal to
#.GT.	.GTS.	Greater than

Use the GOTO command to direct the command procedure to a particular line in your command procedure. A label indicates the exact line in the command procedure. This command has the following format:

```
$ GOTO label
```

The following command directs the command procedure to the START label:

```
$ GOTO START
```

The command procedure reads the GOTO statement and goes to the following line:

```
$ START:
```

The command procedure then reads the lines following the label. Notice that the label requires a colon (:) except when it is in the GOTO command. Each label in a command procedure must be unique.

The following command procedure uses the IF statement to see whether you have specified a file name:

```
+------------------CLEAN.COM---------------------+
| $ ! Print all versions and purge               |
| $ INQUIRE OLD_FILE "File"                      |
| $ IF OLD_FILE .EQS. "" THEN GOTO ERR           |
| $ PRINT 'OLD_FILE';*                           |
| $ PURGE 'OLD_FILE'                             |
| $ EXIT                                         |
| $ ERR:                                         |
| $ WRITE SYS$OUTPUT "No files printed or purged." |
| $ EXIT                                         |
+------------------------------------------------+
```

If OLD_FILE is blank (""), the command GOTO ERR executes. The command procedure continues executing at the line labeled ERR: and displays the message "No file printed or purged." If OLD_FILE is not blank, the file named in OLD_FILE is printed and purged. When the command procedure reads the EXIT command, it exits and returns the user to the DCL command level.

5.2.3 Extracting Information with Lexical Functions

A *lexical function* is like a symbol in that it is equated to a value. However, a lexical function obtains its value by reading arguments supplied by you and by performing a particular operation. Each lexical function performs a specific task. For example, F$LENGTH reads a character string that you supply and then replaces itself with the length of that character string.

To calculate the length of the string MARBLE and equate that value to the symbol SIZE, enter the following command:

```
$ SIZE = F$LENGTH("MARBLE")
```

To see the result of the calculation (the value of the symbol SIZE), enter the following command string:

```
$ SHOW SYMBOL SIZE
       SIZE = 6   Hex = 00000006   Octal = 00000000006
```

Instead of supplying a specific value for the argument for a lexical function, you can use a symbol. Then equate values to the symbol. For example, enter the following command string to equate the symbol ANYTHING to the string "BLAST":

```
$ ANYTHING = "BLAST"
```

Enter the following command string to make F$LENGTH calculate the length of the value of the symbol ANYTHING (in this case, the symbol ANYTHING is equated to the string BLAST):

```
$ SIZE = F$LENGTH(ANYTHING)
```

In the previous example, the symbol ANYTHING equates to the string "BLAST". The lexical function F$LENGTH determines the length of "BLAST" and equates this length to the symbol SIZE.

To see the result of the calculation (the value of the symbol SIZE), enter the following command string:

```
$ SHOW SYMBOL SIZE
       SIZE = 5   Hex = 00000005   Octal = 00000000005
```

When the length of the argument changes, F$LENGTH returns a different number, as the following example shows:

```
$ ANYTHING = "TEA"
$ SIZE = F$LENGTH (ANYTHING)
$ SHOW SYMBOL SIZE
       SIZE = 3   Hex = 00000003   Octal = 00000000003
```

In the previous example, the symbol ANYTHING equates to the string "TEA". The lexical function F$LENGTH determines the length of "TEA" and equates the symbol SIZE to this length.

Table 5–2 lists some commonly used lexical functions with their required arguments.

Table 5–2 Common Lexical Functions

Function	Description
F$EXTRACT(offset,length,string)	Returns a substring
F$LENGTH(string)	Returns the length of a character string
F$LOCATE(substring,string)	Returns the offset position of a substring
F$MODE( )	Returns a character string showing the mode in which a process is executing
F$STRING(expression)	Converts a number to a character string
F$TIME( )	Returns the current date and time

For detailed information about lexical functions and their required arguments, see the *VMS DCL Dictionary*.

5.2.4 What Is a LOGIN.COM File?

If you become a frequent user of the VMS system, you may find that you are entering the same sequence of commands or assignment statements every time you log in. To avoid such repetition, you can place these commands and statements in a special command procedure.

You must name this file containing the command procedure LOGIN.COM, and create it in your default disk directory. Like other command procedures, create your LOGIN.COM file using an editor, such as EDT. When you log in, the system automatically searches for a file with this file name. If the system locates the LOGIN.COM file, it automatically executes the commands within that file.

A LOGIN.COM file might contain the definitions in Figure 5–1.

Figure 5–1 Looking at a LOGIN.COM File

```
$IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
$!symbols for commands I frequently use
$ST=="SHOW TIME"
$SHQU=="SHOW QUEUE"
$!symbols for my directories
$HOME=="SET DEFAULT WORK6:[MALCOLM]"
$TEST=="SET DEFAULT [MALCOLM.TESTFILES]"
$BILLS=="SET DEFAULT [MALCOLM.BILLS]"
$!logical names for people to whom I send mail
$DEFINE AL ALBINONI
$DEFINE BRO BREATH::BROKOWITZ
$DEFINE SAL TITIAN::SALIMONI
```

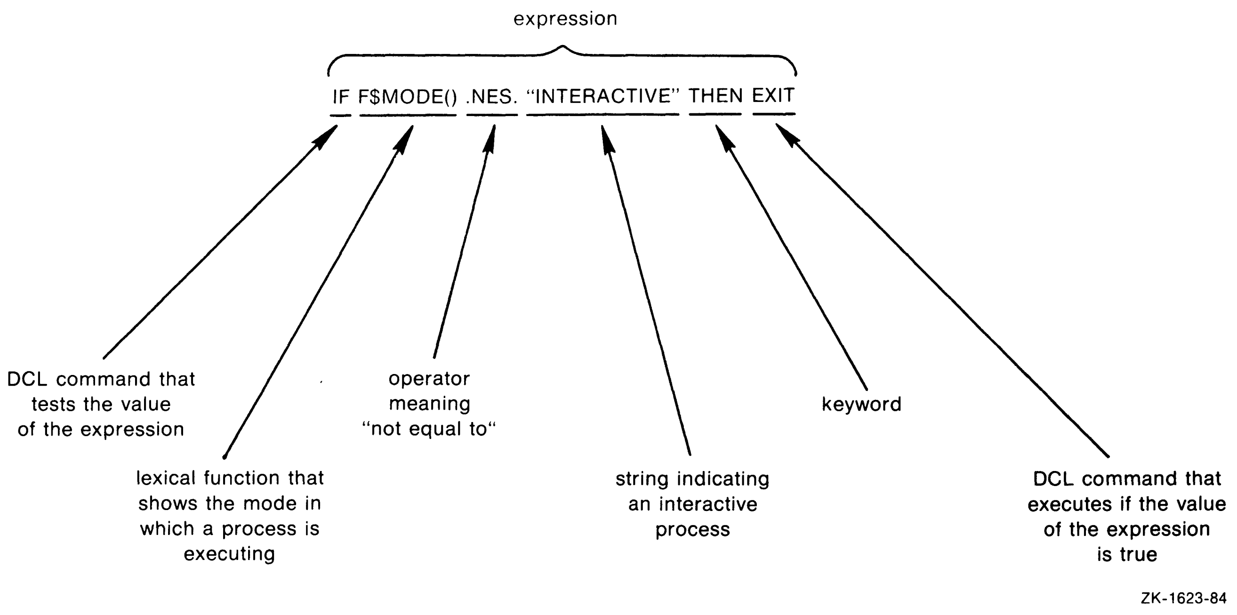
Note that all the symbols defined in the LOGIN.COM file in Figure 5–1 are global symbols, assigned with two equal signs. If these symbols were local (assigned with one equal sign) they would be recognized only within the LOGIN.COM file and would therefore be useless to you.

Also note the first line in the LOGIN.COM file in Figure 5–1. This line translates into the following:

```
"If the current process is not interactive, then ignore
the rest of this LOGIN.COM file".
```

Figure 5–2 shows the labeled line.

Figure 5–2 The First Line of a LOGIN.COM file


expression
IF F$MODE() .NES. "INTERACTIVE" THEN EXIT
DCL command that tests the value of the expression
lexical function that shows the mode in which a process is executing
operator meaning "not equal to"
string indicating an interactive process
keyword
DCL command that executes if the value of the expression is true
ZK-1623-84

You must include this line in your LOGIN.COM file because your LOGIN.COM file may contain commands specific to an interactive environment that would cause a batch or network job to abort. (For information about batch jobs, see Section 5.2.5.)

5.2.5 Submitting Batch Jobs to Avoid Delays

If you have executed a command procedure, you have noticed that while the procedure is executing, you cannot do anything else on your terminal. This is because your process is executing the command procedure and can execute only one command at a time. To avoid this delay, submit a command procedure as a batch job. A batch job is executed in a process of its own: therefore, your process is not kept waiting. However, this means that you cannot use the terminal to send information to or receive information from the command procedure while it executes.

To submit a command procedure as a batch job, use the SUBMIT command. The following example shows how to submit the command procedure CLEAN.COM in the [ACCOUNT] directory as a batch job:

```
$ SUBMIT [ACCOUNT]CLEAN
```

The command procedure is placed in a batch job queue, where it waits to be executed. For more information about batch jobs, see Section 6.2.

When a command procedure is submitted as a batch job, it executes as if you had just logged in. This means that your default directory, which is not necessarily the directory that you want to use, is probably your top level directory. If you want a different default directory, you must include the SET DEFAULT command in the command procedure before you reference any files.

If you are submitting a command procedure that requires parameters, you must use the /PARAMETERS qualifier with the SUBMIT command. The following command submits the command procedure CLEAN.COM in the [ACCOUNT] directory. BILLS.DAT and RECEIPTS.DAT are passed as parameters.

```
$ SUBMIT [ACCOUNT]CLEAN/NOPRINTER/PARAMETERS=(BILLS.DAT,RECEIPTS.DAT)
```

When the batch job is finished, you get a log file containing the output from the command procedure. This log file is given the same name as the command procedure and a file type of LOG. It is placed in your top level directory. By using the /NOPRINTER qualifier, the log file is not printed.

If you want to print and save the log file, use the /KEEP qualifier with the SUBMIT command, as follows:

```
$ SUBMIT/KEEP  [ACCOUNT]CLEAN-
_$/PARAMETERS=(BILLS.DAT,RECEIPTS.DAT)
```

5.2.6 Displaying Command Lines During Execution

By default, the command lines in a command procedure are not displayed as they are executed. When you want to see each command line as it executes, use the DCL command SET VERIFY. When you create your command procedure, add the SET VERIFY command to the top of the list of commands, as follows:

```
+------------------CLEAN.COM---------------------+
| $ ! Print all versions and purge               |
| $ SET VERIFY                                   |
| $ INQUIRE OLD_FILE "File"                      |
| $ IF OLD_FILE .EQS. "" THEN GOTO ERR           |
| $ PRINT 'OLD_FILE';*                           |
| $ PURGE 'OLD_FILE'                             |
| $ EXIT                                         |
| $ ERR:                                         |
| $ WRITE SYS$OUTPUT "No files printed or purged." |
| $ EXIT                                         |
+------------------------------------------------+
```

When you invoke the command procedure CLEAN, you will see every command line as it executes (except the SET VERIFY command, which is not displayed), as follows:

```
$@ CLEAN
!Print all versions and purge
$ INQUIRE OLD_FILE "File"
File:memos.dat
memos.dat
$ IF OLD_FILE .EQS. ""THEN GOTO ERR
$ PRINT MEMOS.DAT
Job MEMOS (queue HAPPY_PRINT, entry 301) started on SPHERE$LPA0
$ PURGE MEMOS.DAT
$ EXIT
$
```

If you want to restore the system to its default after the command procedure executes, use the SET NOVERIFY command. For example, in CLEAN.COM, add the following command directly before the EXIT command:

```
$ SET NOVERIFY
```

If you add this line, the system no longer displays command lines in command procedures as they execute.

6 More About DCL Commands

To become familiar with DCL commands, enter the HELP HINTS command. The available DCL commands are organized by function and are listed in the following categories:

- Batch_and_print_jobs
- Creating_processes
- Files_and_directories
- System_management
- Command_procedures
- Developing_programs
- Logical_names
- Terminal_environment
- Contacting_people
- Executing_programs
- Physical_devices
- User_environment

This chapter introduces some of the DCL commands in these categories. For complete information about all available DCL commands, see the *VMS DCL Dictionary.*

6.1 Printing Files

Although you need to know only the DCL command PRINT to make a hard copy of a file, it is useful to know other commands that allow you to prevent a file from being printed after you have issued the PRINT command. This section describes the PRINT command and related commands.

6.1.1 Sending a File to a Queue

To print a file, enter the DCL command PRINT followed by the name of the file you want to print, as follows:

```
$ PRINT LIST.DAT
```

When you enter the PRINT command, the file you specify is placed in a print queue. A file in the print queue is called a job and has a unique number, called a job entry number. When you enter the PRINT command, the system responds with a message indicating the name and job entry number of the job, as well as the name of the print queue. The following message indicates that a job named LIST with a job entry number of 624 was entered on a print queue named PRINT$FUN.

```
Job LIST (queue PRINT$FUN, entry 624) started on PRINT$FUN
```

For more information about printing files, see Chapter 2. For detailed information about all available qualifiers with the PRINT command, see the *VMS DCL Dictionary*.

6.1.2 Looking at Jobs in the Print Queue

You can see which files are in the print queue by entering the following DCL command:

```
$ SHOW QUEUE queue-name
```

If you do not specify a queue name with the SHOW QUEUE command, you see a list of all available queues. The following example shows how to see the files in the print queue named PRINT$FUN:

```
$ SHOW QUEUE PRINT$FUN
Print queue PRINT$FUN
Jobname      Username     Entry     Status
-------      --------     -----     ------
LIST         BELLINI        624     Printing
```

6.1.3 Removing a Job from the Print Queue

When you have a job waiting in the print queue, and you want to remove it before it starts printing, enter the following DCL command:

```
$ DELETE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your job by entering the SHOW QUEUE command. When you use the DELETE/ENTRY command, the system notifies you that it has deleted the job by issuing the following message:

```
Job NAMES (queue PRINT$WORK, entry 755) completed
%JBC-E-JOBDELETE, job deleted before execution
```

This message indicates that a file named NAMES with a job entry number of 755 was deleted from a print queue named PRINT$WORK before it was printed.

6.1.4 Stopping a Job That is Currently Printing

If you want to stop a file that is currently printing, enter the following DCL command:

```
$ STOP/QUEUE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your job by entering the SHOW QUEUE command. The system notifies you that the job has been stopped with the following message:

```
Job ANOTHER (queue HORACE$PRINT, entry 755) completed
%JBC-E-JOBABORT, job aborted during execution
```

6.2 What Is a Batch Job?

A batch job is a file containing a series of commands (and optionally input data) that is submitted to the operating system for execution. A batch job is executed in a process of its own. Therefore, batch jobs allow you to have two or more processes doing different things at the same time. For example, you may have a command procedure that you want to execute, but you also may want to use your terminal interactively. Instead of waiting for your command procedure to finish executing before doing interactive work, you can submit the command procedure as a batch job. Because a batch job is executed in a process of its own, you can work interactively at the same time.

6.2.1 Starting Batch Jobs

To submit the command procedure LOBSTER.COM as a batch job, enter the following DCL command:

```
$ SUBMIT LOBSTER.COM
```

The SUBMIT command requests the operating system to place a command procedure in a batch job queue and displays the following message:

```
Job LOBSTER (queue BATCH$FUN, entry 442) started on BATCH$FUN
```

If you want the system to notify you when the job is complete, use the /NOTIFY qualifier with the SUBMIT command.

While the system processes your batch job, you can continue interactive use of your terminal.

6.2.2 Looking at Jobs in the Batch Queue

To see the command procedures in the batch queue, enter the following DCL command:

```
$ SHOW QUEUE queue-name
```

The following response indicates that a command procedure named LOOK.COM with a job entry number of 442 was submitted by user SULLIVAN and is currently executing:

```
Jobname      Username      Entry      Status
-------      --------      -----      ------
LOBSTER         SULLIVAN        442      Executing
```

If you specified the /NOTIFY qualifier, the following message is displayed when the batch job finishes executing:

```
Job LOBSTER (queue BATCH$FUN, entry 442) completed
$
```

6.2.3 Removing a Job from the Batch Queue

When you have a batch job waiting in the batch queue, and you want to remove it before it starts executing, enter the following DCL command:

```
$ DELETE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your batch job by entering the SHOW QUEUE command. When you use the DELETE/ENTRY command, the system notifies you that it has deleted the batch job by issuing the following message:

```
Job LOBSTER (queue BATCH$FUN, entry 442) completed
%JBC-E-JOBDELETE, job deleted before execution
```

6.2.4 Stopping a Job That is Currently Executing

If you want to stop a batch job that is currently executing, enter the following DCL command:

```
$ STOP/QUEUE/ENTRY=job-entry-number queue-name
```

You can see the job entry number of your batch job by entering the SHOW QUEUE command. If you specified the /NOTIFY qualifier, the system will notify you that the job has been stopped with the following message:

```
Job LOOK (queue BATCH$FUN, entry 442) completed
%JBC-E-JOBABORT, job aborted during execution
```

6.3 Sorting, Searching, Appending, Comparing, and Copying Files

The following sections introduce five DCL commands to help you when working with files:

- SORT
- SEARCH
- APPEND
- DIFFERENCES
- COPY

6.3.1 Reorganizing Lists

When you want to reorganize a list of items in a file, you can use the DCL command SORT. You must specify an input file to be sorted as well as a name for the newly sorted file, as follows:

```
$ SORT input-file output-file
```

The following example shows a file named SIMPLE_SORT.DAT containing a list of colors:

```
gold
brown
yellow
blue
silver
maroon
green
beige
mauve
```

When you enter the SORT command, the list is reorganized alphabetically according to the first letter of each word.

```
$ SORT simple_sort.dat new_simple_sort.dat
```

The newly sorted file, named NEW_SIMPLE_SORT.DAT, follows:

```
beige
blue
brown
gold
green
maroon
mauve
silver
yellow
```

In a file with more than one column of items, the SORT command moves entire lines of information, not just the first column of items. But, by default, all the columns of items are sorted by the first letter of the first word in the first column. The following example shows a file named SORT_NAMES.DAT containing three columns of information (name, social security number, and profession):

Saxon, Nicholas	749-38-2317	teacher
Able, George	238-90-5674	writer
Drendon, Marka	948-50-3749	writer
Ralston, Celia	263-72-4677	dancer
Briggs, Georgia	374-83-3526	artist
Gregwitz, Marna	478-52-0026	guitarist

When you enter the SORT command, the names are sorted by the first letter of the first word:

```
$ SORT sort_names.dat new_sort_names.dat
```

The newly sorted file, NEW_SORT_NAMES.DAT, follows:

Able, George	238-90-5674	writer
Briggs, Georgia	374-83-3526	artist
Drendon, Marka	948-50-3749	writer
Gregwitz, Marna	478-52-0026	guitarist
Ralston, Celia	263-72-4677	dancer
Saxon, Nicholas	749-38-2317	teacher

By specifying qualifiers with the SORT command, you can sort information by number instead of by character. You can specify that the file be sorted in descending order (ZYXWVUTSR or 87654321) instead of ascending order (ABCDEFG or 12345678). For detailed information about sorting files, see the *VMS Sort/Merge Utility Manual.*

6.3.2 Searching for a String

To find a specific string of text within a file, use the DCL command SEARCH, as follows:

```
$ SEARCH file-spec search-string
```

If the search string contains any lowercase letters or nonalphanumeric characters, enclose it with quotation marks. For example, a file named TEST.DAT contains the following paragraph of text:

```
When you type commands, qualifiers, or parameters you do not
always need to type the full word.  In fact, you never have
to type more than the first four characters, and in many
cases you can type only one or two characters.  The rule to
follow is: you must type at least the minimum number of
characters necessary to make the command unique.
```

To search for the string "characters" in the file named TEST.DAT, enter the following command string:

```
$ SEARCH test.dat "characters"
```

The SEARCH command will display each line containing the string "characters" (lines 3,4, and 6), as follows:

```
to type more than the first four characters, and in many
cases you can type only one or two characters.  The rule to
characters necessary to make the command unique.
```

For detailed information about the SEARCH command, see the *VMS DCL Dictionary.*

6.3.3 Appending Files

When you want to add the contents of one file to the contents of another file, you can use the DCL command APPEND, as follows:

```
$ APPEND input-file-spec output-file-spec
```

The input-file-spec is the name of the file you want to add to the end of the output-file-spec. The system prompts you for the name of the input-file-spec first and then the name of the output-file-spec, as follows:

```
$  APPEND
_From: input-file-spec
_To: output-file-spec
```

The following example shows how to add the contents of a file named ARM.FUN to the end of a file named BODY.FUN. The contents of file ARM.FUN follow:

```
arm arm arm arm
```

The contents of file BODY.FUN follow:

```
body body body
```

Enter the following command string:

```
$ APPEND
_From: ARM.FUN
_To: BODY.FUN
```

Now the contents of the file named ARM.FUN are appended to the end of the file named BODY.FUN, as follows:

```
body body body
arm arm arm arm
```

For detailed information about the qualifiers available with the APPEND command, see the *VMS DCL Dictionary*.

6.3.4 Comparing Files

To compare the contents of one file with the contents of another file, use the DCL command DIFFERENCES, as follows:

```
$ DIFFERENCES
_File 1: first input-file-spec
_File 2: second input-file-spec
```

The DIFFERENCES command lists each record in the first input file that has no match in the second input file. The DIFFERENCES command also lists the line following each unmatched record.

The following example shows two copies of the same letter, COPY1.DAT and COPY2.DAT. There are two differences between them.

```
COPY1.DAT:

     31 December 1988

     Reprint Permission Department
     The Doubleday Company
     2709 Third Avenue
     New York, New York 10022


     Dear Permission Department,

     I  am  currently  writing  a  text  processing  handbook
     for  Digital Equipment  Corporation.   This  handbook
     describes  how to format and process text on the VMS
     ɔperating system.   I  wish  to  make  the handbook more
     interesting by adding the three sentences (marked in your
     copy) from page 22 of your book, "Flying without Fear", as
     part of an example.

COPY2.DAT:

     31 December 1988

     Reprint Permission Department
     The Doubleday Company
     2709 Third Avenue
     New York, New York 10022
     Dear Permissions Department,

     I  am  currently  writing  a  text  processing  handbook
     for  Digital Equipment  Corporation.   This  handbook
     describes  how to format and process text on the VMS
     operating system.   I  wish  to  make  the handbook more
     interesting by adding the three sentences (marked in your
     copy) from page 32 of your book, "Flying without Fear", as
     part of an example.
```

To see the differences between COPY1.DAT and COPY2.DAT, enter the following command string:

```
$ DIFFERENCES
_File 1: COPY1.DAT
_File 2: COPY2.DAT
```

The DIFFERENCES command lists the unmatching lines as it compares both files:

```
************
File DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4
   13        Dear Permission Department,
   14
******
File DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
   13        Dear Permissions Department,
   14
************
************
File DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4
   20        copy) from page 22 of your book, "Flying without Fear", as
   21        part of an example.
******
File DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
   20        copy) from page 32 of your book, "Flying without Fear", as
   21        part of an example.
************
Number of difference sections found: 2
Number of difference records found: 2

DIFFERENCES /IGNORE=()/MERGED=1-
    DISK$DOCUMENT:[KALLAS.BOOK]DIF_COPY.FUN;4-
    DISK$DOCUMENT:[KALLAS.BOOK]NEW_DIF_COPY.FUN;3
```

For detailed information about the qualifiers available with the DIFFERENCES command, see the *VMS DCL Dictionary*.

6.3.5 Copying Files

To copy a file, use the COPY command. You can use it to make copies of files in your default directory, to copy files from one directory to another directory, to copy files from other devices, or to create files consisting of more than one input file.

When you enter the COPY command, specify first the names of the input files you want to copy, then the name of the output file. For example, the following COPY command copies the contents of the file PAYROLL.TST to a file named PAYROLL.OLD:

```
$ COPY PAYROLL.TST PAYROLL.OLD [RET]
```

If a file named PAYROLL.OLD exists, a new version of that file is created with a higher version number.

To copy a file from the directory [MALCOLM] to the subdirectory [MALCOLM.TESTFILES] and give it the new name, OLDFILE.DAT, enter the following command string:

```
$ COPY NEWFILE.DAT [MALCOLM.TESTFILES]OLDFILE.DAT RET
```

When you copy files from devices other than your default disk, you must specify the device name with the COPY command. For example, the following command string copies a file from your default directory onto a disk with the logical name HOMER:

```
$ COPY PAYROLL.TST HOMER: RET
```

Note that the output file specification did not include a file name or file type; the COPY command uses the same directory, file name, and file type as the input file, by default.

Before you can copy any files to or from devices other than system disks, you must gain access to these devices. You do this by following two steps:

1 Mounting the volume with the MOUNT command.

2 Ensuring the volume has a directory for cataloging the file. If no directory exists, use the CREATE command to create one.

Note that the VMS operating system protects against users accessing private volumes and system volumes. For details on the commands and procedures necessary to prepare and use disks and magnetic tapes, see the *Guide to VMS Files and Devices* and the *VMS DCL Dictionary*.

6.4 Controlling the VMS Environment

You can use several DCL commands to modify your user and terminal environments. This section introduces the following commands:

- SET PROMPT
- DEFINE/KEY
- SHOW KEY
- DELETE/KEY
- SHOW PROCESS
- SET PROCESS
- RECALL (CTRL/B)
- SHOW TERMINAL
- SET TERMINAL

6.4.1 Changing the System Prompt

By default, VMS displays a dollar sign ($) when you are at DCL command level. To specify a different prompt for the DCL command level, use the following command:

```
$ SET PROMPT = prompt-string
```

For example, to change your prompt to GOOD_MORNING> , enter the following command string:

```
$ SET PROMPT = good_morning>
GOOD_MORNING>
```

When you want your prompt string to retain lowercase letters, enclose the string in quotation marks ("). Otherwise, letters are automatically converted to uppercase.

When you log out and back in again, the default DCL command prompt, the dollar sign, is restored.

To save a prompt from session to session, enter the SET PROMPT command in your LOGIN.COM file, as follows:

```
$SET PROMPT ="SMILE>"
```

The prompt SMILE> remains in effect until you change or delete the command line in your LOGIN.COM file.

For more information about the SET PROMPT command, see the *VMS DCL Dictionary*.

6.4.2 Saving Time by Defining Keys

You can use the DCL command DEFINE/KEY to assign definitions to keypad keys on VT52, VT100, and VT200 series terminals. Note that the /APPLICATION_KEYPAD qualifier to the SET TERMINAL command must be in effect to use key definitions. (See Section 6.4.6.3.)

When you enter the DEFINE/KEY command, specify a key-name (such as PF1) followed by an equivalence string (such as the DCL command, DIRECTORY):

```
$ DEFINE/KEY key-name "equivalence-string"
```

For example, you can equate the keypad key PF1 to the DCL command DIRECTORY:

```
$ DEFINE/KEY PF1 "DIRECTORY"
```

You must enclose an equivalence string in quotation marks (") if the string contains any spaces.

The following example shows how to equate the keypad key PF2 to the DCL command SHOW TIME:

```
$ DEFINE/KEY PF2 "SHOW TIME"
```

To display a key definition, use the DCL command SHOW KEY followed by the name of the key:

```
$ SHOW KEY PF2
DEFAULT keypad definitions:
  PF2 = "SHOW TIME"
```

To see all currently defined keys, enter the SHOW KEY command with the /ALL qualifier.

```
$ SHOW KEY/ALL
DEFAULT keypad definitions:
  PF1 = "DIRECTORY"
  PF2 = "SHOW TIME"
```

When you enter the SHOW KEY command specifying an undefined key, DCL displays the following message:

```
%DCL-W-UNDKEY, DEFAULT key PF3 is undefined
```

To undefine keys, use the DELETE/KEY command, as follows:

```
$ DELETE/KEY key-name
```

For example, to undefine the key PF1, enter the following command:

```
$ DELETE/KEY PF1
%DCL-I-DELKEY, DEFAULT key PF1 has been deleted
```

To delete all your key definitions, enter the DCL command DELETE/KEY with the /ALL qualifier.

When you log out, any key definitions you have made during the session are deleted. To maintain key definitions from session to session, list them in your LOGIN.COM file. The following example shows three lines containing key definitions from a LOGIN.COM file:

```
$DEFINE/KEY PF1 "DIRECTORY"
$DEFINE/KEY MINUS "SHOW TIME"
$DEFINE/KEY KP1 "SET DEFAULT [TORTELLINI.LETTERS]"
```

For detailed information about available keys for definition and qualifiers to use with the DEFINE/KEY command, see the *VMS DCL Dictionary*.

6.4.3 Looking at Processes

Use the DCL command SHOW PROCESS to display information about your process. Figure 6–1 displays the results of the SHOW PROCESS command with all the parts labeled.

Use the DCL command SET PROCESS to change some of the characteristics of your process. You need various privileges to use all available qualifiers with the SET PROCESS command. For detailed information about restrictions, see the *VMS DCL Dictionary*. If you want to change your process name, enter the SET PROCESS command with the /NAME qualifier, as follows:

```
$ SET PROCESS/NAME = string
```

Figure 6–1 Using the SHOW PROCESS Command

```
❶31-DEC-1988 10:13:40.82   ❷TTA0:              ❸User: FLYNN
❹Pid: 2100044B   ❺Proc. name: Macro_writer    ❻UIC: [DOC1,FLYNN]
❼Priority:   4   ❽Default file spec: WORK9:[FLYNN.EXAMPLES]

❾Devices allocated: TTA0:
```

❶ Date and time the SHOW PROCESS command is issued

❷ Device name of the current SYS$INPUT device (your terminal)

❸ User name

❹ Process identification number, which is a hexadecimal number that uniquely identifies a process

❺ Process name

❻ User identification code specifying the type of access available to the owner, which is either a pair of numbers, or a name (or optionally, a pair of names)

❼ Base execution priority

❽ Default directory

❾ Devices allocated to the process

To set your process name to Macro_Writer, enter the following command:

```
$ SET PROCESS/NAME = "Macro_Writer"
```

To see the newly changed process name, enter the SHOW PROCESS command.

When you log out and back in again, your original default process name returns. To save a process name from session to session, enter the SET PROCESS/NAME command in your LOGIN.COM file, as follows:

```
$SET PROCESS/NAME = "Large_Marge"
```

The process name Marvelous_Mabel will remain in effect until you change or delete the command line in your LOGIN.COM file.

6.4.4 Displaying Previously Entered Commands

During an interactive session, you may enter many command strings. To see these command strings after you have processed them, enter the DCL command RECALL or press CTRL/B. The DCL command RECALL allows you to display a previously entered command. If you want to process the command again, press RETURN. The RECALL command can act as a reminder for previous commands or can save key strokes for long command strings.

The following example shows how to enter a DCL command (SHOW TIME), and then use the RECALL command to display it again:

```
$ SHOW TIME
  31-DEC-1988 13:43:37
$ RECALL
$ SHOW TIME
  31-DEC-1988 13:44:04
```

Add the /ALL qualifier to the RECALL command to display the last twenty commands you have entered. The most recently entered command is number 1. The next to the last command entered is number 2. The RECALL command itself is never assigned a number.

```
$ RECALL/ALL
 1 MAIL
 2 SHOW TIME
 3 SET DEFAULT [BELLINI.OPERA.COSTUMES]
 4 EDIT BUSINESS_MATTERS.DAT
 5 SHOW TIME
 6 MAIL
 7 SET DEFAULT [BELLINI.MUSICAL.STAGING]
 8 SHOW USERS
 9 ERASE
10 EDIT TICKET_SALES.DAT
11 TYPE TICKET_SALES.DAT
12 MAIL
13 SHOW TIME
14 PRINT TICKET_SALES.DAT
15 SHOW TERMINAL
16 ERASE
17 EDIT LETTER.FUN
18 PRINT LETTER.FUN
19 SHOW TIME
20 MAIL
```

You can reenter a command by typing its number with the RECALL command, as the following example shows:

```
$ RECALL 14
$ PRINT TICKET_SALES.DAT
```

You can also enter the first character of a command to recall a command. For example, to recall the last command entered that began with the letter "T" (command number 11), enter the following command:

```
$ RECALL T
$ TYPE TICKET_SALES.DAT
```

If more than one command begins with the letter "T", you must specify more than one character.

6.4.5 Showing Terminal Characteristics

To see the characteristics set for your terminal, enter the DCL command SHOW TERMINAL. This command produces a display similar to the following:

```
Terminal: _VTA732:    Device_Type: VT240_Series     Owner: Macro_writer(FLYNN)

   Input:   9600      LFfill:  0     Width:   80      Parity: None
   Output:  9600      CRfill:  0      Page:   24
```

```
Terminal Characteristics:
   Interactive        Echo                Type_ahead          No Escape
   No Hostsync        TTsync              Lowercase           Tab
   Wrap               Scope               No Remote           No Holdscreen
   No Eightbit        Broadcast           No Readsync         Form
   Fulldup            No Modem            No Local_echo       No Autobaud
   No Hangup          No Brdcstmbx        No DMA              No Altypeahd
   Set_speed          Line Editing        Overstrike editing  No Fallback
   No Dialup          No Secure server    No Disconnect       No Pasthru
   No SIXEL Graphics  No Soft Characters  No Printer Port     Numeric Keypad
   ANSI_CRT           No Regis            No Block_mode       Advanced_video
   No Edit_mode       DEC_CRT
```

6.4.6 Changing Terminal Characteristics

This section introduces the following qualifiers you can use with the SET TERMINAL command to modify your terminal characteristics:

/ECHO
/NOECHO
/INSERT
/OVERSTRIKE
/NUMERIC_KEYPAD
/APPLICATION_KEYPAD
/WIDTH
/WRAP
/NOWRAP

For detailed information about all the available qualifiers, see the *VMS DCL Dictionary*.

6.4.6.1 Using the /ECHO and the /NOECHO Qualifiers

The qualifiers /ECHO and /NOECHO control whether the terminal displays (echoes) the input lines it receives. By default, /ECHO is set, allowing you to see what you type. When you use the /NOECHO qualifier, you do not see what you type.

The following example shows how the /ECHO and /NOECHO qualifiers work. The fourth and sixth lines contain what you type, which is invisible. The first undisplayed command you enter is SHOW TIME (fourth line). The second undisplayed command you enter is SET TERMINAL/ECHO (sixth line). Notice that after you enter the SET TERMINAL/ECHO command on the sixth line, the commands you type are displayed again.

```
$ SHOW TIME
31-DEC-1988  15:43:46
$ SET TERMINAL/NOECHO
$
31-DEC-1988  15:46:15
$
$ SHOW TIME
31-DEC-1988  15:43:46
```

6.4.6.2 Using the /INSERT and the /OVERSTRIKE Qualifiers

The /INSERT and /OVERSTRIKE qualifiers allow you to either insert characters or type over characters when you are editing command strings. By default, the /OVERSTRIKE qualifier is in effect.

Perform the following three steps to see how the /OVERSTRIKE qualifier works:

1 Type the SHOW TIME command, but do not press RETURN.

```
$ SHOW TIME
```

2 Press the left arrow key four times, placing the cursor over the "T" in the word "TIME".

3 Type the word "USERS". As you type each character, "USERS" replaces "TIME".

```
$ SHOW USERS
```

Perform the following four steps to see how the /INSERT qualifier works:

1 Enter the SET TERMINAL/INSERT command.

2 Type the SHOW TIME command, but do not press RETURN.

```
$ SHOW TIME
```

3 Press the left arrow key four times, placing the cursor over the "T" in the word "TIME".

4 Type the word "USERS". As you type each character, the word "TIME" moves to the right.

```
$ SHOW USERSTIME
```

To reinstate the overstrike characteristic, enter the SET TERMINAL/OVERSTRIKE command.

6.4.6.3 Using the /NUMERIC_KEYPAD and /APPLICATION_KEYPAD Qualifiers

The /NUMERIC_KEYPAD and /APPLICATION_KEYPAD qualifiers allow you to use the keys on the keypad to type numbers and punctuation marks or to type defined keys. By default, /NUMERIC_KEYPAD is in effect, enabling you to type the numbers and punctuation marks shown on the keypad. For example, when you press the COMMA key on the keypad, you see a comma (,).

To take advantage of the /APPLICATION_KEYPAD qualifier, define one of the keypad keys using the DEFINE/KEY command. For example, enter the following command string:

```
$ DEFINE/KEY PF1 "SHOW TIME"
%DCL-I-DEFDKEY, DEFAULT key PF1 has been defined
```

Then enter the SET TERMINAL/APPLICATION_KEYPAD command. When you press the PF1 key, you see the SHOW TIME command. See the *VMS DCL Dictionary* for more detailed information about the DEFINE/KEY command and a table listing key designations for various terminals.

6.4.6.4 Using the /WIDTH Qualifier

You can use the /WIDTH qualifier with the SET TERMINAL command to specify the number of characters on each input or output line. Use the following syntax:

```
$ SET TERMINAL/WIDTH=n
```

"N" can be any number from 0 through 255. When you specify a number that is greater than 80, your screen displays "thin" characters (on VT100-type and VT200 series terminals). For example, enter the following command:

```
$ SET TERMINAL/WIDTH=81
```

Immediately, all the characters on your screen become narrow. To return to your previous character size, enter the SET TERMINAL/WIDTH command again specifying a number less than 81.

6.4.6.5 Using the /WRAP and /NOWRAP Qualifiers

The /WRAP and /NOWRAP qualifiers control whether or not the terminal generates a carriage return/line feed when it reaches the end of the line. You can determine the end of a line by setting the terminal width using the /WIDTH qualifier.

To see how a line wraps, press the letter "w" and hold it down. You will see a string of w's on the screen. When the string of w's gets to the end of the line, it goes to the beginning of the next line (wraps). By default, any string of characters you enter wraps unless you enter the SET TERMINAL/NOWRAP command.

6.5 Working with Physical Devices

A physical device is capable of receiving and storing data. Two examples of physical devices are magnetic tapes and disks, which are mass storage devices. You can perform the following tasks by using the corresponding DCL commands:

- Allocate or deallocate a device (ALLOCATE, DEALLOCATE)
- Make a storage device available or unavailable for processing (MOUNT, DISMOUNT)
- Format a storage device (INITIALIZE)
- Save or restore files from storage devices (BACKUP)
- Set characteristics for:
  - Devices (SET DEVICE)
  - Magnetic tapes (SET MAGTAPE)
  - Line printers (SET PRINTER)
  - Mounted volumes (SET VOLUME)

- Display characteristics for:
  - Devices (SHOW DEVICES)
  - Magnetic tapes (SHOW MAGTAPE)
  - Line printers (SHOW PRINTER)

For information on using these DCL commands to manipulate physical devices, see the *Guide to VMS Files and Devices.*

Glossary

account: Enables access to the system software (command interpreters, compilers, utilities).

ASCII: American Standard Code for Information Interchange. ASCII is the standard format for sending readable text. It is a code used by many computers to translate letters, numbers, and symbols from a keyboard into machine code, and back into symbols again.

An ASCII file can be read both by people and by computers.

assembler: Language processor that translates a source program containing assembly language directives and machine instructions into an object module.

assembly language: Machine oriented programming language. VAX MACRO is the assembly language for the VAX computer.

assignment statement: Definition of a symbol name to use in place of a character string or numeric value. Symbols can define synonyms for system commands or can be used for variables in command procedures.

batch: Mode of processing in which all commands to be executed by the operating system and, optionally, data to be used as input to the commands are placed in a file or punched onto cards and submitted to the system for execution.

batch job: A noninteractive process.

baud rate: The speed at which a terminal transmits or receives characters.

buffer: A temporary storage area.

command: An instruction or request for the system to perform a particular action. An entire command string consists of the command name with any parameters and qualifiers.

command interpreter: The operating system component responsible for reading and translating interactive and batch commands. The default command interpreter for the VMS operating system interprets the DIGITAL Command Language (DCL).

command string: A command with any parameters or qualifiers.

command procedure: File containing a sequence of commands to be executed by the operating system. The command procedure can be submitted for execution at the terminal or as a batch job.

compiler: Language processor that translates a source program containing high-level language statements (for example, FORTRAN) into an object module.

concatenate: To link together in a series.

CPU: Central Processing Unit. It is the hardware that handles all calculating and routing of input and output (I/O), as well as executing images. The CPU is the part of the computer that actually computes.

cursor: A flashing indicator used on video terminals to point to the screen position where the next character will appear. It is called a "cursor" because it shows the "course" or direction the printed or typed line will follow.

data: A general term used for any representation of facts, concepts, or instructions in a form suitable for communication, interpretation, or processing. When commands prompt you for command elements, they are asking you for data to process.

DCL: DIGITAL Command Language. It provides a means of communication between the user and the operating system. DCL is designed for ease of use. Commands are English words, and if necessary elements are not typed in, DCL will prompt you for them.

debugger: Interactive program that allows you to display and modify program variables during execution and to step through a program to locate and detect programming errors.

default: Value supplied by the system when a user does not specify a required command parameter or qualifier.

default disk: The disk from which the system reads and to which the system writes, by default, all files that you create. The default is used whenever a file specification in a command does not explicitly name a device.

delimiter: A character that separates, terminates, or organizes elements of a character string or statement. For example, in the file specification, STORIES.DAT, the period (.) is the delimiter that enables the system to tell the difference between the file name STORIES and the file type DAT.

device: Any peripheral hardware connected to the processor and capable of receiving, storing, or transmitting data. Line printers and terminals are examples of record-oriented devices. Magnetic tapes and disks are examples of mass-storage devices. Terminal line interfaces and interprocessor links are examples of communications devices. All devices have names either in the form ddnn:, where dd is a two-letter mnemonic, nn is an octal number, and the colon(:) is a required terminator or as a logical name.

device name: Identification of a physical device (for example, DBA2) or a logical name (for example, SYS$OUTPUT) that is equated to a physical device name.

directory: A file that briefly catalogs a set of files stored on disk or tape. The directory includes the name, type, and version number of each file in the set.

disk: High-speed, random-access devices. There are several kinds of disks. Floppy disks are small, flexible disks. Hard disks are either fixed in place or removable. Removable disk types include a single hard disk enclosed in a protective case and a stacked set of disks enclosed in a protective case.

echo: The display of a character either on the screen or hard copy that was typed on a terminal keyboard. Terminals are dual devices, sending input and receiving output. Typing on the terminal is sending input to the computer. Echoing is receiving output from the computer.

editor: Program that creates or modifies files. In VMS, the default system editor is EDT, which is interactive.

equivalence name: Character string equated to a logical name. When a command or program refers to a file or device by its logical name, the system translates the logical name to its predefined equivalence name.

error message: Sent by the system when some action you have requested fails. Each error message identifies the particular part that detected the error. Most error messages result from typing mistakes or mistakes in syntax. Often, you can correct the error by retyping the command.

field: Usually refers to a portion of a command or a command element. For example, the file name and file type are two fields of a file specification.

file: Collection of data treated as a unit; generally used to refer to data stored on magnetic tapes or disks.

file name: The name component of a file specification.

file specification: Unique identification of a file. A file specification describes the physical location of the file, as well as file name and file type identifiers that describe the file and its contents.

file type: The type component of a file specification. A file type generally describes the nature of a file or how it is used. For example, FOR indicates a FORTRAN source program.

folder: A subdivision of a file in which you can store mail messages.

form feed: Analogous to a line feed, but instead of moving down one line to resume printing, the line printer moves past the perforations in the paper to the top of a new form or page. A form feed consists of a number of line feeds.

function keys: Keyboard keys that send special signals to the operating system. Function keys are referred to as Fx, where x is the number associated with that key. For example, by pressing F9 in MAIL you are telling the system you want to forward a message.

global symbol: A symbol defined with an assignment statement recognized in any command procedure that is executed.

hanging: A terminal or process that appears to be going nowhere or doing nothing. Hung terminals are sometimes described as static, dormant, or locked. Hung terminals may result from a busy system, a crash, or unavailability of system resources.

hardcopy terminal: A terminal that prints output on paper.

hardware: The physical computer equipment, including such mechanical devices as the line printer, the terminals, the mass-storage devices, and so forth.

header page: Printed page at the beginning of a listing that identifies the printed file.

help file: A text file in a format suitable for use with the HELP command. Help files can provide up to nine levels of search.

high-level language: Transportable programming language, such as BASIC, FORTRAN, or COBOL. Programs in these languages are not tied to a particular kind of computer, and must be compiled. Each programming statement in a high-level language is translated into several machine-language instructions.

image: Output from the linker, created from processing one or more object modules. An image is the executable version of a program.

input file: File containing data to be transferred into the computer.

Often input and output files are confused. DCL usually prompts for these files, but most system utilities require you to identify your input and output files by position in a command line. Be sure of the syntax for the command you are using.

interactive: Mode of communication with the operating system in which a user enters a command, and the system executes it and responds.

job: (1) The accounting unit equivalent to a process; jobs are classified as batch or interactive. (2) A print job.

K: A unit for measuring the size of memory or similar resources. K is short for kilo and is used roughly to mean 1000, although actually K is equal to 1024.

keyboard: An input device that can be operated similarly to a typewriter.

keypad: The set of keys next to the main keyboard on a terminal.

keyword: A command name, qualifier, or option. Keywords must be typed verbatim or truncated according to the rules of DCL.

lexical function: A command language construct that the command interpreter evaluates and substitutes before it parses a command string. Lexical functions return information about the current process (for example, the UIC or default directory) and about character strings (for example, their length or the location of substrings).

line editor: Program that allows you to make additions and deletions to a file line by line.

line printer: An output device that prints one line at a time. It is used for printing large amounts of output.

linker: Program that creates an executable program, called an image, from one or more object modules produced by a language compiler or assembler. Programs must be linked before they can be executed.

local symbol: A symbol defined with an assignment statement that is recognized only within the command procedure in which it is defined.

log in: To perform a sequence of actions at a terminal that establishes a user's communication with the operating system and sets up default characteristics for the user's terminal session.

log out: To terminate interactive communication with the operating system. The LOGOUT command executes the procedure and ends a terminal session.

logical name: Substitute character string used to refer to files or devices by other than their original names. Frequently used as a form of shorthand. A command or program can refer to a file by a logical name; the logical name can be equated to an equivalence name at any time; when the command or program refers to the logical name, the system translates the logical name to its defined equivalence name.

logical name table: A table that contains a set of logical names and their equivalence names for a particular process, a particular group, or the system.

machine code: A sequence of binary machine instructions in a form executable by the computer.

magnetic tape: Medium on which data can be stored and accessed.

mass-storage device: An input/output device where data and other types of files are stored while they are not being used. Typical mass-storage devices include disks, magnetic tapes, floppy disks, and DECtapes. Each mass-storage device uses a particular magnetic medium to hold its data.

memory: A series of physical locations into which data or instructions can be placed in the form of binary words. Each location in memory can be addressed, and its contents can be altered.

network: A collection of interconnected computer systems.

node: An individual computer system in a network that can communicate with other computer systems in the network.

node specification: The component of a file specification that identifies the location of a computer system in a network of computer systems.

object module: Output from a language compiler or assembler that can be linked with other object modules to produce an executable image. An object module is a file with a file type of OBJ.

operating system: The system software that controls the operations of the computer.

output file: File to which the computer transfers data.

parameter: Defines what the command acts upon. A parameter can be a file specification, a symbol value passed to a command procedure, or a word defined by DCL.

parse: Separate a command string into its elements to interpret it.

password: Character string that user provides at login to show authorization to access the account.

peripheral devices: Any device used for input/output operations with the CPU. Terminals, line printers, and disks are peripheral devices.

priority: A rank assigned to a process to determine its precedence in obtaining system resources when the process is running.

program: A series of instructions written for the computer to follow.

prompt: A symbol used by the system to signal that the system is ready to accept input from you.

protection code: Specifies what access different categories of system users can have to the file and what they can do to the file when they access it.

qualifier: Describes how the operation of a command occurs. A qualifier is always preceded by a slash character (/).

queue: A line of items waiting to be processed.

random access: A method of accessing files or records in memory or on a device. Using this method, records are accessed using their known address, instead of using the records that precede them in storage. On a random access disk, all information is equally accessible, regardless of its location. Compare to *sequential access.*

range specification: Used with the EDT line editor to define the lines to be affected by the editing command.

real-time system: A system in which inquiries and data entered on a terminal are processed based on need and priority.

reverse video: A feature of some terminals that reverses the default video contrast. If having black figures on a white background is the default, reverse video displays white on black. Reverse video is used with some EDT keypad commands to highlight a range of text.

RMS: Record Management Services. RMS is a sophisticated set of routines used to open and close files, read from files, and extend and delete files.

scrolling: When more than one screen of output is sent to a video terminal, the output scrolls up. New output appears at the bottom of the screen and eventually disappears when it reaches the top.

sequential access: Records or files are read one after another in the order in which they appear in the on the device. Magnetic tape is a sequential-access medium. For example, if you are half way through a tape and wish to read some record that is a third of the way through the tape, you must go back to the beginning of the tape. You must read through until you get to the record that you want or until the end of the tape.

software: The collection of images, procedures, rules, and documentation associated with the operation of a particular computer system. For example, the operating system is software.

source program: A program written in text form that must be compiled or assembled to be used.

string: A sequence of characters. When you use an editor to search for a word or phrase, you are searching for a string. The sequence of characters that forms a command is often called a command string.

subdirectory: A directory within a directory. Subdirectories can contain additional files belonging to the owner of the directory, and offer a method of file organization.

subroutine: A routine that can be used as part of another routine. For instance, you might write a routine to print the time in large numbers on your terminal. You could then call that routine as a subroutine in some task that required printing the time in large numbers.

switch hook character: The Phone Utility prompt (%). When you are "talking" to someone in the PHONE utility, you must type the switch hook character (a percent sign by default) before you enter a PHONE command.

symbol: A variable that is defined or given a value. For instance you can equate the symbol ST with the value SHOW TIME. When you type ST, the computer translates the symbol, giving it the value SHOW TIME.

syntax: The exact form that a command must follow for the system to attempt to execute it. Misspelled words are the most common syntax errors.

system operator: Person responsible for maintaining the system. Within small systems, the job may be combined with that of the system manager or informally divided among several people. The responsibilities of the operator include changing ribbons, rebooting the system, and keeping records.

system manager: Person who makes resources available to users and sets up restrictions governing the use of such resources.

terminal: Hardware communication device with a typewriter-like keyboard that receives and transmits information between users and the system.

timesharing: A system in which each user gets equal computer time in turn. This is in contrast to the allocation based on need and priority in a real-time system.

UIC: User identification code. This code identifies a user by a group number and a member number. (Both numbers are enclosed in brackets.)

user name: Name by which the system identifies a particular user. To gain access to the system, a user specifies a user name followed by a password.

utility: A general-purpose program provided with the operating system that performs tasks. Common utility functions are editing and file handling.

version number: Numeric component of a file specification. When a file is edited, its version number is increased by one.

video display terminal: A terminal where your keystrokes and system responses are displayed on a screen similar to that of a television.

volume: The largest logical unit of the file structure. A volume contains files and corresponds to a physical unit of mass storage.

wildcard character: A symbol that can be used with many DCL commands in place of all or part of a file specification to refer to several files rather than specifying them individually.

Index

Reader's Comments

Introduction to VMS
AA–LA04A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________

______________________________ Phone ______________

— 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

Introduction to VMS
AA–LA04A–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

____

What I like best about this manual is ______________________________

____

____

What I like least about this manual is ______________________________

____

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ____________

Company ______________________________ Date ____________

Mailing Address ______________________________

______________________________ Phone ____________

— 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

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here

Cut Along Dotted Line